RectifyFlow: Plan Upfront, Rectify Selectively for Cost-Efficient Edge–Cloud Tool-Using Agents

Anonymous authors
Paper under double-blind review

Abstract

Tool-using agents increasingly rely on cloud LLMs to execute complex multi-step workflows, but repeated LLM calls incur substantial monetary cost. Edge-side small models (SMs) offer a cheaper alternative, yet their weaker reasoning can compromise execution quality. Edge–cloud collaboration therefore promises a better quality–cost trade-off, but existing approaches remain inefficient for tool use: iterative execution triggers frequent cloud interactions, while local decision errors can propagate through dependent operations and cause costly replanning. We propose `RectifyFlow`, a cost-efficient edge–cloud framework that minimizes cloud involvement while containing edge-side execution errors. First, Dual-View Plan Construction produces structured plans that guide the edge SM through multiple dependent operations with limited cloud intervention, avoiding repeated cloud reasoning during execution. Second, when execution deviates from the plan, Dependency-Aware Plan Rectification identifies the deviation source, traces its downstream impact, and selectively repairs only the affected plan components, rather than regenerating the entire plan. Across BFCL and ToolSandbox, `RectifyFlow` retains 94.6% and 91.8% of Cloud-only performance while reducing cloud API cost by approximately 2.5× and 2.1×, respectively. Against nine representative baselines, `RectifyFlow` achieves approximately 1.5× higher task performance at 1.3× lower cost, demonstrating a substantially improved quality–cost trade-off for tool-using agents.

1 Introduction

Recent advances in large language models (LLMs) have enabled tool-use agents capable of autonomously coordinating multi-step tool operations through execution feedback (Drouin et al., 2024; Yang et al., 2024). Such agents have demonstrated practical potential in emerging applications, including software engineering automation (*e.g.*, OpenAI's Codex) and personal assistance (*e.g.*, Meta's Muse). To ensure high-quality task execution, existing tool-use agent systems typically rely on cloud LLMs. However, these advanced LLMs introduce prohibitive monetary costs. To mitigate this cost, existing studies explore edge–cloud collaboration as a promising paradigm by deploying small models (SMs) on edge devices and dynamically deciding whether each task step should be executed by the SM or cloud LLM based on its complexity. This paradigm leverages SMs for low-cost execution of simple steps and LLMs for complex steps requiring stronger reasoning capability, thereby reducing inference costs while maintaining task quality (Shao et al., 2025; Dong et al., 2026; Liu et al., 2026).

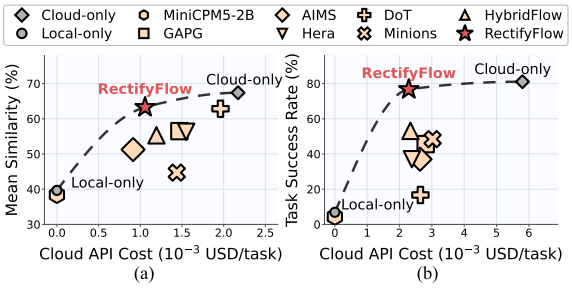

Figure 1: `RectifyFlow` achieves a favorable quality–cost trade-off on both datasets.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Existing edge–cloud collaboration methods typically follow two paradigms: iterative model routing (Liu et al., 2026; Zhang et al., 2026) and subtask allocation (Shao et al., 2025; Dong et al., 2026). The former progressively determines whether each step should be processed by an edge SM or a cloud LLM during execution, while the latter first decomposes a task into multiple subtasks and then assigns appropriate models to different subtasks. Despite their demonstrated effectiveness (Dong et al., 2026), dynamic dependencies in tool use introduce limitations when applying these methods to tool-use agents: `(L1)` Intermediate tool outputs make subsequent operations and their complexity difficult to anticipate, requiring frequent edge–cloud interactions during execution. This increases cloud invocation overhead and limits the achievable reduction in cost. `(L2)` Assigning more execution to edge-side SMs can reduce cloud usage, but their lower decision reliability may introduce latent errors into interdependent tool operations (Li et al., 2025; Ye et al., 2025). Such errors may remain latent until they affect subsequent dependent operations, requiring additional cloud-side correction or re-execution and increasing overall cost.

To address these limitations, we propose `RectifyFlow`, a novel cost-efficient edge–cloud collaborative agent framework for tool use. Specifically, inspired by the Plan-and-Execute paradigm (Erdogan et al., 2025; Xu et al., 2023), we leverage a cloud LLM to generate multi-step execution plans in advance and delegate their execution to an edge-side SM, thereby reducing frequent edge–cloud interactions. Meanwhile, the generated plans provide execution guidance for SMs, constraining their execution process and improving local execution reliability. Together, these designs aim to address the aforementioned limitations, thereby enabling a better quality–cost trade-off. However, realizing this objective still presents two key challenges:

- **Providing sufficient execution guidance.** The plan should enable the SM to reliably execute as many steps as possible under limited reasoning capability. However, tool outcomes are unavailable during planning, requiring the plan to capture operational dependencies while leaving runtime decisions to the SM. Insufficient guidance increases the SM's reasoning burden and error risk, whereas incorrect assumptions may affect subsequent dependent operations.
- **Recovering from execution deviations.** Runtime observations may reveal conditions not captured during planning, causing deviations in tool-use tasks. Recovering from such deviations requires identifying their source and affected downstream operations. Without this analysis, replanning may retain invalid decisions or unnecessarily modify valid steps, leading to repeated recovery, redundant execution, and additional cloud invocations.

To this end, `RectifyFlow` introduces two complementary mechanisms: *(i) Dual-View Plan Construction*, where the cloud LLM captures tool dependencies in a DAG-based dependency view and derives a linear execution view to guide reliable edge-side execution with fewer cloud interactions. *(ii) Dependency-aware Plan Rectification*, where `RectifyFlow` performs dependency-aware rectification when edge execution deviates by localizing the deviation source and its downstream impact, and then selectively patches only the affected nodes. This preserves valid execution results and avoids unnecessary replanning, redundant execution, and additional LLM invocations. As shown in Fig. 1, extensive experiments against nine representative baselines show that `RectifyFlow` achieves the best observed quality–cost trade-off among the compared methods, retaining about 93% of cloud-only performance while reducing API cost by about 56% on average.

2 Related Work

Tool-Using Agents. Tool-using agents augment LLMs with external tools for complex task execution (Qin et al., 2024; Lu et al., 2025). Recent studies further improve tool selection, invocation, and multi-step execution through structured tool representations and reinforcement learning (Qian et al., 2025; Wang et al., 2025; Liu et al., 2025). However, these methods primarily focus on improving the tool-use capability of individual models and often rely on cloud LLMs for strong task performance, incurring substantial monetary costs.

Edge–Cloud Collaboration for LLM Agents. Edge–cloud collaboration aims to balance performance and cost through model routing and task allocation. RouteLLM learns query routing policies from preference data (Ong et al., 2025), while DoT, AIMS, and Minions optimize subtask allocation or cloud–edge coordination through dependency analysis and local processing (Shao et al., 2025; Liu et al., 2026; Narayan et al., 2025). HybridFlow and Hera further explore dependency-aware routing

and cost-aware optimization for efficient collaboration (Dong et al., 2026; Zhang et al., 2026). However, as shown in §5, these methods mainly target static reasoning scenarios and remain challenged by tool-use tasks with continuous interactions and complex dependencies.

Plan-and-Execute Methods. Existing planning and execution methods improve agent execution accuracy through explicit execution guidance and error recovery. ReAct (Yao et al., 2023), ReWOO (Xu et al., 2023), and LLMCompiler (Kim et al., 2024) enhance reasoning, planning efficiency, and tool scheduling, while Plan-and-Act (Erdogan et al., 2025) and AgentFlow (Li et al., 2026) further optimize planning through replanning and learning. For execution recovery, AdaPlanner (Sun et al., 2023) refines plans using feedback, and ADaPT (Prasad et al., 2024) recursively decomposes failed subtasks for continued execution. However, applying these methods to edge–cloud collaborative tool use remains challenging due to plan uncertainty in dynamic environments and the need for precise, low-cost plan correction.

3 Problem Formulation

We consider tool-use tasks where an agent $\mathcal{M}$ completes a user task q by iteratively selecting tools from a candidate set $\mathcal{T}$ and constructing their arguments $\mathbf{a}_t$ based on interaction history h_t: $(u_t, \mathbf{a}_t) = \mathcal{M}(q, h_t)$. Each tool invocation produces feedback that is appended to the history for subsequent decisions, forming an iterative tool-use process. To reduce the high API cost of repeated cloud LLM calls, we consider an edge–cloud collaborative agent consisting of a cloud LLM $\mathcal{M}_{\mathrm{C}}$ and an edge-side SM $\mathcal{M}_{\mathrm{E}}$, where $\mathcal{M}_{\mathrm{C}}$ first generates an initial plan $\mathcal{P}^{(0)}$ for task q and $\mathcal{M}_{\mathrm{E}}$ executes it locally based on the plan and interaction history.

Despite this arrangement, tool outcomes and environment changes remain unknown during cloud-side planning, which may invalidate planning assumptions and cause execution deviations. Meanwhile, the limited capability of the edge SM makes reliable execution challenging. The first problem is therefore to construct a plan that captures task dependencies while providing sufficient execution guidance for $\mathcal{M}_{\mathrm{E}}$. We denote such a plan as $\mathcal{P}^{(0)} = (p_1, \ldots, p_{n_0})$, where each $p_r \in \mathcal{P}^{(0)}$ is a planned step node that provides the edge SM with explicit execution constraints and task-relevant information, thereby enabling reliable plan-guided execution.

Correspondingly, the second problem is to localize execution deviations and selectively rectify affected plan components. When the edge SM cannot proceed, it reports execution evidence ξ_ℓ to the cloud LLM, which identifies the deviation source, determines the affected node set $\mathcal{R} \subseteq \{p_1, \ldots, p_{n_0}\}$, and generates a selective rectification δ that updates affected nodes to obtain the revised plan $\mathcal{P}^{(1)}$. The edge model then resumes execution with the updated plan, and the process iterates until task completion or the predefined limit is reached.

Our goal is to address these two problems and achieve an effective quality–cost trade-off in tool-use tasks. We measure this trade-off by task quality $Q = \mathbb{E}_q[\mathrm{Score}(q)]$ and cloud API cost $C = \mathbb{E}_q[C_{\mathrm{cloud}}(q)]$. Ideally, Q approaches the quality of Cloud-only execution (only $\mathcal{M}_{\mathrm{C}}$), while C approaches the cloud cost of Local-only execution (only $\mathcal{M}_{\mathrm{E}}$).

4 RectifyFlow

4.1 Overview

We propose `RectifyFlow`, a cost-efficient edge–cloud collaborative agent framework for tool use, as illustrated in Fig. 2. ❶ `RectifyFlow` first performs Dual-View Plan Construction: the cloud LLM organizes the required tool operations and their dependencies into a DAG-based *dependency view*, and derives a linear *execution view* to guide reliable edge-side execution. ❷ The edge SM follows the execution view sequentially, grounding arguments and invoking applicable tools based on runtime observations. It performs bounded local recovery for recoverable argument errors; when execution cannot be restored locally, it sends the execution context and interruption evidence to the cloud LLM. ❸ The cloud LLM performs dependency-aware plan rectification by tracing backward from the interrupted node to localize the deviation source, then analyzing its downstream dependencies to determine the candidate reconsideration scope and selectively revise affected nodes. ❹ Finally, the cloud LLM returns the revised plan and execution instruction. The edge SM updates

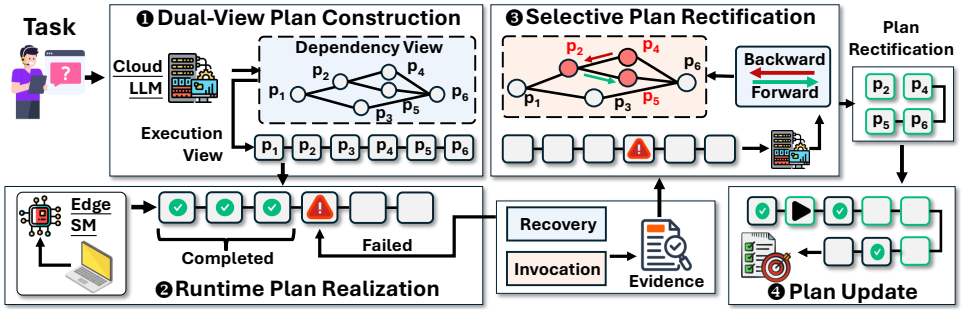

Figure 2: **Overview of `RectifyFlow`.**

its local view, preserves valid results, and resumes execution from the specified point. The process iterates until task completion.

4.2 Dual-View Plan-Guided Edge–Cloud Execution

In this section, our goal is to construct a plan that enables reliable edge SM execution with limited cloud intervention. Rule-based executors handle runtime variations through predefined logic with adaptation limited to the cases covered by their rules (Kim et al., 2024), whereas edge SMs can leverage plan guidance and feedback to adapt tool arguments and recover from minor execution errors. However, their limited reasoning capability requires explicit execution constraints. We therefore introduce a Dual-View Plan that provides step-wise guidance for execution.

Dual-View Plan Construction. Tool-use tasks often involve interdependent operations, with earlier tool outputs or state changes constraining subsequent actions. We capture these dependencies with a directed acyclic graph (DAG) (Ye et al., 2025). Given a user task q, the cloud LLM $\mathcal{M}_{\mathrm{C}}$ selects task-relevant tools $\mathcal{U} \subseteq \mathcal{T}$ and constructs an initial plan as a dependency-annotated DAG: $\mathcal{G}^{(0)} = \big(\{p_1, \ldots, p_{n_0}\}, E^{(0)}\big)$. This graph serves as the plan's *dependency view*, representing the planned tool operations and their inferred dependencies. Each node $p_r \in \mathcal{G}^{(0)}$ specifies a semantic objective d_r and a selected tool $u_r \in \mathcal{U}$ for achieving it, while $E^{(0)}$ records the data, state, and control dependencies among operations.

The dependency view $\mathcal{G}^{(0)}$ captures operation relationships but does not specify how the edge SM should execute each operation under runtime conditions. Without explicit guidance, the SM must infer dependencies and track execution states, leading to long contexts that strain its limited capacity and increase hallucination risk. `RectifyFlow` therefore constructs a linear *execution view* $\mathcal{P}^{(0)} = (p_1, \ldots, p_{n_0})$ by selecting a valid topological order from $\mathcal{G}^{(0)}$. It guides $\mathcal{M}_{\mathrm{E}}$ through step-wise execution with explicit dependency information.

In the execution view, `RectifyFlow` maps the dependencies in $E^{(0)}$ into output references, argument-grounding specifications, and execution criteria, which respectively encode required tool outputs, guide argument construction, and enable $\mathcal{M}_{\mathrm{E}}$ to select applicable nodes based on tool feedback:

Task. Check Beijing's weather on D; if rain is forecast, set a 9 a.m. umbrella reminder.

Node p_1:
d_1: Retrieve the weather forecast.
u_1: `get_weather` (`city`; `date`)
θ_1: Set `city` to Beijing; format D as `YYYY-MM-DD` for `date`.
ϕ_1: $\varnothing$ (unconditional).
Outputs: `date`; `forecast`

Node p_2:
d_2: Remind the user to bring an umbrella.
u_2: `create_reminder` (`time`; `message`)
θ_2: Format p_1[`date`] and 09:00 as `YYYY-MM-DD HH:mm` for `time`. Set `message` to an umbrella reminder with a brief summary of p_1[`forecast`].
ϕ_2: p_1[`forecast`] indicates rain.
...

- **Output references.** If operation p_r depends on output field f from a preceding operation p_i, the dependency is represented in p_r by the symbolic reference $p_i[f]$. After p_i is executed, its result is stored in the binding map $\mathcal{B}$, allowing $\mathcal{M}_{\mathrm{E}}$ to directly retrieve $\mathcal{B}(p_i[f])$ when executing p_r. This enables required but unavailable information to be specified in advance, supporting continuous guidance throughout execution.

- **Argument grounding.** $\mathcal{M}_{\mathrm{C}}$ further generates an argument-grounding specification θ_r for each node p_r, describing how to construct the arguments of tool u_r from the task information and outputs of preceding nodes in the execution view. At runtime,

$\mathcal{M}_{\mathrm{E}}$ follows θ_r and uses the preceding-node outputs in $\mathcal{B}$, together with the semantic objective d_r, to instantiate the concrete tool arguments $\mathbf{a}_r$.

• **Execution criteria.** $\mathcal{M}_{\mathrm{C}}$ generates an execution criterion ϕ_r for each branch-dependent node p_r, encoding the corresponding branch condition in the dependency view, while ϕ_r is left empty for other nodes. At runtime, $\mathcal{M}_{\mathrm{E}}$ evaluates ϕ_r using the results recorded in $\mathcal{B}$. If the condition is not satisfied, p_r is skipped and execution proceeds to the next node p_{r+1}.

Combining these components, each executable node in the execution view is represented as:

$$p_r = (d_r, u_r, \theta_r, \phi_r), \quad p_r \in \mathcal{P}^{(0)}. \tag{1}$$

We jointly represent the complementary views as $\Pi^{(0)} = (\mathcal{G}^{(0)}, \mathcal{P}^{(0)})$. While $\mathcal{P}^{(0)}$ provides execution order and node-level guidance for $\mathcal{M}_{\mathrm{E}}$, $\mathcal{G}^{(0)}$ guides cloud-side rectification by localizing affected dependencies and avoiding unnecessary updates and re-execution (see §4.3). Since $\mathcal{G}^{(0)}$ may also contain errors, both views are updated consistently during rectification.

Runtime Plan Realization with Conditional Cloud Invocation. After receiving the initial dual-view plan $\Pi^{(0)}$, $\mathcal{M}_{\mathrm{E}}$ sequentially executes only $\mathcal{P}^{(0)}$, while retaining $\mathcal{G}^{(0)}$ for subsequent cloud-side rectification. For each node p_r, $\mathcal{M}_{\mathrm{E}}$ first evaluates ϕ_r. If ϕ_r is empty or satisfied, it instantiates $\mathbf{a}_r = \mathcal{M}_{\mathrm{E}}(d_r, u_r, \theta_r, \mathcal{B})$ and invokes u_r; otherwise, p_r is skipped. The outputs of each executed node are recorded in $\mathcal{B}$, while h_r maintains the interaction history up to p_r, including tool calls, arguments, observations, and recovery outcomes.

When tool u_r reports invalid arguments $\mathbf{a}_r$, such as missing fields, malformed formats, or incompatible values, $\mathcal{M}_{\mathrm{E}}$ attempts local recovery to obtain $\widetilde{\mathbf{a}}_r$ by reinterpreting θ_r and the tool specification to identify formatting or computation errors. The cloud model $\mathcal{M}_{\mathrm{C}}$ is invoked when *(i)* feedback at the interruption reveals missing or incorrect plan nodes that block further execution, or *(ii)* $\mathcal{M}_{\mathrm{E}}$ fails to complete the current node within the predefined recovery budget. In either case, $\mathcal{M}_{\mathrm{C}}$ performs dependency-aware plan rectification based on the accumulated execution evidence.

To support plan rectification by $\mathcal{M}_{\mathrm{C}}$, $\mathcal{M}_{\mathrm{E}}$ provides the following revision context:

$$\chi = \left(\Pi^{(0)}, \mathcal{B}, h, \xi\right), \tag{2}$$

where interruption evidence ξ_ℓ identifies the interrupted node and summarizes the corresponding tool feedback and recovery results.

4.3 Dependency-aware Plan Rectification

Backward Deviation Localization. When $\mathcal{M}_{\mathrm{C}}$ receives the evidence χ, where ξ identifies p_r as the interrupted node, `RectifyFlow` compares the planned execution in $\mathcal{P}^{(0)}$ with the observed outputs in $\mathcal{B}$ and interaction history h. It then traces backward from p_r through the dependency structure in $\mathcal{G}^{(0)}$ to identify the earliest node whose execution behavior conflicts with the observed evidence, which is treated as the deviation source p_s. By preserving $\mathcal{G}^{(0)}$ as the dependency reference, `RectifyFlow` avoids repeated dependency reconstruction from $\mathcal{P}^{(0)}$, reducing cloud reasoning overhead and preventing inconsistent dependency updates during successive rectifications:

$$p_s = \mathcal{M}_{\mathrm{C}}\big(\chi_\ell, p_r\big), \qquad p_s \rightsquigarrow^* p_r, \tag{3}$$

where $\rightsquigarrow^*$ denotes reflexive reachability in the dependency view. p_s denotes the earliest deviation source that explains the observed inconsistency. It may correspond to an existing node in $\mathcal{P}^{(0)}$, or a missing dependency node revealed by execution feedback. In the latter case, $\mathcal{M}_{\mathrm{C}}$ augments the dependency structure by introducing the missing node and establishing its relation to p_r before subsequent rectification.

Forward Impact Analysis. After identifying the deviation source p_s, `RectifyFlow` performs forward impact analysis over the dependency structure. Although the deviation is observed at the interrupted node p_r, the underlying issue at p_s may invalidate the execution guidance of subsequent unexecuted nodes in the original plan $\Pi^{(0)}$. Therefore, `RectifyFlow` identifies all downstream nodes that depend on p_s in the dependency view as candidates for reconsideration:

$$\mathcal{R} = \left\{ p_j \in \mathcal{P}^{(\ell)} \mid p_s \rightsquigarrow^* p_j \right\}. \tag{4}$$

Selective Plan Rectification. The candidate scope $\mathcal{R}_\ell$ identifies nodes whose execution may be affected by the deviation. However, modifying these nodes alone may not fully restore execution consistency, since previously executed operations may have introduced environment changes recorded in the interaction history h. Therefore, $\mathcal{M}_{\mathrm{C}}$ jointly considers the dependency-constrained scope $\mathcal{R}$ and the observed state changes in h to generate a structured rectification δ:

$$\delta = \mathcal{M}_{\mathrm{C}}\left(\Pi^{(0)}, \mathcal{R}_\ell, h\right). \tag{5}$$

where δ denotes a rectification patch consisting of the nodes selected for plan modification and state-consistency recovery, *i.e.*, $\delta = (\delta^{plan}, \delta^{state})$. The plan-level rectification δ^{plan} is a subset of nodes selected from the candidate scope $\mathcal{R}$ that require modification to restore execution consistency, while preserving unaffected nodes. The state-consistency rectification δ^{state} contains additional rollback nodes introduced to address side effects from previously executed operations recorded in h. Based on δ, the initial plan is updated to obtain the rectified plan $\Pi^{(1)}$.

Once rectification is complete, $\mathcal{M}_{\mathrm{C}}$ sends the updated dual-view plan $\Pi^{(1)}$ together with an execution instruction to $\mathcal{M}_{\mathrm{E}}$. The instruction specifies how to resume execution, including the starting node, nodes that have been successfully completed and can be preserved, and nodes requiring re-execution after rectification. Based on the updated plan and execution instruction, $\mathcal{M}_{\mathrm{E}}$ performs plan realization again, while retaining unaffected nodes and valid execution results. This process iterates over multiple rectification rounds until the task is completed or the execution terminates.

5 Experiment

5.1 Experimental Setup

Datasets and Splits. We evaluate `RectifyFlow` on two tool-use benchmarks: ToolSandbox (Lu et al., 2025) and BFCL-V4 (Patil et al., 2025). ToolSandbox is designed for stateful, multi-turn tool-use scenarios and contains 1,032 tool-use tasks. BFCL-V4 includes 800 multi-turn function-calling samples covering four categories. For baselines requiring training, we randomly sample tasks from each category to construct training and test splits. Specifically, ToolSandbox is divided into 206 training tasks and 826 test tasks, while BFCL-V4 is divided into 160 training tasks and 640 test tasks. All methods are evaluated on the same test sets, whereas `RectifyFlow` requires no additional training and is directly evaluated in the test phase.

Baselines. We first include Cloud-only and Local-only as performance boundary baselines. Cloud-only relies entirely on cloud LLMs for reasoning and execution, while Local-only performs all inference using edge-side SMs. We then compare `RectifyFlow` with nine representative baselines, including edge–cloud collaboration approaches (Hera (Zhang et al., 2026), AIMS (Liu et al., 2026), Minions (Narayan et al., 2025), GAPG (Fang et al., 2025), DoT (Shao et al., 2025), and HybridFlow (Dong et al., 2026)), plan-and-execute methods (Plan-and-Act (Erdogan et al., 2025) and ReWOO (Xu et al., 2023)), and MiniCPM5-2B (MiniCPM, 2025), an edge-oriented agent model used to examine the impact of stronger SM capabilities on tool-use performance.

Models and Hyperparameters. We use DeepSeek-V4-Flash (Xu et al., 2026) as the default cloud LLM and Qwen3-8B (Yang et al., 2025) as the default edge-side SM. To evaluate the generality of `RectifyFlow`, we further consider different edge models, including Qwen3-1.7B (Yang et al., 2025), Qwen3-4B (Yang et al., 2025), and quantized Llama-3-8B-Instruct (Q4_K_M) (Grattafiori et al., 2024), as well as GPT-5.6-Terra as the cloud LLM. For hyperparameter settings, we set the maximum local recovery budget of the edge SM to one attempt, balancing edge computation overhead and execution reliability.

Metrics. For ToolSandbox, we adopt its official similarity score, which measures the consistency between execution trajectories and target states through milestone completion and final task achievement. For BFCL-V4, we use the official task success rate. Both metrics are denoted as **Avg. Perf.** for unified comparison. We estimate API cost in USD based on official model pricing and consumed input/output tokens. For system overhead, we report latency, communication volume, RTT, and P95 latency. RTT measures cumulative edge–cloud round-trip communication time, while P95 latency captures tail execution behavior caused by unstable or repeated interactions.

Table 1: Average performance and cloud API cost on BFCL and ToolSandbox.

Methods	BFCL			ToolSandbox		
	Avg. Perf.↑ ($\times 10^{-2}$)	API Cost↓ ($\times 10^{-3}$ \$)	vs. Cloud-only Perf.↑ / Cost↓	Avg. Perf.↑ ($\times 10^{-2}$)	API Cost↓ ($\times 10^{-3}$ \$)	vs. Cloud-only Perf.↑ / Cost↓
Cloud-only	81.09	5.803	100.00% / 100.00%	70.65	2.173	100.00% / 100.00%
Local-only	6.88	0.000	8.48% / 0.00%	38.26	0.000	54.16% / 0.00%
MiniCPM5-2B	4.38	0.000	5.39% / 0.00%	38.20	0.000	54.07% / 0.00%
GAPG	45.63	2.809	56.26% / 48.41%	59.95	1.469	84.85% / 67.59%
AIMS	37.03	2.635	45.66% / 45.41%	53.90	0.914	76.30% / 42.06%
Hera	37.03	2.383	45.66% / 41.06%	58.78	1.550	83.20% / 71.35%
DoT	16.72	2.646	20.62% / 45.60%	66.70	1.962	94.41% / 90.28%
Minions	48.44	3.023	59.73% / 52.10%	44.59	1.438	63.11% / 66.15%
HybridFlow	53.13	2.337	65.51% / 40.27%	55.85	1.194	79.05% / 54.94%
`RectifyFlow`	**76.72**	**2.286**	**94.61% / 39.39%**	**64.83**	**1.057**	**91.77% / 48.64%**

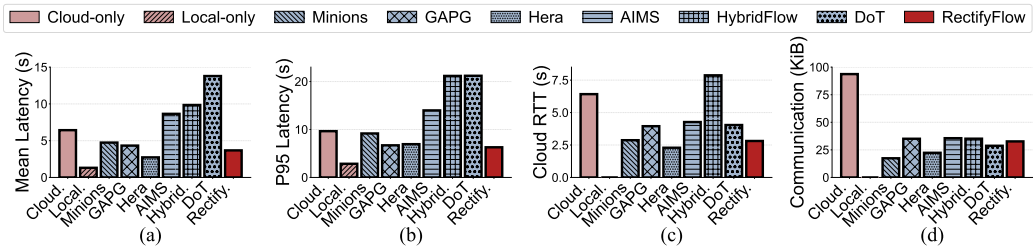

Figure 3: Comparison of latency and communication overhead across different methods.

All edge-side components are deployed on a single PC with an NVIDIA RTX 4090 GPU. Unless stated otherwise, models use default configurations without fine-tuning. We report the mean results over three runs. Cloud API costs are estimated from recorded input/output tokens using official pricing. Detailed dataset and baseline settings are provided in Appendix A.

5.2 Numerical Results

Performance Comparison. We first evaluate the quality–cost trade-off of `RectifyFlow` against representative edge–cloud collaboration baselines. As shown in Tab. 1, `RectifyFlow` achieves competitive performance with substantially reduced cloud API costs. In BFCL, it reaches 76.72 Avg. Perf., retaining 94.61% of Cloud-only performance with only 39.39% of its cost. On ToolSandbox, it achieves 64.83 Avg. Perf. while reducing cost by 51.36% and preserving 91.77% of Cloud-only performance. Compared with existing approaches, `RectifyFlow` provides a more favorable quality–cost balance. MiniCPM5-2B eliminates cloud usage but suffers from limited performance, while other collaboration methods improve performance at the expense of higher API costs caused by frequent interactions. For example, DoT achieves 66.70 Avg. Perf. on ToolSandbox but requires 1.962×10^{-3} \$, while `RectifyFlow` achieves comparable performance with only 1.057×10^{-3} \$. This advantage stems from `RectifyFlow`'s ability to reduce redundant cloud interactions while maintaining reliable tool execution.

Latency and Communication Overhead. To evaluate the practical applicability of `RectifyFlow`, we further analyze latency and communication overhead under different edge–cloud collaboration strategies. As shown in Fig. 3, `RectifyFlow` achieves a lower execution latency while maintaining moderate communication overhead. It reaches 3.68s average latency and 6.29s P95 latency, outperforming methods such as HybridFlow and DoT that rely on frequent cloud decisions. This improvement is enabled by plan-guided execution, where the cloud LLM provides structured guidance in advance, allowing the edge model to execute multiple dependent operations locally with fewer interactions. Meanwhile, `RectifyFlow` introduces 32.68 KiB/task communication overhead, comparable to other collaboration methods. By triggering cloud-side plan rectification only when local execution becomes unreliable, it balances communication efficiency and execution reliability. Additional edge–side resource analysis shows that `RectifyFlow` maintains low computational and memory overhead, with detailed results provided in Appendix A.3.

Table 2: Performance and cloud API cost of `RectifyFlow` with different edge models.

Edge Model	ToolSandbox			BFCL		
	Avg. Perf.↑ ($\times 10^{-2}$)	**API Cost↓** ($\times 10^{-3}$ \$)	**vs. Cloud-only** Perf.↑ / Cost↓	**Avg. Perf.↑** ($\times 10^{-2}$)	**API Cost↓** ($\times 10^{-3}$ \$)	**vs. Cloud-only** Perf.↑ / Cost↓
Cloud-only	70.65	2.173	100.00% / 100.00%	81.09	5.803	100.00% / 100.00%
Qwen3-1.7B	64.23	1.085	90.91% / 49.93%	75.63	2.325	93.27% / 40.07%
Qwen3-4B	**65.96**	1.167	**93.36%** / 53.70%	73.44	2.448	90.57% / 42.19%
Qwen3-8B	64.83	1.057	91.76% / 48.64%	**76.72**	**2.286**	**94.61% / 39.39%**
Llama3-8B	65.64	**1.043**	92.91% / **48.00%**	76.41	2.306	94.23% / 39.74%

Table 3: Performance and cloud API cost comparison with Plan-and-Execute methods.

Methods	ToolSandbox			BFCL		
	Avg. Perf.↑ ($\times 10^{-2}$)	**API Cost↓** ($\times 10^{-3}$ \$)	**vs. Cloud-only** Perf.↑/Cost↓	**Avg. Perf.↑** ($\times 10^{-2}$)	**API Cost↓** ($\times 10^{-3}$ \$)	**vs. Cloud-only** Perf.↑/Cost↓
Cloud-only	70.65	2.173	100.00%/100.00%	81.09	5.803	100.00%/100.00%
ReWOO	70.10	2.402	99.22%/110.54%	69.22	4.052	85.36%/69.83%
Plan-and-Act	69.07	2.143	97.76%/98.62%	76.41	7.173	94.23%/123.61%
ReWOO (SM)	61.06	1.704	86.43%/78.42%	48.28	2.137	59.54%/36.83%
Plan-and-Act (SM)	63.14	1.561	89.37%/71.84%	44.38	4.197	54.73%/72.32%
`RectifyFlow`	**64.83**	**1.057**	**91.76%/48.64%**	**76.72**	**2.286**	**94.61%/39.39%**

Generalization Analysis. We evaluate `RectifyFlow` under different LLM settings.

Edge SM Generalization. We evaluate `RectifyFlow` with four edge-side SMs: Qwen3-1.7B, Qwen3-4B, Qwen3-8B, and quantized Llama3-8B. As shown in Tab. 2, all configurations achieve comparable performance while substantially reducing API costs relative to Cloud-only. These results suggest that `RectifyFlow`'s quality–cost benefits generalize across the evaluated SMs.

Cloud LLM Generalization. We further evaluate `RectifyFlow` with GPT-5.6-Terra as the cloud LLM while keeping the edge model and all framework components unchanged. As shown in Fig. 4, `RectifyFlow` consistently achieves a favorable quality–cost trade-off under the new cloud model, approaching the performance of Cloud-only execution while substantially reducing API expenditure. This result indicates that the effectiveness of `RectifyFlow` does not rely on a specific cloud LLM, but mainly stems from its framework-level design, including dual-view planning and dependency-aware rectification. By separating task-level guidance from execution-time correction, `RectifyFlow` can effectively exploit different cloud model capabilities while maintaining reliable tool execution. Detailed latency, communication, and edge-side resource analyses under GPT-5.6-Terra are provided in Appendix A.3, further demonstrating consistent system efficiency across different cloud configurations.

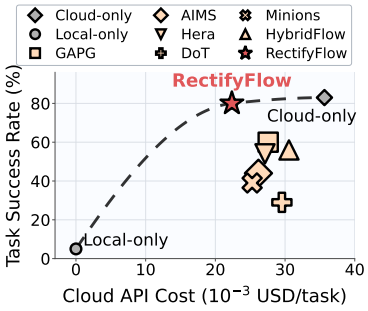

Figure 4: Cost–performance comparison on BFCL.

Comparison with Plan-and-Execute. Tab. 3 compares `RectifyFlow` with Plan-and-Execute methods adapted to edge–cloud collaboration. We consider cloud-based variants with cloud planning/execution and SM-based variants with cloud planning and edge execution. Cloud-based variants achieve high performance but incur substantial API costs; *e.g.*., Plan-and-Act retains 94.23% of Cloud-only performance on BFCL with 123.61% cost. SM-based variants reduce cloud usage but suffer from limited execution capability. These results highlight the challenges of applying Plan-and-Execute to tool-use tasks (see §1). `RectifyFlow` overcomes these limitations through dependency-aware planning and selective rectification.

Case Study. Fig. 5 presents representative execution trajectories, where ❶, ❷ and ❸ correspond to DoT, Minions, and HybridFlow, respectively. Existing methods frequently alternate between cloud LLM and edge SM, causing repeated cloud calls and redundant execution. In contrast, `RectifyFlow` enables continuous local execution through plan guidance and selectively rec-

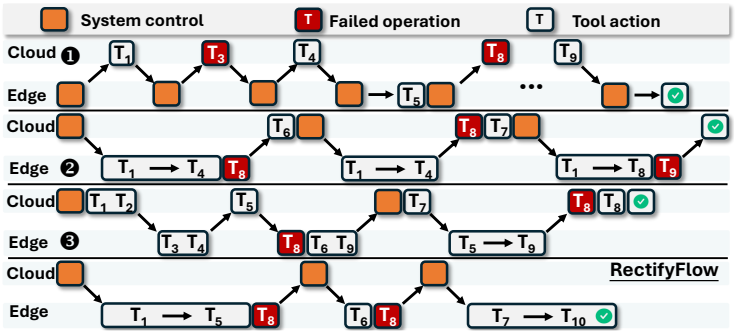

Figure 5: Execution trajectories of different methods.

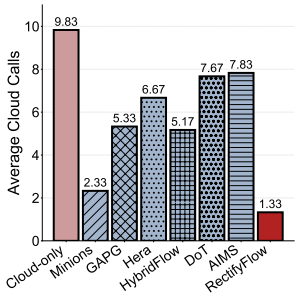

Figure 6: Average cloud LLM calls.

Table 4: Sensitivity analysis and ablation study of `RectifyFlow`.

(A) Hyperparameter Analysis on BFCL				(B) Ablation Study on ToolSandbox			
Budget	Avg. Perf.↑ ($\times 10^{-2}$)	Budget	Avg. Perf.↑ ($\times 10^{-2}$)	Variant	Avg. Perf.↑ ($\times 10^{-2}$)	API Cost↓ (10^{-3} \$)	Cloud Calls↓ (per task)
1	76.72	4	77.03	`RectifyFlow`	**64.83**	**1.057**	**1.39**
2	74.38	5	76.25	w/o DPC	50.60	2.052	2.60
3	76.88	w/o Repair	73.28	w/o DPR	61.93	1.254	1.63

tifies affected nodes, preserving execution progress with fewer cloud interactions. As shown in Fig. 6, `RectifyFlow` requires only 1.33 cloud LLM calls on average, compared with 9.83 calls for Cloud-only, demonstrating that plan-guided execution and dependency-aware rectification effectively reduce cloud overhead while maintaining reliable execution. Detailed analysis of plan quality and rectification effectiveness is provided in Appendix A.3, showing that `RectifyFlow` progressively improves plan alignment through effective rectification. The complete prompts are provided in Appendix A.3.

Sensitivity Analysis. We analyze the effect of the local recovery budget on BFCL. As shown in Tab. 4 (A), `RectifyFlow` remains stable across different budgets. One recovery attempt improves performance to 76.72%, compared with 73.28% without recovery, while larger budgets yield limited gains because complex deviations require dependency-aware plan rectification rather than repeated local retries. Thus, a small recovery budget is sufficient for effective local error handling.

Ablation Study. We conduct ablation studies to analyze the contributions of Dual-View Plan Construction (DPC) and Dependency-aware Plan Rectification (DPR), with results summarized in Tab. 4 (B). Removing DPC decreases performance from 64.83 to 50.60 and increases average cloud calls from 1.39 to 2.60, indicating that DPC is crucial for reliable edge execution. Removing DPR (*i.e.*, using full plan updates) maintains comparable performance but increases cloud calls to 1.63 and incurs higher API costs, showing that global updates introduce unnecessary overhead. Overall, DPC reduces cloud dependency through structured execution guidance, while DPR improves correction efficiency through selective plan updates. A fine-grained ablation analysis of components is provided in Appendix A.3, demonstrating the necessity of each component in `RectifyFlow`.

6 Conclusion

The paper addressed the high cost of tool-use agents by proposing a new edge–cloud collaborative framework named `RectifyFlow`. Specifically, by integrating Dual-View Plan Construction and Dependency-aware Plan Rectification, `RectifyFlow` improves edge-side execution reliability and enables precise, efficient selective plan updates, substantially reducing cloud API costs with minimal performance degradation. Extensive experiments on tool-use benchmarks demonstrate that `RectifyFlow` achieves a favorable quality–cost trade-off compared with existing approaches, retaining approximately 93% of Cloud-only performance while reducing cloud API costs by about 56% on average.

REFERENCES

Jiangwen Dong, Jiayu Li, Tianhang Zheng, and Wanyu Lin. Hybridflow: Resource-adaptive subtask routing for efficient edge-cloud llm inference. *arXiv preprint arXiv:2512.22137*, 2026.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vázquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks? In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, 2025.

Wenzhi Fang, Dong-Jun Han, Liangqi Yuan, Evan Chen, and Christopher Brinton. Bridging on-device and cloud llms for collaborative reasoning: A unified methodology for local routing and post-training. *arXiv preprint arXiv:2509.24050*, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.

Xingzuo Li, Kehai Chen, Yunfei Long, Xuefeng Bai, Yong Xu, and Min Zhang. Generator-assistant stepwise rollback framework for large language model agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, 2025.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. In *International Conference on Learning Representations (ICLR)*, 2026.

Shiyi Liu, Haiying Shen, Shuai Che, Mahdi Ghandi, and Mingqin Li. AIMS: cost-efficient llm-based agent deployment in hybrid cloud-edge environments. In *Proceedings of the 21st European Conference on Computer Systems, EuroSys 2026, McEwan Hall/The University of Edinburgh, Edinburgh, Scotland, UK, April 27-30, 2026*, 2026.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, 2025.

Team MiniCPM. Minicpm4: Ultra-efficient llms on end devices. *arXiv preprint arXiv:2506.07900*, 2025.

Avanika Narayan, Dan Biderman, Sabri Eyuboglu, Avner May, Scott W. Linderman, James Zou, and Christopher Ré. Cost-efficient collaboration between on-device and cloud language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, 2025.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, 2024.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024.

Chenyang Shao, Xinyuan Hu, Yutang Lin, and Fengli Xu. Division-of-thoughts: Harnessing hybrid language model synergy for efficient on-device agents. In *Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025*, 2025.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. Toolhop: A query-driven benchmark for evaluating large language models in multi-hop tool use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, 2025.

Yuxin Zhang, Mengxue Hu, Zheng Lin, Xiaoyi Fan, Fan Xie, Zihan Fang, Jing Yang, Wenjun Zhu, Zhiwen Chen, Chengfei Lv, et al. Hera: Learning long-horizon coordination for device-cloud collaborative llm agents. *arXiv preprint arXiv:2605.24598*, 2026.

A Appendix

AI use statement

AI tools were used in this work to assist with language refinement, including grammar correction, sentence restructuring, and improving the clarity and fluency of expression. All AI-assisted modifications were carefully reviewed and verified by the authors. The authors take full responsibility for the final content and claims presented in this paper.

Reproducibility Statement

To facilitate reproducibility, we provide detailed descriptions of the proposed framework, experimental settings, and evaluation protocols in the paper and appendix. The source code, experiment configurations, and scripts will be publicly available through the anonymous repository: anonymous repository.

A.1 Algorithm of RectifyFlow

Algorithm 1 illustrates the overall workflow of `RectifyFlow`. Given a task q, the cloud model $\mathcal{M}_{\mathrm{C}}$ first constructs an initial dual-view plan $\Pi^{(0)} = (\mathcal{G}^{(0)}, \mathcal{P}^{(0)})$, where the dependency view captures tool dependencies and the execution view provides an ordered sequence of executable nodes with node-level guidance. Each node is augmented with output references, argument-grounding specifications, and execution criteria, and is then executed sequentially by the edge model $\mathcal{M}_{\mathrm{E}}$. During execution, $\mathcal{M}_{\mathrm{E}}$ maintains the binding map $\mathcal{B}$ and interaction history h, which record generated outputs, execution progress, tool feedback, and environment changes. For each node, the edge model evaluates its execution criterion, constructs tool arguments according to the provided guidance, and performs local recovery within a limited budget for recoverable failures. When execution cannot proceed, `RectifyFlow` collects the execution evidence $\chi_\ell = (\Pi^{(\ell)}, \mathcal{B}, h, \xi_\ell)$ and invokes the cloud model for dependency-aware rectification, as detailed in Algorithm 2. The rectification process first identifies the deviation source by tracing backward over the dependency view, then analyzes downstream impact to determine the candidate scope, and finally generates a selective rectification patch that combines necessary plan updates and state-consistency recovery operations. The revised dual-view plan $\Pi^{(\ell+1)}$ and execution instruction are sent back to the edge model, which preserves valid results, applies required updates, and resumes execution from the specified point. This iterative process continues until the task is completed or termination is reached.

A.2 Details of the experimental setup

A.2.1 Dataset Details.

ToolSandbox (Lu et al., 2025). ToolSandbox is a benchmark for evaluating stateful and multi-turn tool-use agents, where agents interact with external tools under evolving task states. Following the official evaluation protocol, we report the similarity score, which measures the alignment between agent execution trajectories and target milestones. ToolSandbox evaluates tool-use capability

Algorithm 1 `RectifyFlow`

Require: Task q, tools $\mathcal{T}$, models $\mathcal{M}_{\mathrm{C}}, \mathcal{M}_{\mathrm{E}}$, local repair budget K
Ensure: Task result or termination status for q

$\ell \leftarrow 0$; $r \leftarrow 1$
$\mathcal{B} \leftarrow \varnothing$; $h \leftarrow \varnothing$
Cloud: dual-view plan construction
$\mathcal{M}_{\mathrm{C}}$ selects $\mathcal{U} \subseteq \mathcal{T}$ and constructs $\mathcal{G}^{(0)}$ for q
$\mathcal{M}_{\mathrm{C}}$ selects a topological order $\mathcal{P}^{(0)}$ of $\mathcal{G}^{(0)}$
for each node p in $\mathcal{P}^{(0)}$ **do**
    $\mathcal{M}_{\mathrm{C}}$ encodes upstream output references and generates θ_p, ϕ_p
    $p \leftarrow (d_p, u_p, \theta_p, \phi_p)$
$\Pi^{(0)} \leftarrow (\mathcal{G}^{(0)}, \mathcal{P}^{(0)})$
Send $\Pi^{(0)}$ to $\mathcal{M}_{\mathrm{E}}$
Edge: sequential execution and local recovery
while $r \leq |\mathcal{P}^{(\ell)}|$ **and** execution has not terminated **do**
    $p \leftarrow \mathcal{P}^{(\ell)}[r]$
    if p has a retained valid completed result **then**
        $r \leftarrow r + 1$; **continue**
    $\mathcal{M}_{\mathrm{E}}$ evaluates ϕ_p using $\mathcal{B}$
    if ϕ_p evaluates to false **then**
        Record p as skipped in h
        $r \leftarrow r + 1$; **continue**
    Mark p as incomplete; $k \leftarrow 0$
    if ϕ_p is empty or true, and required bindings are available **then**
        $\mathbf{a} \leftarrow \mathcal{M}_{\mathrm{E}}(d_p, u_p, \theta_p, \mathcal{B})$
        Invoke $u_p(\mathbf{a})$; record the call and feedback in h
    else
        Record the unmet requirements in h
    while p is incomplete **and** $k < K$ **and** feedback indicates a repairable argument error **do**
        $\mathcal{M}_{\mathrm{E}}$ revises $\mathbf{a}$ using θ_p, the tool specification, and feedback
        Invoke $u_p(\mathbf{a})$; record the retry and feedback in h
        $k \leftarrow k + 1$
    if p completes successfully **then**
        Bind its output fields in $\mathcal{B}$
        Record completion in h; $r \leftarrow r + 1$
    else
        Form ξ_ℓ from p, unmet requirements, and recovery feedback
        $\chi_\ell \leftarrow (\Pi^{(\ell)}, \mathcal{B}, h, \xi_\ell)$
        Send χ_ℓ to $\mathcal{M}_{\mathrm{C}}$
    Cloud: dependency-aware rectification
        $(\Pi^{(\ell+1)}, I_\ell) \leftarrow \mathrm{DPR}(\mathcal{M}_{\mathrm{C}}, \chi_\ell)$
        Send $(\Pi^{(\ell+1)}, I_\ell)$ to $\mathcal{M}_{\mathrm{E}}$
    Edge: resume the revised plan
        Adopt $\Pi^{(\ell+1)}$ and apply I_ℓ
        Retain valid results and discard invalidated bindings
        Mark nodes requiring re-execution according to I_ℓ
        $r \leftarrow$ the starting position specified by I_ℓ
        $\ell \leftarrow \ell + 1$
return Task result or termination status for q, based on $\mathcal{B}$ and h

from two perspectives: scenario complexity and tool robustness. The scenario-based evaluation contains seven categories: Single Tool Call (STC) and Multiple Tool Call (MTC) measure tool invocation complexity; Single User Turn (SUT) and Multiple User Turn (MUT) evaluate interaction length; State Dependency (SD) tests execution under evolving states; Canonicalization (C) assesses input/output normalization; and Insufficient Information (II) evaluates the ability to identify missing information and request necessary clarification. For robustness evaluation, ToolSandbox introduces controlled tool perturbations. The distraction tool settings (0 DT, 3 DT, and 10 DT) vary the number of irrelevant tools to increase selection difficulty, while Tool Name Scrambling (TNS), Tool Description Scrambling (TDS), Argument Description Scrambling (ADS), and Argument Type Scrambling (ATS) evaluate robustness to ambiguous tool semantics and inconsistent tool specifications.

BFCL-V4 (Patil et al., 2025). BFCL-V4 is a benchmark for evaluating function-calling capability under realistic multi-turn scenarios. We use the official task success rate as the evaluation metric. The multi-turn test subset consists of four categories: Base, which contains standard function-calling tasks without additional perturbations; Long Context, which evaluates tasks with extended contex-

Algorithm 2 Dependency-aware Plan Rectification (DPR)

Require: Cloud model $\mathcal{M}_{\mathrm{C}}$, context $\chi_\ell = (\Pi^{(\ell)}, \mathcal{B}, h, \xi_\ell)$
Ensure: Revised plan $\Pi^{(\ell+1)}$ and execution instruction I_ℓ
$(\mathcal{G}^{(\ell)}, \mathcal{P}^{(\ell)}) \leftarrow \Pi^{(\ell)}$
Identify interrupted node p_r from ξ_ℓ
Backward deviation localization
Compare $\mathcal{P}^{(\ell)}$ with $\mathcal{B}$, h, and ξ_ℓ
Trace predecessor paths of p_r in $\mathcal{G}^{(\ell)}$
$\mathcal{M}_{\mathrm{C}}$ identifies the earliest evidence-supported deviation source p_s
if p_s is a missing prerequisite **then**
    Add p_s and its dependencies to $\mathcal{G}^{(\ell)}$
Forward impact analysis
$\mathcal{R}_\ell \leftarrow \{p_j \in V(\mathcal{G}^{(\ell)}) \mid p_s \rightsquigarrow^* p_j\}$
Selective plan rectification
$\delta_\ell \leftarrow \mathcal{M}_{\mathrm{C}}(\chi_\ell, \mathcal{R}_\ell)$
    $\delta_\ell = (\delta_\ell^{\mathrm{plan}}, \delta_\ell^{\mathrm{state}})$
$\delta_\ell^{\mathrm{plan}}$ specifies necessary edits within $\mathcal{R}_\ell$
$\delta_\ell^{\mathrm{state}}$ specifies recovery nodes for side effects recorded in h
Apply δ_ℓ to obtain $\Pi^{(\ell+1)}$, preserving unaffected components
Update dependency relations, execution order, and node specifications consistently
Generate I_ℓ: starting node, preserved completed nodes, and nodes to re-execute
return $\Pi^{(\ell+1)}, I_\ell$

tual information; Missing Functions, where relevant functions are unavailable and agents must handle incomplete tool specifications; and Missing Parameters, where agents are required to identify and recover missing function arguments.

A.2.2 Baseline Details.

We compare `RectifyFlow` with representative edge–cloud collaboration approaches and reference systems. Cloud-only uses the cloud LLM to perform all reasoning and tool-use decisions, providing an upper-bound performance reference without edge-side cost reduction. Local-only relies solely on the edge-side small model to complete planning and tool execution, representing the low-cost but capability-limited setting.

MiniCPM5-2B (MiniCPM, 2025) is a lightweight model optimized for edge-side agent scenarios, which is included as an additional local-model baseline to evaluate the impact of different SM capabilities. GAPG (Fang et al., 2025) reduces cloud inference cost by learning a model selection policy that determines whether each execution step should be handled by the edge model or the cloud model. AIMS (Liu et al., 2026) performs adaptive model selection during agent execution by routing requests between edge and cloud models according to estimated execution requirements. Hera (Zhang et al., 2026) introduces an edge–cloud collaborative execution strategy that dynamically assigns computation between local and cloud models to balance performance and inference cost. Minions (Narayan et al., 2025) decomposes agent tasks into smaller units and allocates selected subtasks to local models to reduce cloud dependency. DoT (Shao et al., 2025) adopts a planning-based approach that generates intermediate execution structures and assigns tool-use operations between edge and cloud models. HybridFlow (Dong et al., 2026) performs hybrid execution by combining cloud reasoning and edge-side execution through task decomposition and execution allocation. Plan-and-Act (Erdogan et al., 2025) decomposes tool-use tasks into planning and execution stages, where the LLM iteratively generates plans and selects actions based on execution feedback. ReWOO (Xu et al., 2023) separates reasoning from tool execution by generating a tool-use plan in advance and reducing intermediate LLM calls through deferred execution.

For baselines that require training, we use the category-wise randomly sampled training split to train their method-specific edge–cloud allocation components. All baselines are reproduced by following their official open-source implementations or the procedures described in the original papers as closely as possible. We use consistent dataset splits and evaluation protocols across methods to ensure a fair and accurate comparison.

Table 5: Performance across ToolSandbox scenario categories and average cloud API cost.

Method	ToolSandbox									
	Performance ($\times 10^{-2}$)↑								API Cost↓ ($\times 10^{-3}$ \$)	Relative to Cloud-only Perf.↑ / Cost↓
	STC	MTC	SUT	MUT	SD	C	II	Avg.		
Cloud-only	95.01	76.48	85.58	63.51	81.75	73.18	19.04	70.65	2.173	100.00% / 100.00%
Local-only (Qwen3-8B)	70.67	19.86	33.95	16.27	33.38	14.77	78.94	38.26	0.000	54.16% / 0.00%
MiniCPM5-2B	57.86	22.10	31.72	20.42	43.83	16.42	75.03	38.20	0.000	54.07% / 0.00%
GAPG	94.74	53.84	67.34	44.59	81.83	40.27	37.03	59.95	1.469	84.85% / 67.59%
AIMS	80.11	41.84	52.31	39.51	74.45	29.41	59.70	53.90	0.914	76.30% / 42.06%
Hera	82.38	54.32	65.12	43.43	79.96	42.57	43.67	58.78	1.550	83.20% / 71.35%
DoT	93.47	70.51	79.13	62.24	78.64	66.15	16.78	66.70	1.962	94.41% / 90.28%
Minions	65.56	33.39	40.01	37.90	41.90	28.58	64.78	44.59	1.438	63.11% / 66.15%
HybridFlow	87.14	40.39	51.96	41.18	59.77	31.83	78.66	55.85	1.194	79.05% / 54.94%
`RectifyFlow`	92.48	65.33	76.47	52.78	72.36	58.74	35.67	**64.83**	**1.057**	**91.77% / 48.66%**

Table 6: Performance across ToolSandbox tool augmentations and average cloud API cost.

Method	ToolSandbox									
	Performance ($\times 10^{-2}$)↑								API Cost↓ ($\times 10^{-3}$ \$)	Relative to Cloud-only Perf.↑ / Cost↓
	0 DT	3 DT	10 DT	TNS	TDS	ADS	ATS	Avg.		
Cloud-only	70.96	66.44	68.84	59.16	68.29	67.17	71.77	67.52	2.173	100.00% / 100.00%
Local-only (Qwen3-8B)	39.15	40.87	27.66	46.47	33.49	43.12	37.66	38.35	0.000	56.79% / 0.00%
MiniCPM5-2B	38.57	39.39	56.50	41.80	34.35	38.46	33.94	40.43	0.000	59.88% / 0.00%
GAPG	64.98	50.99	56.28	51.17	57.04	55.92	57.79	56.31	1.469	83.40% / 67.59%
AIMS	56.80	52.28	59.08	45.54	51.03	51.82	47.04	51.94	0.914	76.93% / 42.06%
Hera	60.32	61.20	48.90	55.42	54.19	49.46	59.12	55.52	1.550	82.22% / 71.35%
DoT	67.07	62.61	65.66	57.48	60.67	66.01	61.87	63.05	1.962	93.39% / 90.28%
Minions	45.45	43.68	57.17	50.12	38.47	42.42	46.38	46.24	1.438	68.49% / 66.15%
HybridFlow	53.97	55.21	52.58	56.34	51.20	57.99	58.34	55.09	1.194	81.59% / 54.94%
`RectifyFlow`	69.59	63.21	52.09	57.93	62.82	62.62	64.99	**61.89**	**1.057**	**91.67% / 48.66%**

A.3 Details of the experimental results

Detailed Performance Analysis. To provide a more fine-grained analysis, we further report the performance of different methods across individual task categories in ToolSandbox and BFCL. These results complement the overall comparison in § 5 by analyzing the robustness of each approach under different tool-use scenarios.

ToolSandbox Scenario Analysis. Tab. 5 presents the performance across different ToolSandbox scenario categories. `RectifyFlow` consistently achieves strong performance across diverse task types, including single-tool calling (STC), multi-tool calling (MTC), and single- and multi-turn user interactions (SUT/MUT). In particular, it achieves an average performance of 64.83 while requiring only 1.057×10^{-3} USD/task, preserving 91.77% of Cloud-only performance with less than half of its API cost. Compared with other collaboration methods, `RectifyFlow` provides a more balanced trade-off across different categories rather than relying on improvements from specific task types.

Tool Augmentation Analysis. Tab. 6 further evaluates robustness under different tool augmentation settings. As the number and complexity of tool dependencies increase, existing methods exhibit larger performance variations due to frequent routing decisions or insufficient local execution capability. In contrast, `RectifyFlow` maintains stable performance across different augmentation settings, achieving 61.89 average performance with reduced cloud cost. This indicates that dependency-aware planning and selective rectification effectively handle increasing execution complexity.

BFCL Multi-turn Analysis. Tab. 7 reports results on the BFCL multi-turn subset, which requires continuous interactions under tool execution. `RectifyFlow` achieves 76.72 average performance, approaching Cloud-only execution (81.09) while reducing API cost from 5.803×10^{-3} to 2.286×10^{-3} USD/task. It also maintains strong performance under challenging cases involving long context, missing functions, and missing parameters, demonstrating that dependency-aware exe-

Table 7: Task performance on BFCL and average cloud API cost.

Method	BFCL						
	Performance ($\times 10^{-2}$)↑					API Cost↓ ($\times 10^{-3}$ \$)	Relative to Cloud-only Perf.↑ / Cost↓
	Base	Long Context	Missing Functions	Missing Parameters	Avg.		
Cloud-only	91.88	75.62	76.88	80.00	81.09	5.803	100.00% / 100.00%
Local-only (Qwen3-8B)	8.75	4.38	6.88	7.50	6.88	0.000	8.48% / 0.00%
MiniCPM5-2B	5.00	3.75	4.38	4.38	4.38	0.000	5.39% / 0.00%
GAPG	52.50	38.75	45.62	45.62	45.63	2.809	56.26% / 48.41%
AIMS	46.88	22.50	36.88	41.88	37.03	2.635	45.66% / 45.41%
Minions	50.00	48.12	45.62	50.00	48.44	3.023	59.73% / 52.10%
Hera	43.75	31.25	36.88	36.25	37.03	2.383	45.66% / 41.06%
DoT	18.12	16.88	13.12	18.75	16.72	2.646	20.62% / 45.60%
HybridFlow	56.88	47.50	52.50	55.62	53.13	2.337	65.51% / 40.27%
`RectifyFlow`	76.88	72.50	78.12	79.38	**76.72**	**2.286**	**94.61% / 39.39%**

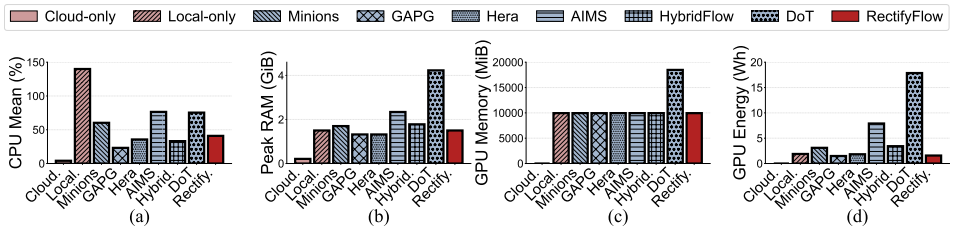

Figure 7: Comparison of Edge overhead across different methods.

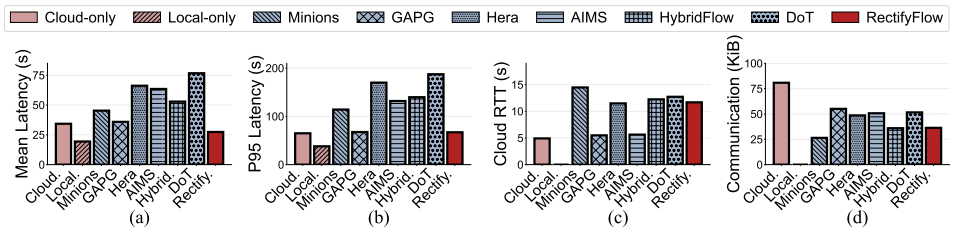

Figure 8: Comparison of latency and communication overhead across different methods with GPT–5.6–Terra.

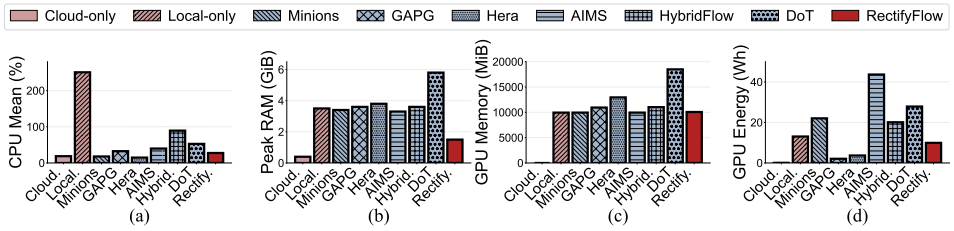

Figure 9: Comparison of Edge overhead across different methods with GPT–5.6–Terra.

cution guidance and targeted rectification can effectively mitigate execution deviations in multi-turn scenarios.

Edge-side Resource Overhead. We analyze the edge-side resource consumption under the default model configuration. Fig. 7 compares CPU utilization, memory consumption, GPU memory usage, and energy consumption across different methods. `RectifyFlow` introduces moderate edge-side overhead while enabling reliable local execution. It requires 40.94% average CPU utilization and 1.50 GiB peak RAM, avoiding the high computation burden of Local-only execution and the additional coordination overhead of several collaborative methods. Its GPU memory consumption is 9.9

Table 8: Fine-grained component ablation of RectifyFlow on ToolSandbox. ✓ and ✗ indicate enabled and disabled components, respectively.

Variant	DPC			DPR			Evaluation		
	Output References	Argument Grounding	Execution Criteria	Backward Localization	Forward Impact Analysis	Selective Patch	Avg. Perf.↑ ($\times 10^{-2}$)	API Cost↓ (10^{-3} \$)	Cloud Calls↓ (per task)
RectifyFlow	✓	✓	✓	✓	✓	✓	64.83	1.057	1.39
A. Ablating DPC									
w/o output references	✗	✓	✓	✓	✓	✓	58.32	1.889	1.98
w/o argument grounding	✓	✗	✓	✓	✓	✓	56.61	1.694	2.13
w/o execution criteria	✓	✓	✗	✓	✓	✓	61.68	1.153	1.54
w/o DPC	✗	✗	✗	✓	✓	✓	50.60	2.052	2.60
B. Ablating DPR									
w/o backward localization	✓	✓	✓	✗	✓	✓	59.28	1.174	1.53
w/o forward impact analysis	✓	✓	✓	✓	✗	✓	62.17	1.094	1.40
w/o selective plan rectification	✓	✓	✓	✓	✓	✗	63.26	1.217	1.42
w/o $\mathcal{G}$	✓	✓	✓	✗	✗	✓	61.83	1.262	1.59
w/o DPR	✓	✓	✓	✗	✗	✗	61.93	1.254	1.63

GiB, comparable to other edge-assisted approaches and substantially lower than DoT (18.5 GiB). Moreover, RectifyFlow consumes only 1.576 Wh GPU energy, reducing energy consumption by 49.1%, 54.0%, and 80.2% compared with Minions, HybridFlow, and DoT, respectively. These results demonstrate that RectifyFlow achieves reliable tool execution without introducing excessive edge-side resource overhead.

System Overhead under GPT-5.6-Terra. To further evaluate the robustness of RectifyFlow under different cloud LLM configurations, we replace the default cloud model with GPT-5.6-Terra while keeping the edge model and framework unchanged. Fig. 8 presents the corresponding latency and communication overhead. RectifyFlow achieves 27.4s average latency and 67.1s P95 latency, which remain lower than most collaborative baselines that require frequent cloud interactions. The average cloud RTT is 11.68s, and the communication overhead is 36.31 KiB/task, indicating that the plan-guided execution strategy continues to reduce unnecessary cloud exchanges under different cloud models. Fig. 9 further reports the edge-side resource consumption with GPT-5.6-Terra. RectifyFlow maintains a similar resource profile, requiring only 27.6% CPU utilization, 1.5 GiB peak RAM, and 10.07 GiB GPU memory. Its GPU energy consumption is 9.929 Wh, remaining lower than most collaborative baselines. These results confirm that the efficiency advantages of RectifyFlow are not tied to a specific cloud model and remain stable across different cloud LLM configurations.

Ablation Implementation. To provide a more fine-grained understanding of each component, we conduct a detailed ablation study by individually removing the key components of DPC and DPR. For DPC ablations, the removed components are excluded from the execution view, requiring the edge SM to resolve the missing information from available observations. For DPR ablations, removed mechanisms are replaced with simplified recovery strategies: the interrupted node is treated as the deviation source without backward localization, the entire plan is considered as the candidate scope without impact analysis, all candidate nodes are modified without selective rectification, and the dependency graph is removed during graph-free recovery.

Tab. 8 evaluates the contributions of DPC and DPR. The complete framework achieves the highest performance (64.83%) with the lowest API cost (1.057×10^{-3} USD/task) and cloud invocation count (1.39), demonstrating the effectiveness of combining structured execution guidance with dependency-aware rectification. For DPC, removing the complete guidance mechanism decreases performance to 50.60% and increases API cost to 2.052×10^{-3} USD/task, confirming that execution guidance substantially reduces the reasoning burden on the edge SM. Among individual components, output references and argument grounding have the largest impact, indicating the importance of explicit dependency binding and tool-argument construction. For DPR, removing backward localization causes the largest performance drop (64.83% to 59.28%), showing that accurately identifying deviation sources is critical for effective recovery. Forward impact analysis further improves recovery quality by reconsidering affected downstream nodes. Removing selective rectification increases API cost from 1.057 to 1.217×10^{-3} USD/task, demonstrating that minimal updates reduce rectification overhead while preserving execution progress. Finally, removing dependency information

Table 9: Analysis of plan Accuracy and the effectiveness of plan rectifications.

Stage	Sequence Similarity		Mean LCS Recall	Mean Tool Recall
	Mean	Median		
Initial plan (P_0)	91.18	92.31	90.97	91.83
First-patch stage (P_1)	92.69	94.74	94.37	95.26
Second-patch stage (P_2)	93.50	100.00	95.99	96.94

results in lower performance (61.83%) and higher cost (1.262×10^{-3} USD/task), indicating that dependency-aware analysis is essential beyond selective patching alone.

Plan Quality and Rectification Effectiveness. Tab. 9 analyzes the structural quality of generated plans and subsequent rectifications by comparing them with the corresponding ground-truth on the BFCL test set. The Initial stage is evaluated over all test instances, whereas the First-patch stage includes only instances that undergo at least one rectification, and the Second-patch stage further focuses on those requiring at least two rectifications. The initial plans already achieve high structural alignment, with 91.18% sequence similarity, 90.97% LCS recall, and 91.83% tool recall, indicating that the dual-view plan construction provides reliable initial execution guidance for edge-side SMs. For instances requiring rectification, the First-patch stage achieves 92.69% sequence similarity, 94.37% LCS recall, and 95.26% tool recall. Among instances undergoing at least two rectifications, the Second-patch stage further reaches 93.50%, 95.99%, and 96.94% on these metrics, respectively. These results show that `RectifyFlow` produces structurally reliable initial plans and that its dependency-aware rectification can progressively improve plan consistency without frequent full replanning.

Prompt Design. To instantiate `RectifyFlow`, we design four role-specific prompts for dual-view plan construction, edge execution, dependency-aware rectification, and final response generation. All prompts enforce structured JSON outputs to facilitate reliable interaction between model decisions and the execution engine. The cloud planning prompt guides $\mathcal{M}_{\mathrm{C}}$ to construct the dual-view plan by selecting relevant tools, building the dependency DAG, deriving the execution order, and annotating nodes with output references, argument-grounding specifications, and execution criteria. The edge execution prompt guides $\mathcal{M}_{\mathrm{E}}$ to execute nodes according to the execution view, instantiate tool arguments, evaluate execution conditions, and perform bounded local recovery for tool-supported errors. The rectification prompt guides $\mathcal{M}_{\mathrm{C}}$ to identify deviation sources through dependency analysis, determine affected nodes, and generate selective plan updates with state-consistency recovery when necessary. The complete prompts are provided below.

Cloud Prompt: Dual-View Plan Construction

You are the cloud planner in `RectifyFlow`. Return JSON only.

Given the task and available tool schemas, construct a dual-view plan.

1. Select task-relevant tools and define the required operations. Assign each operation a stable node ID, a semantic objective, and one selected tool.

2. Construct a dependency DAG with explicit data, state, and control dependencies. Select a topological execution order containing every node exactly once.

3. For each executable node, specify its objective, selected tool, argument-grounding specification, and execution criterion. The grounding specification must identify argument names, sources, formats, and necessary transformations. Use a null criterion for unconditional nodes.

4. Represent unavailable upstream outputs with symbolic references to their producer nodes and output fields. Do not invent their values during planning.

5. Keep dependency edges, output references, and execution order consistent. Execution criteria determine branch applicability, not post-call success.

Use only available tools and supported task information. Do not invent identifiers, states, or tool outputs.

Input: User task; initial context; available tool schemas.

Output: A JSON object containing `dependency_view` (nodes and typed edges) and `execution_view` (annotated nodes and execution order).

Edge Prompt: Node Execution and Local Recovery

You are the edge execution model in `RectifyFlow`. Return JSON only.

Realize the current node using its execution specifications and the supplied runtime evidence.

1. Evaluate the execution criterion using relevant bindings. A null criterion is unconditional. Skip the node only when its condition is false.

2. If information required to evaluate the criterion or construct arguments is missing, report an interruption. Do not interpret missing information as a false condition.

3. For an applicable node, instantiate concrete arguments according to its argument-grounding specification. Use exact tool argument names and supported values.

4. After a failed call, correct only argument errors supported by the tool schema and observed feedback, including formatting, type, enum, or computation errors. Preserve the node objective, selected tool, and plan.

5. Return `plan_defect` if the evidence reveals a missing or incorrect operation or state prerequisite. Return `cannot_repair` if no supported local correction is available.

Do not add tools, modify dependencies, guess missing values, or alter user-provided facts.

Input: Task; current node; current tool schema; relevant bindings; runtime observations; previous failure and local attempts, if any.

Output: A JSON object with `decision` (`execute`, `skip`, `plan_defect`, or `cannot_repair`), `arguments`, and a brief evidence-based `reason`.

Cloud Prompt: Dependency-aware Plan Rectification

You are the cloud rectification model in `RectifyFlow`. Return JSON only.

Revise the current dual-view plan using execution evidence. Preserve unaffected operations and valid completed results.

Backward deviation localization.
1. Identify the interrupted node. Compare the execution view with bindings, execution history, and interruption evidence.

2. Trace relevant predecessor paths in the supplied dependency graph. Identify the earliest evidence-supported deviation source. Use the supplied graph as the dependency reference instead of reconstructing dependencies from execution order alone.

3. If a prerequisite is missing, introduce a node with a stable ID and establish its dependency relations before analyzing downstream impact. Correct dependencies only when supported by evidence.

Forward impact analysis.
4. Identify the deviation source and all nodes reachable from it as candidates for reconsideration, including affected nodes on other branches and previously executed nodes.

Selective plan rectification.
5. Modify only candidate operations requiring correction. Update node specifications, output references, and dependencies consistently.

6. Inspect recorded side effects. Add state-recovery operations when needed, and place them before operations requiring the restored state. Editing the plan does not undo an executed tool action.

7. Preserve unaffected components and still-valid results. Identify invalidated results even when the corresponding node specifications remain unchanged.

8. Construct the updated dependency DAG and a compatible execution view. Specify the starting node, results to preserve, and nodes requiring re-execution. Re-evaluate previous skip decisions when their supporting conditions have changed.

Do not invent facts, tool outputs, or unobserved side effects.

Input: Task; current dual-view plan; bindings; execution history; interruption evidence; available tool schemas.

Output: A JSON object containing: `diagnosis` (source, evidence, and affected nodes); `rectification` (plan changes and state-recovery nodes); `updated_dual_view_plan`; and `resume` (starting node, preserved results, and nodes requiring re-execution or re-evaluation).