Failure-Aware Dual-Flow Control for Computer-Use Agents

Anonymous ACL submission

Abstract

Computer-use agents (CUAs) operate by directly interacting with graphical user interfaces in real operating system environments, where execution failures frequently prevent progress. When actions fail to induce observable state changes, existing agents often continue conditioning on unchanged observations and repeatedly generate similar decisions, exhausting execution budgets without effective recovery. We frame this behavior as a control problem: current agentic frameworks lack explicit mechanisms to regulate how decision-making should adapt under execution failure. To address this gap, we propose **Failure-aware Dual-flow Control (FDC)**, a control framework that explicitly separates standard execution from adaptive recovery and regulates transitions between them using action-grounded failure signals. FDC integrates action-grounded dual failure evaluation with a hierarchical adaptive recovery mechanism, enabling targeted decision revision only upon detected failure. Evaluated on OSWorld, a large-scale benchmark for real OS interaction, FDC consistently improves success rates across both open-weight and closed-source backbones while incurring substantially lower inference-time cost than scaling-based alternatives. These results demonstrate that explicit failure-aware control is critical for robust and efficient computer-use agents.

1 Introduction

Computer-use agents(CUAs) are autonomous systems that accomplish user-specified tasks by directly interacting with graphical user interfaces (GUIs) (Cheng et al., 2024; Yang et al., 2025; Xie et al., 2025; Wu et al., 2025a; Liu et al., 2025; Wang et al., 2025b; Song et al., 2025; Gonzalez-Pumariega et al., 2025). Rather than relying on application-specific APIs or predefined scripts, CUAs operate at the interface level, executing low-level actions such as clicking, typing, and navigation. This interaction paradigm enables agents to function in real operating-system environments where internal program states and control interfaces are not explicitly exposed (Xie et al., 2024; Zhou et al., 2023).

Recent advances in large language models (LLMs) have significantly expanded the capability of CUAs (Madaan et al., 2023; Shinn et al., 2023; Yao et al., 2022; Schick et al., 2023; Yang et al., 2023). Given a natural language instruction, modern agents can interpret user intent and generate sequences of executable GUI actions. These advances have made it possible to deploy agents in open-ended desktop environments involving diverse applications and extended execution trajectories. However, operating reliably in such environments requires not only strong planning ability, but also effective control over execution under uncertainty.

Existing CUAs approaches can be broadly distinguished by how they organize perception, reasoning, and action during execution. End-to-end methods integrate these components within a single model, directly mapping visual observations and task instructions to executable actions (Hong et al., 2024; Liu et al., 2024; Hong et al., 2024; Wang et al., 2025b; Yang et al., 2025; Qin et al., 2025; Wang et al., 2025a; Ye et al., 2025; Fu et al., 2025). In contrast, agent-based frameworks decompose decision-making into interacting components, typically separating high-level reasoning from low-level execution (Agashe et al., 2024; Gonzalez-Pumariega et al., 2025; Guo et al., 2025; Song et al., 2025; Agashe et al., 2025),. A common design follows a planner–executor paradigm (Cheng et al., 2024; Wu et al., 2024), where the planner determines what action to take and the executor grounds it on the screen. This modular structure improves flexibility and interpretability, but most existing frameworks still treat execution as a single, continuous process. Failure handling and recovery are often implicitly entangled with action generation.

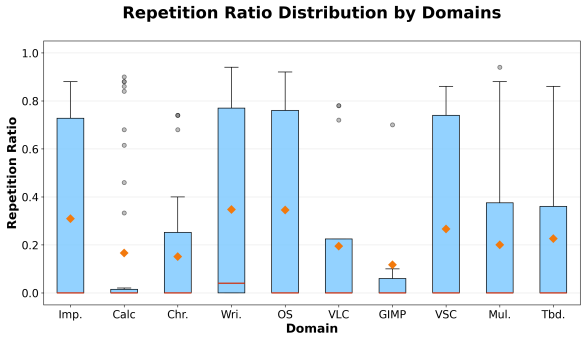

Figure 1: Distribution of plan repetition ratios in failed Agent S3 trajectories on OSWorld, grouped by application domain. Each box shows the distribution of the fraction of execution steps whose plan content repeats earlier plans within the same task. Orange diamonds indicate mean values.

Despite recent progress in agent-based frameworks, reliable execution remains a fundamental challenge for computer-use agents operating in real OS environments. In practice, agents often act under partial observability, where the effects of an action may be delayed, subtle, or entirely absent from the screen. When an executed action fails to induce a meaningful state change, the subsequent screenshots observation can remain unchanged, providing no new evidence to guide decision-making.

Under such conditions, we observe a recurring execution pattern in which the agent continues to condition on unchanged observations and regenerates similar plans across consecutive steps. Without a mechanism to recognize that the current decision flow has become ineffective and to deliberately shift into a different mode of reasoning, the agent may persist in repeating unproductive decisions until the execution budget is exhausted.

In an analysis of failed tasks produced by a representative agent framework (Agent S3) on OSWorld, we find that repeated plan content accounts for approximately one-third of all execution steps in failed trajectories (Figure 1). This quantitative evidence highlights a systematic gap in existing agent designs: while modern agents are capable of generating plausible plans, they lack explicit control over how decision-making should adapt once execution failures prevent progress.

These observations point to a missing component in existing agent designs: explicit control over the agent's decision process when execution failures occur. To address this need, we propose **Failure-aware Dual-flow Control (FDC)**, a control framework that explicitly separates routine execution from adaptive recovery. Under FDC, the agent follows a standard execution flow during normal progress, and transitions into a dedicated recovery flow only when concrete failure signals are observed. By treating recovery as a distinct decision regime with its own objectives and constraints, FDC enables targeted intervention under execution failure while preserving efficiency during normal operation.

In summary, this work makes the following contributions: (i) We identify and quantitatively characterize a prevalent failure pattern in computer-use agents, where decision-making stagnates under execution failure due to repeated plan generation conditioned on unchanged observations. (ii) We propose Failure-aware Dual-flow Control(FDC), a control framework that explicitly regulates when an agent follows standard execution and when it transitions into recovery, enabling selective intervention under execution failure. (iii) We demonstrate on the OSWorld benchmark that FDC substantially improves execution reliability and efficiency across different model backbones, outperforming existing agentic frameworks while incurring significantly lower inference-time overhead than scaling-based alternatives.

2 Related Work

2.1 Computer-Use Agent Frameworks

Early approaches operate on structured representations such as DOM trees or accessibility metadata, enabling symbolic command execution but limiting applicability to web or instrumented environments (Nakano et al., 2021; Xu et al., 2023). With recent advances in vision–language models, end-to-end approaches have emerged that directly map screenshots and task instructions to executable GUI actions (Wang et al., 2025b; Qin et al., 2025; Wang et al., 2025a). While these models demonstrate strong visual grounding, their tightly coupled perception–action pipelines lack explicit control when execution deviates from expectation.

To improve flexibility, agent-based frameworks decompose decision-making into interacting components, separating high-level reasoning from low-level execution in a planner–executor paradigm. However, most existing frameworks still treat execution as a single, continuous decision process. Failure handling is commonly embedded within the same execution flow, relying on implicit recovery or repeated action sampling rather than explicit

control over how decision-making should adapt when execution failures occur.

2.2 Training-Based Error Correction

Several studies address agent failures by incorporating error correction directly into training, including BacktrackAgent (Wu et al., 2025b), InfiGUI-R1 (Liu et al., 2025), and GUI Reflection (Wu et al., 2025a) augment agents with reflection or judgment capabilities learned from curated datasets. These approaches can improve robustness within the scope of their training distributions, particularly in simulated or reversible environments where actions can be undone or replayed.

However, training-based correction methods inherently depend on data coverage and environment assumptions. In real OS settings, many actions are irreversible, and failure recovery cannot rely on resetting system state or replaying trajectories. In contrast, our work focuses on inference-time control, explicitly regulating the agent's decision process under execution failure without assuming reversible environments or additional training data.

2.3 Inference-Time Verification and Recovery

Another line of work improves reliability through inference-time verification and scaling (Snell et al., 2024). Multi-agent frameworks and test-time scaling methods evaluate multiple candidate trajectories via parallel rollouts (Liu et al., 2025)or stepwise judging (Liu et al., 2025), selecting actions based on retrospective comparison. While effective in controlled benchmarks, these approaches are often impractical in real OS environments, where parallel execution can interfere with shared system state.

More importantly, inference-time scaling treats recovery as a byproduct of extensive sampling rather than as a controlled decision process. Verification is typically applied uniformly at every step, regardless of whether execution is progressing normally or stagnating. In contrast, our Failure-aware Dual-flow Control framework performs selective intervention: recovery reasoning is triggered only upon concrete failure signals and operates as a distinct decision regime. This design enables efficient and targeted recovery without relying on exhaustive sampling or continuous verification.

3 Method

We formulate CUA as decision-making systems operating under partial observability, where system state changes are not guaranteed to be reversible. The core challenge addressed in this work is not action generation itself, but controlling how the agent's decision process adapts when execution failures prevent progress.

To this end, we present **Failure-aware Dual-flow Control (FDC)**, a control framework depicted in Figure 2 that explicitly regulates when the agent follows standard execution and when it transitions into recovery. FDC introduces a state-dependent control signal that switches between two execution flows: a lightweight standard execution flow for routine interaction, and a recovery flow that performs targeted reasoning and strategy revision upon detected failure.

3.1 Failure-Aware Dual-Flow Control

We model the computer-use agent as operating under a partially observable Markov decision process (POMDP) (Kaelbling et al., 1998). At each step t, the agent receives a screen observation o_t and maintains a compressed interaction history H_t. Given a task instruction $\mathcal{I}$, the agent generates an executable action a_t.

FDC explicitly controls the agent's decision process by switching between two decision regimes. Under the standard execution flow π_{std}, the agent generates actions for routine progress. When execution failures are detected, control transitions to a recovery flow π_{rec}, which performs targeted reasoning and strategy revision before returning control to standard execution. The action selection at step t is defined as:

$$a_t = \begin{cases} \pi_{\text{rec}}(\mathcal{I}, H_t, o_t, \mathcal{S}_{\text{layer}}), & \text{if } T(H_t, o_t) = 1, \\ \pi_{\text{std}}(\mathcal{I}, H_t, o_t), & \text{otherwise.} \end{cases} \tag{1}$$

Here, $\mathcal{S}_{\text{layer}}$ denotes the hierarchical recovery strategy templates provided to the recovery flow. The transition signal $T(H_t, o_t)$ determines whether control remains in standard execution or shifts to recovery. Importantly, recovery is not invoked continuously but only when concrete failure evidence is observed, allowing FDC to preserve efficiency during normal execution.

3.2 Action-Grounded Failure Evaluation

FDC derives its control signal from two complementary failure evaluations that assess execution outcomes at different levels. Together, these evaluations provide explicit evidence for determining

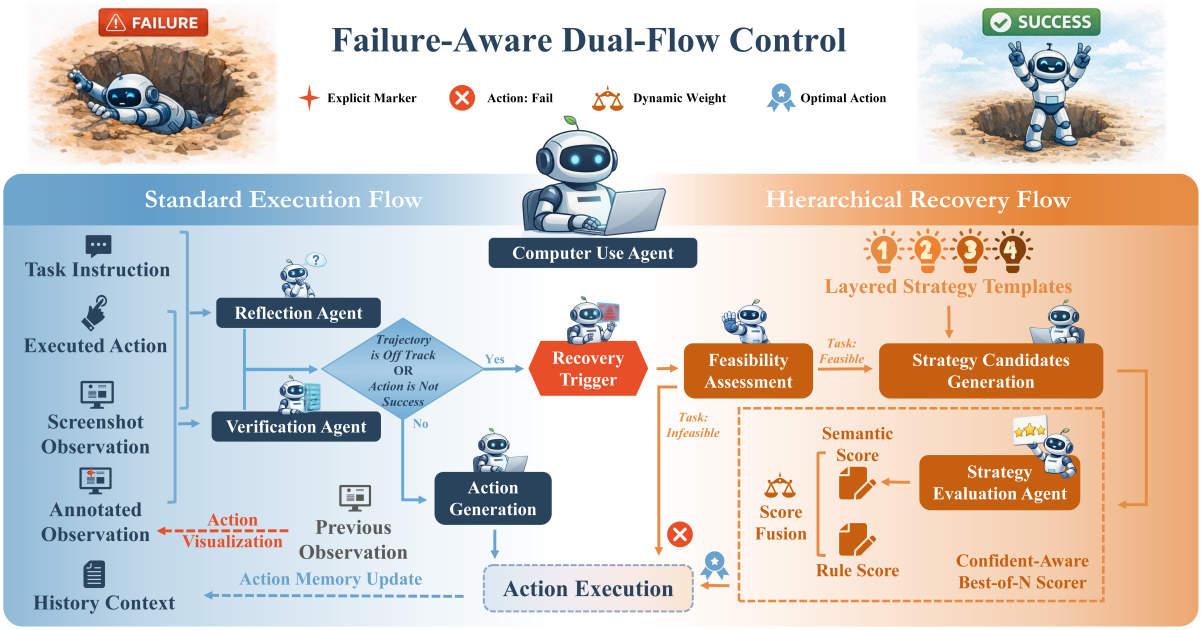

Figure 2: Overview of Failure-Aware Dual-Flow Control (FDC). The agent is explicitly controlled by switching between two execution flows. In the standard execution flow (left), actions are generated and assessed via action-grounded verification and trajectory-level reflection. Upon detected failure, a control signal transitions the agent into the hierarchical recovery flow (right): a feasibility assessment first determines whether recovery is possible; then *layered strategy templates* are injected as hints to guide the agent to generate a set of recovery strategy candidates conditioned on the current task and context; finally, a confidence-aware scorer selects one candidate for execution before returning control to standard execution.

whether the current execution flow remains effective.

Action-Level Verification. To assess whether an executed action produces the intended effect, we employ an action-level verification process grounded in visual evidence. An Action Visualizer overlays the executed action's spatial location on the previous screen observation o_{t-1}, producing an annotated observation o'_{t-1}. In parallel, a Screenshot Diff tool computes a summary of pixel-level changes δ_{diff} between consecutive observations.

A dedicated Verification Agent V evaluates whether the executed action a_{t-1} successfully induced a meaningful state change:

$$v_t = V(a_{t-1}, o'_{t-1}, o_t, \delta_{\text{diff}}) \tag{2}$$

where v_t includes a binary success indicator and supporting evidence. This evaluation focuses strictly on execution outcomes and does not assess task-level progress.

Trajectory-Level Reflection. Complementary to action-level verification, a Reflection Agent R evaluates whether the current execution trajectory remains aligned with the task objective. Given the task instruction $\mathcal{I}$, current observation o_t, and interaction history H_t, the reflection process determines whether execution is making progress toward task completion:

$$r_t = R(\mathcal{I}, o_t, H_t) \tag{3}$$

The output r_t classifies the trajectory state as `on_track`, `off_track`, or `completed`, accompanied by explanatory reasoning. This assessment is independent of individual action outcomes and captures higher-level execution failure modes.

Control Signal. The control signal $T(H_t, o_t)$ triggers the recovery flow when either action execution fails or the execution trajectory deviates from the task objective:

$$T(H_t, o_t) = \mathbb{I}(\neg v_t.\text{success} \vee r_t = \text{off\_track}) \tag{4}$$

3.3 Hierarchical Adaptive Recovery

When control transitions into the recovery flow, FDC either terminates early if the task is deemed infeasible, or executes a recovery policy designed to revise the agent's decision process under execution failure. Rather than exhaustively replanning or

sampling actions, recovery is structured hierarchically to regulate how far the agent deviates from prior decisions.

Feasibility Assessment. Upon entering the recovery flow, FDC first performs a feasibility assessment to determine whether the task is fundamentally unrecoverable under the current failure context. This assessment aims to filter out impossible tasks—e.g., when required files, UI objects, or resources are demonstrably absent from the environment given the current observation and interaction history. If the task is deemed infeasible, the agent terminates recovery early by outputting "agent.fail" action to prevent futile execution loops and to save computational resources.

Hierarchical Strategy Generation. Upon entering the recovery flow, FDC expands the agent's exploration space by injecting layered strategy templates as *hints* that guide how recovery candidates should be generated under the current task and context. The strategy templates are organized into four layers, ranging from local refinement of the current plan, to switching interaction modalities, restructuring the navigation path, and finally reconsidering the task or terminating gracefully when recovery is deemed infeasible. The active layer is determined by the cumulative failure count c_f, which increases with consecutive failures and resets upon a successful execution. To encourage diversity without excessive exploration, the strategy generator produces multiple candidates using detailed guidance from the active layer, while incorporating concise hints from shallower layers.

Adaptive Strategy Selection. Given a set of candidate recovery strategies $\mathcal{C}$, FDC selects a single strategy for execution using a confidence-aware scoring mechanism. The goal is to identify an effective recovery action without relying on parallel execution or exhaustive sampling in the environment. Each candidate strategy is evaluated along two complementary dimensions. First, a semantic evaluation agent assesses the plausibility of the strategy with respect to the task instruction and current context. To improve robustness, the continuous evaluation output is discretized into a three-valued semantic score $S_{\text{semantic}} \in \{1.0, 0.5, 0.0\}$, representing strong support, neutrality, and rejection, respectively. The agent also outputs a confidence estimate $\gamma \in \{\text{high}, \text{medium}, \text{low}\}$. Second, a rule-based score S_{rule} captures execution-oriented constraints, including safety and novelty considerations, discouraging repetitive or high-risk recovery attempts. The final selection score is computed as a dynamic-weighted combination of semantic and rule-based evaluations:

$$S_{\text{final}} = \alpha(\gamma) \cdot S_{\text{semantic}} + (1 - \alpha(\gamma)) \cdot S_{\text{rule}}, \quad (5)$$

where the weight $\alpha(\gamma)$ decreases as semantic confidence decreases, shifting reliance toward rule-based constraints when the semantic judgment is uncertain.

4 Experiments

4.1 Experimental Settings

Environment. We evaluate our framework on OSWorld (**?**), a widely adopted benchmark for computer-use agents operating in real OS environments. OSWorld provides a fully functional operating system and requires agents to interact through low-level mouse and keyboard actions across a broad range of native applications (e.g., Chrome, VS Code, GIMP, Thunderbird), rather than relying on application-specific APIs or simulators. Tasks span office workflows, software development, and media processing, often involving cross-application interaction and file I/O. Due to this breadth and realism, OSWorld has become a standard testbed for evaluating robustness and general-purpose computer-use agents. We evaluate on 361 tasks following standard protocols, excluding Google Drive tasks due to authentication instability.

Baselines. We compare our method against representative end-to-end models and agentic frameworks. End-to-end baselines include Qwen3-VL-30B-A3B-Instruct and the UI-TARS series. Agentic baselines include Agent S2.5 (the predecessor to S3 without scaling or code-agent capabilities), Agent S3, GTA1, and Agentic Lybic. For Agent S3, we report No Scaling (N=1) and Best-of-N configurations (N=3, 10). The N=3 setting provides a compute-matched comparison, as our framework explores an average of 2.13 recovery strategies per step. GTA1 is evaluated with its original N=8 configuration.

Evaluation Metrics. We use Success Rate (SR) as the primary metric, determined by domain-specific evaluation scripts that verify final environment states. To assess efficiency, we additionally report average completion steps and total token consumption per task.

Implementation Details. Each task is executed with a maximum of 50 steps. Due to environment

stochasticity, results are averaged over two runs. We use UI-TARS-1.5-7B as the executor. For planning and recovery, we evaluate both Qwen3-VL-30B-A3B-Instruct and OpenAI o3.

4.2 Main Results

Table 1 reports Success Rate (SR) on OSWorld. FDC consistently outperforms both end-to-end models and prior agentic frameworks across backbones, demonstrating the effectiveness of failure-aware dual-flow control.

Open-Weight Models. With Qwen3-VL-30B-A3B-Instruct, FDC achieves an average SR of 30.19%, surpassing Agent S3 with Best-of-N scaling (N=10, 27.70%) while exploring substantially fewer candidates (2.13 vs. 10 per step). This indicates that selectively triggering recovery under execution failure is more effective than uniformly increasing inference-time sampling. In contrast, search-heavy frameworks such as GTA1 degrade on weaker backbones, as the model struggles to reliably generate and select among a large number of candidate plans at each step.

Reasoning Models. With OpenAI o3, FDC further scales to 58.72% SR, outperforming Agentic Lybic (56.90%) and Agent S2.5 (54.20%). Gains are most pronounced in domains requiring precise control and structured reasoning, such as VS Code and GIMP, where recovery frequently involves switching interaction modalities or restructuring execution paths. A representative example illustrates how failure-aware recovery improves execution. In a VS Code task requiring global text replacement ("text" $\rightarrow$ "test"), repeated attempts to use a small GUI icon failed due to unreliable visual grounding. After detecting persistent execution failure, our agent transitioned into recovery and switched interaction modality, invoking a command-line operation (`sed -i`) to complete the task. This behavior highlights how controlled escalation enables the agent to bypass fragile GUI interactions rather than repeatedly retrying the same action.

4.3 Efficiency Analysis

Token Cost. Table 2 compares inference cost and success rate. We define M as the average number of recovery strategies explored per execution step. Our framework maintains a low exploration density ($M \approx 2.13$), as recovery is triggered only upon detected failure. In contrast, inference-time scaling methods incur substantially higher costs with diminishing returns. For example, Agent S3 (N=10) increases token usage by over 20$\times$ compared to no scaling, while yielding a smaller SR gain. This super-linear surge stems from the auxiliary overhead of behavior narrative generation and comparative evaluations required for processing parallel trajectories. FDC achieves higher SR with comparable token cost to unscaled baselines.

Completion Steps. We further measure average completion steps as a proxy for physical efficiency. As shown in Figure 3, our framework consistently reduces execution length across domains. This reduction reflects the ability of hierarchical recovery to break unproductive execution patterns early, rather than repeatedly retrying similar actions.

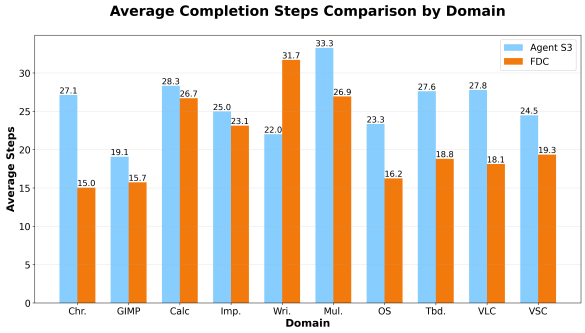

Figure 3: Average completion steps by domain for Agent S3 and FDC. Lower values indicate shorter execution trajectories. Across most domains, FDC completes tasks with fewer steps, reflecting earlier disruption of unproductive execution patterns through failure-aware hierarchical recovery.

4.4 Ablation Studies

4.4.1 Effect of Steps Budget

We first examine how performance scales with the execution steps budget. Table 3 reports success rates under step budgets of 15, 50, and 100 steps for both Agent S3 and FDC, using Qwen3-VL-30B-A3B-Instruct as the planner and UI-TARS-1.5-7B as the executor. For Agent S3, increasing the step budget yields marginal improvements: performance saturates at 23.82% once the limit exceeds 50 steps. This behavior suggests that additional steps primarily prolong unproductive execution rather than enabling effective recovery. In contrast, our framework exhibits a markedly different scaling pattern. While the gap is modest under a tight 15-step budget (22.71% vs. 20.78%), performance improves substantially as more steps become available, reaching 30.19% at 50 steps and 31.86% at 100 steps.

Method	Chr.	GIMP	Calc	Imp.	Wri.	Mul.	OS	Tbd.	VLC	VSC	Avg. (SR)
End-to-End Models											
Qwen3-VL-30B-A3B-Instruct	17.39	30.77	**19.15**	**36.17**	**52.17**	8.60	20.83	40.00	29.41	17.39	24.10
UI-TARS-72B-DPO*	33.24	61.54	12.77	25.45	43.48	6.71	33.33	33.33	23.53	47.83	25.80
UI-TARS-1.5-7B	28.80	50.00	4.26	36.15	39.13	9.77	25.00	46.67	18.75	47.83	25.07
Frameworks with Qwen											
Agent S2.5	**36.96**	42.31	10.64	17.02	30.43	11.83	33.33	53.33	17.65	47.83	24.65
GTA1 (N=8)	15.22	61.54	6.38	6.38	8.70	7.53	29.17	20.00	29.41	39.13	17.17
Agent S3 (N=1)	34.78	57.69	10.64	17.02	21.74	10.75	33.33	40.00	23.53	39.13	23.82
Agent S3 (N=3)	32.61	53.85	17.02	17.02	34.78	9.68	29.17	53.33	23.53	43.48	25.21
Agent S3 (N=10)	34.78	61.54	12.77	23.40	34.78	12.90	45.83	**53.33**	11.76	43.48	27.70
FDC	34.78	**65.38**	12.77	21.28	34.78	**16.13**	**45.83**	33.33	**35.29**	**65.22**	**30.19**
Frameworks with OpenAI o3											
GTA1*	34.78	73.08	40.43	44.60	60.74	37.05	58.33	86.67	35.47	82.61	48.59
Agent S2.5*	52.09	76.92	55.32	55.32	47.83	**39.53**	75.00	73.33	42.00	73.91	54.20
Agentic Lybic*	**60.78**	84.62	51.06	**59.48**	69.56	35.56	79.17	73.33	63.75	73.91	56.90
FDC	58.70	**92.31**	**55.32**	55.32	**73.91**	38.71	**79.17**	**86.67**	**64.71**	**82.61**	**58.72**

Table 1: Success rate (SR, %) on OSWorld across application domains. Results are grouped by end-to-end models and agentic frameworks under two planner backbones (Qwen3-VL-30B-A3B-Instruct and OpenAI o3). * denotes results reported on the public OSWorld leaderboard. Best results within each block are in **bold**.

Table 2: Efficiency comparison on OSWorld. Token Cost is the average total input and output tokens per task. N is the number of candidates used by scaling-based baselines. For FDC, M is the average number of explored recovery strategies per execution step.

Model	Token Cost	SR (%)
Agent S3 (No Scaling)	2,305,613	23.82
GTA1 ($N = 8$)	6,767,093	17.17
Agent S3 ($N = 3$)	14,879,269	25.21
Agent S3 ($N = 10$)	48,577,441	27.70
FDC ($M = 2.13$)	**2,333,364**	**30.19**

Method	15 steps	50 steps	100 steps
Agent S3	20.78	23.82	23.82
FDC	22.71	30.19	31.86

Table 3: Effect of execution steps budget on SR(%).

4.4.2 Module Ablation

We next conduct a module-level ablation to isolate the contribution of the Hierarchical Recovery Flow and the Adaptive Strategy Selection mechanism. All experiments are conducted with a 50-step limit.

As shown in Table 4, removing the Hierarchical Recovery Flow entirely, while retaining Action-Grounded Dual Failure Evaluation and feeding its signals back to the agent, reduces the success rate from 30.19% to 25.21%. In this setting, the agent is informed of failures but lacks an explicit mechanism to escape. These results indicate that failure awareness without an explicit recovery mechanism yields limited improvement.

Removing Adaptive Strategy Selection results in a comparable degradation (26.04%). In this variant, the agent still generates multiple recovery candidates, but relies solely on an LLM-as-a-Judge to select among them, without incorporating rule-based safety and novelty constraints. We observe that this substantially weakens the effectiveness of the recovery flow. Without explicit penalties on repetitive or high-risk actions, the judge will select recovery plans that map to the same action types or target elements that have already failed in recent steps, leading to recurrence of the same failure loops observed in other agent frameworks.

Together, these ablations show that hierarchical recovery and adaptive strategy selection are jointly necessary for performance gains. Simply increasing the number of candidate strategies is insufficient if the agent lacks reliable criteria to discriminate among them. The adaptive combination of novelty and safety constraints with semantic judgment is therefore essential to translating recovery attempts into actual performance gains.

4.4.3 Backbone Model Ablation

We further evaluate the robustness of our framework under different planner and executor configurations. Table 5 reports results across three settings. Replacing the planner with a stronger reasoning model (OpenAI o3) while keeping the executor fixed (UI-TARS-1.5-7B) yields a substantial improvement (30.19% to 58.72%), indicating that

Module Configuration	SR(%)
w/o Hierarchical Recovery Flow	25.21
w/o Adaptive Strategy Selection	26.04
Ours (Full)	**30.19**

Table 4: Module ablation on OSWorld with a fixed 50-step budget (Qwen3-VL-30B-A3B-Instruct as planner and recovery modules, UI-TARS-1.5-7B as executor).

Planner	Executor	SR
Qwen3-VL-30B	UI-TARS-1.5	30.19%
OpenAI o3	UI-TARS-1.5	58.72%
Qwen3-VL-30B	Qwen3-VL-8B	32.13%

Table 5: Backbone configuration ablation on OSWorld. We vary the planner (Qwen3-VL-30B-A3B-Instruct vs. OpenAI o3) and the executor (UI-TARS-1.5-7B vs. Qwen3-VL-8B-Instruct) to assess the effect of planning and execution backbones on SR.

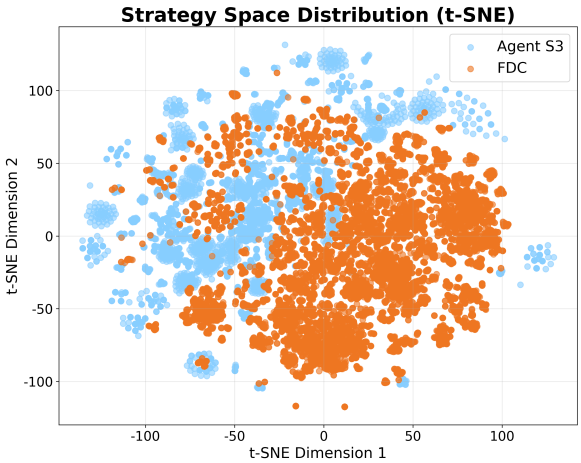

Figure 4: Trajectory-level visualization of plan embeddings using t-SNE. Each point corresponds to a plan generated at one execution step, encoded with Sentence-BERT.

stronger reasoning amplifies. Replacing the executor with Qwen3-VL-8B-Instruct while keeping the planner fixed results in a modest gain (30.19% to 32.13%), consistent with its stronger execution reliability reported on public leaderboard. Overall, performance improvements from stronger planners or executors are additive, while the primary gains stem from the failure-aware control structure rather than reliance on a specific model configuration.

4.5 Exploratory Analysis

To better understand how different frameworks explore the strategy space during execution, we perform a trajectory-level analysis of generated plans. We extract the textual plan produced by the agent and encode it into a semantic embedding using Sentence-BERT. We then apply t-SNE to project these embeddings into a two-dimensional space. Figure 4 visualizes the resulting distributions for Agent S3 and FDC. The baseline exhibits dense clusters, while FDC produces a more dispersed distribution, suggesting exploration of a broader set of recovery strategies.This analysis is diagnostic rather than evaluative: the embedding space does not indicate whether the explored strategies are optimal, but highlights how the agents' exploration behaviors differ.

5 Conclusion

This work addresses a fundamental limitation of existing computer-use agent frameworks: the lack of explicit control over how decision-making adapts when execution failures prevent progress. Based on empirical analysis on OSWorld, we show that representative agentic baselines can stall when execution actions fail to induce meaningful environment changes, causing the agent to repeatedly condition on unchanged observations and regenerate similar plans without effective recovery. To address this issue, we propose Failure-aware Dual-flow Control (FDC), which explicitly separates routine execution from adaptive recovery and regulates transitions between them using action-grounded failure signals. FDC integrates action-grounded dual failure evaluation with a hierarchical adaptive recovery mechanism to revise decisions only when execution becomes ineffective, while avoiding the overhead of continuous replanning or inference-time scaling. Extensive experiments demonstrate that this control-centric design consistently improves execution reliability and efficiency across model backbones, highlighting explicit failure-aware control as a key design principle for robust computer-use agents in real OS environments.

Limitations

While Failure-Aware Dual-Flow Control improves robustness and execution efficiency, several limitations remain. First, the hierarchical recovery strategies are guided by predefined templates that encode general patterns of escalation. Although this design provides a clear control interface and stabilizes recovery behavior, the templates themselves are static and do not adapt based on accumulated experience. Learning or dynamically updating recovery templates, potentially via online adaptation or meta-learning, remains an open direction. Second, by explicitly expanding the agent's exploration space during recovery, FDC generates diverse, structured recovery trajectories. While we exploit this structure purely at inference time, it also suggests a promising avenue for data synthesis and training-time supervision, which we do not explore in this work. Finally, while our evaluation focuses on GUI-based interaction in OSWorld, other environments may expose different interaction primitives, such as API-level actions, command-line operations, or structured tool invocations, under which the effectiveness of failure-aware dual-flow control has not yet been evaluated.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2024. Agent s: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*.

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, Bangkok, Thailand. Association for Computational Linguistics.

Tianyu Fu, Anyang Su, Chenxu Zhao, Hanning Wang, Minghui Wu, Zhe Yu, Fei Hu, Mingjia Shi, Wei Dong, Jiayao Wang, and 1 others. 2025. Mano technical report. *arXiv preprint arXiv:2509.17336*.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. The unreasonable effectiveness of scaling agents for computer use. *arXiv preprint arXiv:2510.02250*.

Liangxuan Guo, Bin Zhu, Qingqian Tao, Kangning Liu, Xun Zhao, Xianzhe Qin, Jin Gao, and Guangfu Hao. 2025. Agentic lybic: Multi-agent execution system with tiered reasoning and orchestration. *arXiv preprint arXiv:2509.11067*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, and 1 others. 2024. Autoglm: Autonomous foundation agents for guis. *arXiv preprint arXiv:2411.00820*.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. 2025. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, and 1 others. 2025. Coact-1: Computer-using agents with coding as actions. *arXiv preprint arXiv:2508.03923*.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025a. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, and 1 others. 2025b. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*.

Penghao Wu, Shengnan Ma, Bo Wang, Jiaheng Yu, Lewei Lu, and Ziwei Liu. 2025a. Gui-reflection: Empowering multimodal gui models with self-reflection behavior. *arXiv preprint arXiv:2506.08012*.

Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. 2025b. Backtrackagent: Enhancing gui agent with error detection and backtracking mechanism. *arXiv preprint arXiv:2505.20660*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2024. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, and 1 others. 2025. Scaling computer-use grounding via user interface decomposition and synthesis. *arXiv preprint arXiv:2505.13227*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, and 1 others. 2023. Lemur: Harmonizing natural language and code for language agents. *arXiv preprint arXiv:2310.06830*.

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. 2025. Aria-UI: Visual grounding for GUI instructions. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22418–22433, Vienna, Austria. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, and 1 others. 2025. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Appendix

A Hierarchical Strategy Templates

This appendix provides additional details on the hierarchical strategy templates used in the recovery flow of Failure-aware Dual-flow Control (FDC). These templates guide the generation of candidate recovery strategies at different levels of deviation from the current execution, complementing the high-level description in Section 3.

Layer 1: Local Refinement. This layer focuses on improving the current execution approach without altering its overall structure. Typical strategies involve refining action targets, adjusting interaction parameters, or correcting minor execution-level mistakes. The goal is to preserve the original plan while resolving local failures.

Layer 2: Interaction Modality Shift. This layer encourages switching the interaction modality or control method when repeated failures suggest limitations of the current approach. Examples include replacing pointer-based interactions with keyboard-driven commands or alternative input patterns to bypass interaction bottlenecks.

Layer 3: Navigation Reconfiguration. This layer aims to break out of a stalled execution state by reconstructing the navigation path. Strategies may involve refreshing the interface, backtracking to a previously stable state, or jumping directly to the target location to escape local dead-ends.

Layer 4: Task Reconsideration or Termination. This layer represents the most aggressive form of recovery. The agent either reformulates the execution plan at a higher level while preserving the original task objective, or determines that the task is unrecoverable under the current context and terminates gracefully.

Template Injection. During recovery, the active strategy layer is determined by the cumulative failure count c_f, which increases with consecutive failures and resets upon a successful execution step. To encourage strategy diversity without excessive exploration, the strategy generator uses detailed guidance corresponding to the active layer, while simultaneously injecting concise guidance from shallower layers. This design biases candidate generation toward minimally sufficient deviations, while retaining the ability to escalate when simpler interventions fail.

B Adaptive Strategy Scoring

This appendix provides the complete formulation of the adaptive strategy scoring mechanism used in the recovery flow of Failure-aware Dual-flow Control (FDC). The main text presents only the high-level aggregation logic for clarity, while detailed scoring components are specified here for reproducibility.

B.1 Semantic Strategy Evaluation

Each candidate recovery strategy $p \in \mathcal{C}$ is evaluated by a Strategy Evaluation Agent, which assesses its plausibility given the task instruction $\mathcal{I}$ and the current execution context H_t. The raw output of the evaluator is discretized into a three-valued semantic score:

$$S_{\text{semantic}}(p) = E_{\text{LLM}}(\mathcal{I}, H_t, p) \in 1.0, 0.5, 0.0 \tag{6}$$

corresponding to strong support, neutrality, and rejection, respectively.

In addition, the evaluator outputs a confidence level

$$\gamma \in \text{high}, \text{medium}, \text{low} \tag{7}$$

which reflects the reliability of the semantic judgment and is used to modulate score aggregation.

B.2 Rule-Based Scoring

To complement semantic evaluation with execution-oriented constraints, each candidate strategy is assigned a rule-based score S_{rule}, defined as a weighted combination of safety and novelty components:

$$S_{\text{rule}} = 0.5 \cdot S_{\text{safety}} + 0.5 \cdot S_{\text{novelty}} \tag{8}$$

Safety Score. The safety score S_{safety} acts as a soft constraint on operational risk. Action types that have been repeatedly verified as failures in recent execution steps are assigned a score of zero. For other actions, the safety score decays as the recovery layer deepens, reflecting increased tolerance for deviation when simpler recovery attempts fail:

$$S_{\text{safety}} = \begin{cases} 1.0, & \text{Layer 1}, \\ 0.8, & \text{Layer 2}, \\ 0.6, & \text{Layer 3}, \\ 0.4, & \text{Layer 4}. \end{cases} \tag{9}$$

The novelty score discourages repetitive recovery attempts by penalizing strategies that resemble

recent failures. It is defined as:

$$S_{\text{novelty}} S_{\text{novelty}} = 1 - 0.5 \cdot \mathbb{I}(a_{\text{type}} \in H_{\text{type}}) - 0.5 \cdot \text{Sim}(e_{\text{desc}}, H_{\text{desc}}) \quad (10)$$

where $\mathbb{I}(\cdot)$ denotes whether the proposed action type has appeared in recent failed steps, and $\text{Sim}(\cdot)$ computes string similarity between the target element description e_{desc} and recently attempted descriptions H_{desc} using sequence matching.

B.3 Confidence-Aware Score Fusion

The final score used to select a recovery strategy combines semantic and rule-based evaluations:

$$S_{\text{final}} = \alpha(\gamma) \cdot S_{\text{semantic}} + \big(1 - \alpha(\gamma)\big) \cdot S_{\text{rule}}, \quad (11)$$

where the confidence-dependent weight $\alpha(\gamma)$ is defined as:

$$\alpha(\gamma) = \begin{cases} 0.8, & \gamma = \text{high}, \\ 0.7, & \gamma = \text{medium}, \\ 0.6, & \gamma = \text{low}. \end{cases} \quad (12)$$

When the semantic evaluator exhibits low confidence, the weighting shifts toward the rule-based score, reducing reliance on potentially unreliable semantic judgments and improving robustness under uncertainty.

C Implementation Details

This section provides additional implementation details for the main experiments reported in Section 4, including execution settings, model configurations, and evaluation protocols.

Execution Budget and Termination. All agents are evaluated with a maximum execution budget of 50 steps per task. A task terminates successfully if the final environment state satisfies the task-specific evaluation script, and fails otherwise. Early termination occurs if the agent explicitly concludes that the task is infeasible or if the environment becomes unrecoverable (e.g., required files are missing).

Agent Configuration. Our framework follows a planner–executor architecture. The executor is responsible for grounded GUI actions and remains fixed across experiments, while the planner and recovery-related components vary by backbone model. Unless otherwise specified, UI-TARS-1.5-7B is used as the executor. For planning and recovery reasoning, we evaluate two backbones: Qwen3-VL-30B-A3B-Instruct and OpenAI o3.

Decoding Configuration. All language model components in our framework operate under deterministic decoding to reduce stochastic variance during evaluation. Unless otherwise specified, we set the temperature to 0 and top-p to 1.0 for all planner, verification, reflection, and recovery-related prompts. This configuration ensures that observed performance differences primarily reflect architectural and control design choices rather than sampling randomness.

Failure Signals. Failure-aware transitions are triggered by action-grounded verification or trajectory-level reflection. Action verification relies on screenshot differencing and action localization, while reflection evaluates task progress based on the current observation and execution history. The cumulative failure count c_f increases only when consecutive failures are detected and resets upon a verified successful step.

Evaluation Protocol. All results are reported as mean success rates over two independent runs to account for environment stochasticity. We follow the standard OSWorld evaluation protocol and exclude Google Drive tasks due to authentication instability, consistent with prior work. Token usage is measured by aggregating all model input and output tokens incurred during execution, including recovery-related reasoning.