DUOGATE: ACTION AND CONTEXT CONTROL FOR RELIABLE LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents can report success without executing the actions a task requires. When runtime checks correct such failures, their explanations provide learning signals beyond the final task outcome. Using this feedback effectively requires deciding when an interaction warrants learning and which lessons to retain. We introduce DuoGate, a framework that connects action control with selective context adaptation through execution feedback. Its action branch checks selected proposals against available evidence, directs recovery toward detected gaps, and preserves correction feedback for learning. Its context branch uses outcomes and execution signals to select learning updates. For these updates, a reflection and curation workflow derives candidate guidance from the interaction and correction reasons. Separate admission checks screen this guidance before it enters a playbook for later prompts, so an intervention need not become a retained lesson. We evaluate DuoGate on AppWorld, τ^2-Bench, and Terminal-Bench 2, finding higher mean completion than the adaptive context baseline across all three benchmarks. With MiniMax M2.7, DuoGate raises AppWorld task completion from the baseline's 60.7% to 69.2% and scenario completion from 41.7% to 54.2%. These results demonstrate the complementary benefits of action control and context adaptation, highlighting correction feedback as an effective signal for learning from execution. Code: `https://anonymous.4open.science/r/DuoGate-56CF/`.

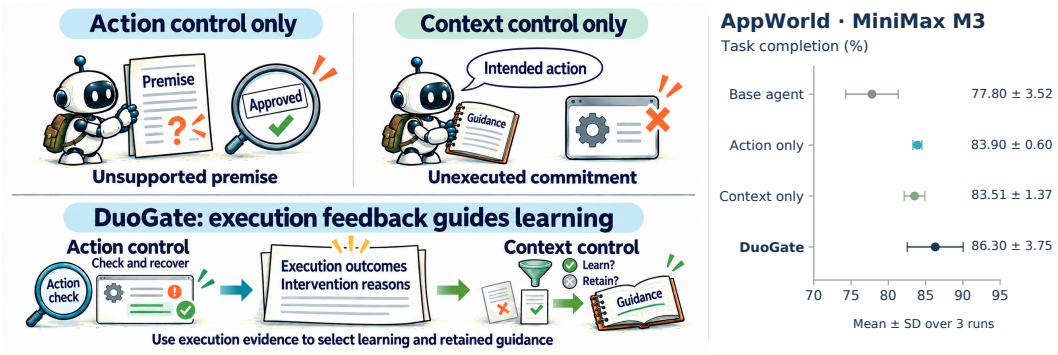


Figure 1: DuoGate connects action control with selective context adaptation. Action checks can overlook unsupported task assumptions, and guidance alone does not ensure that intended operations are executed. Outcomes and execution signals inform learning eligibility, correction reasons inform reflection, and admission checks screen candidate guidance. On AppWorld with MiniMax M3, DuoGate achieves higher mean task completion than either branch alone. Points and error bars show means and sample standard deviations over three runs; Action only and Context only denote DuoGate-Action and DuoGate-Context.

1 INTRODUCTION

LLM agents carry out tasks through sequences of actions, using feedback from code execution and tool calls to guide subsequent steps (Yao et al., 2023; Wang et al., 2024b). They can also use this feedback to adapt their context, reflecting on their interactions and retaining the resulting lessons as reusable guidance (Shinn et al., 2023; Zhao et al., 2024; Suzgun et al., 2026; Zhang et al., 2026;

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.

Ouyang et al., 2026). By bringing earlier experience into later decisions, this guidance can help an agent choose an appropriate action, yet the agent may claim to have performed that action without actually executing it.

Runtime monitors address such execution gaps by checking proposed steps against contextual support, procedural constraints, or execution risk (She et al., 2026; Elkoussy & Perez, 2026; Wang et al., 2026; Yang, 2026a). A check might catch a mismatch between a reported count and the retrieved records, yet accept a numerically correct count based on the wrong year. Retained guidance can prompt the agent to establish the relevant dates before retrieving records, addressing an assumption that this check leaves unexamined. When a check triggers a correction, its explanation also provides feedback that the agent can use to improve guidance for later decisions.

Learning from this feedback is not equally useful for every interaction. Some interactions provide too little usable evidence to justify reflection, and even informative feedback can lead to advice that duplicates existing guidance or omits conditions needed for reuse (Suzgun et al., 2026; Zhang et al., 2026). The challenge is to turn correction feedback into reusable guidance without reflecting on every interaction or retaining every proposed lesson.

To address this challenge, we introduce DuoGate, a framework that connects evidence-based action control with selective updates to adaptive context through execution feedback. Its action branch checks selected code and tool actions, outcome claims, and completion decisions against available evidence, guiding recovery when checks identify missing support and preserving correction feedback for learning. The context branch uses recorded outcomes and execution signals to select interactions for learning. For each selected interaction, a reflection and curation workflow derives candidate guidance from the interaction and correction feedback. Admission checks then determine which candidates enter the playbook, a collection of reusable guidance included in later prompts. The agent draws on this playbook to propose later steps, which encounter action control again and generate experience for subsequent learning decisions.

We evaluate DuoGate on AppWorld (Trivedi et al., 2024), τ^2-Bench (Barres et al., 2026; Shi et al., 2026), and Terminal-Bench 2 (Merrill et al., 2026). DuoGate achieves higher mean completion than the adaptive context baseline across all three benchmarks and all four evaluated AppWorld base LLMs. With MiniMax M2.7, it raises AppWorld task completion from the baseline's 60.7% to 69.2% and scenario completion from 41.7% to 54.2%. AppWorld comparisons further show higher mean completion when learning receives original correction explanations or generic verification feedback than when explanations are omitted or mismatched.

Our contributions are as follows.

- We introduce DuoGate, a framework that connects evidence-based action control with selective updates to adaptive context. Its context branch separately determines when to learn from an interaction and which resulting guidance to retain.
- Experiments on AppWorld, τ^2-Bench, and Terminal-Bench 2 demonstrate that DuoGate achieves higher mean completion than the adaptive context baseline, with improvements across all four evaluated AppWorld base LLMs. Component comparisons show complementary benefits from combining action and context control.
- We show that action-correction feedback benefits learning beyond its role in immediate recovery. Incorporating correction explanations into context updates improves mean task and scenario completion, supporting the use of runtime feedback to shape guidance for later decisions.

2 Related Work

Feedback can improve an agent's current response and inform later decisions. Self-correction methods revise outputs using self-feedback or tool-assisted critiques (Madaan et al., 2023; Gou et al., 2024). Beyond revising individual outputs, agents can retain experience as executable skills (Wang et al., 2024a). Adaptive guardrails similarly retain safety checks, defense rules, and judgment precedents to inform subsequent oversight (Luo et al., 2025; Liu et al., 2026; Yang, 2026b).

At execution time, oversight centers on whether a proposed action should proceed. AgentTrust assesses tool-use risk (Yang, 2026a), and AgentLTL enforces procedural constraints over action traces (Elkoussy & Perez, 2026). ProvenanceGuard checks contextual support (She et al., 2026), and

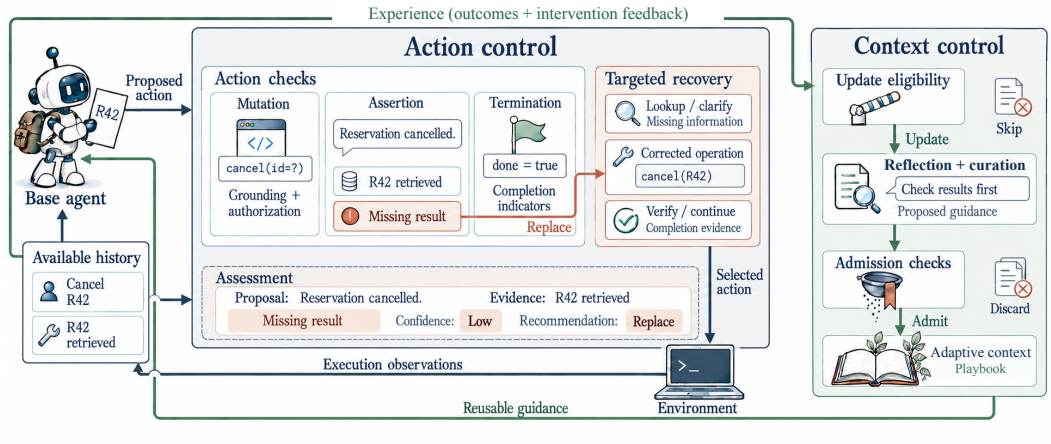


Figure 2: Overview of DuoGate. Action control checks selected proposals against available evidence. The highlighted path shows recovery from an unsupported cancellation claim; unchanged proposals proceed directly. Outcomes and recorded controller feedback inform context control, which selects learning updates, derives guidance through reflection and curation, and screens candidate guidance before adding it to the playbook. Skip bypasses reflection and curation; discard omits a candidate. Checks and learning triggers vary by interface.

AgentSpec provides configurable enforcement through triggers and predicates (Wang et al., 2026). DuoGate connects this oversight to selective learning, using the resulting interaction to update the base agent's reusable task guidance.

Context adaptation shapes later proposals by retaining such guidance as reflections and insights (Shinn et al., 2023; Zhao et al., 2024), strategies and workflows (Ouyang et al., 2026; Wang et al., 2025), or conditional guidelines (Fu et al., 2024). To refine this guidance, Dynamic Cheatsheet's cumulative variant DC(CU) curates accumulated advice (Suzgun et al., 2026), GEPA searches over prompt revisions (Agrawal et al., 2026), and ACE incrementally updates a playbook through reflection and curation (Zhang et al., 2026). DuoGate connects this workflow to action control, using recorded correction explanations to derive reusable guidance. Outcomes and execution signals determine when to initiate learning, and separate admission checks determine which guidance enters the playbook.

3 Method

3.1 Overview

DuoGate connects action control with context control through the feedback generated during task execution (Figure 2). Given a task request, the base agent proposes a step using its interaction history and a playbook of reusable guidance. Before the step proceeds, the action control branch routes recognized operations, outcome claims, and completion requests to the applicable evidence checks. It then allows the proposal to proceed unchanged or selects a recovery step to address detected gaps. Observations from the selected step and recorded controller feedback, including correction reasons, enter the interaction history, informing subsequent proposals and providing experience for learning.

The context control branch decides whether to learn from this experience (Figure 3). Recorded outcomes and execution signals, including interventions and errors, inform this eligibility decision. When learning is selected, we derive candidate guidance from the interaction and recorded correction reasons using a reflection and curation workflow adapted from ACE (Zhang et al., 2026). Admission checks then determine which candidate entries are merged into the playbook. If learning is skipped or no candidate is admitted, the agent continues with the existing playbook. Admitted guidance is supplied in later prompts, shaping subsequent proposals under continued action control without changing model weights.

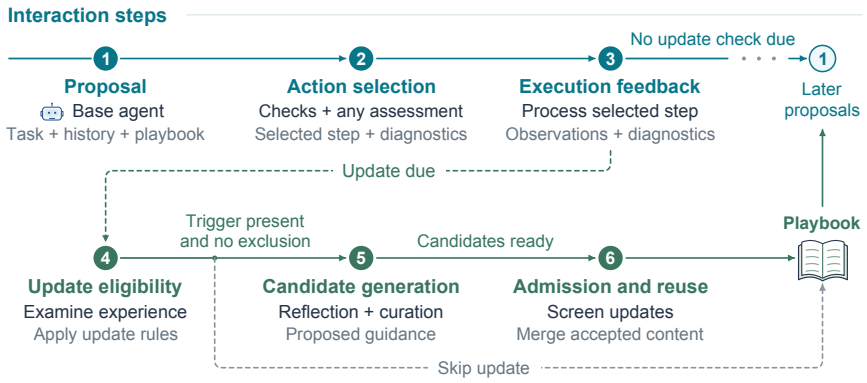


Figure 3: Execution and conditional learning in DuoGate. The agent proposes a step, action control selects what proceeds, and observations combine with controller feedback (diagnostics) to form execution feedback (stages 1 to 3). When a learning update is considered, the update gate determines eligibility for reflection and curation, and admission checks screen the resulting candidates (stages 4 to 6). Later proposals use the existing playbook unless new guidance is admitted.

3.2 Action Control

DuoGate's action controller checks proposed operations and claims against available evidence and directs recovery toward detected gaps. It recognizes information-gathering actions, mutations that change task state, assertions about operations or outcomes, and completion requests. Since a code block can combine several of these components, the controller builds an evidence summary from the task, interaction history, and proposal, bringing together relevant argument values, the user's request, previous operations and results, and progress signals.

The evidence summary supports four types of checks.

- **Grounding** checks compare argument values with recorded evidence.
- **Authorization** checks examine whether a mutation matches the user's request.
- **Outcome** checks assess whether recorded operations and results support a claim.
- **Completion** checks use progress and verification signals to decide whether the agent should stop, continue, or verify.

The checks distinguish identifying an action's target, establishing permission, and confirming execution. For example, retrieving a reservation identifier identifies the booking but does not authorize its cancellation, and permission to cancel does not establish that cancellation occurred. AppWorld and τ^2-Bench combine local rules with an LLM assessor for targeted assessment. Local rules resolve proposals by allowing recognized low-risk actions or triggering interventions without a separate assessor call. The controller selects proposals for assessment based on operation type and recent execution signals. For each selected proposal, the assessor considers the task, evidence summary, and supplied history or guidance, returning a confidence category or execution recommendation with reasons and recovery advice. The controller combines the assessment with applicable local checks to allow the proposal or select recovery. Local fallback rules handle proposals when assessment is unavailable. Terminal-Bench 2 uses a local completion gate (Appendix A).

Recovery addresses the gap identified during action selection. The controller gathers missing information through a lookup or clarification, corrects unsupported operations or responses, or requests continued execution or verification when completion evidence is insufficient. It screens replacement steps according to their operation type. For example, replacing an unsupported cancellation claim with a cancellation call involves checking the call's arguments and scope before execution. The selected step is an executable action, a response, or corrective feedback that defers an operation. Appendices A.1, A.2, and A.4 detail evidence extraction, recovery screening, and regeneration or intervention limits.

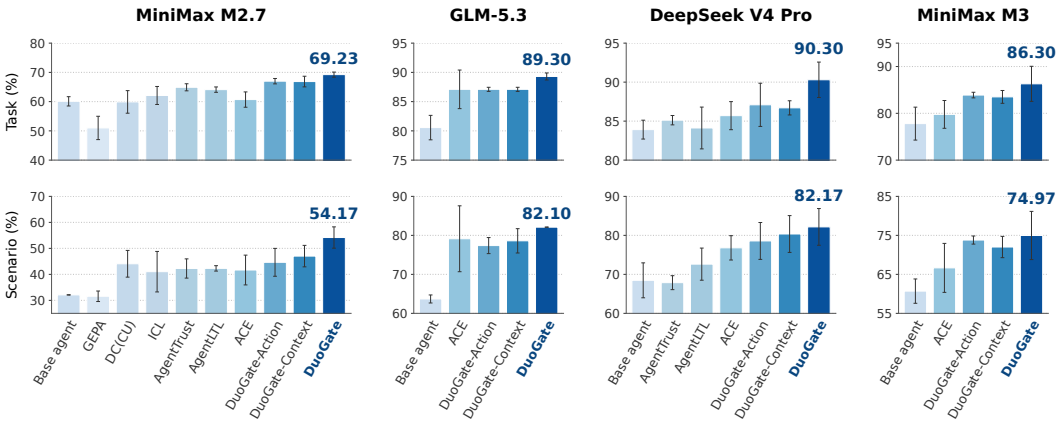


Figure 4: AppWorld completion across four base LLMs. Bars and whiskers show mean ± sample SD over three runs, and numeric labels mark the highest mean in each panel. Both rows use the same method order; y-axis ranges vary by panel. Full numerical results, including recorded tokens, are in Table 10 (Appendix C.1).

3.3 Execution Feedback

Execution feedback connects action control to subsequent proposals and context updates by preserving what happened and why the controller intervened. It combines observations from the selected step with recorded controller feedback, including correction reasons, in the interaction history used by the base agent. Context control forms a learning record from this history or an execution summary, drawing on the proposal, selected step, observed result, and controller feedback (Appendix A.5).

Context control uses the record's outcomes and execution signals, including interventions, errors, and changes in output, to select learning updates. For an eligible update, reflection examines the actions and results together with correction reasons to derive lessons (Shinn et al., 2023; Zhao et al., 2024). Correction reasons explain the gap that prompted recovery, distinguishing an unsupported outcome claim from a missing action argument even when both interactions end in success. They therefore guide reflection toward what needed correction, information that the final outcome alone does not provide. Section 4.4 evaluates how correction reasons affect completion.

3.4 Context Control

Context control manages playbook updates through the three stages of reflection, curation, and admission. An update gate first evaluates outcomes and execution signals in the learning record to decide whether to run this process. The gate passes eligible records to reflection and curation and skips both stages for ineligible records, leaving the playbook unchanged.

Reflection derives lessons from the interface-specific learning record: the interaction and current playbook on AppWorld and τ^2-Bench, or an iteration summary on Terminal-Bench 2. The lessons describe what to repeat, avoid, or verify in later interactions. This stage supports playbook updates; reflection within the base agent's execution loop remains separate (Appendix A.5).

Curation converts the reflected lessons into proposed playbook updates, expressing them as reusable procedures, constraints, environment facts, or verification practices. For example, a lesson about a missing operation result can become a candidate entry instructing the agent to verify that result before reporting completion. These candidate entries then undergo admission checks.

Admission checks determine which candidate entries enter the playbook by screening their structure, content, size, and duplication. It rejects incomplete entries and unsupported update types, and filters recognized advice that bypasses completion requirements. Duplicate handling omits repeated guidance or merges it into an existing entry. Candidates omitted at admission follow the discard path in Figure 2; admitted candidates add or revise playbook entries. If none are admitted, the playbook remains unchanged. The agent uses the resulting playbook in later prompts, connecting retained guidance to subsequent proposals under continued action control (Appendix A.5).

Table 1: Domain success and recorded tokens on τ^2-Bench using MiniMax M2.7. Values are mean $\pm$ sample standard deviation over three runs. Tokens are in millions per run, summed over domains. Bold marks the best mean (higher success, lower usage); light blue shading marks full DuoGate.

Agent method	Airline (%) ↑	Retail (%) ↑	Telecom (%) ↑	Banking (%) ↑	Recorded tokens (M) ↓	
					Input	Output
DC(CU)	78.67 ±4.16	75.73 ±3.54	88.01 ±3.96	6.87 ±3.32	96.30 ±11.57	1.48 ±0.03
mini-SWE-agent v2	80.00 ±2.00	72.22 ±3.09	91.22 ±2.63	10.65 ±2.14	98.63 ±4.60	1.58 ±0.03
Terminus-2	74.67 ±1.15	72.79 ±6.12	91.22 ±0.88	8.94 ±0.60	105.94 ±0.98	1.56 ±0.01
Base agent	72.67 ±4.62	72.51 ±2.82	92.11 ±0.88	10.65 ±1.19	88.84 ±2.93	1.48 ±0.07
AgentTrust	74.67 ±5.03	73.10 ±2.68	91.81 ±3.95	10.65 ±1.57	99.72 ±7.72	1.75 ±0.08
AgentLTL	75.33 ±7.02	78.07 ±3.16	92.98 ±3.04	10.31 ±2.73	96.81 ±4.98	1.73 ±0.15
ACE	80.67 ±1.15	75.15 ±3.65	93.27 ±2.68	5.84 ±1.20	94.49 ±8.24	1.47 ±0.04
DuoGate-Action	76.67 ±5.03	76.61 ±4.05	95.32 ±1.34	6.53 ±2.60	**69.92** ±32.41	**1.38** ±0.39
DuoGate-Context	74.00 ±4.00	78.07 ±4.64	94.74 ±0.88	8.25 ±1.78	105.31 ±3.73	1.51 ±0.02
DuoGate (ours)	**81.33** ±4.16	**78.36** ±2.53	**95.91** ±0.51	**12.37** ±1.78	86.23 ±2.98	1.45 ±0.05

4 EXPERIMENTS

We evaluate DuoGate on AppWorld, τ^2-Bench, and Terminal-Bench 2 with four base LLMs. Across every setting, it outperforms adaptive context and runtime oversight baselines (Section 4.2), and ablations show that both branches contribute to these gains (Section 4.3). We then test the value of correction feedback for learning beyond immediate recovery (Section 4.4). We separately examine learning-update selection and guidance admission (Section 4.5). DuoGate also uses fewer recorded tokens than ACE in five of six settings (Section 4.6).

4.1 EXPERIMENTAL SETUP

Benchmarks and models. We evaluate on AppWorld (Trivedi et al., 2024), whose 168 test-normal tasks form 56 scenarios of three related variants; τ^2-Bench (Barres et al., 2026; Shi et al., 2026), covering the Airline, Retail, Telecom, and retrieval-based Banking domains; and the 89 tasks of Terminal-Bench 2 (Merrill et al., 2026). Playbooks persist across tasks on AppWorld and τ^2-Bench, and are reset between tasks and updated within each task on Terminal-Bench 2. MiniMax M2.7 is the common base LLM; AppWorld additionally uses MiniMax M3, GLM-5.3, and DeepSeek V4 Pro.

Baselines. ACE (Zhang et al., 2026) is the primary baseline because DuoGate adapts its reflection and curation workflow. We compare ACE with the base agent and three DuoGate configurations: DuoGate-Action and DuoGate-Context, which enable only the action or the context branch, and full DuoGate. ACE and the DuoGate configurations share task order and execution limits. AgentTrust (Yang, 2026a) and AgentLTL (Elkoussy & Perez, 2026) provide runtime oversight. Further baselines are Dynamic Cheatsheet (DC(CU); Suzgun et al., 2026) on all benchmarks; GEPA (Agrawal et al., 2026) and fixed demonstrations (ICL; Brown et al., 2020) on AppWorld with MiniMax M2.7; and mini-SWE-agent v2 (Yang et al., 2024) and Terminus-2 (Merrill et al., 2026), the terminal base agent, on τ^2-Bench and Terminal-Bench 2. Coverage and configurations are detailed in Appendix B, and controller protocols in Appendix A.

Metrics. We report task completion and, on AppWorld, scenario completion, which requires all three variants of a scenario to succeed in the same run. Numbers in the text are three-run means, and differences between them are in percentage points (pp); tables add sample standard deviations, and Appendix F lists individual runs. Tokens are recorded input and output tokens per run for agent execution (including recovery), reflection, and curation, excluding assessor calls (Appendix D).

4.2 COMPARISON WITH ADAPTIVE CONTEXT AND RUNTIME OVERSIGHT

DuoGate achieves higher mean completion than every baseline in most evaluated settings (Figure 4; Tables 1 and 2). On AppWorld, the gains over ACE hold for both task and scenario completion across all four evaluated base LLMs. With MiniMax M2.7, DuoGate improves task completion

Table 2: Terminal-Bench 2 with MiniMax M2.7; Terminus-2 is the base agent. Tokens are in millions per run. Statistics, bold, and shading follow Table 1.

Agent method	Success (%) ↑	Recorded tokens (M) ↓	
		Input	Output
mini-SWE-agent v2	46.82 ±3.43	115.94 ±4.01	**1.73** ±0.06
DC(CU)	44.20 ±4.25	110.24 ±11.31	2.13 ±0.18
Base agent	46.05 ±4.05	97.12 ±13.97	1.94 ±0.03
AgentTrust	47.19 ±2.25	147.56 ±34.41	2.18 ±0.17
AgentLTL	46.82 ±1.29	130.45 ±45.97	2.08 ±0.13
ACE	47.57 ±4.25	89.40 ±10.33	2.62 ±0.50
DuoGate-Action	47.19 ±1.12	106.22 ±16.71	2.35 ±0.55
DuoGate-Context	44.94 ±2.97	105.46 ±4.11	2.68 ±0.04
DuoGate (ours)	**50.19** ±2.34	**88.46** ±4.19	2.34 ±0.50

by 8.5 pp and scenario completion by 12.5 pp over ACE. The scenario gain means that DuoGate completes more groups of related tasks with all three variants succeeding in the same run. These improvements across base LLMs and both completion metrics establish a consistent advantage over the adaptive-context workflow on which DuoGate builds. DuoGate's advantage also holds against baselines that explicitly check agent behavior. It exceeds AgentTrust and AgentLTL in both task and scenario completion on AppWorld with MiniMax M2.7 and DeepSeek V4 Pro (Table 10). The combined framework therefore improves completion over evaluated methods from both the adaptive-context and runtime-oversight families.

The gains extend across code, dialogue, and terminal interfaces. With MiniMax M2.7 as the common base LLM, DuoGate achieves the highest mean completion in every τ^2-Bench domain and on Terminal-Bench 2, where it improves task completion over ACE by 2.6 pp (Tables 1 and 2). DuoGate improves completion both when it reuses guidance across tasks, as in the dialogue benchmark, and when it adapts guidance within a single task, as in the terminal benchmark.

4.3 Contributions of the action and context branches

We find that combining the branches improves mean completion over either branch alone in all 13 comparisons of completion metrics across the evaluated base LLMs and benchmarks (Figure 4 and Tables 1 and 2). Action control and retained guidance address different sources of failure: action control corrects execution gaps, while guidance informs the reasoning that produces proposals. In a τ^2-Bench Telecom task, DuoGate-Context announces a transfer to a human agent without calling the transfer tool and fails, whereas DuoGate identifies the missing operation and executes the call (Appendix E.1).

Retained guidance can address assumptions that local action checks leave unresolved. In an AppWorld counting task, the agent computes an answer over the wrong year, and action-only control accepts it. Full DuoGate instead consults retained time-anchoring guidance, checks timestamped records, and answers correctly (Appendix E.1). Guidance thus influences the reasoning that produces a proposal, including steps that proceed without intervention. On AppWorld with MiniMax M3, action control replaces code in only 3.3% of recorded decisions.

In Banking, action control alone reaches 6.5% mean completion and context control alone reaches 8.3%. Both fall below the base agent's 10.7%, yet combining the branches raises completion to 12.4%. Combining the branches improves completion even when neither branch alone improves over the base agent. In the reported means for Airline, Banking, and Terminal-Bench 2, DuoGate's gain over the base agent also exceeds the sum of the gains from each branch alone. DuoGate connects these branches by using recovery experience for learning and retained guidance to inform subsequent proposals. Complete numerical comparisons appear in Appendix C.4 (Table 11).

4.4 Correction feedback as a learning input

Correction explanations improve learning beyond their role in immediate recovery. On AppWorld with MiniMax M2.7, four conditions differ only in the explanations that reflection and curation re-

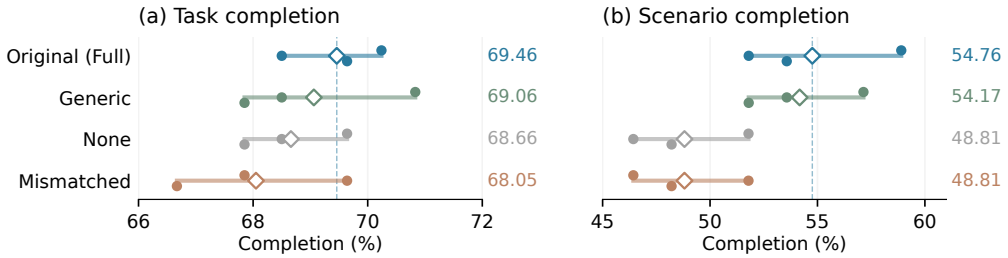


Figure 5: Learning-input explanation ablation on AppWorld with MiniMax M2.7, with three runs of 168 tasks and 56 scenarios per condition. Circles show individual runs, diamonds their means, and segments observed ranges. Dashed lines mark Original means. Vertical offsets do not indicate paired runs.

ceive. *Original* retains them, *Generic* substitutes fixed generic verification feedback, *Mismatched* substitutes a diagnostic template from a reason category the controller did not detect, and *None* removes explanations while preserving intervention indicators and execution feedback. All conditions preserve controller explanations in the Agent's runtime feedback and use the same action-control, recovery, eligibility, and admission policies. Each uses three complete 168-task runs with matched task order, initial playbook, and budgets. Original is a fresh rerun of full DuoGate (Appendix C.5.1).

The completion gain from correction explanations is larger at the scenario level: *Original* exceeds *None* by 6.0 pp in mean scenario completion and 0.8 pp in mean task completion. An independent run set shows the same gains at this precision (Figure 5 and Appendix C.5.2). The larger scenario gain suggests that reusable guidance helps the agent complete all three related variants under the same runtime control policies. For example, in an archived trajectory, DuoGate retains a checklist for rejecting an incorrect payment request and sending the corrected amount after one variant, then follows it successfully when the checklist appears in the next variant's input.

Generic verification feedback recovers most of the completion gain from original correction explanations. *Generic* comes within 0.4 pp of *Original* in task completion and 0.6 pp in scenario completion. *Mismatched* offers no observed gain over *None*: both reach 48.8% scenario completion, with lower mean task completion under mismatched explanations (Appendix C.5.1). Generic verification feedback improves subsequent completion over intervention indicators alone or mismatched reasons.

4.5 Selecting learning updates and admitting guidance

Selecting learning opportunities and screening the resulting guidance each improve mean completion (Table 3). A 2×2 ablation on AppWorld with MiniMax M2.7 compares the two decisions independently and in combination. *Selective* applies the default eligibility policy, while *Always* learns at every normal task ending. *Full* applies the default admission checks, while *Minimal* removes duplicate and content filters while preserving structural requirements and size limits. Each configuration uses three fresh runs, separate from the main benchmark and explanation-ablation runs, with matched task order, initial playbook, execution and context limits, and all other control policies and correction explanations (Appendix C.6).

Table 3: Update-selection and admission ablation on AppWorld with MiniMax M2.7. Values are mean $\pm$ sample SD over three fresh runs per configuration. Tokens include Agent (with recovery), Reflector, and Curator, excluding assessor calls, in millions per run. Light blue shading marks default DuoGate (Selective + Full). Bold marks the best mean in each column.

Selection + Admission	Task (%) ↑	Scenario (%) ↑	Input (M) ↓	Output (M) ↓
Always + Minimal	67.26 ± 1.19	44.05 ± 4.49	53.76 ± 1.67	0.84 ± 0.05
Selective + Minimal	68.07 ± 1.25	47.62 ± 2.73	52.99 ± 1.52	0.81 ± 0.03
Always + Full	68.07 ± 0.70	50.60 ± 2.07	53.01 ± 1.60	$\mathbf{0.80} \pm 0.04$
Selective + Full	$\mathbf{69.26} \pm 1.36$	$\mathbf{54.17} \pm 1.03$	$\mathbf{51.93} \pm 3.50$	0.82 ± 0.02

With full admission, selective learning improves scenario completion by 3.57 pp over always-on learning. With selective learning, full admission improves scenario completion by 6.55 pp over

minimal admission checks. Both decisions also improve task completion. Selection also improves completion under minimal admission, and full admission improves completion under always-on learning. Selection and admission provide complementary benefits by controlling which interactions trigger learning and which resulting lessons enter the playbook. Default DuoGate (Selective + Full) combines the highest completion means with the lowest total recorded token usage among these configurations.

4.6 Token cost of selective adaptation

DuoGate uses fewer recorded tokens on average than ACE in five of the six benchmark and base LLM combinations, with the largest reduction on AppWorld with MiniMax M3 (65%, Table 10). Overall usage depends on both learning updates and Agent execution, where the agent repeatedly receives retained guidance. The component breakdown shows how savings from reflection and curation combine with changes in execution usage (Figure 6).

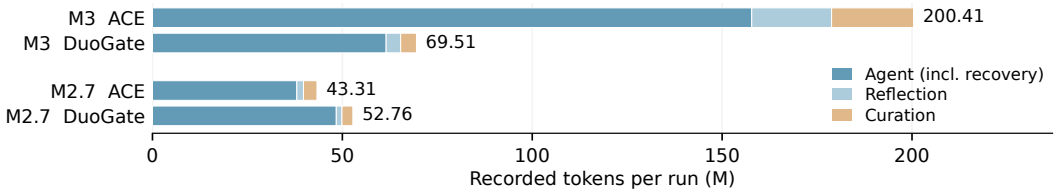


Figure 6: Recorded AppWorld tokens per run by component for ACE and DuoGate with MiniMax M3 and MiniMax M2.7. Bars show three-run means, with totals in millions. Agent includes recovery. We exclude assessor calls. Component values are in Tables 19 and 20.

DuoGate controls learning costs at two stages. Update selection avoids reflection and curation calls: with MiniMax M3, DuoGate skips these calls for 216 of 435 successful task-runs. Admission limits the guidance that learning adds to future prompts: 75 of 256 updates with filtering records leave no new guidance after screening (Appendix E.5). The two stages thus control both the cost of producing guidance and the context the agent carries into later execution. Compared with ACE, DuoGate makes fewer reflection–curation calls and supplies shorter playbooks, with per-run median lengths of 33k–36k characters for DuoGate versus 163k–256k for ACE. Because the agent receives playbook guidance in subsequent prompts, shorter playbooks can reduce token usage throughout execution. Agent execution accounts for 73.5% of DuoGate's total recorded token saving over ACE, despite DuoGate making more Agent calls (Appendix D).

5 Discussion

DuoGate's results support coupling runtime oversight with selective context adaptation. Combining the branches improves mean completion over either branch alone, while correction feedback helps the agent improve subsequent completion under the same runtime control policies. Together, these findings give oversight a role beyond immediate recovery: identifying what the agent should check in future interactions. Task outcomes indicate whether the agent succeeded, while correction feedback identifies execution gaps that future guidance can address. Turning that feedback into useful guidance, however, requires deciding both when to learn and what to retain. The selection and admission ablation supports separating these decisions, since an interaction worth learning from can still produce redundant or unsuitable advice.

More broadly, execution feedback could turn retained guidance into knowledge that an agent continually tests and revises through use. A lesson that resolves one failure may need additional conditions when the agent encounters a different task or environment. Subsequent successes and corrections could help identify those conditions, supporting revisions to existing guidance as well as new additions. This would extend DuoGate's selection and admission mechanisms into a continuing cycle of learning, evaluation, and revision. Such revision also matters for efficiency because retained guidance incurs prompt costs across later interactions. DuoGate's shorter playbooks accompany lower execution token usage even with more agent calls, motivating future work that evaluates accumulated guidance by both its contribution to task completion and its recurring computational cost.

AI USE STATEMENT

We used generative AI tools to retrieve related literature, assist with adapting baseline code, generate case study figures, refine the method figure layout, and check grammar and polish the draft. We reviewed all AI-assisted work. In particular, we carefully reviewed the code throughout the adaptation process, checked the cited sources, and checked the polished text and figures against the experimental records. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

DuoGate aims to reduce unsupported actions and completion claims by checking available evidence before selected proposals proceed. These checks are partial and can fail, so improved benchmark completion does not guarantee safe or authorized behavior. Reusable guidance can also retain incorrect or sensitive information from interaction histories. Applications involving real users would therefore require permissions, privacy controls, and monitoring appropriate to their setting, including review of stored guidance and shared traces. Our evaluation does not establish the safety of deployment in such settings.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide the AppWorld implementation, prompts, and a configuration for full DuoGate at `https://anonymous.4open.science/r/DuoGate-56CF/`. Section 4.1 defines the benchmarks, base LLMs, baselines, evaluation metrics, and aggregation across runs. Appendix A details action control, recovery, and context updates for each interface. Appendix B documents recorded experimental settings, implementation parameters, task order, execution budgets, and playbook initialization and reset policies. The correction-feedback and update-selection/admission ablation protocols and individual runs appear in Appendix C, while Appendix F reports individual runs for the main benchmark comparisons. Appendix D specifies token-counting conventions and component usage, and Appendix E presents trajectory evidence connecting action corrections, learning decisions, and subsequent guidance reuse.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. In *International Conference on Machine Learning*, 2026.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Laïla Elkoussy and Julien Perez. AgentLTL: A trace-verification framework for measuring, enforcing, and training procedural compliance in tool-using LLM agents. *arXiv preprint arXiv:2607.02599*, 2026. URL `https://arxiv.org/abs/2607.02599v1`.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 119919–119948. Curran Associates, Inc., 2024. doi: 10.52202/079017-3811.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024.

Zhe Liu, Zonghao Ying, Wenxin Zhang, Quanchen Zou, Deyue Zhang, Dongdong Yang, Xiangzheng Zhang, and Hao Peng. SafeHarbor: Defining precise decision boundaries via hierarchical memory-augmented guardrail for LLM agent safety. In *International Conference on Machine Learning*, 2026.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. AGrail: A lifelong agent guardrail with effective and adaptive safety detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8104–8139. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.399.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong (Ryan) Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026.

Yining She, Yiliang Liang, and Eunsuk Kang. Safeguarding LLM agents from misalignment through provenance analysis. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2026. doi: 10.1145/3832783.3837535. URL `https://arxiv.org/abs/2607.01236`. To appear.

Quan Shi, Alexandra Zytek, Pedram Razavi, Karthik Narasimhan, and Victor Barres. τ-Knowledge: Evaluating conversational agents over unstructured knowledge. In *International Conference on Machine Learning*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-long.333.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, pp. 2938–2950. ACM, 2026. doi: 10.1145/3744916.3764546.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR, 2024b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025.

Chenglin Yang. AgentTrust: Runtime safety evaluation and interception for AI agent tool use. *arXiv preprint arXiv:2605.04785*, 2026a. URL `https://arxiv.org/abs/2605.04785v1`.

Chenglin Yang. AgentTrust: A self-improving trust layer for AI-agent actions. *arXiv preprint arXiv:2606.08539*, 2026b. URL `https://arxiv.org/abs/2606.08539v1`.

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*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

APPENDIX CONTENTS

A Additional method details 14
A.1 AppWorld controller 14
A.2 τ^2-Bench controller 15
A.3 Domain checks 16
A.4 Terminal-Bench 2 controller 17
A.5 Context-update procedure 18
A.6 Illustrative decision paths 19

B Experimental setup and configurations 20
B.1 Benchmarks, models, and execution settings 20
B.2 Baselines and branch configurations 20
B.3 Task order and budgets 21
B.4 Prompt inputs and initial playbooks 21
B.5 Implementation parameters 22

C Additional results and ablations 24
C.1 AppWorld 24
C.2 τ^2-Bench 25
C.3 Terminal-Bench 2 26
C.4 Complete branch comparisons 27
C.5 Correction explanations in learning 29
C.6 Update selection and admission 32
C.7 Error-window sensitivity 32

D Resource consumption and token accounting 34
D.1 Counting unit and scope 34
D.2 Context length and learning frequency 34
D.3 Accounting conventions by interface 35

E Case studies and observed action–learning paths 35
E.1 Contrasting action and context control in archived trajectories 35
E.2 Learning from a recovered bulk import 38
E.3 Reusing failure-derived guidance during regex repair 39
E.4 Illustrative action recovery and supported continuation 39
E.5 Update eligibility and admission in recorded trajectories 40

F Per-run experimental results 42
F.1 AppWorld repetitions 42
F.2 τ^2-Bench repetitions 43
F.3 Terminal-Bench 2 repetitions 46

G Limitations 46

A ADDITIONAL METHOD DETAILS

This appendix expands the shared action-control and context-update sequence introduced in Section 3. Table 4 maps its stages to the evidence, decisions, and guidance reuse supported by each interface. The following subsections describe the corresponding controllers, domain checks, and update and admission policies, then illustrate how action and learning decisions can differ for the same interaction. Appendix B reports the experimental configurations, and Appendix E examines observed action–learning paths.

Each interface instantiates the action-control procedure of Section 3. The controller builds an evidence summary from the task, interaction history, and proposal, applies the grounding, authorization, outcome, and completion checks relevant to the recognized operations and claims, and allows the proposal or selects recovery. Local rules resolve recognized proposals directly, and proposals selected for targeted assessment receive an LLM assessment (Appendices A.1 and A.2). Observations from the selected step and the controller's feedback, including correction reasons, enter the interaction history used by the base agent. As in Figure 2, the specific checks and learning triggers vary by interface, following the operations and evidence that each interface exposes.

Table 4: Shared stages of DuoGate and their implementation across interfaces. Each interface connects action assessment and recovery to conditional learning, admission, and guidance reuse. Checks, learning triggers, and playbook lifetimes vary by interface; Appendix A.5 details the update procedure.

Stage	AppWorld	τ^2-Bench	Terminal-Bench 2
Action assessment	Parsed API calls, task and conversation evidence, and execution outputs support grounding, authorization, outcome, and completion checks.	Typed tool calls, user intent, domain policy, and tool results support grounding, authorization, outcome, and completion checks.	Command and output history, progress, and verification signals support outcome and completion checks on completion requests; progress signals can trigger recovery feedback.
Recovery feedback	Replacement code or corrective instructions; controller feedback with correction reasons enters the interaction history.	Corrected tool calls or responses and targeted questions; controller feedback with correction reasons enters the interaction history.	Feedback to continue or verify, with the reason for the intervention; a rejected completion request defers its command batch.
Update eligibility	Task-end outcome and execution signals, including interventions and errors, select learning, with optional periodic updates.	Recorded task outcomes select learning; timeouts and infrastructure errors are excluded.	Iteration-level failures, changes in output, and completion requests select learning, subject to cooldown rules.
Reflection and curation	Interaction history with correction reasons and the current playbook inform reflection; curation proposes playbook entries.	Trajectory with its recorded outcome, correction reasons, and the current playbook inform reflection; curation proposes playbook entries.	Iteration summary with correction reasons and the current playbook inform reflection; curation converts the reflected lessons into typed entries.
Admission	Admission checks screen proposed entries; duplicates are omitted.	Admission checks screen proposed entries; duplicates are omitted.	Admission checks screen proposed entries; duplicates merge into existing entries.
Guidance reuse	Updated playbook is available to later tasks within a run.	Updated playbook is available to later tasks within a run.	Entries are reused across iterations of the same task; each task starts empty.

A.1 APPWORLD CONTROLLER

For AppWorld's Python API interface (Trivedi et al., 2024), the controller receives the task instruction, proposed code, surrounding assistant response, and conversation history. It parses an abstract

syntax tree to extract direct calls rooted at `apis`, retaining their names, source locations, and positional and keyword arguments. Literal values remain explicit; expressions that require execution are marked as dynamic. API metadata and operation-name rules classify the calls as observations, mutations, completion calls, or unknown operations, with the supervisor completion call recognized separately.

To assess these calls, the controller builds an evidence summary of previously seen API documentation, identifiers and values, calls, errors, pagination signals, and mutation outcomes. It associates each execution output with the code block that produced it and records the recognized operations in that block together with their execution status, including any detected errors. Grounding checks compare argument values with the recorded identifiers and values, authorization checks compare mutations with the task request, and outcome and completion checks compare claims and completion calls with the recorded operations and results, the task text, and recent messages.

Recent-error detection supplies one signal for assessment and completion checks. For a history of n messages, the latest detected error at message index e remains flagged when $e \geq n - w$, where w is the recent-error window. When this flag is absent, the unresolved-error check scans the last six messages, excluding recognized controller-block echoes and benign verification errors. Thus, $w = 0$ disables the variable-window flag but retains the six-message scan; smaller values of w do not shorten every error-detection path. A detected unresolved error can defer `complete_task()` and prompt further checking or recovery.

In control mode, the proposal follows the decision order below. An early return skips the later stages.

1. Apply task-dependent completion normalization and collection checks. A recognized collection mismatch can trigger an immediate intervention.
2. Allow recognized low-risk proposals, those reaching the ordinary read-only path or containing no extracted API calls, without a separate assessor call. Remaining proposals encounter applicable payment and local completion checks before assessment.
3. Request assessment using the task, code, assistant response, extracted calls, risk category, evidence summary, and recent-history summary. The structured response provides `confidence`, `decision`, `should_execute`, `reason`, `missing_evidence`, and `suggested_code`. The controller combines this response with the applicable local checks to allow the proposal or select recovery.
4. If assessment is unavailable, apply the local completion and mutation fallback rules. The mutation fallback checks dynamic arguments against entity evidence, and repeated blocks of the same proposal are subject to an intervention limit.

Recovery addresses the detected gap. It can request API documentation or a lookup, provide a corrective instruction, replace the proposal with corrected code, or remove an unsupported completion call from a larger code block. Replacement code is screened according to its operation type and may contain recognized API operations or permitted computation. The selected code enters the interaction history together with the controller's feedback, which identifies the gap in the original proposal and gives the correction reason, so that subsequent proposals can use it. The learning record formed from this history therefore draws on the proposal, the selected step, its observed result, and this feedback (Appendix A.5).

A.2 τ^2-BENCH CONTROLLER

For the dialogue interface of τ^2-Bench (Barres et al., 2026), the controller receives typed assistant messages containing text or tool calls. It builds an evidence summary of entity values, operation records and their results, user intent and confirmation signals, and recent tools, then checks recent progress. Grounding checks compare selected arguments, such as identifiers, with this summary, authorization checks compare mutations with the user's request and confirmation, and outcome checks compare claims with recent tool records. Domain checks handle domain-specific fields (Appendix A.3).

The controller uses this evidence in the following order.

1. Handle terminal and loop conditions. A successful prior transfer can end the conversation. A detected loop can produce a targeted clarification or terminal response if the proposal does not break the loop.

2. Select proposals for assessment under `risk_only` or `all`. Selective assessment covers recognized mutations, outcome claims, terminal actions, schema-critical operations, critical lookups, and loop signals. Ordinary low-risk proposals can bypass assessment, but applicable outgoing domain guards still run.
3. Assess selected proposals using the proposed message, domain policy, tools, playbook, recent history and tool results, evidence summary, argument-grounding result, and loop state. The response includes confidence, check results, `recommended_control`, and `best_recovery_action`. A mutation judged ready can proceed; an unsupported completion claim can instead prompt a corrected tool call, followed by argument and domain checks.
4. If that correction is unavailable or fails its checks, or another gap needs recovery, select a lookup, verification, targeted question, or revised response. For a regenerated mutation, a second assessment accepts or rejects the candidate without requesting another regeneration. Other replacement types follow their respective screening rules.

When assessment is unavailable, local fallback rules use the risk category and the grounding result for critical arguments. Low-risk lookups proceed directly, and other proposals proceed when the applicable local checks find no missing critical argument, subject to outgoing domain guards. The selected assistant action enters the policy-visible history together with the controller's feedback, including the correction reason when the controller intervenes, and the decision is logged. Appendix A.3 describes the additional domain guards.

A.3 Domain checks

Domain checks apply the check types of Section 3 to specific operations, using recognized tool names, argument fields, and conversation patterns. Table 5 lists, for each domain, the check type and condition examined and the recovery selected when a gap is detected. Each decision path applies the checks relevant to its recognized operations.

In AppWorld, completion handling also distinguishes action tasks from tasks requiring a textual answer.

Table 5: Domain-specific action checks. Each row gives the check type (Section 3), the condition examined, and the recovery selected when the check detects a gap. Each decision path applies the relevant checks to recognized operations and available evidence.

Domain or operation	Check and condition examined	Recovery
AppWorld payments	*Authorization:* a recognized payment mutation must match the request-versus-send direction of the task. *Grounding:* sending needs receiver and amount evidence; acting on a request needs its identifier.	Replace a direction mismatch or unsupported argument with recovery instructions.
AppWorld collection completion	*Completion and outcome:* compare expected and proposed identifier and action sets from the task, recent output, and code.	Missing targets or wrong actions block completion; suspected extras prompt verification.
AppWorld collection mutation	*Authorization:* check explicit ineligible targets and wrong actions against the extracted sets.	Block recognized violations.
τ^2 retail	*Grounding and authorization:* check order and item identity against available evidence and the latest request, coverage of multi-object requests, profile versus order address, variant constraints, and mutation side effects.	Identify missing objects, suggest a read-only lookup, or ask for clarification.
τ^2 airline	*Grounding:* compare booking and update arguments with recent reservation and flight evidence, including known flight numbers and route endpoints.	Constrain recognized booking and in-place update operations to arguments supported by this evidence.
τ-Banking	*Grounding:* check transfer-reason values and wrapper arguments against the tool's expected fields and the available account and tool evidence.	Correct argument values to the expected fields, or gather missing information through a lookup or targeted question.
τ^2 telecom	*Grounding:* check selected critical arguments against recorded evidence, and check that a proposed tool is available to the assistant rather than the user.	Correct missing arguments from recorded evidence, or correct a call to a user-side tool into a response instructing the user.

A.4 TERMINAL-BENCH 2 CONTROLLER

The controller extends Terminus-2 for Terminal-Bench 2 (Merrill et al., 2026). It recognizes the proposed command batch and completion requests, and builds its evidence summary from the command and output history, progress signals, and verification signals. Outcome and completion checks use this evidence to decide whether the agent should stop, continue, or verify. Local rules resolve completion requests before the proposed command batch executes. Under the defaults, completion requires a score of at least 0.72 and a positive verification signal within the last four iterations. If either condition is unmet, the controller clears the command batch, resets the completion flag, and returns feedback that states the reason and asks the agent to continue or verify. Commands bundled with a rejected completion request are therefore deferred as well.

Each score calculation starts from a base value, the confidence stated in the current response or 0.65 if none is stated, and applies the progress, verification, and budget adjustments below before clipping the result to $[0, 1]$; adjustments do not accumulate across iterations. The score is a decision value rather than a calibrated probability. Context pressure depends on the model's effective context limit and reserved token margin; time pressure depends on the available timeout budget.

Detected condition	Adjustment
Repeated commands	-0.20
Current failure	-0.20
Repeated observations	-0.15
Empty proposal without completion or a stated wait condition	-0.15
Context pressure	-0.15
Approaching the time budget	-0.15
Completion requested without recent verification	-0.20
Recent positive verification	$+0.10$

After each iteration, the controller records command and observation fingerprints, error signatures, and verification signals, then recomputes the score. A low score produces recovery feedback for a nonterminal iteration if no intervention has already been recorded for it and the recovery-prompt budget remains; completion blocks have a separate counter. The original assistant response remains in the interaction history, followed by the controller's feedback, which states the correction reason and the requested recovery. The trace records the same reason together with the intervention type, verification signals, and token usage.

Verification signals come from output phrases or a standalone marker. The marker is tested first; otherwise, recognized failure text suppresses positive phrase detection for that observation. The recent-verification condition is satisfied if any of the last four iterations has a positive verification signal, and it is combined with the other adjustments above in the completion score.

A.5 Context-update procedure

Context control adapts ACE's reflection and curation (Zhang et al., 2026) to each interface and organizes playbook updates into the three stages of Section 3: reflection, curation, and admission, preceded by an update gate (Figures 2 and 3). The learning record is formed from the interaction history or, on Terminal-Bench 2, from an execution summary of each iteration. It draws on the original proposal, the selected step, its observed result, and the controller's feedback, including correction reasons. The update gate evaluates the outcomes and execution signals in this record, passes eligible records to reflection and curation, and skips both stages for ineligible records, leaving the playbook unchanged. Reflection examines the actions, results, and recorded correction reasons together with the current playbook and derives lessons about what to repeat, avoid, or verify. Curation converts these lessons into candidate entries that express reusable procedures, constraints, environment facts, or verification practices. Admission screens each candidate's structure, content, size, and duplication: it rejects structurally incomplete entries and unsupported update types, and filters recognized advice that bypasses completion requirements. Duplicate handling omits repeated guidance or merges it into an existing entry. Admitted candidates add or revise playbook entries, and the resulting playbook is supplied in later prompts. Following the common sequence in Table 4, the procedures below specify how each interface forms the learning record, selects updates, generates candidate guidance, and decides what to admit. Initialization and resets are described in Appendix B.

Reflection in this procedure supports playbook updates and is separate from reflection within the base agent's execution loop, such as the analysis and plan that precede the commands in a Terminus-2 response or the reasoning that accompanies an AppWorld code proposal. In-loop reflection informs the agent's next proposal and becomes part of the recorded interaction, but it does not modify the playbook.

In AppWorld, ACE, DuoGate-Action, DuoGate-Context, and DuoGate receive the task instruction and the same categories of execution feedback, including API responses, execution outputs, and runtime errors. The interaction history contains generated code, any recorded analysis or plans, and, when action control is enabled, the controller's feedback and correction reasons. Configurations with learning enabled form the learning record from this history. All no-GT learning templates leave reference-solution code and complete test reports unpopulated.

The AppWorld update gate evaluates the task-end outcome together with execution signals: interventions (replaced code, blocked completion, and collection interventions), execution errors, and high step counts. Failures and interactions with any of these signals are eligible, including successful interactions that required correction; successes without such signals skip the update, and unknown outcomes remain eligible. Reflection receives the conversation, including correction reasons, and the current playbook. Curation receives the reflection together with the task context, playbook, and

conversation, and returns a reasoning string and candidate entries, each with a section and content. Permitted sections cover procedures, API guidance, mistakes, verification, and related notes. Admission checks structure (a permitted section and nonempty content), content (recognized completion-workaround advice, code fences, selected structured-content patterns, and keyword-based value screening), size (per-entry and per-update limits), and duplication (normalized-text and prefix comparisons with existing entries, which omit repeated guidance). Admitted entries are merged into the playbook, which is supplied to later tasks within the run. When the playbook exceeds the retrieval size target, the prompt includes the entries most relevant to the task, selected by app mentions, task terms, and priorities, together with a prefix of critical guidance.

In τ^2-Bench, the learning record is the task trajectory, which contains the dialogue, tool calls and results, and the controller's feedback, including correction reasons, together with the task's recorded outcome. The update gate evaluates the recorded outcome and error signals of each task. Tasks that end in failure are selected for reflection and curation, whereas tasks terminated by a timeout or an infrastructure error are excluded. Reflection receives the trajectory with its correction reasons and the current playbook. Curation receives the parsed reflections that request an update, together with the current playbook, and proposes candidate entries. Admission checks include structure (a supported update type), content (nonempty guidance), and duplication against existing entries, and admitted entries form a new playbook version that is available to subsequent tasks within the run.

In Terminal-Bench 2, the learning record is an execution summary of each iteration: the analysis, plan, commands, output, completion request, parser feedback, and the controller's feedback, including correction reasons. The update gate evaluates each iteration's execution signals. Failures, changes in output, and completion requests select learning, including completion requests that the controller rejects. A cooldown suppresses updates triggered only by output changes when the playbook is full and reflection occurred recently; failure and completion triggers are exempt. Reflection uses the same underlying model interface as the base agent and receives the task, the iteration summary, and the current playbook. Reference solutions and final verifier reports are not supplied to reflection; final verification runs after agent execution. Reflection returns its lessons in the format

```
{"items": [{"kind": "...", "text": "...", "evidence": "..."}]}.
```

Curation converts these lessons into typed playbook entries within the update step. Kinds cover diagnostics, constraints, environment facts, success and failure patterns, verification, and task strategy. Admission checks structure (a permitted kind with nonempty text and evidence), content (recognized full-solution patterns are rejected), size (limits on retained fields, entries, and the rendered playbook), and duplication. A normalized-text duplicate revises the existing entry, updating its recency, use count, and evidence instead of adding a new entry. When the entry or rendered-size limit is reached, pruning retains recent and reused entries. The resulting playbook is supplied in the agent's subsequent observations within the same task.

A.6 Illustrative decision paths

Table 6 uses three hypothetical interactions to illustrate how action control and learning eligibility respond to the same episode. Each row specifies the conditions for its path and the admission checks applied if learning produces candidate guidance.

Table 6: Illustrative action and context decisions. Each hypothetical interaction specifies the proposal, evidence, action decision, and conditions for learning and admission.

Proposal and evidence	Action decision	Context decision
AppWorld task requests money, but the recognized payment API sends money.	The authorization check replaces the code with recovery instructions and records the replacement and its reason.	The replacement makes the interaction eligible for learning even if the task then succeeds. Reflection uses the correction reason; proposed payment-direction advice receives structure, content, size, and duplication checks.
τ^2-Bench agent claims cancellation after retrieving a reservation, without a cancellation result. Assume the proposal reaches assessment.	The outcome check finds no cancellation result, and the controller requests a corrected cancellation call. It proceeds only if the applicable grounding and domain checks pass.	The correction and its reason enter the interaction history and the learning record. When the update gate selects the task, reflection uses the correction reason to propose verification advice, which curation turns into a candidate entry for admission.
Terminal-Bench 2 agent requests completion with a stated confidence of 0.90 and no recent verification. Assume default settings and no other score adjustments.	The missing-verification adjustment gives 0.70. The completion check defers completion and the command batch, then supplies feedback stating the missing verification.	The completion request triggers reflection. Candidate entries need a permitted kind, text, and evidence; duplicates revise existing entries.

B Experimental setup and configurations

This appendix expands the evaluation protocol in Section 4.1. It brings together benchmark coverage, baseline and branch configurations, execution budgets, prompts, and playbook initialization. Recorded run settings are distinguished from implementation defaults throughout.

B.1 Benchmarks, models, and execution settings

Table 7 summarizes task coverage, models, and execution settings for each benchmark, and Table 9 lists implementation defaults and fixed limits. A run evaluates the complete task set once; an attempt or trial is one evaluation of a task within a run. Appendix F reports every run. Some settings are missing or only partially preserved in the run metadata, and we do not substitute implementation defaults for them: the τ^2-Bench source commit; AppWorld execution timeouts and output-token limits; auxiliary-call budgets, model-client retries, and other launch-level fields; the source tasks of the AppWorld ICL demonstrations; and the Banking corpus revision and index-construction parameters.

B.2 Baselines and branch configurations

Table 8 distinguishes the reflection and curation workflow inherited from Agentic Context Engineering (ACE) (Zhang et al., 2026) from the control decisions introduced in DuoGate. DuoGate-Action enables action control without adaptive context. DuoGate-Context adapts the ACE workflow using DuoGate's update eligibility and admission, with action control disabled. Full DuoGate additionally supplies the recorded correction reasons to learning. Outcomes and execution signals inform update eligibility, while correction reasons inform reflection on selected experience. Admission then determines which proposed guidance enters the playbook. The base agent and ICL use neither action control nor adaptive context. Within each benchmark and base-LLM setting, ACE, DuoGate-Action, DuoGate-Context, and DuoGate use the same task order and execution limits, as well as the same retrieval settings for Banking.

DuoGate-Context is evaluated on AppWorld with MiniMax M2.7, GLM-5.3, DeepSeek V4 Pro, and MiniMax M3, and on τ^2-Bench and Terminal-Bench 2 with MiniMax M2.7. The AppWorld and τ^2-Bench runs use `risk_only` assessment selection. Update eligibility, assessment selection, and admission follow the benchmark-specific stages in Appendix A, where unknown outcomes are handled by the update logic rather than automatically treated as successes without execution signals.

Table 7: Task coverage, models, and execution settings for each benchmark. Implementation defaults are listed separately in Table 9.

Interface	Data and task coverage	Models and execution settings
AppWorld	`test_normal`, 168 tasks per run. Code/evaluator version `0.1.4.dev0` and data version `0.1.0`, as recorded in the MiniMax M3 runs.	Agent, Reflector, and Curator use the evaluated base LLM. Seed 100, as recorded in the MiniMax M3 ACE, DuoGate-Context, and DuoGate runs. Each task is limited to 40 steps.
τ^2-Bench, DuoGate	Three runs each for Airline, Retail, Telecom, and Banking, with 50, 114, 114, and 97 tasks per run, respectively (375 in total).	Agent and user simulator: MiniMax M2.7 across all four domains; seed 300; one trial per task; 200 maximum steps, 10 maximum tool-execution errors, and a 600-second simulation timeout.
Terminal-Bench 2	`terminal-bench@2.0`; 89 locked tasks per run; three ACE and three DuoGate runs. Harbor version `0.13.1`.	Agent: MiniMax M2.7; serial execution; one attempt per task; framework retry limit 0; timeout multiplier 1, with no explicit task-timeout override.

AgentTrust and AgentLTL provide runtime-oversight comparisons on AppWorld with MiniMax M2.7 and DeepSeek V4 Pro, and on τ^2-Bench and Terminal-Bench 2 with MiniMax M2.7. Each DeepSeek V4 Pro runtime-oversight baseline is evaluated over three AppWorld runs, with aggregate results in Table 10 and individual repetitions in Appendix F.1.

The additional comparisons examine other sources of guidance and agent implementations. DC(CU) provides an alternative adaptation baseline across all three benchmarks; GEPA and fixed-demonstration ICL are evaluated on AppWorld with MiniMax M2.7. The dialogue and terminal evaluations also include mini-SWE-agent v2 and Terminus-2; Terminus-2 serves as the terminal base agent. In-context learning (ICL) supplies task demonstrations without updating model parameters (Brown et al., 2020); mini-SWE-agent v2 (Yang et al., 2024) and Terminus-2 (Merrill et al., 2026) provide additional agent baselines. The two AppWorld explanation experiments retain both DuoGate branches and runtime feedback and vary only the correction explanations supplied to learning: the four-condition experiment compares Original, Generic, None, and Mismatched explanations, and the earlier explanation-removal experiment compares full DuoGate with DuoGate no explanation (Appendix C.5).

B.3 Task order and budgets

ACE, DuoGate-Action, DuoGate-Context, and DuoGate follow the same fixed task order within each benchmark, without shuffling. AppWorld follows the line order of task IDs in `test_normal.txt`. For τ^2-Bench, Airline, Retail, and Telecom retain the order of the `base` subset in each domain's task file; Banking tasks follow ascending `task_*.json` filenames. Terminal-Bench 2 runs the locked 89-task manifest serially in alphabetical task-name order. The Terminal-Bench dataset commit is `69671fbaac6d67a7ef0dfec016cc38a64ef7a77c`.

Each AppWorld task has a 40-step limit. Each τ^2-Bench simulation has a 200-step limit, terminates when the tool-execution error count reaches 10, and uses a 600-second timeout. Terminal-Bench 2 retains each task's execution timeout from `task.toml`, with timeout multiplier 1 and no explicit timeout override or additional run-specific turn limit. Each terminal task has one attempt and the framework retry limit is zero; this does not specify retries internal to a model client.

Terminal-Bench 2 uses an effective context limit of 180,000 tokens, recorded for 266 of the 267 DuoGate task-runs; this is a context limit, not a per-call output budget. Component token accounting is given in Appendix D.

B.4 Prompt inputs and initial playbooks

The Agent receives the benchmark task, interface instructions and tools, and the configured demonstrations or playbook. The anonymous repository includes the AppWorld prompts. Assessment payloads and learning inputs are detailed in Appendices A.1, A.2, and A.5.

Table 8: Shared information and execution conditions, and differences among ACE and the DuoGate configurations. Merged cells apply to all four methods. DuoGate adapts ACE's reflection and curation workflow, with separate decisions on update eligibility and admission. Recorded correction reasons are learning inputs. The base agent and ICL omit all adaptive stages and DuoGate action control.

Component or condition	ACE	DuoGate-Action	DuoGate-Context	DuoGate
Task inputs and feedback	Task instructions and standard interaction feedback. Benchmark-specific learning inputs are described in Appendix A.5; DuoGate-Action does not perform learning.			
Task order	The same fixed order within each benchmark, without shuffling (Appendix B.3).			
Execution limits	AppWorld: 40 steps per task. τ^2-Bench: 200 steps, 10 tool-execution errors, and 600 seconds per simulation. Terminal-Bench 2: task-specific timeouts at multiplier 1, one attempt per task, and no framework retries.			
Banking retrieval	BM25, top-$k = 10$, full document bodies, no reranker, and no cross-query deduplication of document bodies.			
Action control	Disabled	Enabled	Disabled	Enabled
Reflection and curation	ACE workflow	None	Adapted from ACE	Adapted from ACE
Update eligibility	ACE procedure	No updates	DuoGate policy	DuoGate policy
Correction reasons in learning	None from DuoGate	No learning	None from DuoGate	Recorded correction reasons
Admission	ACE curation and merging	None	DuoGate checks	DuoGate checks

Within each benchmark and base LLM, all compared methods that use a playbook, including ACE and DuoGate, start from the same initial playbook, which is restored at the beginning of each of the three independent runs; no learned guidance is transferred between runs. On AppWorld, we use the eight-entry `appworld_initial_playbook.txt` distributed with the ACE-AppWorld repository at commit `9f3e92155345`.[1] This file provides general API-use and execution instructions, including API documentation lookup, pagination, credential handling, and task completion. For τ^2-Bench, the initial playbook consists of manually authored, instance-independent operational guidelines. As in AppWorld, this guidance provides a procedural foundation for agent execution and contains no evaluation-instance-specific information or reference solutions. In both benchmarks, configurations with online learning pass the updated playbook to subsequent tasks within the same run. Terminal-Bench 2 instead starts each task-run with an empty playbook and reuses learned guidance only across iterations within that task.

AppWorld ICL demonstrations and Banking retrieval. The AppWorld ICL prompt places three worked examples before the evaluated task: counting Spotify playlists, finding the most-liked song across playlists, and sending movie recommendations from Simple Note by SMS. For Banking, ACE, DuoGate-Action, DuoGate-Context, and DuoGate use the same default BM25 retrieval over the domain knowledge base through `KB_search`, with the settings listed in Table 8.

B.5 Implementation parameters

Table 9 lists implementation defaults, fixed helper limits, and the AppWorld task-step budget used in the comparisons; its Source column distinguishes the run setting from defaults and fixed limits. The AppWorld configuration in the anonymous repository enables action control with selective assessment.

The one-per-turn regeneration budget applies only to regeneration attempts; other model calls are governed separately, and token accounting includes every model call regardless of this budget. The recovery-prompt budget does not cap completion blocks. AppWorld retrieval targets can be ex-

[1] ACE-AppWorld initial playbook at the specified commit.

Table 9: Controller and context-update parameters. Shared execution budgets are described in Appendix B.3, and recorded run settings in Table 7.

Interface	Setting	Value	Source
AppWorld	Task-step budget	40 steps	Run setting
	Update trigger	Failure or execution signals	Default
	Skip successes without execution signals	On	Default
	High-step update trigger	25 steps	Default
	Periodic updates	Off	Default
	Periodic interval when enabled	20 tasks	Default
	New entries per update	2	Default
	Candidate entry limit	800 characters	Default
	Total update limit	1,600 characters	Default
	Oversized-playbook retrieval size target	60,000 characters	Default
	Oversized-playbook retrieval bullet target	80 bullets	Default
τ^2-Bench	Assessment selection	`risk_only`	Default
	Regeneration budget	1 per turn	Default
	Update trigger	Failed task outcome	Default
	Exclusions	Timeout, infrastructure error	Default
	Assessment history limit	24 messages	Fixed limit
	Assessment recent-tool-result limit	8 results	Fixed limit
Terminal-Bench 2	Completion score threshold	0.72	Default
	Recovery score threshold	0.55	Default
	Verification window	4 iterations	Default
	Repetition thresholds	2 occurrences	Default
	Recovery-prompt budget	12 prompts	Default
	Reserved context margin	40,000 tokens	Default
	Soft-timeout ratio	0.85	Default
	Playbook entry limit	24 entries	Default
	Rendered playbook limit	2,400 characters	Default
	Entry text and evidence limit	500 characters each	Fixed limit
	Output-only cooldown when playbook is full	1 iteration	Default

ceeded by prefixed guidance. In Terminal-Bench 2, an admitted candidate can revise an existing entry or later be pruned, so admitted-candidate counts need not equal the increase in stored entries.

C ADDITIONAL RESULTS AND ABLATIONS

This appendix examines the performance and cost tradeoffs behind Section 4.2, then expands the branch comparisons and the two explanation experiments in Sections 4.3 and 4.4. A separate factorial ablation examines update selection and admission (Appendix C.6), followed by recent-error-window sensitivity. Individual repetitions are reported in Appendix F, with the explanation-content runs in Table 14 and the update-selection and admission runs in Table 16. Comparisons of three-run means and standard deviations are descriptive and do not establish statistical significance.

C.1 APPWORLD

Table 10: AppWorld completion and recorded tokens per run. Values are mean ± sample standard deviation over three runs; tokens are in millions. Bold marks the best mean within each base LLM (higher completion, lower usage); light blue shading marks full DuoGate. Token accounting is detailed in Appendix D.

Agent method	Base LLM	Task (%) ↑	Scenario (%) ↑	Recorded tokens (M) ↓	
				Input	Output
GEPA	MiniMax M2.7	51.00 ±3.99	31.57 ±2.02	27.50 ±0.44	0.73 ±0.02
DC(CU)		59.90 ±3.86	44.07 ±5.14	48.52 ±1.89	2.47 ±0.13
ICL		62.10 ±3.08	41.03 ±7.79	28.96 ±0.86	0.53 ±0.01
Base agent		60.10 ±1.59	32.10 ±0.00	**21.18** ±0.29	0.57 ±0.01
AgentTrust		64.90 ±1.21	42.26 ±3.70	23.52 ±0.76	**0.47** ±0.02
AgentLTL		64.10 ±0.92	42.30 ±1.04	23.18 ±1.17	**0.47** ±0.01
ACE		60.70 ±2.62	41.67 ±5.73	42.62 ±0.76	0.69 ±0.04
DuoGate-Action		66.99 ±0.92	44.63 ±5.36	42.72 ±1.92	0.73 ±0.04
DuoGate-Context		66.86 ±1.82	47.02 ±4.13	43.49 ±1.09	0.67 ±0.03
DuoGate (ours)		**69.23** ±0.87	**54.17** ±4.10	51.95 ±3.37	0.82 ±0.02
Base agent	GLM-5.3	80.56 ±2.09	63.69 ±1.03	**22.74** ±0.34	**0.70** ±0.02
ACE		87.10 ±3.30	79.13 ±8.45	84.76 ±7.48	1.55 ±0.01
DuoGate-Action		87.10 ±0.35	77.38 ±2.06	43.44 ±1.09	1.09 ±0.01
DuoGate-Context		87.10 ±0.35	78.60 ±3.12	66.76 ±2.74	1.23 ±0.05
DuoGate (ours)		**89.30** ±0.60	**82.10** ±0.00	62.81 ±3.26	1.72 ±0.13
Base agent	DeepSeek V4 Pro	83.91 ±1.21	68.47 ±4.48	**25.08** ±0.38	**0.56** ±0.02
AgentTrust		85.12 ±0.60	67.87 ±1.79	31.13 ±1.89	2.12 ±0.10
AgentLTL		84.12 ±2.68	72.62 ±4.12	34.18 ±3.03	2.20 ±0.21
ACE		85.70 ±1.80	76.80 ±3.12	82.00 ±5.03	0.64 ±0.01
DuoGate-Action		87.09 ±2.77	78.57 ±4.73	57.12 ±0.84	3.35 ±0.39
DuoGate-Context		86.70 ±0.91	80.35 ±4.72	68.63 ±0.48	3.66 ±0.02
DuoGate (ours)		**90.30** ±2.27	**82.17** ±4.71	62.44 ±4.99	1.52 ±0.22
Base agent	MiniMax M3	77.80 ±3.52	60.70 ±3.12	**21.07** ±0.48	**0.46** ±0.02
ACE		79.77 ±2.97	66.66 ±6.28	198.72 ±35.45	1.69 ±0.07
DuoGate-Action		83.90 ±0.60	73.80 ±1.04	43.17 ±0.48	0.60 ±0.04
DuoGate-Context		83.51 ±1.37	72.01 ±2.73	63.43 ±1.90	1.19 ±0.04
DuoGate (ours)		**86.30** ±3.75	**74.97** ±6.18	68.24 ±3.18	1.27 ±0.11

DuoGate achieves the highest mean task and scenario completion among the evaluated methods with all four AppWorld base LLMs (Table 10). With DeepSeek V4 Pro, adding action control to DuoGate-Context raises task completion from 86.70% to 90.30% and scenario completion from 80.35% to 82.17%, gains of 3.60 and 1.82 percentage points. With GLM-5.3, DuoGate-Context matches ACE's task completion at 87.10% and has lower scenario completion (78.60% versus 79.13%), while full DuoGate reaches 89.30% and 82.10%, respectively. The combined configuration therefore improves both metrics even where context control alone does not exceed ACE. Figure 7 displays both criteria alongside recorded usage for the base agent, ACE, and the three DuoGate configurations.

The DeepSeek V4 Pro runtime-oversight results provide additional comparisons. AgentTrust achieves $85.12 \pm 0.60\%$ task and $67.87 \pm 1.79\%$ scenario completion; AgentLTL achieves $84.12 \pm 2.68\%$ and $72.62 \pm 4.12\%$, respectively. Both DuoGate-Action and full DuoGate have higher means on both criteria. AgentTrust slightly exceeds the base agent on task completion but falls below it

on scenario completion, whereas AgentLTL has higher means than the base agent on both. These comparisons describe the observed three-run means and do not establish statistical significance.

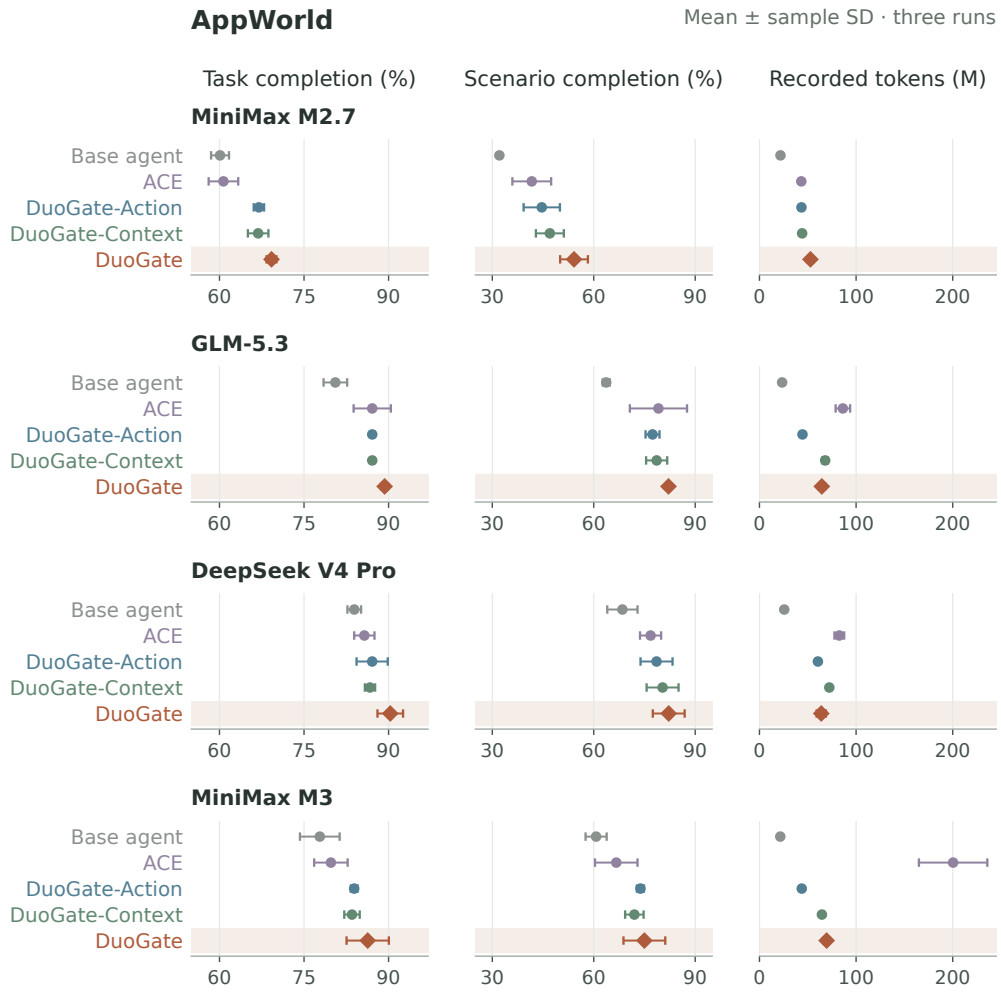


Figure 7: Task completion, scenario completion, and recorded tokens for five common AppWorld configurations across four base LLMs. Markers and whiskers show mean ± sample standard deviation over three runs; shading marks DuoGate. Token totals sum input and output within each run before computing variability. Table 10 includes the additional adaptation and oversight baselines.

The resource comparison also depends on the reference configuration. DuoGate improves both completion metrics over ACE with all four base LLMs and uses fewer recorded tokens on three, with MiniMax M2.7 the exception. Relative to either individual branch, however, its higher completion on the MiniMax base LLMs comes with additional usage. AgentTrust and AgentLTL offer still lower usage on MiniMax M2.7, at fewer than half DuoGate's tokens, with lower completion on both metrics. The DeepSeek V4 Pro comparisons also show lower recorded usage for these baselines: AgentTrust uses 31.13M input and 2.12M output tokens per run, and AgentLTL uses 34.18M and 2.20M, compared with 62.44M and 1.52M for DuoGate. Their input savings outweigh their higher output usage, while both completion metrics remain below DuoGate. With DeepSeek V4 Pro, full DuoGate improves both completion metrics over context-only control while using fewer recorded tokens (63.96M versus 72.29M per run). These comparisons separate the benefit over ACE from the choice between lower-cost configurations and higher completion; Appendix D examines the measured sources of expenditure.

C.2 τ^2-BENCH

The dialogue domains distinguish improvements over adaptive context from improvements over the base agent (Table 1). Banking makes this distinction particularly clear: ACE reaches 5.84% success,

below the base agent's 10.65%, whereas DuoGate reaches 12.37% and exceeds both. By contrast, the leading Retail configurations are closely matched, with DuoGate at 78.36% and both AgentLTL and DuoGate-Context at 78.07%. The domain breakdown therefore identifies where the combined configuration improves on an already competitive alternative and where it recovers the performance lost by individual adaptations.

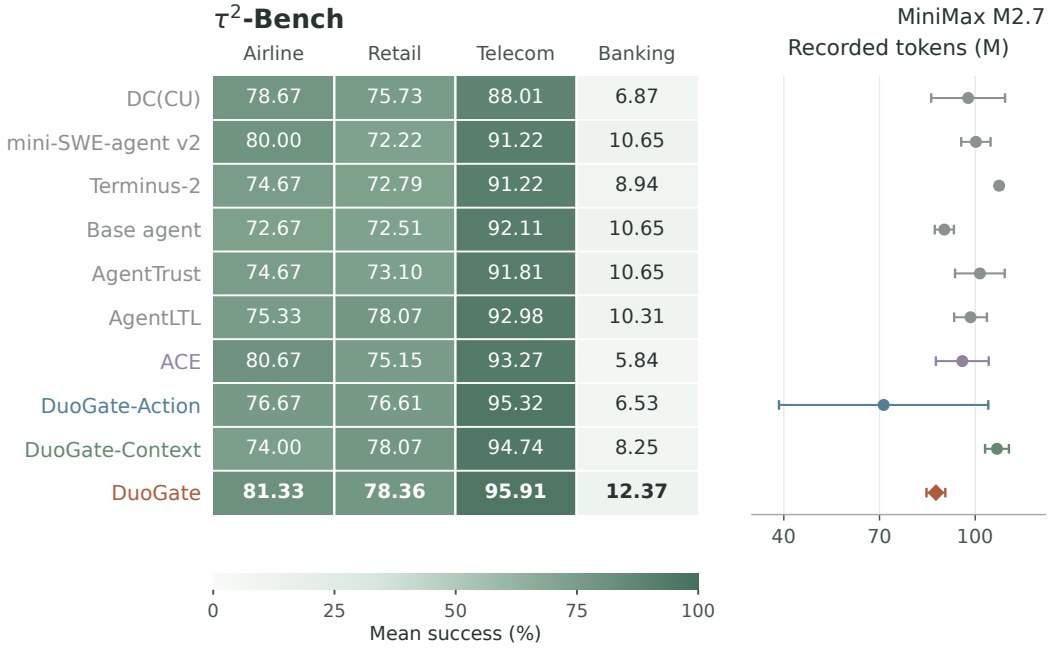


Figure 8: Domain success and recorded token usage on τ^2-Bench with MiniMax M2.7. Cells and colors show mean success in percent over three runs. The right panel shows mean $\pm$ sample standard deviation of recorded input-plus-output tokens, summed over all four domains within each run. All ten configurations are included; Table 1 reports success standard deviations and token coverage.

Combining the branches improves the success–usage tradeoff over context control alone across the four dialogue domains (Figure 8). DuoGate has higher mean success in each domain and uses 87.68M recorded tokens per run in aggregate, compared with 106.83M for DuoGate-Context and 95.95M for ACE. Against action control alone, the success gains instead accompany higher usage, at 87.68M versus 71.29M tokens. Thus, the lower recorded cost relative to context-only adaptation does not make the full configuration the least expensive option. These totals sum all four domains within each run; they do not locate the saving in a particular domain.

C.3 Terminal-Bench 2

On Terminal-Bench 2, DuoGate combines the highest mean success (50.19%) with the lowest mean recorded token usage (90.80M) among the comparable three-run methods using MiniMax M2.7 (Table 2). It improves both measures over either individual branch, whereas the gains over action control alone on the MiniMax AppWorld evaluations and τ^2-Bench require more tokens. The benefit of combining the branches therefore need not entail higher recorded usage, but the cost balance depends on the execution setting.

The total-token advantage is concentrated in input usage (Figure 9). DuoGate has the lowest mean input usage and reduces both input and output usage relative to ACE. Its output usage is nevertheless higher than mini-SWE-agent v2's, at 2.34M versus 1.73M, so the lowest total does not require the lowest value in every component. Across the three runs, DuoGate also has smaller observed spreads than ACE, with success standard deviations of 2.34% and 4.25% and total-token standard deviations of 4.00M and 10.12M. Figure 9 shows the mean comparisons and run variability together.

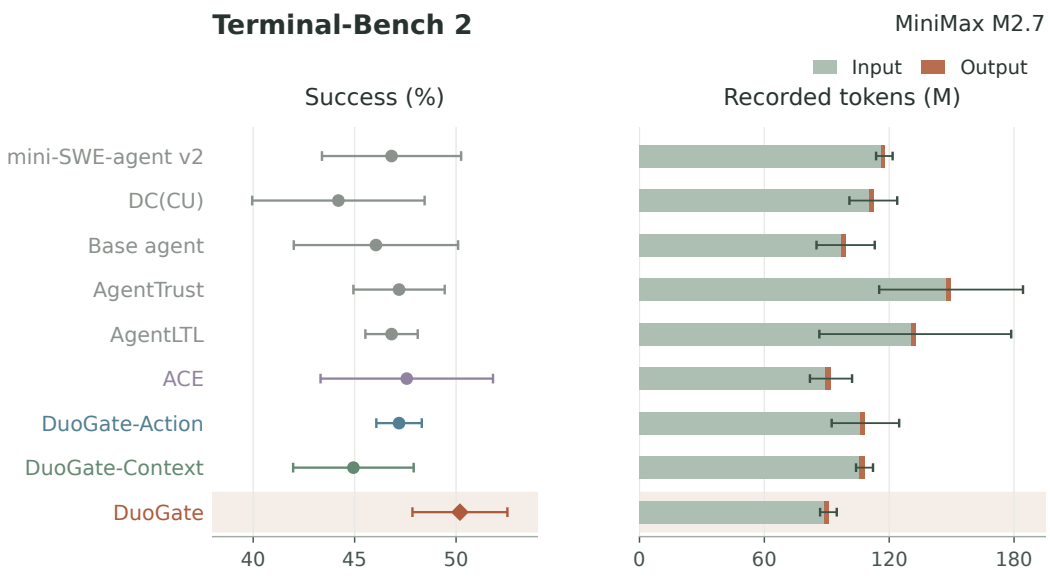


Figure 9: Completion and recorded tokens for all nine Terminal-Bench 2 configurations with MiniMax M2.7. The left panel shows mean success ± sample standard deviation over three runs. The right panel separates mean input and output tokens, with whiskers for the sample standard deviation of their per-run sum. Terminus-2 is the base agent. Token coverage follows Table 2.

C.4 Complete branch comparisons

We compare the base agent, DuoGate-Action, DuoGate-Context, and DuoGate to examine each individual branch and the combined configuration. Table 11 covers AppWorld with all four base LLMs and τ^2-Bench and Terminal-Bench 2 with MiniMax M2.7. All four configurations have three reported runs for every listed base LLM and benchmark, including DuoGate-Context with GLM-5.3. Individual repetitions appear in Appendix F.

Table 11: Completion with action control alone (DuoGate-Action), context control alone (DuoGate-Context), and both branches (DuoGate), compared with the base agent. Values are mean ± sample standard deviation over three runs. The last two columns give differences from the base agent and DuoGate-Action in percentage points (pp), calculated from the displayed means. Dashes suppress self-comparisons. Bold marks the highest reported mean within each benchmark, base LLM, and metric comparison. Light blue shading marks full DuoGate.

Metric	Agent method	Completion (%) ↑	vs. Base agent (pp) ↑	vs. DuoGate-Action (pp) ↑
AppWorld – MiniMax M2.7				
Task	Base agent	60.10 ± 1.59	—	—
	DuoGate-Action	66.99 ± 0.92	+6.89	—
	DuoGate-Context	66.86 ± 1.82	+6.76	−0.13
	DuoGate (ours)	$\mathbf{69.23 \pm 0.87}$	+9.13	+2.24
Scenario	Base agent	32.10 ± 0.00	—	—
	DuoGate-Action	44.63 ± 5.36	+12.53	—
	DuoGate-Context	47.02 ± 4.13	+14.92	+2.39
	DuoGate (ours)	$\mathbf{54.17 \pm 4.10}$	+22.07	+9.54
AppWorld – GLM-5.3				
Task	Base agent	80.56 ± 2.09	—	—
	DuoGate-Action	87.10 ± 0.35	+6.54	—
	DuoGate-Context	87.10 ± 0.35	+6.54	0.00
	DuoGate (ours)	$\mathbf{89.30 \pm 0.60}$	+8.74	+2.20

Continued on next page

Table 11 (continued)

Metric	Agent method	Completion (%) ↑	vs. Base agent (pp) ↑	vs. DuoGate-Action (pp) ↑
Scenario	Base agent	63.69 ± 1.03	—	—
	DuoGate-Action	77.38 ± 2.06	$+13.69$	—
	DuoGate-Context	78.60 ± 3.12	$+14.91$	$+1.22$
	DuoGate (ours)	$\mathbf{82.10 \pm 0.00}$	$+18.41$	$+4.72$
AppWorld – DeepSeek V4 Pro				
Task	Base agent	83.91 ± 1.21	—	—
	DuoGate-Action	87.09 ± 2.77	$+3.18$	—
	DuoGate-Context	86.70 ± 0.91	$+2.79$	-0.39
	DuoGate (ours)	$\mathbf{90.30 \pm 2.27}$	$+6.39$	$+3.21$
Scenario	Base agent	68.47 ± 4.48	—	—
	DuoGate-Action	78.57 ± 4.73	$+10.10$	—
	DuoGate-Context	80.35 ± 4.72	$+11.88$	$+1.78$
	DuoGate (ours)	$\mathbf{82.17 \pm 4.71}$	$+13.70$	$+3.60$
AppWorld – MiniMax M3				
Task	Base agent	77.80 ± 3.52	—	—
	DuoGate-Action	83.90 ± 0.60	$+6.10$	—
	DuoGate-Context	83.51 ± 1.37	$+5.71$	-0.39
	DuoGate (ours)	$\mathbf{86.30 \pm 3.75}$	$+8.50$	$+2.40$
Scenario	Base agent	60.70 ± 3.12	—	—
	DuoGate-Action	73.80 ± 1.04	$+13.10$	—
	DuoGate-Context	72.01 ± 2.73	$+11.31$	-1.79
	DuoGate (ours)	$\mathbf{74.97 \pm 6.18}$	$+14.27$	$+1.17$
τ^2-Bench – MiniMax M2.7				
Airline	Base agent	72.67 ± 4.62	—	—
	DuoGate-Action	76.67 ± 5.03	$+4.00$	—
	DuoGate-Context	74.00 ± 4.00	$+1.33$	-2.67
	DuoGate (ours)	$\mathbf{81.33 \pm 4.16}$	$+8.66$	$+4.66$
Retail	Base agent	72.51 ± 2.82	—	—
	DuoGate-Action	76.61 ± 4.05	$+4.10$	—
	DuoGate-Context	78.07 ± 4.64	$+5.56$	$+1.46$
	DuoGate (ours)	$\mathbf{78.36 \pm 2.53}$	$+5.85$	$+1.75$
Telecom	Base agent	92.11 ± 0.88	—	—
	DuoGate-Action	95.32 ± 1.34	$+3.21$	—
	DuoGate-Context	94.74 ± 0.88	$+2.63$	-0.58
	DuoGate (ours)	$\mathbf{95.91 \pm 0.51}$	$+3.80$	$+0.59$
Banking	Base agent	10.65 ± 1.19	—	—
	DuoGate-Action	6.53 ± 2.60	-4.12	—
	DuoGate-Context	8.25 ± 1.78	-2.40	$+1.72$
	DuoGate (ours)	$\mathbf{12.37 \pm 1.78}$	$+1.72$	$+5.84$
Terminal-Bench 2 – MiniMax M2.7				
Success	Base agent	46.05 ± 4.05	—	—
	DuoGate-Action	47.19 ± 1.12	$+1.14$	—
	DuoGate-Context	44.94 ± 2.97	-1.11	-2.25
	DuoGate (ours)	$\mathbf{50.19 \pm 2.34}$	$+4.14$	$+3.00$

The individual branches usually improve on the base agent, but neither does so uniformly. Each comparison uses one completion metric for a given base LLM and benchmark or domain. DuoGate-Action exceeds the base agent in twelve of thirteen comparisons, with Banking the exception as success falls from 10.65% to 6.53%. DuoGate-Context improves eleven of thirteen comparisons, with lower success in Banking (8.25% versus 10.65%) and Terminal-Bench 2 (44.94% versus 46.05%). These counts summarize metric comparisons rather than independent benchmarks.

Combining the branches improves all thirteen completion comparisons against both DuoGate-Action and DuoGate-Context. With DeepSeek V4 Pro, the gains over action-only control are 3.21 percentage points in task completion and 3.60 points in scenario completion; the corresponding gains over context-only control are 3.60 and 1.82 points. Banking illustrates why the individual-branch results alone do not predict the combined outcome. Both branches fall below the base agent when used separately, whereas DuoGate reaches 12.37% and exceeds all three. These contrasts concern complete configurations and do not isolate the effects of individual checks.

C.5 Correction explanations in learning

Two AppWorld experiments with MiniMax M2.7 test whether the correction explanations attached to action-control interventions help learning beyond execution outcomes and intervention indicators. Both keep action control, recovery, runtime feedback, update eligibility, and admission unchanged and vary only the explanations supplied to the Reflector and Curator. The four-condition experiment compares original, generic, mismatched, and absent explanations; the earlier explanation-removal experiment compares full DuoGate with a variant that omits them. Table 12 summarizes how the two experiments differ. They use separate runs and token scopes, and their results are not pooled.

Table 12: The two explanation experiments on AppWorld with MiniMax M2.7. Each configuration has three runs of 168 tasks in 56 scenarios.

	Four-condition experiment	Earlier explanation-removal experiment
Configurations	Original (full DuoGate), Generic, None, and Mismatched	Full DuoGate and DuoGate no explanation
Full DuoGate runs	Rerun for this experiment	Runs of the main comparison (Table 10)
Runs without explanations	None	DuoGate no explanation, a run set separate from None
Token scope	Agent (including recovery), Reflector, and Curator	Agent only
Reported in	Section 4.4 and Figure 5	Section 4.4, as a replication

C.5.1 Four-condition explanation-content experiment

Treatments. The four conditions differ only in the correction explanations supplied to the Reflector and Curator:

- **Original (Full)** retains the controller's original explanations. "Original" refers to their provenance and does not imply that every explanation has been independently verified.
- **Generic** replaces each available explanation with the same statement: "The proposed action required additional verification before proceeding." It retains a generic verification cue without the original diagnostic content.
- **Mismatched** substitutes a diagnostic template selected deterministically from a reason category not detected in the source explanation. Keyword patterns identify source categories, and a hash of the source explanation selects among the remaining templates. The templates contain no task-specific identifiers or answers. This is a category-mismatched treatment, rather than a permutation of explanations between tasks or an independently annotated set of false statements.
- **None** removes correction explanations and attributable copies from learning inputs while preserving intervention indicators and execution feedback.

Controlled configuration. The transformation is applied after ordinary history trimming so that shorter explanations do not expose additional execution history to learning. It changes the learning-visible record, including the history supplied to curation, while leaving the Agent's runtime history and the controller's execution path unchanged. All conditions use MiniMax M2.7 and retain the same action-control, recovery, update-eligibility and admission policies, context limits, task order, initial playbook, and configured budgets. Policies are fixed, but learned guidance, subsequent trajectories, and realized learning-call counts can differ during online execution. Outcomes, actions, and

runtime feedback can still convey reasons indirectly; None therefore does not remove all inferable intervention information.

Evaluation and accounting. Each condition is evaluated over three runs of the complete AppWorld test-normal sequence (168 tasks in 56 scenarios), with the playbook reset to the same initial state for every run. Task completion is the fraction of tasks that pass the evaluator, and scenario completion requires all three tasks of a scenario to pass in the same run. We report means and sample standard deviations across runs; repetition indices do not imply paired random seeds across conditions. All four conditions share one token scope, Agent execution (including recovery) plus Reflector and Curator calls, with input and output reported separately in millions per run (Appendix D). Table 13 reports the aggregate results, and Table 14 gives all twelve runs.

Table 13: Explanation-content ablation on AppWorld with MiniMax M2.7. Entries are mean ± sample SD over three runs per condition, each with 168 tasks and 56 scenarios. All token columns include Agent (with recovery), Reflector, and Curator, in millions per run. Bold marks the highest completion mean or lowest token mean, including ties at the reported precision. Light blue shading marks Original, full DuoGate rerun for this experiment.

Learning explanation	Task (%) ↑	Scenario (%) ↑	Input (M) ↓	Output (M) ↓
Original (Full)	$\mathbf{69.46} \pm 0.88$	$\mathbf{54.76} \pm 3.70$	52.02 ± 1.91	0.82 ± 0.02
Generic	69.06 ± 1.57	54.17 ± 2.72	$\mathbf{47.25} \pm 1.34$	$\mathbf{0.72} \pm 0.02$
None	68.66 ± 0.91	48.81 ± 2.73	47.90 ± 4.04	0.74 ± 0.03
Mismatched	68.05 ± 1.50	48.81 ± 2.73	47.96 ± 2.12	$\mathbf{0.72} \pm 0.05$

Results. Original has the highest mean on both completion metrics. Its task-completion advantage is 0.40 percentage points over Generic, 0.80 over None, and 1.41 over Mismatched; the corresponding scenario-completion differences are 0.59, 5.95, and 5.95 points. Generic remains close to Original on both means. None reaches higher mean task completion than Mismatched (68.66% versus 68.05%), while both have 48.81% mean scenario completion with a sample SD of 2.73 percentage points. These results indicate that correction explanations contribute useful learning information, and that a generic verification cue recovers most of this benefit. As with the other three-run comparisons, the differences are descriptive and do not establish statistical significance.

Original's higher completion means are accompanied by greater recorded token use. Its mean input consumption is 52.02M tokens, compared with 47.25M for Generic, 47.90M for None, and 47.96M for Mismatched; mean output consumption is 0.82M versus 0.72M, 0.74M, and 0.72M, respectively. Since each condition learns online, these usage differences reflect both the altered learning inputs and the resulting execution and learning trajectories, rather than the length of explanation text alone.

C.5.2 Earlier explanation-removal experiment

This experiment compares full DuoGate with *DuoGate no explanation*, which removes correction explanations from Reflector and Curator inputs, including traceable copies in histories and intermediate learning text. Action control, recovery, runtime feedback, and the update and admission policies remain active. Both configurations use three runs of the same 168 tasks with matched models, task order, initial playbook, and configured budgets, and the full DuoGate runs are those of the main comparison (Table 10). Although the policies are fixed, changes in learned guidance can alter subsequent trajectories and update counts.

Completing all variants of a scenario. Correction explanations yield a larger gain in scenario completion. Full DuoGate completes 91 of 168 scenario-runs, compared with 81 for DuoGate no explanation, a gain of 5.95 percentage points. Each completed scenario requires all three task variants to succeed in the same run. Task completion increases from 345 to 349 of 504 task-runs, a gain of 0.79 points. These differences are calculated from evaluator counts before rounding. Table 15 reports the success frequencies of individual tasks and scenarios across the three runs.

Distribution of the scenario gains. Completing a remaining variant can turn partial task success into a completed scenario. In `9dabbc9`, both configurations pass the first two variants in all three runs, whereas the third succeeds in $3/3$ full DuoGate runs and $0/3$ no-explanation runs, adding three scenario completions. Across all 56 scenarios, full DuoGate has higher completion frequency on 16,

Table 14: Individual explanation-content ablation runs on AppWorld with MiniMax M2.7. Each run evaluates 168 tasks and 56 scenarios. Input and output include Agent (with recovery), Reflector, and Curator, in millions per run. Values are shown at the precision of the run records; run numbers identify repetitions within a condition, not paired random seeds. Light blue shading marks Original (full DuoGate).

Learning explanation	Run	Task (%)	Scenario (%)	Input (M)	Output (M)
Original (Full)	R1	69.64	53.57	54.22	0.83
	R2	68.50	51.80	50.77	0.80
	R3	70.24	58.90	51.06	0.82
Generic	R1	67.85	51.80	47.75	0.72
	R2	68.50	53.57	45.73	0.70
	R3	70.83	57.14	48.26	0.73
None	R1	67.85	48.21	45.26	0.74
	R2	68.50	46.43	45.88	0.71
	R3	69.64	51.79	52.55	0.76
Mismatched	R1	66.67	48.21	49.69	0.76
	R2	69.64	51.79	45.59	0.67
	R3	67.85	46.43	48.59	0.74

Table 15: Outcome frequencies in the earlier explanation-removal experiment. Higher, equal, and lower compare full DuoGate's success frequency (0–3) with that of DuoGate no explanation for each task or scenario. Net is the difference in total successful task-runs or scenario-runs.

	Successes (R1, R2, R3)		Full vs. no explanation			
Unit	Full DuoGate	No explanation	Higher	Equal	Lower	Net
Tasks (168/run)	118, 115, 116	117, 112, 116	36	96	36	+4
Scenarios (56/run)	29, 29, 33	30, 22, 29	16	33	7	+10

equal frequency on 33, and lower frequency on seven, for a net gain of ten scenario completions. The largest positive contributions are `9dabbc9` (+3), `042a9fc` (+2), and `634f342` (+2); together they account for seven of the 20 additional completions across improved scenarios (35%), with the other 13 improved scenarios each contributing one.

Retained guidance and subsequent behavior. Representative full DuoGate trajectories also show guidance reuse. In R1, `9dabbc9_2` retains a checklist for rejecting an incorrect payment request, sending the corrected amount, and verifying the recipient, amount, and description. The checklist appears in the next variant's input, whose agent follows the sequence and verifies the corrected transaction; the no-explanation runs of that variant fail the required-description check. In the playlist-archiving scenario, guidance retained after `634f342_1` appears in `634f342_3`, where the agent explicitly cites the entry, scans the source playlists, deduplicates the target songs, and completes the task. Together, these cases illustrate the reuse of retained procedural guidance during successful execution of later task variants.

Tokens use the Agent-only scope available for both configurations. DuoGate no explanation uses 47.93 ± 3.24M input and 0.73 ± 0.03M output tokens per run (48.66 ± 3.27M in total), compared with 47.69M input and 0.71M output Agent tokens (48.40M) for full DuoGate (Table 20). Lower completion is therefore not accompanied by lower mean Agent usage. Including reflection and curation, full DuoGate uses 52.76M tokens per run.

The individual runs of DuoGate no explanation appear as `AW-10-R1–R3` in Figure 14.

C.6 UPDATE SELECTION AND ADMISSION

Purpose. This appendix details the update-selection and admission experiment in Section 4.5. We separate two decisions in context control: selecting experience for a learning update, and screening the guidance produced by that update. A 2×2 ablation crosses selective versus always-on learning with full versus minimal admission checks. This tests whether both decisions contribute to downstream completion under the same action-control policy and context limits, and whether the default combination offers a favorable balance of completion and recorded token usage.

Treatments. *Selective* uses DuoGate's default eligibility policy (Appendix A.5): failures and interactions with recorded execution signals, including controller interventions, execution errors, and high step counts, trigger learning; successes without such signals skip it, and unknown outcomes remain eligible. Periodic updates are disabled. *Always* invokes reflection and curation once at every normal task ending.

Full applies the default admission checks: structural checks, normalized-text and prefix deduplication, keyword-based value screening, exclusion of code examples and selected structured-content patterns, filters for recognized advice that bypasses completion requirements, and size limits. *Minimal* retains the structural requirements (valid, nonempty entries in a permitted section) and the same size limits, while disabling the duplicate and content filters. Both admit at most two new entries per update, with at most 800 characters per entry and 1,600 characters in total. Thus, minimal admission changes screening rather than allowing an unbounded playbook update. Selective + Full is the default DuoGate configuration.

Controlled configuration and evaluation. All four configurations are rerun with MiniMax M2.7 for the Agent, Reflector, Curator, and Confidence assessor. Each configuration has three runs of the full AppWorld test-normal sequence: 168 tasks in 56 scenarios, with the same task order and initial playbook reset for each run. Action control, recovery, original correction explanations, no-GT learning prompts, the 40-step task limit, history trimming, and playbook retrieval settings remain fixed. The configured retrieval target is 60,000 characters and 80 entries; the common critical-guidance prefix is unchanged. The runs learn online, so subsequent trajectories and realized updates may differ. These runs, including the default control, are separate from the main benchmark and explanation-ablation runs.

Task completion uses the benchmark evaluator; scenario completion requires all three variants to succeed in the same run. Tables 3 and 16 report means and sample standard deviations and all individual runs, respectively. Input and output count Agent execution (including recovery), Reflector, and Curator usage in millions per run, excluding Confidence assessor calls, as in Appendix D.1. Cached input and reasoning output are not added again. Run indices identify repetitions within each configuration, not paired random seeds.

The aggregate comparison appears in Section 4.5 and Table 3. Table 16 below preserves all individual runs. Default DuoGate has the highest mean task and scenario completion and the lowest mean total recorded usage; each of its three scenario scores also exceeds every ablated run's score. These comparisons are descriptive and do not establish statistical significance or isolate individual admission filters.

C.7 ERROR-WINDOW SENSITIVITY

The recent-error window determines how long a detected execution error remains flagged for action assessment and completion checks. To examine whether extending this window improves completion enough to justify additional token use, we vary $w \in \{0, 2, 5, 10, 20, 40\}$ for DuoGate on AppWorld with MiniMax M2.7, using three runs per setting. The window counts conversation messages, and a separate six-message error scan remains active at every setting, including $w = 0$ (Appendix A.1).

Extending the error window beyond the default does not improve mean completion in this sweep (Figure 10 and Table 17). At $w = 5$ and $w = 10$, task completion is nearly identical at 69.25% and 69.24%, but scenario completion falls from 53.60% to 47.60%, with recorded usage increasing slightly from 52.62M to 52.81M tokens. Extending the window to 20 or 40 messages further lowers task completion and increases usage, with scenario completion remaining below 48%. Shorter windows reduce usage at the expense of task completion. In particular, $w = 2$ matches the default's 53.60% scenario completion but reaches 68.05% task completion, using 52.34M tokens compared

Table 16: Individual runs of the update-selection and admission ablation. Each run uses 168 AppWorld tasks in 56 scenarios and MiniMax M2.7. Token scope matches Table 3; values retain the reporting precision of the run summaries. Light blue shading marks Selective + Full (default DuoGate).

Selection + Admission	Run	Task (%)	Scenario (%)	Input (M)	Output (M)
Selective + Full	R1	70.83	53.57	53.01	0.82
	R2	68.50	53.57	54.77	0.83
	R3	68.45	55.36	48.02	0.80
Always + Full	R1	68.45	51.80	54.86	0.85
	R2	68.50	51.80	52.18	0.79
	R3	67.26	48.21	51.99	0.77
Selective + Minimal	R1	69.05	48.21	54.41	0.84
	R2	68.50	50.00	51.39	0.78
	R3	66.67	44.64	53.18	0.81
Always + Minimal	R1	66.07	39.29	54.19	0.85
	R2	67.26	44.64	51.92	0.78
	R3	68.45	48.21	55.18	0.88

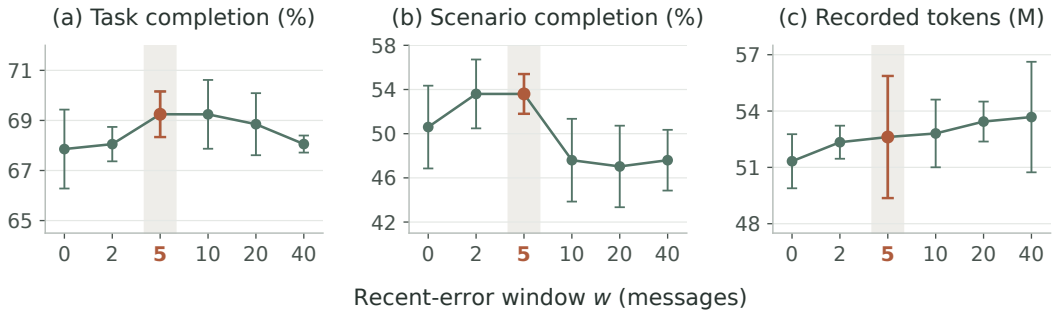


Figure 10: Recent-error-window sensitivity on AppWorld with MiniMax M2.7. Markers and error bars show mean ± sample standard deviation over three runs; shading marks the default $w = 5$. Windows count messages and are displayed at equal spacing. Task completion is nearly unchanged between $w = 5$ and $w = 10$, but scenario completion is lower at $w = 10$. Recorded tokens sum input and output per run.

Table 17: Recent-error-window sweep for DuoGate on AppWorld with MiniMax M2.7. Light blue shading marks the default $w = 5$. Values are mean ± sample standard deviation over three runs; recorded tokens sum input and output within each run, in millions. This sweep uses separate runs from the main comparison in Table 10.

Window w	Task (%) ↑	Scenario (%) ↑	Recorded tokens (M) ↓
0	67.86 ± 1.58	50.60 ± 3.75	51.33 ± 1.44
2	68.05 ± 0.69	53.60 ± 3.12	52.34 ± 0.88
5	69.25 ± 0.91	53.60 ± 1.80	52.62 ± 3.25
10	69.24 ± 1.37	47.60 ± 3.75	52.81 ± 1.80
20	68.85 ± 1.24	47.03 ± 3.69	53.44 ± 1.06
40	68.06 ± 0.34	47.60 ± 2.75	53.68 ± 2.94

with 52.62M at $w = 5$. We therefore retain $w = 5$ as the default, which achieves the highest mean task completion and ties for the highest mean scenario completion, using fewer recorded tokens than the longer windows.

D RESOURCE CONSUMPTION AND TOKEN ACCOUNTING

This appendix details the resource comparisons in Section 4.6. It defines the counting scope and reports component usage, examines context length and learning frequency, and then specifies how usage is counted for each interface.

D.1 COUNTING UNIT AND SCOPE

For each run, we sum recorded input and output usage for distinct model calls, and then average the run totals. Table 18 compares ACE and DuoGate across the four AppWorld base LLMs. Tables 19 and 20 separate usage by component for MiniMax M3 and MiniMax M2.7. Totals are computed before rounding. These are logical token counts, not provider bills or cache-adjusted monetary costs. Assessor-call tokens are not included in the reported token totals.

Relative to the base agent, DuoGate uses 2.4–3.2× the recorded tokens on AppWorld across all four base LLMs. ACE uses 2.0× with MiniMax M2.7, 3.2× with DeepSeek V4 Pro, 3.7× with GLM-5.3, and 9.3× with MiniMax M3 (Table 10). Across the four base LLMs, 29–46% of task-runs skip learning and 20–29% of filtered updates produce empty output (Appendix E.5).

The explanation experiments use their own token scopes (Table 12); apart from the main full DuoGate runs, their runs are not included in the component measurements below. The separately rerun update-selection and admission ablation uses the common Agent (including recovery), Reflector, and Curator scope (Appendix C.6); its runs are also excluded from the component measurements below.

Table 18: Recorded AppWorld tokens per run, in millions. Each entry shows ACE → DuoGate means over three runs; full results and sample SDs are in Table 10.

Base LLM	Input tokens ↓	Output tokens ↓
MiniMax M2.7	42.62 → 51.95	0.69 → 0.82
GLM-5.3	84.76 → 62.81	1.55 → 1.72
DeepSeek V4 Pro	82.00 → 62.44	0.64 → 1.52
MiniMax M3	198.72 → 68.24	1.69 → 1.27

Table 19: Recorded component token usage on AppWorld with MiniMax M3. Entries are means over three runs, in millions of input or output tokens per run. Agent usage includes recovery.

Method	Component	Input	Output
ACE	Agent execution, including recovery	157.444	0.346
	Reflection	20.292	0.739
	Curation	20.987	0.600
	Recorded sum	198.722	1.685
DuoGate	Agent execution, including recovery	61.034	0.493
	Reflection	3.385	0.441
	Curation	3.818	0.340
	Recorded sum	68.237	1.274

Table 20: Recorded component token usage on AppWorld with MiniMax M2.7. Entries are means over three runs, in millions of input or output tokens per run. Agent usage includes recovery.

	ACE		DuoGate	
Component	Input	Output	Input	Output
Agent execution, including recovery	37.480	0.534	47.690	0.710
Reflection	1.692	0.059	1.449	0.042
Curation	3.445	0.102	2.808	0.064
Recorded sum	42.617	0.695	51.947	0.816

D.2 CONTEXT LENGTH AND LEARNING FREQUENCY

For MiniMax M3, the per-run median playbook lengths in initial Agent prompts are approximately 216k, 163k, and 256k characters for ACE and 36k, 33k, and 35k for DuoGate. The respective

maxima across these runs are approximately 578k and 60.8k characters. These character counts measure the supplied guidance and are not substituted for token usage. Across the same three runs, Agent calls increase from 6,621 to 7,576, while Reflector–Curator pairs decrease from 498 to 270. Shorter supplied playbooks and fewer learning updates thus accompany the recorded token reduction despite more Agent calls. The component measurements locate the change in expenditure, but do not separate the causal effects of context length, intervention, or individual admission checks.

D.3 Accounting conventions by interface

AppWorld. Duplicate copies of a request record, identified by request identifier, are counted once per run. Independently executed calls, including separately recorded retries and unsuccessful calls, are each counted once. Cached input tokens are already part of the reported input usage, and reasoning tokens are already part of the output usage, so neither is added again. Recovery is included in Agent usage. For the DeepSeek V4 Pro AgentTrust and AgentLTL baselines, we use the per-run input and output totals reported by those runs.

Terminal-Bench 2. Curation runs within the update step and records no separate model tokens. Agent usage is taken from each trial's final metrics, which also cover context compression and other model calls beyond the visible execution steps. Reflector usage is added once, including failed attempts, even when a curation event repeats it; 4,195 Reflector calls are therefore counted, whereas the learning-decision records contain 4,184 updates. The resulting totals match the per-trial aggregate usage. Step-level usage identifies subsets without being added again. For example, the Agent iterations immediately following an intervention (1,895 iterations across the three DuoGate runs) use a mean of 16.33M input and 0.48M output tokens per run, which are already included in Agent usage.

τ^2-Bench. Input and output tokens come from assistant-message usage and from the Banking run summaries. Domain totals are summed within each run before computing the three-run mean and sample standard deviation in Table 1.

E Case studies and observed action–learning paths

We use archived execution, learning, and model-request records to examine the information flow between action control and learning. A task-run denotes one task evaluated in one run; counts pooled across three runs do not denote distinct tasks. We distinguish the proposed action, the executed recovery, the final evaluator outcome, the learning decision, the logged filtering output, and any directly observed inclusion in a later prompt. The presence of one stage does not imply that subsequent stages occurred.

E.1 Contrasting action and context control in archived trajectories

Figure 11 presents two task-level comparisons using concrete messages, code, controller records, and evaluator outcomes. The Telecom example contrasts a missing tool call under DuoGate-Context with a recorded action correction under DuoGate. The AppWorld example contrasts a wrong temporal premise accepted by DuoGate-Action with a DuoGate trajectory that explicitly consults retained guidance and checks that premise. Each comparison uses the same task and base LLM (and, for Telecom, the same seed) but comes from separate runs with different execution histories. We use these cases to examine observable behavior, not to estimate a causal effect from the selected successes and failures.

Action control: from a transfer announcement to an executed call. Figure 11(a) concerns a Telecom task with MiniMax M2.7 and the Hard persona. After earlier troubleshooting, the SIM remains PIN-locked and the required action is escalation to a human agent. In DuoGate-Context rep3, the assistant recognizes that it cannot unlock the SIM and announces a transfer, but no `transfer_to_human_agents` call occurs in the trajectory. The user's subsequent `###TRANSFER###` marker ends the dialogue without satisfying the required action. The evaluator assigns an action-check value of zero and final reward zero.

In DuoGate rep3, gate step 15 records a text-only transfer proposal with no tool call and flags `intent_action_consistency=fail`. Its explanation explicitly refers to the playbook requirement to call the transfer tool. The controller generates a corrected mutation; the executed

assistant message contains `transfer_to_human_agents(summary=...)`, followed by the tool response "Transfer successful." The action check and final reward are both one. This establishes the correction and executed transfer within that DuoGate trajectory. Success here means the required escalation, not restored cellular service; the task's expected environment state still includes `no_service`. The figure excerpts the final transfer segment, not the earlier interventions.

Context control: checking the premise of a count. Figure 11(b) uses AppWorld task `dac78d9_2` with MiniMax M3, which asks how many Venmo friendships began since the start of October last year. DuoGate-Action rep1 retrieves 15 friends and uses the appropriate `friends_since` field, but sets the lower bound to `datetime(2024, 10, 1)`. The resulting count is zero. Gate step 10 accepts `complete_task(answer=0)` with high confidence, reasoning that the observed friendship dates precede that cutoff. The formal evaluator marks the answer incorrect and expects eight. Thus, agreement between the calculation and a chosen cutoff does not establish that the cutoff correctly interprets "last year."

DuoGate rep1 begins with retained time-anchoring and API guidance in its actual initial Agent request. Its first response explicitly references `cms-00080` for cross-checking the date and `api-00093` for filtering on `friends_since`. The trajectory examines timestamped Phone, Gmail, and Venmo records; the displayed Gmail and Venmo observations are from May 2023. It infers the current year as 2023, uses `datetime(2022, 10, 1)`, counts eight friends, and submits eight. The formal evaluator reports success. No code is replaced by the action controller in this target task. These are checks of timestamped records, not a direct read of an authoritative environment clock.

From an earlier learning update to explicit reference and use. The guidance shown in Figure 11(b) can be traced to the earlier task `b6d1104_2` in the same DuoGate archive. That task succeeds but records `high_risk_success_or_unknown` as its learning reason, with API/runtime-error and recovery signals. No controller code replacement is recorded for this task, so this learning event is not attributed to a code-replacement intervention. The Curator proposes cross-checking an inferred time boundary against another timestamped source. A retained form appears as `[cms-00080]` in the next task's first Agent request and in the initial request for `dac78d9_2`; it is absent from the source task's initial request. Additional guidance in the target request addresses annual boundaries and the distinction between `friends_since` and `registered_at`.

The target Agent's explicit reference and subsequent timestamp checks provide evidence beyond prompt inclusion alone. They do not identify the marginal benefit of one entry, of filtering, or of the context branch relative to an otherwise identical execution. Retention also does not certify every instruction: the full time-anchoring entry names an unavailable Gmail `list_threads` endpoint, which produces an error before the agent discovers supported endpoints and continues. The figure shows the cross-checking clause and the later relevant observations; it does not imply an error-free trajectory.

(a) Action control: executing a transfer that was only announced

τ^2-Bench Telecom / MiniMax M2.7. SIM PIN lock, Hard persona: the required action is escalation to a human agent.

DuoGate-Context

Observation. "The SIM card is locked with a PIN code."

Assistant. "I'm going to transfer you to a human agent who can help you with this SIM PIN issue."

Execution trace: no `transfer_to_human_agents` call.

Dialogue ending. The user emits `###TRANSFER###`.

FAIL: action check = 0; reward = 0.

DuoGate

Gate 15 proposal (summary). A text-only transfer announcement.

Gate reason (excerpt). "announces transfer without calling `transfer_to_human_agents`"

Executed call.

```
transfer_to_human_agents(summary=...)
```

Tool result. "Transfer successful"

PASS: action check = 1; reward = 1.

(b) Context control: checking the year before committing to a count

AppWorld / MiniMax M3 / `dac78d9_2`. "How many venmo friends have I made since the start of October last year?"

DuoGate-Action

Agent 8: date cutoff.

```
cutoff_date = datetime(2024, 10, 1)
```

Observation (excerpt). "Number of friends since start of October last year: 0"

Gate 10.

```
confidence:  high
decision:  execute_original
```

Executed completion.

```
apis.supervisor.complete_task(
  answer=0)
```

Gate reason (excerpt). "before the cutoff of October 1, 2024, and correctly computed 0."

FAIL: answer 0; evaluator expects 8.

DuoGate

Retained guidance: [cms-00080] (excerpt).

"cross-check the inferred period boundary with at least one OTHER timestamped source"

Agent 1 (excerpt). "determining 'today' via cross-check per cms-00080"

Observed timestamps (abridged).

Gmail: `2023-05-18`; Venmo: `2023-05-14`.

Agent 9: date cutoff.

```
october_last_year_start = datetime(
  2022, 10, 1)
```

Executed completion.

```
apis.supervisor.complete_task(
  answer=8)
```

PASS: answer 8; no code replacement.

Guidance provenance in (b). Earlier task `b6d1104_2`: recorded execution signals trigger learning → guidance appears as `[cms-00080]` in subsequent prompts → the target Agent explicitly references it and checks timestamps.

Figure 11: Action and context control in two archived task comparisons. (a) A transfer announcement without a tool call, and a logged correction that executes the transfer. (b) A wrong date premise accepted by action control, and explicit use of retained guidance to check the year. Quotes and code are abridged and typeset for readability; unquoted labels are summaries. The exact DuoGate proposal wording in (a) is not reconstructed. These selected archived runs are not same-state counterfactual replays.

E.2 LEARNING FROM A RECOVERED BULK IMPORT

Figure 12 traces action recovery and the resulting context update in AppWorld task `0d01c76_3` (DuoGate, MiniMax M3, run 1). The task requires one Simple Note per Markdown source file, preserving content and deriving titles from filenames. At step 8, the controller replaces a loop that interleaves reads and writes with a read-only preflight of all 26 sources. The next block creates notes from the prepared records; all 26 creations succeed and all six evaluator checks pass.

Despite the successful outcome, `confidence_code_replacement` triggers learning. The actual Reflector request includes the intervention explanation. A verification checklist is subsequently retained as `[vc-00033]`, including clauses to match the returned-item count and derived title set to the sources. This entry is absent from the source task's initial request and present in the next task's initial request. The case shows how a successful recovery supplies concrete material for retained guidance; the later request documents its availability for reuse.

(a) Action control: preflight the complete batch before writing

AppWorld / MiniMax M3 / `0d01c76_3`. Import 26 Markdown files as separate Simple Notes.

Proposed block and gate decision

Proposal (abridged).

```
for file_path in files:
  ...show_file(...)
  ...create_note(...)
```

Step 8. Reads and note creation are interleaved in one loop.

Gate reason (excerpt). "Split it so all source files are preflighted first, then execute the mutations separately."

Decision. Replace with read-only preparation.

Executed recovery and result

Replacement. Read every source, check that its title is nonempty and content is available, and populate `verified_markdown`.

Next executed block (abridged).

```
for item in verified_markdown:
  ...create_note(...)
```

Observed returns. 26 successful creations; note IDs 3084–3109.

PASS: all 6 evaluator checks.

(b) Context control: learn from a successful recovery

The final success is accompanied by a recorded intervention and a subsequent retained checklist.

Learning decision and evidence

Task outcome. Success.

Learning decision.

```
run_learning_update:  true
```

Recorded execution signal.
`confidence_code_replacement`

Reflector input. Contains the correction reason for separating source inspection from note creation.

Learning source. An executed recovery with a verified outcome.

Retained verification guidance

[vc-00033], count and title clauses (excerpt).

"confirm `len(returned_target_ids) == len(source_files)` exactly"
"confirm the derived title set equals the expected title set"

Observed persistence. The complete entry appears in the initial Agent request for the next task, `ff58e36_1`.

Retained and available in a later prompt.

Figure 12: From batch-action recovery to retained verification guidance. The controller separates source preflight from 26 note creations; the successful task still enters learning because its code was replaced. The retained checklist is then observed in a later Agent request. Code and quotes are abridged; only the count and title clauses of the longer checklist are shown.

E.3 Reusing failure-derived guidance during regex repair

Figure 13 shows an action–learning path within Terminal-Bench 2 task `regex-log` (DuoGate, MiniMax M2.7, run 1). The task asks for the last valid date on each line containing a valid IPv4 address. The initial expression instead captures the first date in two of the agent's 21 local tests. At logged episode 1, the controller records a recovery intervention, while the learning trace records reflection on the failure and admission of six guidance items. One retained item recommends a negative lookahead after the date group.

At the next repair step, the agent explicitly states that this fix is "directly suggested by the ACE Context Playbook." It uses greedy matching and a negative lookahead for subsequent valid dates, then reruns the same local suite: all 21 tests pass. The separate benchmark verifier also passes and assigns reward 1. This case connects a concrete failure, retained guidance, an explicit reference to that guidance, and the corresponding executed repair within one trajectory.

(a) Failure feedback activates recovery and learning

Terminal-Bench 2 / MiniMax M2.7 / `regex-log`. Match only the last valid date on a line containing a valid IPv4 address.

Observed error and recovery signal

One failing local input.

```
192.168.1.1
2023-02-28 2023-03-15
```

Displayed on two lines for readability.

Expected: `2023-03-15`
Returned: `2023-02-28`

Local suite. 19 pass; 2 fail.

Controller record. `intervention: recovery`; reason: "recent failure signal."

Retained diagnosis and strategy

Learning record. Reflection runs; six items are admitted.

Failure-pattern entry (excerpt). "Regex matches first date instead of last date"

Strategy entry (excerpt). "To match only the LAST date, add a negative lookahead after the date group"

Retention. The strategy is present in the saved playbook with creation episode 1.

Guidance is tied to the observed mismatch.

(b) The next repair explicitly references the retained guidance

The same local tests are rerun after the expression is revised; the benchmark verifier evaluates the final artifact separately.

Explicit reference and executed change

Agent's recorded reason (excerpt).

"the fix (adding negative lookahead for subsequent dates) is directly suggested by the ACE Context Playbook."

Executed repair (summary). Replace the lazy prefix with greedy matching and add a negative lookahead that rejects a captured date when another valid date follows.

Next action. Rewrite `regex.txt` and rerun the local suite.

Verified continuation and final outcome

Same local suite after repair.

"Results: 21 passed, 0 failed"

Previously failing example. The returned date is now `2023-03-15`.

Separate benchmark verification.

`test_regex_matches_dates`: passed.

PASS: formal reward = 1.

Figure 13: Failure-derived guidance used in a subsequent repair. The recovery record, admitted strategy, explicit playbook reference, executed edit, and successful verification occur within the same task. The 21 local checks are agent-authored tests; the formal benchmark verifier separately records one passed test and reward 1. Quotes are excerpted and unquoted labels summarize the logs.

E.4 Illustrative action recovery and supported continuation

Table 21 presents positive illustrative cases from the three full DuoGate runs on AppWorld with MiniMax M2.7. For each case, we inspect the task requirement, evidence available before the decision, and the subsequent executed action. These trajectories show both recovery before commitment and continuation once the relevant evidence is available.

Table 21: Illustrative DuoGate action decisions. The first two rows trace an intervention and subsequent recovery; the last two trace allowed actions. Run IDs, task IDs, and environment steps identify the underlying records.

Decision and record	Task requirement and proposal evidence	Observed continuation
Intervene: album playback R2, `0de03ea_3` Steps 8–12	The task requires 20 minutes of downloaded music; the original proposal plays album 2.	After the block, the agent checks downloaded durations: 8 minutes for album 2 and 20 for album 4. It verifies album 4 and executes playback of that album.
Intervene: private requests R1, `3b8fb7a_3` Steps 8–10	The task requires private payment requests, but the proposed calls omit the privacy parameter.	Recovery requests verification of payment fields. The agent reads the API specification and successfully creates a request with `private=True`.
Allow: corrected payment R1, `9dabbc9_1` Steps 8–9	The recipient and original $500 request are established. The specified 5% increase gives a corrected payment of $525.	The gate allows the payment. The API reports success, and a subsequent query confirms the recipient, amount, and requested note.
Allow: verified completion R3, `8749218_1` Steps 9–13	All 23 recommended songs have been added to the reset queue. Shuffle, playback, and the final queue count are observed before completion.	The gate allows `complete_task()` after the recorded operations and final read-only check.

The observed recovery sequences include concrete checks of execution targets and task-required parameters, followed by corrected actions. The allowed cases show grounded payments and verified completion proceeding through the same action-control framework.

E.5 Update eligibility and admission in recorded trajectories

Case study: when reflection is skipped. Two DuoGate episodes with MiniMax M2.7 illustrate how update eligibility responds to the evidence available at task end.

A completed, low-risk interaction. In AppWorld run 2, task `afc4005_3` asks for the Sunday workout duration stored in Simple Note. The agent retrieves the "Weekly Exercise Plan", reads the Sunday entry, and completes the task with the answer 50 minutes. All five execution steps finish, and both evaluator checks pass: the answer is correct and application data remain unchanged. The learning record explicitly sets `run_learning_update=false` with reason `clean_low_risk_success`, an empty list of high-risk signals, and zero API errors, Python errors, or code replacements. The task's model-call log contains five Agent calls and no Reflector or Curator call. This completed interaction therefore skips learning under the eligibility policy because it supplies no recorded failure or execution signal that warrants an update.

An incomplete interaction without execution feedback. In τ^2-Bench Retail run 2, task 107 terminates with `timeout` after only the assistant's greeting and the user's opening description, before any tool call or response. The learning summary records `skipped_timeout`, with both `reflected=false` and `curated=false`, while online adaptation is enabled. The eligibility gate thus excludes this incomplete trajectory before reflection and candidate generation.

Together, these cases show selective reflection in two complementary settings: a routine successful interaction without a qualifying learning signal, and an interrupted interaction without usable execution feedback.

AppWorld with MiniMax M3. The AppWorld records show that selecting an episode for learning and admitting the resulting guidance are distinct decisions. With MiniMax M3, 219 of 435 successful task-runs are selected for learning on recorded execution signals, and 216 skip learning. Failed task-runs contribute 51 further updates, giving 270 recorded updates overall. Of the 256 updates with filtering records, 181 have nonempty output and 75 have empty output. A selected learning update therefore need not produce new guidance, just as successful execution alone does not determine update eligibility (Table 22).

Code-replacement events give a more specific view of this sequence. Across 504 task-runs, 250 of 7,572 recorded gate decisions replace code, affecting 87 task-runs, of which 59 ultimately succeed and 28 fail. All 59 successful task-runs with code replacement are selected for learning, so a successful outcome does not prevent learning after an action correction; among the 58 with filtering records, 46 have nonempty output and 12 have empty output. Failed task-runs with code replacement contribute 13 further updates, of which 11 have nonempty output and two have empty output. Even after an action correction and a selected update, the recorded filtering output can therefore be empty. These outcome counts describe the observed paths rather than a causal recovery rate.

Other AppWorld base LLMs. The same separation appears with the other AppWorld base LLMs (Table 22). MiniMax M2.7, GLM-5.3, and DeepSeek V4 Pro have 341, 259, and 287 updates, respectively, alongside 145, 230, and 192 explicit skips. Their available filtering records include 46, 52, and 66 empty outputs. Action-gate coverage differs: MiniMax M2.7 records code replacement in 195 of 504 task-runs, ending in 114 successes and 81 failures; DeepSeek has gate traces for all 504 task-runs and records 170 replacement events. The GLM-5.3 archive lacks the corresponding action-gate traces. We therefore compare learning-stage counts across base LLMs and report action replacements only where those records are available.

Table 22: Recorded learning decisions for DuoGate on AppWorld, with 504 task-runs per base LLM. Updates and Skips count task-runs with a recorded learning decision. Filtering coverage is the number of available filtering records divided by the number of updates. Nonempty and Zero count records with nonempty or empty filtering output; they do not count persistent guidance entries.

Base LLM	Updates	Skips	Filtering coverage	Nonempty	Zero
MiniMax M2.7	341	145	168/341	122	46
MiniMax M3	270	216	256/270	181	75
GLM-5.3	259	230	256/259	204	52
DeepSeek V4 Pro	287	192	286/287	220	66

Terminal-Bench 2. In Terminal-Bench 2, interventions and learning updates also follow different selection rules. Joining records by trial and iteration matches 1,921 of 1,937 interventions to a learning decision: 1,292 coincide with updates and 629 with explicit skips. Of the 1,292 updates, 1,280 have nonempty retained output and 12 have empty output. The full archive contains controller and learning traces for 266 of 267 task-runs, with 4,184 updates and 3,195 explicit skips among 7,379 learning decisions. These are iteration-level counts because guidance is updated and reused within each task.

Completion checks intervene in 283 of 643 completion requests, involving 157 task-runs, of which 110 end with reward 1 and 47 with reward 0. Across all intervention types, 260 of 267 task-runs contain at least one intervention, ending in 129 successes, 128 failures, and three missing rewards. These final outcomes do not identify the effect of a particular block. Playbooks reset between tasks, and the archive does not preserve every full model request, so these counts describe local update and retention paths without establishing later prompt inclusion for a specific entry.

τ^2-Bench. The dialogue records show how update eligibility and admission respond to different domain outcomes. Across three runs, Airline, Retail, Telecom, and Banking comprise 1,125 task-runs. Banking selects 255 of its 291 task-runs for reflection and curation; 91 of these updates admit new entries, totaling 120 entries, and the other 164 add no entry. Airline selects 28 of its 150 task-runs, and every selected update admits new entries. All three Airline runs enable online playbook updates between tasks.

Retail and Telecom, where success rates are higher, select fewer task-runs for learning. Of 342 Retail task-runs, 58 undergo reflection and curation and 16 are excluded after a timeout; 51 updates admit new entries and seven add none. Of 342 Telecom task-runs, eight undergo reflection and curation and six are excluded after a timeout; five updates admit new entries and three add none. Together with Banking, these records distinguish the frequency of learning from the frequency of admitted additions. They report applied playbook updates rather than a complete intervention-to-prompt trajectory.

F PER-RUN EXPERIMENTAL RESULTS

Figures 14–19 report the individual runs underlying the main benchmark comparisons, the earlier explanation-removal experiment, and the error-window sweep; the four-condition explanation experiment is reported in Table 14. Each run ID combines a benchmark prefix (`AW`, `T2`, or `TB`), the configuration number shown in brackets beside each method name, and a repetition index (`R1`–`R3`); repetition indices are not paired across methods. Dialogue IDs add `A`, `R`, `T`, or `B` for Airline, Retail, Telecom, or Banking, and `AW-W` identifies window-sweep settings. The AppWorld plot abbreviates DuoGate-Action and DuoGate-Context as DG-Action and DG-Context; the dialogue and terminal plots use the full method names.

Every reported run evaluates its full configured task set, so $N_{\text{evaluate}} = N_{\text{set}}$, with set sizes given in the captions. Token totals sum input and output within each repetition before computing across-run means and sample standard deviations. Token accounting follows Appendix D.

F.1 APPWORLD REPETITIONS

Figure 14 covers all four base LLMs in Table 10, including three runs each for GLM-5.3 DuoGate-Context (configuration 26) and the DeepSeek V4 Pro AgentTrust and AgentLTL baselines (configurations 27 and 28). Completion and recorded tokens appear in separate panels with independent horizontal axes and repeated method labels. DuoGate no explanation (DG w/o expl.†, configuration 10) from the earlier explanation-removal experiment reports Agent-only tokens, the scope of that experiment (Table 12). The window sweep is plotted at the precision of Table 17.

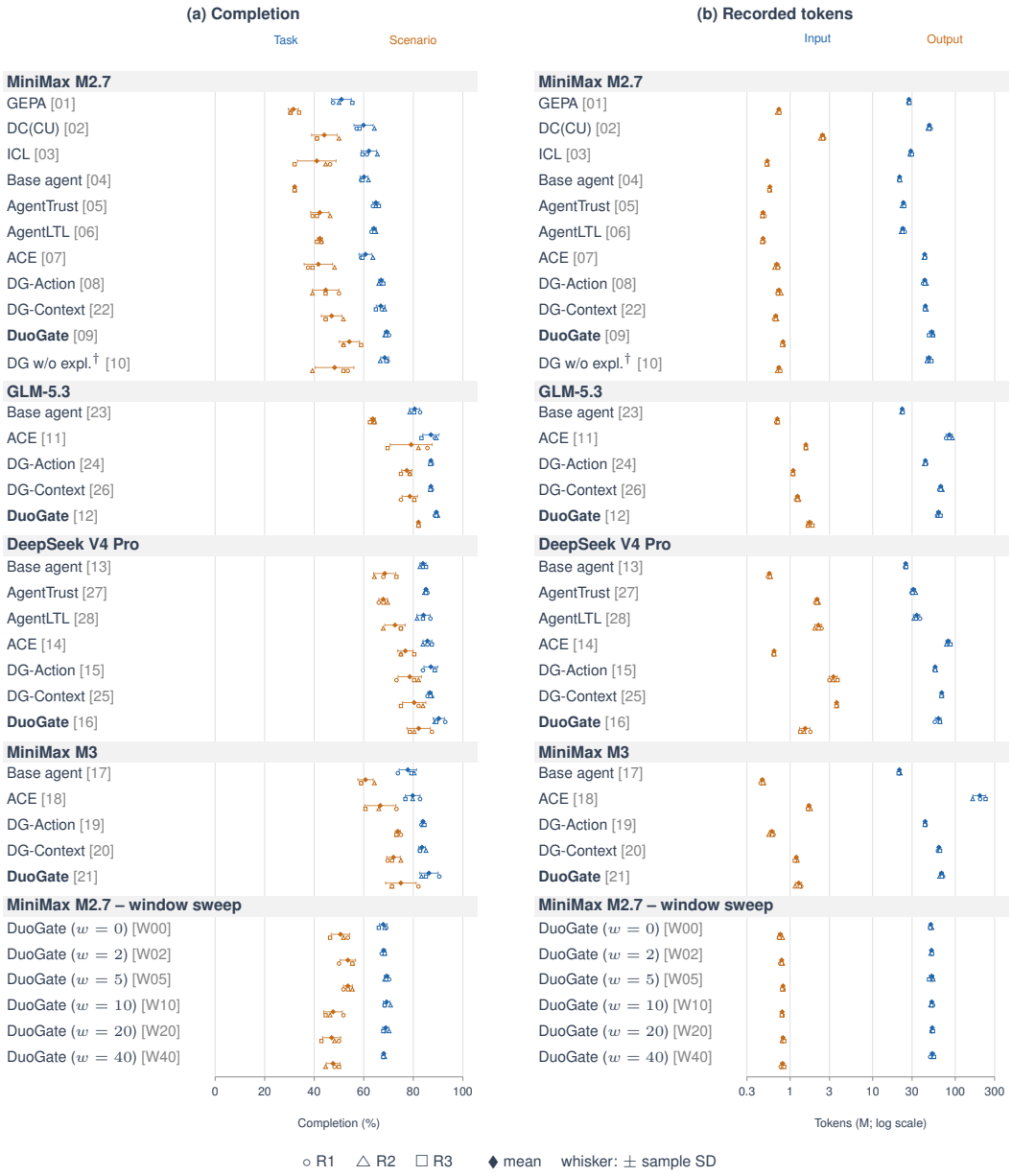


Figure 14: AppWorld repetitions ($N_{\text{set}} = 168$). Figure 14(a) shows task and scenario completion on a linear percentage axis; Figure 14(b) shows recorded input and output tokens on a separate logarithmic axis in millions. Both panels retain the same method order and all three repetitions, including the DeepSeek V4 Pro AgentTrust and AgentLTL baselines. Small circles, triangles, and squares denote R1, R2, and R3; diamonds and whiskers show the reported mean and sample SD. Configuration suffixes give the run IDs: [01] denotes `AW-01-R1-R3`, [26] denotes GLM-5.3 DuoGate-Context, [27] and [28] denote DeepSeek V4 Pro AgentTrust and AgentLTL, and [W05] denotes `AW-W05-R1-R3`. [10] denotes DuoGate no explanation from the earlier explanation-removal experiment; † marks its Agent-only accounting. The four-condition experiment, including None, appears in Table 14. The window sweep remains separate from the main comparisons.

F.2 τ^2-BENCH REPETITIONS

Figures 15–18 show the four domains with MiniMax M2.7 in a common method order. They retain reported success means and standard deviations alongside the individual repetitions, and provide domain-level token measurements that complement the aggregate comparison in Table 1.

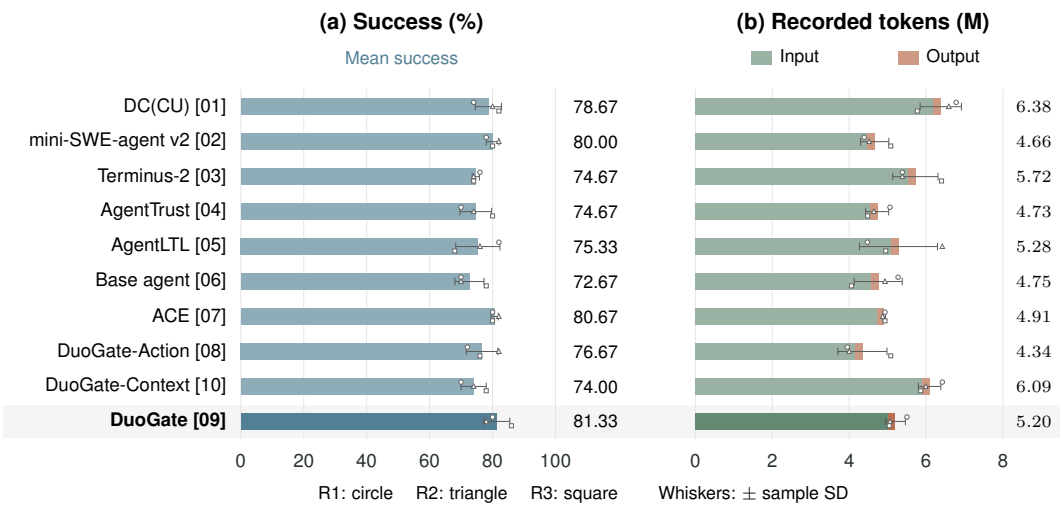


Figure 15: τ^2-Bench: **Airline** with MiniMax M2.7 ($N_{\text{set}} = 50$). Left: success over three runs. Right: mean input and output tokens, with markers and whiskers for their per-run sum. Numeric labels give the means. Configuration suffixes recover IDs such as `T2-09-A-R1-R3` for DuoGate.

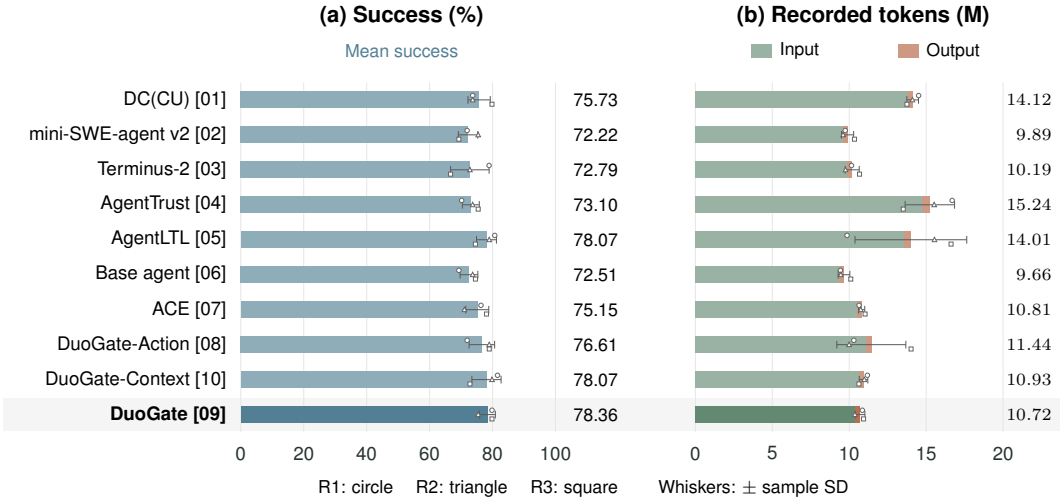


Figure 16: τ^2-Bench: **Retail** with MiniMax M2.7 ($N_{\text{set}} = 114$). Left: success over three runs. Right: mean input and output tokens, with markers and whiskers for their per-run sum. Numeric labels give the means. Configuration suffixes recover IDs such as `T2-09-R-R1-R3` for DuoGate.

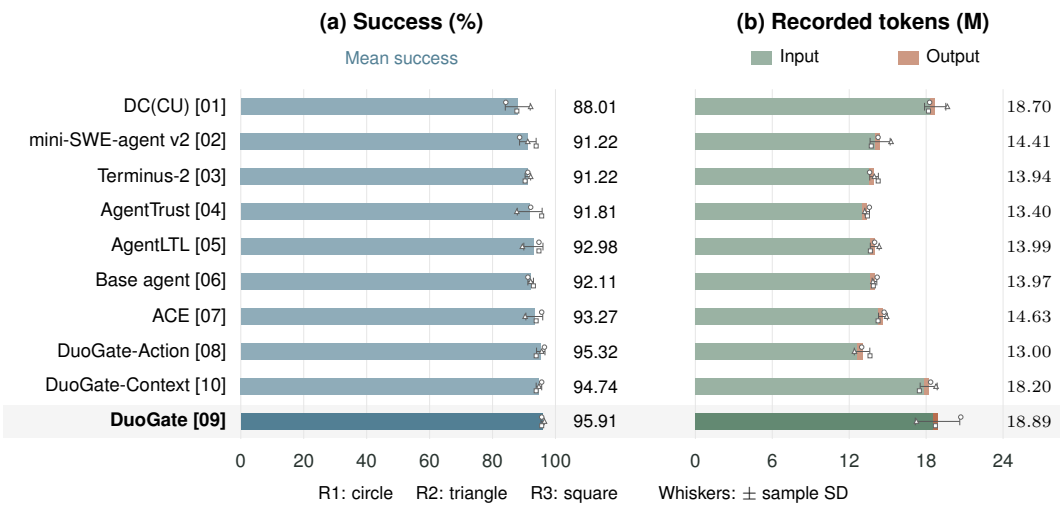


Figure 17: τ^2-Bench: **Telecom** with MiniMax M2.7 ($N_{\text{set}} = 114$). Left: success over three runs. Right: mean input and output tokens, with markers and whiskers for their per-run sum. Numeric labels give the means. Configuration suffixes recover IDs such as `T2-09-T-R1-R3` for DuoGate.

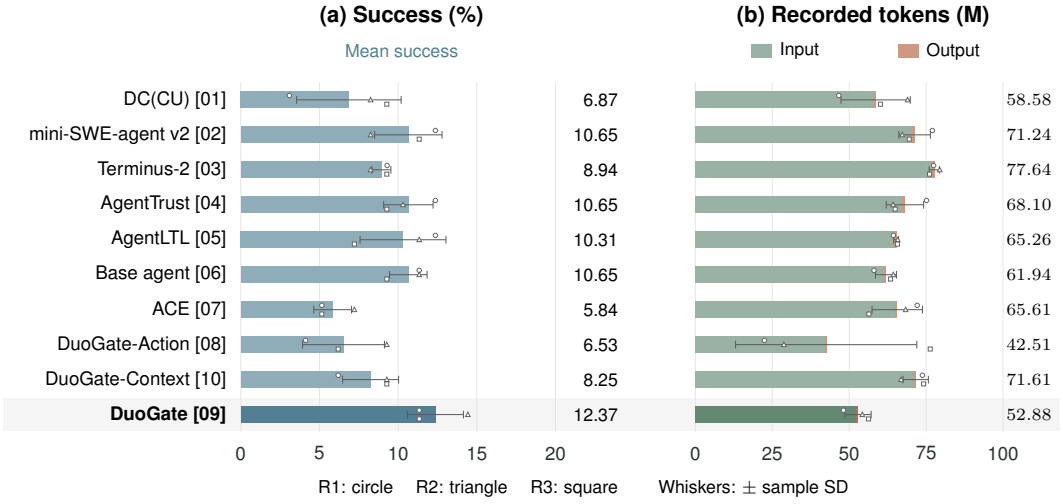


Figure 18: τ^2-Bench: **Banking** with MiniMax M2.7 ($N_{\text{set}} = 97$). Left: success over three runs. Right: mean input and output tokens, with markers and whiskers for their per-run sum. Numeric labels give the means. Configuration suffixes recover IDs such as `T2-09-B-R1-R3` for DuoGate.

F.3 TERMINAL-BENCH 2 REPETITIONS

Figure 19 shows all three repetitions for the nine MiniMax M2.7 configurations in Table 2, where the corresponding means and sample standard deviations are reported. Terminus-2 is the base agent.

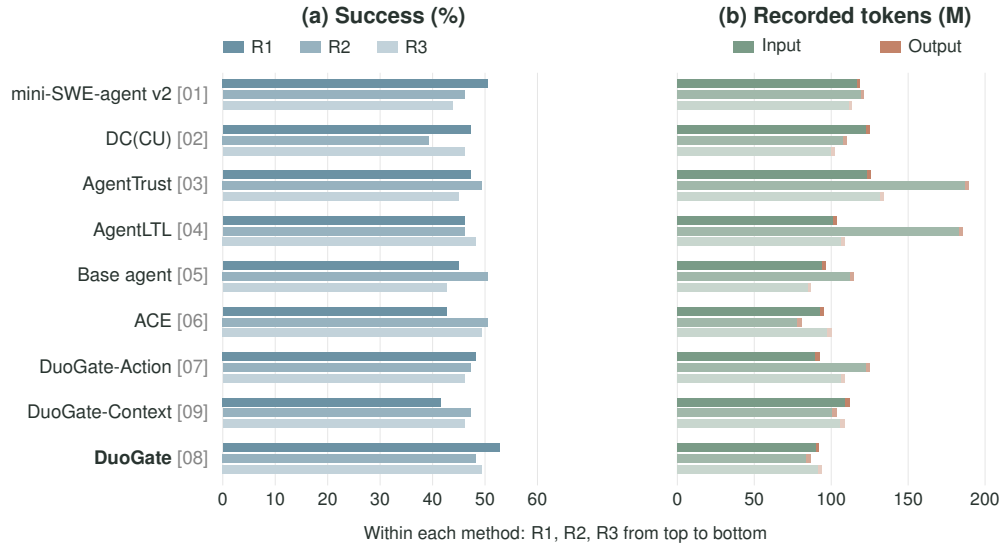


Figure 19: Terminal-Bench 2 repetitions with MiniMax M2.7 ($N_{\text{set}} = 89$). Each method has three horizontal bars showing individual-run success (left) and input-plus-output tokens (right). Both axes begin at zero. Input tokens dominate usage, so output segments are narrow on the shared linear token scale. Configuration suffixes preserve IDs such as `TB-08-R1-R3` for DuoGate. The Base agent is Terminus-2.

G LIMITATIONS

Our evaluation leaves two limitations for future work. First, DuoGate operates on predefined code, dialogue, and terminal interfaces, where each controller draws on interface-specific operations and execution records. Evaluating transfer to unseen tools and changing API schemas would clarify how broadly the action–context control framework applies and motivate more reusable evidence-extraction mechanisms. Second, our experiments study adaptation over fixed benchmark sequences, with playbooks reset between runs or individual terminal tasks. Longer evaluations under changing task distributions are needed to understand how the usefulness of retained guidance evolves as experience accumulates. Future work could use subsequent execution feedback to revise, consolidate, or retire existing guidance, and assess whether this process sustains completion while controlling context growth and computational cost.