# SYNCHRONIZING WORK WITH RESULTS IN LONG-HORIZON COMPUTER-USE AGENTS VIA DEMAND-LINKED EXECUTION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Long-horizon computer-use agents (CUAs) carry out work across applications, using findings and intermediate conclusions to make decisions and update artifacts. These results must remain connected to the work that needs them and to the earlier decisions that rely on them as execution moves on. Yet work can remain unfinished after a needed result arrives, and earlier decisions can go unreviewed when that result changes. We identify this phenomenon as ***Result–Work Desynchronization*** (***RWD***) and analyze its associated decision costs. In this paper, we propose ***Demand-Linked Execution*** (***DLE***) to mitigate the RWD phenomenon by preserving result uses together with each work's evolving state. DLE records why results are requested and which decisions and artifacts rely on each adopted version. When results arrive or change, the agent's model uses these records and current software observations to resume waiting work and determine which earlier outcomes to retain or revise. On representative long-horizon CUA benchmarks spanning desktop (OSWorld 2.0), web (Odysseys), and mobile (MobileWorld) environments, DLE improves task success over Vanilla by 6.5, 9.0, and 10.9 percentage points, respectively. Controlled probes compare DLE with baselines that make the current result's use explicit or maintain a persistent task graph. DLE's revision advantage persists as intervening work increases.

## 1 INTRODUCTION

Long-horizon computer-use agents (CUAs) gather information, make decisions, and update files or application objects to carry out user requests. This work can span many interactions across applications and websites, as reflected in desktop, web, and mobile benchmarks (Yuan et al., 2026; Jang et al., 2026; Kong et al., 2026). Through these interactions, agents obtain findings and intermediate conclusions that support further decisions and changes to artifacts. These results can remain relevant after execution moves to other work. Some work may still await a result, while earlier decisions may need review when a result they relied on changes.

Figure 1 illustrates two situations in a long-horizon computer-use task involving course planning. An agent needs a course schedule to create a calendar event and turns to other work while waiting. The schedule arrives, but the agent may not return to create the event. In the second situation, the agent has used the schedule to assign the course to Spring and create a 09:00 calendar event. A later correction changes the time to 10:00, but the agent may leave the existing event at 09:00. We call this ***Result–Work Desynchronization*** (***RWD***): work remains unfinished after a needed result arrives, or earlier decisions go unreviewed when that result changes.

Prior work has developed memory systems to manage and retrieve past information, including associations that evolve as new memories arrive (Packer et al., 2023; Xu et al., 2025). Planning and dependency-aware orchestration use feedback and intermediate results to revise task arrangements (Sun et al., 2023; Kim et al., 2024; Koh et al., 2026). Correction and verification help assess outputs and maintain task progress (Gou et al., 2024; Wu et al., 2026). The question here is how execution keeps track of what a result is needed for and what has already been decided on its basis.

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.

Before a result arrives, its intended use identifies the unfinished judgment it is meant to inform. After adoption, the particular version used must remain connected to the judgments and artifacts that rely on it. A later correction may leave some of these outcomes supported while requiring others to change. Because the work may have progressed in the meantime, assessing each use requires its original purpose together with the work's current judgments and unresolved questions.

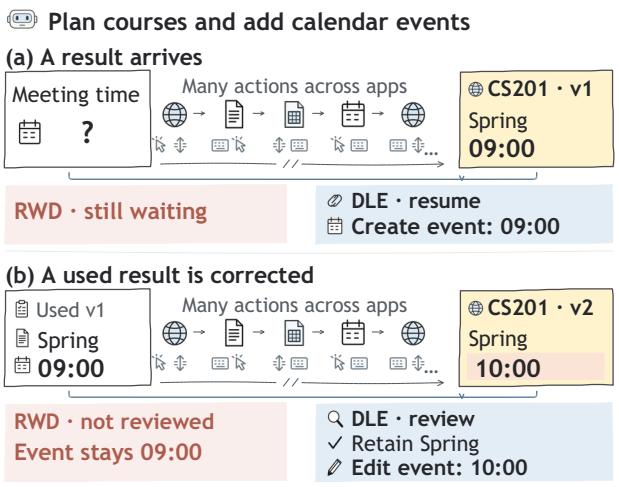



Figure 1: Example of RWD. (a) DLE resumes work when the requested result arrives. (b) After a correction, review retains the semester assignment and updates the calendar time.

In this paper, we propose ***Demand-Linked Execution*** (***DLE***) to mitigate the RWD phenomenon by preserving the intended and actual uses of results together with the evolving state of the work they support. For each work, DLE maintains a ***Work State*** record containing its current judgments, open questions, and supporting evidence. When a work requests a result, DLE's ***Result Return*** mechanism preserves why it is needed and restores that intended use with the work's latest state when the result arrives. After adoption, DLE records ***Use Links*** between the exact result version and its actual uses, enabling affected work to be located for review after a revision. The agent's model uses these records and current software observations to assess returned or revised results and decide how to proceed (Figure 3).

Our main contributions are:

1. We identify RWD in long-horizon computer-use agents: work can remain unfinished after a needed result arrives, and earlier decisions can go unreviewed when that result changes. We analyze the decision costs associated with this phenomenon (Section 4).
2. We propose DLE to mitigate the RWD phenomenon by recording what a result is needed for and which decisions or artifacts later rely on it. These records let the agent resume waiting work when the result arrives and revisit affected work when it changes (Section 5).
3. We evaluate DLE on representative long-horizon CUA benchmarks spanning desktop (OSWorld 2.0), web (Odysseys), and mobile (MobileWorld) environments. DLE improves task success over Vanilla by 6.5, 9.0, and 10.9 percentage points, respectively (Section 6).

## 2 Preliminaries

**Task interaction.** Given a user request $g$, an agent interacts with software environment $E$ under an outer-decision budget $B$. At step $t$, policy $\pi$ uses the available interaction history and observation $o_t$ to choose a decision $d_t$. A decision may issue a native action or action batch $a_t$, or perform internal processing. Observations include screenshots and permitted tool returns, following the observation–decision–action interaction studied by ReAct and CodeAct (Yao et al., 2023; Wang et al., 2024). They reveal only part of the environment. Files and application objects can persist beyond the view in which they were created, and resumed work encounters the current software state.

**Work.** We use work $w_i$ to denote an activity pursuing request $q_i$, with $i$ indexing the work. A work may pursue the original user request or a prerequisite request arising during execution. It can span multiple decisions and continue after execution has turned elsewhere. Work may obtain information, form judgments, or change external artifacts such as documents and calendar entries. Its progress includes the supported judgments and unresolved conditions relevant to its request. This separates the work still to be done from the objects already created, allowing an earlier artifact to remain relevant while other work proceeds.

**Results.** A result is an answer or intermediate conclusion obtained from observations, tool returns, or synthesis of available information. It has a scope, supporting evidence, and any remaining gaps. Later evidence may supplement or correct the answer. We use source identity $k$ to track an answer across revisions, denote version $v$ by $r_k^{(v)}$, and write its exact reference as $(k, v)$. The source identifies the continuing answer, while webpages, files, and observations provide its evidence. A new version records a revision of the answer, which may follow from correcting an earlier interpretation of unchanged external facts.

**Result use.** An intended use specifies what answer an unfinished judgment needs and how the answer will inform it. Adoption occurs when a result supplies the basis for a judgment, another answer, or an external artifact. A recorded use explicitly associates that purpose with the adopted version. We call work that produces a result its producer and work that uses it a consumer. Once a result is available, the model assesses it against the intended use to decide how to continue. A correction after adoption calls for review of the earlier use. Because one result can support several purposes, the same correction can leave some outcomes supported while requiring others to change.

## 3 Related Work

**Computer-Use Agents.** SeeClick and UGround connect language descriptions to visible interface targets, while ShowUI and UI-TARS develop visual action models (Cheng et al., 2024; Gou et al., 2025; Lin et al., 2025; Qin et al., 2025). Agent S uses experience-augmented hierarchical planning, and Agent S2 combines general planning with specialist components (Agashe et al., 2025a;b). OS-Copilot accumulates skills across tasks, while Cradle combines planning with screenshot-based keyboard and mouse control (Wu et al., 2024; Tan et al., 2025).

**Agent Memory.** MemGPT manages information across memory tiers to extend usable context (Packer et al., 2023). Generative Agents retrieves experiences and reflections for planning, while Reflexion retains verbal feedback for subsequent attempts (Park et al., 2023; Shinn et al., 2023). Agent Workflow Memory extracts reusable routines to guide later actions (Wang et al., 2025). A-MEM links memories and uses new information to update earlier representations, making memory organization itself responsive to accumulated experience (Xu et al., 2025).

**Task Orchestration.** AdaPlanner revises plans from environmental feedback, and ReCode uses recursive code to express plans and actions at different granularities (Sun et al., 2023; Yu et al., 2025). LLMCompiler schedules interdependent function calls (Kim et al., 2024). MACU maintains a task graph for computer use, passes findings to downstream agents, and revises the graph as new results arrive (Koh et al., 2026). Maintaining dependencies also has classical foundations: truth maintenance records reasons for beliefs, adaptive computation updates outputs after input changes, and provenance represents the inputs contributing to query outputs (Doyle, 1979; Acar et al., 2002; Green et al., 2007).

**Agent Self-Correction.** Self-Refine and CRITIC revise outputs using model-generated and tool-based feedback, respectively (Madaan et al., 2023; Gou et al., 2024). RARR uses external evidence to correct unsupported text while preserving other content (Gao et al., 2023). For GUI tasks, GUI-Shepherd supplies process feedback and inference-time verification, while action-effect verification checks operations to guide recovery (Chen et al., 2025; Zhang et al., 2026). StructAgent coordinates planning, acting, and verification through a shared task-progress state, with verifier-backed updates supporting checkpointing and targeted recovery (Wu et al., 2026).

DLE organizes continued execution around the particular uses of results. It records what a result is needed for before arrival and which judgments or artifacts rely on each adopted version. These uses remain available together with the work's current state to guide continuation when results return and review when they change.

## 4 Result–Work Desynchronization

We analyze RWD in Vanilla trajectories on OSWorld 2.0 with Qwen3.8-27B. An episode comprises one result transition and its relevant uses, including all uses affected by the same correction. Post-use revision includes both changes in an external source and later evidence correcting an adopted

answer. The relevant work should continue or be reviewed before renewed reliance on that answer or final delivery. For a partial answer, this response can include identifying a remaining gap and seeking further evidence. Necessary information gathering, independent work, and failures confined to native action execution are excluded from RWD.

Table 1: Incidence of result–work desynchronization.

| Result transition | Eligible episodes | Episodes with RWD | Eligible runs | Runs with RWD |
|---|---|---|---|---|
| Result availability | 2,368 | 684 (28.9%) | 108 | 82 (75.9%) |
| Post-use revision | 91 | 27 (29.7%) | 31 | 17 (54.8%) |
| Either transition | 2,459 | 711 (28.9%) | 108 | 86 (79.6%) |

RWD occurs in 28.9% of eligible episodes and affects 79.6% of the analyzed runs (Table 1). Episode incidence and trajectory coverage use their respective eligible populations. Appendix B.2 details the event boundaries, attribution, and aggregation.

RWD also consumes execution effort. We measure decisions spent reacquiring available content, recovering lost connections to established uses, continuing with superseded inputs, and repairing the consequences. For each outcome group, the cost is the number of these decisions divided by all outer decisions across the runs in that group. Under Vanilla execution, these costs account for roughly one quarter of decisions in unsuccessful runs and one sixth even in successful runs (Figure 2(a)).

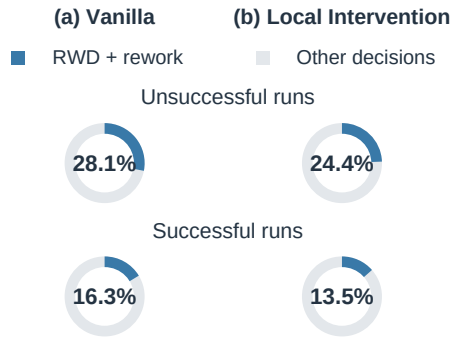



Figure 2: Decision cost associated with RWD. Shares use each configuration's own successful and unsuccessful groups.

We compare Vanilla with Local Intervention to examine the role of keeping the current result's use explicit. Local Intervention retains the current result, its intended use, and the subsequent feedback in a temporary entry. The entry is replaced when execution moves to the next result and use, with earlier content retained in the history. Figure 2(b) reports the decision shares under this intervention. The lower shares within its own outcome groups are consistent with a benefit from directing attention to the current result's use. Once the entry changes, however, a later correction may concern an earlier judgment or artifact whose connection must be recovered from history.

This motivates keeping pending and adopted uses available beyond the current interaction, together with the state needed to resume or review the relevant work.

## 5 Demand-Linked Execution

DLE starts with a root work responsible for the user request $g$. The same model advances the active work using its request, current record, and software observation. Figure 3 illustrates how a result remains connected to work throughout execution. Before requesting an answer, the runtime saves its intended use in the waiting work (①). When the answer returns, the runtime restores that use for the model to assess the answer (②). Adoption records the result version and its actual use as the work continues (③). After the model identifies a revision, the runtime locates the earlier uses and marks them for review (④). On revisiting the affected work, the model uses its latest state and current observation to retain or revise the outcome, or seek further evidence (⑤). Results obtained directly within a work are maintained through the same state and use records.

### 5.1 Work State

For work $w_i$ pursuing request $q_i$, the record $W_i$ organizes what is needed to continue: the answer being sought, supported judgments, unresolved conditions, and their evidence. Current judgments

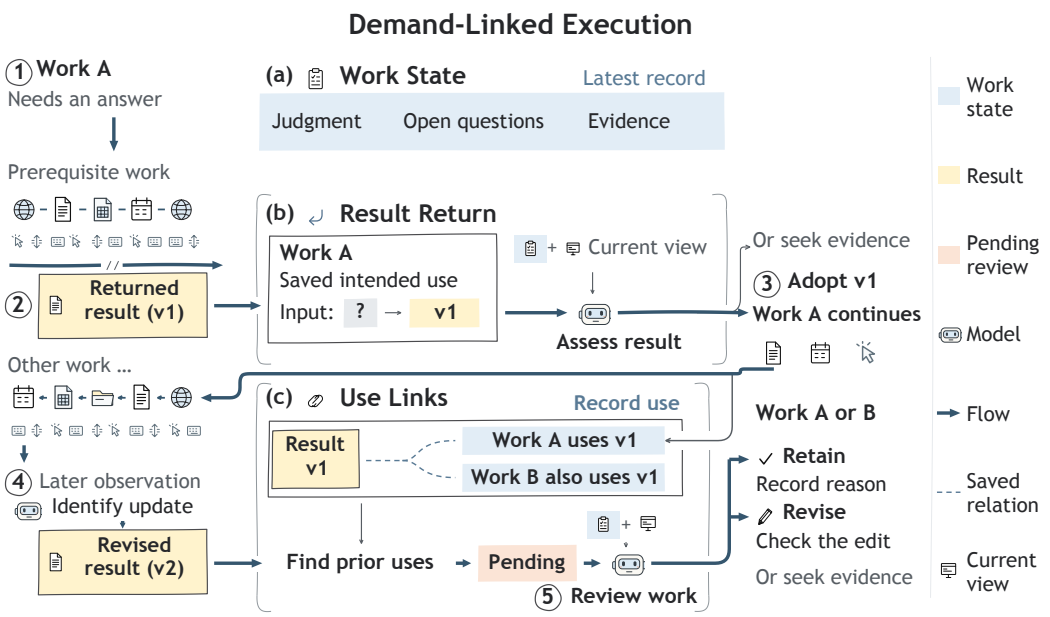



Figure 3: Demand-Linked Execution. (a) Work State maintains each work's current judgments, open questions, and evidence. (b) Result Return restores the saved intended use together with the latest work state when a requested result returns. (c) Use Links record adopted result versions and their uses, enabling affected work to be located after a revision. Repeated model icons denote the same model operating with the current software view.

anchor subsequent decisions, unresolved conditions identify missing information, and evidence references preserve their basis, including uncertainty and unchecked source content. The record is initialized from the request and available inputs, with observations and detailed history retained as supporting material.

Using the current observation, $W_i$, and relevant results, the model records which question new evidence answers, which judgment it changes, or which gap remains. Unresolved conditions persist until evidence addresses them. The runtime stores these updates and preserves earlier records for inspection. An action that leaves the work's judgments unchanged can retain the existing record.

On revisiting work, its latest record supplies the current judgments against which the model assesses a returned answer or a pending review. The model can then continue an action, revise a judgment, or seek further evidence. The record identifies what remains to be done, while the current software view determines how to carry it out.

### 5.2 Result Return

A work can request a relatively independent prerequisite answer from another work. For request event $j$, the requesting work is the caller and the work obtaining the answer is the callee. The model specifies the needed answer, constraints, selected input versions, and intended use. Before switching control, the runtime preserves the caller's record and saves

$$K_j = (i, u_j, s_j), \tag{1}$$

where $i$ identifies the caller, $u_j$ describes the pending use, and $s_j$ is the slot that will receive the answer. This return point keeps the caller's unfinished judgment identifiable while the callee proceeds. Activating the callee switches the active work index $i$ to that work.

The callee returns a result version $r_k^{(v)}$ with the request it answers, supporting evidence, applicable scope, and remaining gaps. The scope tells the caller which judgments the answer can support, while the gaps identify conditions that remain unresolved. A return may provide a complete answer, a partial answer, or an account of a limitation. It also preserves the input versions used to produce the result, retaining its historical basis as later versions become available.

On return, the runtime recovers the caller index $i$ from $K_j$, places the result and $u_j$ in $s_j$, and restores the caller's latest $W_i$ alongside the current software observation. The model compares the answer's scope and gaps with the pending judgment. Sufficient evidence permits explicit adoption of the version and continuation of the supported work. A partial answer leaves its unmet conditions open, while missing or conflicting evidence calls for further checking or a more focused prerequisite request. After adoption, Use Links preserve the version and its actual purpose so that the judgment can be reconsidered if the answer later changes.

### 5.3 USE LINKS

When a result supports a judgment, another answer, or an artifact, the model declares its exact version and the purpose it serves. Let $D(w)$ be the set of source–version references recorded for work $w$. Each use is stored separately with the judgment or artifact it supports. Let $\mathrm{uses}(d_t)$ contain the references adopted in decision $d_t$, together with the declared input references carried by those results. The runtime updates the reference set as

$$D(w_i) \leftarrow D(w_i) \cup \mathrm{uses}(d_t). \quad (2)$$

The runtime checks these references against the latest registered versions and marks an already superseded version for review. Including the declared input references allows upstream revisions to reach indirect consumers. Historical references are preserved, with later review conclusions recorded separately.

**Algorithm 1** Demand-Linked Execution

**Input:** Request $g$, policy $\pi$, environment $E$, budget $B$
**Output:** Final answer or incomplete outcome

Create root work $w_i$ for $g$
**for** $t = 1, \ldots, B$ **do**
  Assemble $C_i$ from $w_i$'s request, $W_i$, selected inputs, returned content, and pending reviews
  $(W_i, d_t) \sim \pi(\cdot \mid C_i, \mathrm{Observe}(E))$
  Store result versions declared in $d_t$
  $D(w_i) \leftarrow D(w_i) \cup \mathrm{uses}(d_t)$
  Mark stale $\mathrm{uses}(d_t)$ for review in $w_i$
  $A_t \leftarrow \{w : \exists (k, v) \in D(w), k \in \mathrm{revised}(d_t) \ \wedge \ v \neq v_{\mathrm{new}}(k)\}$
  Mark affected uses for review in each $w \in A_t$
  **if** $d_t$ requests prerequisite $j$ **then**
    $K_j \leftarrow (i, u_j, s_j)$
    Activate the callee and set $i$ to its work index
  **else if** $d_t$ returns $r_k^{(v)}$ for $j$ **then**
    $(i, u_j, s_j) \leftarrow K_j$
    Place $(r_k^{(v)}, u_j)$ in caller slot $s_j$
  **else**
    $i \leftarrow \mathrm{Execute}(d_t, i, E)$
  **if** root work is complete **then**
    **return** final answer
**return** incomplete

For new evidence, the model determines whether it supplements or corrects the same answer by considering the original question, object, and scope. It identifies the existing source and submits revised content and evidence, or registers a separate result when the answer is distinct. The runtime stores a new version under the declared source identity. Let $\Delta_t = \mathrm{revised}(d_t)$ contain the sources revised in this decision and let $v_{\mathrm{new}}(k)$ denote the identifier of the new version registered for source $k$ in this decision. The runtime identifies candidate work as

$$A_t = \{w : \exists (k, v) \in D(w),\ k \in \Delta_t,\ v \neq v_{\mathrm{new}}(k)\}. \quad (3)$$

This set identifies work whose recorded basis has changed. A source can support several purposes, and a correction may leave the conditions for some of them intact. Recovering the original use lets the model determine which judgments remain supported and which outcomes require an edit.

For each affected use, the runtime creates a pending review containing the old and new versions, original use, and owning work. The runtime lists affected work, and the model selects which to resume. On revisiting the work, the model receives the latest $W_i$, the change, and the current software observation. It can continue independent work, but must address the relevant review before relying again on the questioned basis or making final delivery.

The model assesses the change against the request, original use, and current outcome. If the required conditions still hold, it retains the outcome and records the supporting reason. Otherwise, it locates and modifies the affected object in the current software state, then checks the effect through a new observation. Insufficient evidence leaves the review open while the model investigates further. The review conclusion is recorded for this use and update. Later source revisions can create new reviews.

**Completion.** The root checks the original requirements and delivery-relevant pending reviews before returning a final answer. Unresolved requirements or reviews lead to further work, while a

callee's return resumes its caller. All decisions share the run-level budget. Exhaustion yields an incomplete outcome. The common delivery checks are detailed in Appendix A.3.

Algorithm 1 summarizes the execution loop, where $i$ denotes the currently active work index. At each decision step, the policy $\pi$ receives the assembled work context $C_i$ and current observation $o_t = \mathrm{Observe}(E)$, then produces an updated $W_i$ and decision $d_t$. The decision specifies the next operation and any result or use declarations. The $\mathrm{Execute}$ operation handles the remaining native actions and work-control operations and returns the next active work index.

# 6 EXPERIMENTS

## 6.1 EXPERIMENTAL SETUP

**Setup.** We evaluate on OSWorld 2.0, with 108 desktop workflows and a 500-decision budget (Yuan et al., 2026), Odysseys, with 200 web tasks and a 100-decision budget (Jang et al., 2026), and MobileWorld, with 201 mobile tasks and its default 50-step budget (Kong et al., 2026). Main results use two frozen backbone configurations, ***Qwen3.5-9B*** and ***Qwen3.8-27B*** (Qwen Team, 2026a;b). Task-level comparisons, controlled probes, and ablations use Qwen3.8-27B. Methods in each configuration share the model, decoding, initial environment, and native action interfaces. All receive screenshots. Resource budgets and retry rules are matched, with all control calls and reviews charged to the budget. The same delivery checks and benchmark-native evaluator apply to every method.

**Baselines.** We compare representative execution mechanisms within a shared agent runtime to examine how they support continued work when results arrive or change. ***Persistent Memory*** tests retaining and retrieving task information, following MemGPT (Packer et al., 2023). ***Adaptive Replanning*** tests revising the plan from environmental feedback, following AdaPlanner (Sun et al., 2023). ***Feedback-based Correction*** tests critique and revision of outputs using tool feedback, following CRITIC (Gou et al., 2024). Two further comparisons address how results connect to work: ***Explicit Dependency*** maintains a task/dataflow graph, including completed nodes, following LLM-Compiler (Kim et al., 2024), while ***Local Intervention*** records the current result's immediate use and feedback (Section 4). These are mechanism-level adaptations of the cited ideas. ***Vanilla*** provides the native execution baseline.

**Metrics.** OSWorld Binary, Odysseys Perfect, and MobileWorld Overall Success Rate measure task completion under each benchmark's native rules. OSWorld Partial and Odysseys Rubric Average retain graded progress, with Odysseys crediting rubric items satisfied at any point in a trajectory. Task-level comparisons pair the same tasks across methods. Appendix B details the comparisons, scoring, and budget accounting.

## 6.2 MAIN RESULTS

DLE achieves the highest scores among the compared methods on every metric under both tested backbone configurations (Table 2). The strongest alternative depends on the outcome: Explicit Dependency leads on strict completion, while Feedback-based Correction leads on OSWorld Partial and Odysseys Rubric Average. DLE improves on both, extending the gains to complete tasks as well as individual requirements.

With Qwen3.8-27B, DLE raises OSWorld Partial from 52.3% to 57.6% and Odysseys Rubric Average from 69.4% to 73.5% over their strongest respective alternatives. MobileWorld success improves by 6.0 percentage points over Explicit Dependency. The Qwen3.5-9B configuration shows the same direction of improvement across desktop, web, and mobile tasks.

**Task-level gains and regressions.** Figure 4 shows how these aggregate gains are distributed across paired tasks. We compare against Feedback-based Correction for OSWorld Partial and Odysseys Rubric, and Explicit Dependency for MobileWorld SR. Improvements outnumber regressions in all three comparisons, alongside substantial groups with unchanged scores.

On MobileWorld, for example, DLE completes 25 tasks that Explicit Dependency fails, while the reverse occurs on 13 tasks. The net gain therefore includes both newly completed tasks and lost successes. On OSWorld and Odysseys, the bars instead describe changes in graded progress. We

Table 2: Main results on desktop, web, and mobile benchmarks under two backbone configurations. Methods within each model block share decoding, native action interfaces, and resource budgets. Scores are percentages, with higher values indicating better performance. DLE rows are shaded. Within each block, bold marks the best score and italics the strongest baseline.

| | OSWorld 2.0 | | Odysseys | | MobileWorld |
|---|---|---|---|---|---|
| Method | Binary ↑ | Partial ↑ | Perfect ↑ | Rubric Avg. ↑ | SR ↑ |
| **Qwen3.5-9B** | | | | | |
| Vanilla | 3.7 | 8.0 | 13.5 | 42.6 | 8.5 |
| Persistent Memory | 3.7 | 8.7 | 14.0 | 45.0 | 9.5 |
| Adaptive Replanning | 4.6 | 7.9 | 17.0 | 44.1 | 9.0 |
| Feedback-based Correction | 3.7 | *9.3* | 15.0 | *47.0* | 10.0 |
| Explicit Dependency | *5.6* | 9.1 | *18.5* | 46.1 | *11.4* |
| Local Intervention | 3.7 | 8.9 | 16.5 | 46.3 | 10.4 |
| **DLE** | **7.4** | **13.8** | **24.5** | **52.8** | **13.4** |
| **Qwen3.8-27B** | | | | | |
| Vanilla | 19.4 | 48.0 | 41.5 | 65.0 | 20.4 |
| Persistent Memory | 20.4 | 50.8 | 42.5 | 68.2 | 21.9 |
| Adaptive Replanning | 22.2 | 48.5 | 45.5 | 66.0 | 22.4 |
| Feedback-based Correction | 20.4 | *52.3* | 44.0 | *69.4* | 23.4 |
| Explicit Dependency | *23.1* | 51.2 | *47.0* | 68.0 | *25.4* |
| Local Intervention | 20.4 | 49.6 | 43.5 | 67.2 | 23.9 |
| **DLE** | **25.9** | **57.6** | **50.5** | **73.5** | **31.3** |

next examine whether result changes continue to reach the relevant work after execution has moved elsewhere.

### 6.3 Synchronization across Dependency Delays

We test result–work synchronization as the amount of independent intervening work increases within matched episodes. In availability episodes, a result becomes available for waiting work. In revision episodes, a correction arrives after an earlier version has been used. Methods start before demand formation or adoption and build their own records from ordinary observations and tool returns.

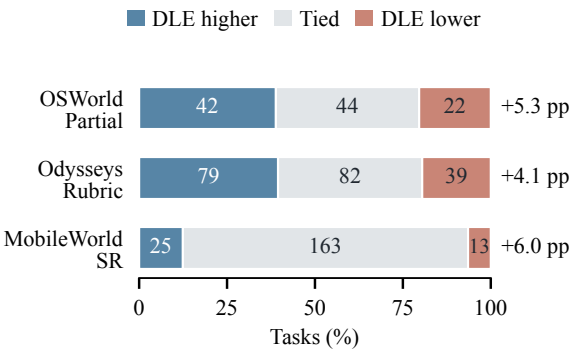



Figure 4: Task-level changes with Qwen3.8-27B against Feedback-based Correction (OSWorld, Odysseys) and Explicit Dependency (MobileWorld). Labels give task counts and aggregate native-score gains.

We use 80 revision episodes from at least 40 OSWorld-style tasks, with at most two episodes per task, and a separate set of 80 availability episodes. Each episode runs at delays of 0, 20, 50, and 100 outer decisions. Matched runs share the starting environment, intervening work, result-change trigger, and remaining budget. For availability, independent decisions separate establishing the need from result arrival. For revision, they separate adoption from correction. Correct consumer links and source mappings are withheld from all methods. Availability success requires using the result to advance the intended work without redundant acquisition or damage to other requirements. Revision success requires all necessary consumer changes before renewed reliance or delivery, while preserving unaffected work. Environment outcomes or independent rubrics determine success.

Figure 5(a) shows that DLE sustains revision success better as intervening work increases. At delay 100, it achieves 63.8%, compared with 45.0% for Local Intervention and 33.8% for Explicit Depen-

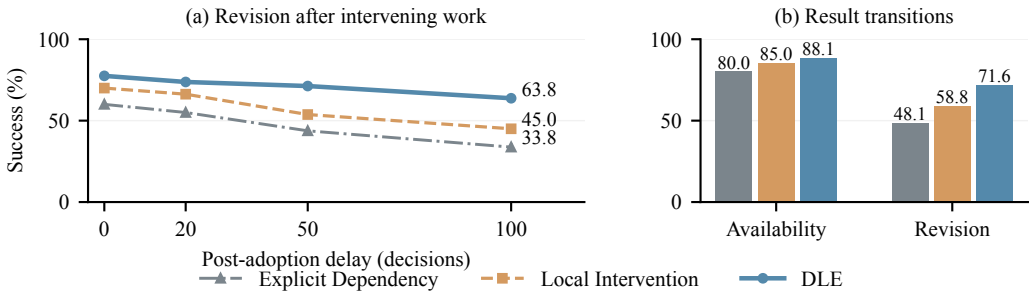



Figure 5: Controlled result–work synchronization probes with Qwen3.8-27B. Delays count independent decisions from establishing a need to result arrival (availability), or from adoption to correction (revision). (a) Semantic revision success as delay increases. (b) Success for each transition type, averaged equally over four delay levels.

dency. Across the delay range, DLE declines by 13.8 percentage points, versus 25.0 and 26.2 points for the alternatives. The advantage thus persists beyond an immediate response to the correction and grows in the reported rates as more work separates adoption from revision.

Figure 5(b) distinguishes initial availability from revision after use. Averaged equally over delays, DLE's advantage over Local Intervention is 3.1 points on availability and 12.8 points on revision. In these probe sets, the additional benefit is more pronounced when an already used result changes. Explicit Dependency retains its task graph, and Local Intervention makes the current use explicit. DLE additionally retains version-specific uses and brings earlier work back for review. The larger revision advantage is consistent with keeping those uses actionable after execution has moved on.

### 6.4 Core Ablations

Table 3 tests the execution support provided by Work State, Result Return, and Use Links. Each variant retains the original request and history, so the model can reconstruct information that the removed support would otherwise supply. Without carried work state, it reconstructs current judgments and open questions. Without the saved return point, it receives the result and returns to the caller but must recover the unfinished use. Without automatic affected-work lookup, it retains exact versions and historical uses but must locate consumers itself, without automatic review injection. All variants share the delivery checks of full DLE.

Table 3: Core ablations with Qwen3.8-27B. Scores are percentages. Full DLE is shaded.

| Variant | OSWorld Partial | Odysseys Rubric | Mobile SR |
|---|---|---|---|
| **DLE** | **57.6** | **73.5** | **31.3** |
| w/o carried work state | 54.6 | 69.8 | 26.9 |
| w/o saved return point | *55.1* | 68.9 | 27.4 |
| w/o automatic affected-work lookup | 52.9 | *71.0* | *28.4* |

Each removal lowers scores across the three benchmarks. Removing state support reduces MobileWorld success by 4.5 percentage points. Removing the saved return point lowers Odysseys Rubric by 4.6 points, while removing automatic affected-work lookup lowers OSWorld Partial by 4.7 points.

## 7 Conclusion

In this paper, we identify RWD in long-horizon computer-use agents, where available or revised results fail to reach the work that depends on them. To mitigate the RWD phenomenon, we propose DLE, which preserves result uses together with each work's evolving state so agents can resume waiting work and revisit earlier decisions. Experiments across desktop, web, and mobile benchmarks show improved task success, while controlled probes show that its advantage persists as independent work separates adoption from revision.

## AI USE STATEMENT

We used large language models to improve the language quality of the manuscript, including grammar correction, wording refinement, and clarity enhancement, for limited coding assistance, such as debugging and code implementation suggestions, and for trajectory annotation. All AI-assisted manuscript content and code were reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

## REFERENCES


Umut A. Acar, Guy E. Blelloch, and Robert Harper. Adaptive functional programming. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 247–259. ACM, 2002. doi: 10.1145/503272.503296.

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S: An Open Agentic Framework that Uses Computers Like a Human. In *The Thirteenth International Conference on Learning Representations*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=lIVRgt4nLv.

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S2: A Compositional Generalist-Specialist Framework for Computer Use Agents. In *Second Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=zg5is4GJ3R.

Cong Chen, Kaixiang Ji, Hao Zhong, Muzhi Zhu, Anzhou Li, Guo Gan, Ziyuan Huang, Cheng Zou, Jiajia Liu, Jingdong Chen, Hao Chen, and Chunhua Shen. GUI-Shepherd: Reliable Process Reward and Verification for Long-Sequence GUI Tasks. *arXiv preprint arXiv:2509.23738*, 2025. doi: 10.48550/arxiv.2509.23738. URL https://arxiv.org/abs/2509.23738.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.505. URL https://aclanthology.org/2024.acl-long.505/.

Jon Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12(3):231–272, 1979. doi: 10.1016/0004-3702(79)90008-0.

Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Y. Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, and Kelvin Guu. RARR: Researching and Revising What Language Models Say, Using Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16477–16508. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.910. URL https://aclanthology.org/2023.acl-long.910/.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the Digital World as Humans Do: Universal Visual Grounding for GUI Agents. In *The Thirteenth International Conference on Learning Representations*. OpenReview.net, 2025. URL https://openreview.net/forum?id=kxnoqaisCT.

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 *The Twelfth International Conference on Learning Representations*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 31–40. ACM, 2007. doi: 10.1145/1265530.1265535.

Lawrence Keunho Jang, Jing Yu Koh, Daniel Fried, and Ruslan Salakhutdinov. Odysseys: Benchmarking Web Agents on Realistic Long Horizon Tasks. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=oGWqU9VHQ6.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM Compiler for Parallel Function Calling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24370–24391. PMLR, 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

Jing Yu Koh, Ruslan Salakhutdinov, and Daniel Fried. Multi-Agent Computer Use. *arXiv preprint arXiv:2606.01533*, 2026. doi: 10.48550/arxiv.2606.01533. URL https://arxiv.org/abs/2606.01533.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. MobileWorld: Benchmarking Autonomous Mobile Agents in Agent-User Interactive and MCP-Augmented Environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6142–6167. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.278. URL https://aclanthology.org/2026.acl-long.278/.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. ShowUI: One Vision-Language-Action Model for GUI Visual Agent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19498–19508. Computer Vision Foundation / IEEE, 2025. doi: 10.1109/cvpr52734.2025.01816. URL https://openaccess.thecvf.com/content/CVPR2025/html/Lin_ShowUI_One_Vision-Language-Action_Model_for_GUI_Visual_Agent_CVPR_2025_paper.html.

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. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*, 2023. doi: 10.48550/arxiv.2310.08560. URL https://arxiv.org/abs/2310.08560v2.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 2:1–2:22. ACM, 2023. doi: 10.1145/3586183.3606763.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: Pioneering Automated GUI Interaction with Native Agents. *arXiv preprint arXiv:2501.12326*, 2025. doi: 10.48550/arxiv.2501.12326. URL https://arxiv.org/abs/2501.12326.

Qwen Team. Qwen3.5-9B. Model card, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-9B.

Qwen Team. Qwen3.8-27B. Model card, 2026b. URL https://huggingface.co/Qwen/Qwen3.8-27B.

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. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive Planning from Feedback with Language Models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 58202–58245. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/b5c8c1c117618267944b2617add0a766-Abstract-Conference.html.

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, Ruyi An, Molei Qin, Chuqiao Zong, Longtao Zheng, Yujie Wu, Xiaoqiang Chai, Yifei Bi, Tianbao Xie, Pengjie Gu, Xiyun Li, Ceyao Zhang, Long Tian, Chaojie Wang, Xinrun Wang, Börje F. Karlsson, Bo An, Shuicheng Yan, and Zongqing Lu. Cradle: Empowering Foundation Agents towards General Computer Control. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 58658–58725. PMLR, 2025. URL https://proceedings.mlr.press/v267/tan25h.html.

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, 2024. URL https://proceedings.mlr.press/v235/wang24h.html.

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. URL https://proceedings.mlr.press/v267/wang25bx.html.

Wenyi Wu, Sibo Zhu, Kun Zhou, Aayush Salvi, Zixuan Song, and Biwei Huang. StructAgent: Harness Long-horizon Digital Agents with Unified Causal Structure. *arXiv preprint arXiv:2607.11388*, 2026. doi: 10.48550/arxiv.2607.11388. URL https://arxiv.org/abs/2607.11388.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. OS-Copilot: Towards Generalist Computer Agents with Self-Improvement. *arXiv preprint arXiv:2402.07456*, 2024. doi: 10.48550/arxiv.2402.07456. URL https://arxiv.org/abs/2402.07456.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic Memory for LLM Agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 17577–17604. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhaoyang Yu, Jiayi Zhang, Huixue Su, Yufan Zhao, Yifan Wu, Mingyi Deng, Jinyu Xiang, Yizhang Lin, Lingxiao Tang, Yuyu Luo, Bang Liu, and Chenglin Wu. ReCode: Unify Plan and Action for Universal Granularity Control. *arXiv preprint arXiv:2510.23564*, 2025. doi: 10.48550/arxiv.2510.23564. URL https://arxiv.org/abs/2510.23564v4.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, Dunjie Lu, Haikong Lu, Qi Zhen, Xinyuan Wang, Jiaqi Deng, Yuhao Yang, Cheng Chen, Boyuan Zheng, Alex Su, Xiao Yu, Hao Zou, Saaket Agashe, Xing Han Lu, Manpreet Kaur, Zhengyang Qi, Vincent Sunn Chen, Frederic Sala, Dayiheng Liu, Junyang Lin, Zhou Yu, Yu Su, Siva Reddy, Xin Eric Wang, Peng Qi, Tianbao Xie, and Tao Yu. OSWorld 2.0: Benchmarking Computer Use Agents on Long-Horizon Real-World Tasks. *arXiv preprint arXiv:2606.29537*, 2026. URL https://arxiv.org/abs/2606.29537.

Yuzhe Zhang, Xianwei Xue, Xingyong Wu, Mengke Chen, Chen Liu, Xinran He, Run Shao, Feiran Liu, Huanmin Xu, Qiutong Pan, and Haiwei Wang. Don't Act Blindly: Robust GUI Automation

via Action-Effect Verification and Self-Correction. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28924–28941. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1335. URL https://aclanthology.org/2026.acl-long.1335/.

# A Implementation of Demand-Linked Execution

This appendix explains how DLE's work records and versioned uses enter policy decisions, and how the runtime handles returns, reviews, and completion.

## A.1 Work records and control flow

Table 4: Objects used by the implementation listings. Historical support and the basis established by a later review are stored separately.

| Store | Object | Contents and role |
|---|---|---|
| `W` | Work | Request, current judgments, named needs, evidence, and returned inputs. State patches merge needs by their slot. |
| `K` | Call continuation | Caller, callee, question, receiving slot, and intended purpose for each invocation. |
| `R` | Result version | Immutable answer content, source identity, status, gaps, evidence, and declared input versions. |
| `U` | Recorded use | Consumer work, directly adopted reference, slot, purpose, artifact or judgment targets, and historical support. |
| `I` | Source index | Recorded uses whose support includes a source, enabling lookup after an update. |
| `Q` | Review | Uses grouped by work, source, previous basis, and new version. |

`review_context` assembles each review's old and new versions and recorded uses. `merge_named_needs` patches needs by slot while preserving other entries. `append_version` stores the declared result and advances its source version. `Record` and identifier allocation construct the stored objects.

Bringing retained work, returned inputs, and reviews into the next policy decision.

```
def step(w, env):
    reviews = [review_context(k) for k in Q if k[0] == w]
    d = policy(request=W[w].request, state=W[w].state,
        returned=[(item, R[item.ref]) for item in W[w].inbox],
        reviews=reviews,
        affected_work={k[0] for k in Q}, observation=env.observe())
    W[w].state = merge_named_needs(W[w].state, d.state_patch)
    for result in d.results:
        ref = append_version(result)
        mark_affected(ref.source)
    for use in d.uses:
        adopt(w, use.ref, use.slot, use.purpose, use.targets)
    op = d.operation
    if op.kind == "call":
        c = fresh_call_id()
        K[c] = (w, op.child, op.slot, op.question, op.purpose)
        return op.child
    if op.kind == "return":
        caller, child, slot, question, purpose = K.pop(op.call)
        W[caller].inbox.append(Record(ref=op.ref, slot=slot,
            question=question, purpose=purpose,
            status=R[op.ref].status, gaps=R[op.ref].gaps))
        return caller
    if op.kind == "resume":
        return op.work
    if op.kind == "conclude_review":
        conclude(op.key, op.outcome, op.reason, op.evidence)
    if op.kind == "act":
        env.execute(op.action)
    return w
```

Each call creates a continuation for that invocation. Returning consumes the continuation and delivers an exact result reference with its status and gaps to the caller. The reference enters the next policy input, where the caller can assess it before adoption.

### A.2 Adopted inputs and review

Adoption includes the cited result and its transitive declared inputs. Thus an upstream correction can reach work that used a derived answer. Each recorded use retains its actual purpose and targets, while the source index locates the relevant records when a version changes. Adopting an already superseded version invokes the same lookup.

Recording an adopted basis, retrieving its consumers, and recording review conclusions.

```
def adopt(w, ref, slot, purpose, targets):
    support, frontier = set(), [ref]
    while frontier:
        r = frontier.pop()
        if r not in support:
            support.add(r)
            frontier.extend(R[r].inputs)
    u = fresh_use_id()
    U[u] = Record(work=w, direct=ref, slot=slot, purpose=purpose,
                  targets=targets, support=support, reviewed={})
    for source, version in support:
        I[source].add(u)
    for source in {s for s, v in support}:
        mark_affected(source)

def mark_affected(source):
    new = latest[source]
    for u in I[source]:
        for s, original in U[u].support:
            basis = U[u].reviewed.get(s, original)
            if s == source and basis != new:
                key = (U[u].work, source, basis, new)
                Q.setdefault(key, set()).add(u)

def conclude(key, outcome, reason, evidence):
    if outcome == "blocked":
        return
    w, source, old, new = key
    conclusions[key] = (outcome, reason, evidence)
    for u in Q.pop(key):
        U[u].reviewed[source] = max(new, U[u].reviewed.get(source, 0))
```

The `support` field preserves the original basis. The `reviewed` field records the version against which the use was subsequently assessed. Lookup compares the latest source version with that reviewed basis when available, or with the original adopted version otherwise. Review keys group the affected uses of one work under a source transition.

A review can span several policy decisions: inspection, a native edit, and an observation of its effect. Closing it advances the reviewed basis of its uses while preserving their historical support. A later source version can therefore trigger another review.

### A.3 Completion and policy prompt

The shared delivery guard uses the original requirements, final evidence-inspection opportunity, and validation budget. DLE associates each required outcome with its responsible work and records checks of the target properties, observations, status, and input versions. The root assesses both requirement coverage and the current support for these checks. Completion requires satisfied requirements and resolved delivery-relevant reviews.

An artifact modification requires observation of its effect. A source update requires reconsidering the associated checks, allowing supported outcomes to be retained. The native evaluator separately scores the delivered environment. All calls, retries, reviews, and control operations consume the common budget, with cumulative use preserved across continuation. Exhaustion yields an incomplete outcome.

At a return, the policy receives the saved question and purpose alongside the answer's evidence, scope, and gaps. During review, it receives the original uses with the old and new versions. Work-state refresh incorporates new observations, while recorded-result assessment uses evidence already attached to findings. Fresh interface inspection occurs in the execution view. These views share the same policy and budget and follow the prompt below.

**General DLE Prompt**

You are carrying out the user's request through the permitted native interfaces. Use the current software state and actual observations. Keep the original request and its unresolved requirements in view throughout execution.

**Work State**

Maintain a compact judgment, its evidence and uncertainty, open questions, coverage of the relevant sources, and the next useful direction for each work. Preserve unresolved requirements when updating another part of the record. A planned action is not an observed outcome. Update a record when information or verified progress changes it; unchanged local actions need not repeat the record. When work resumes, combine its latest record with the current observation. The software has not been restored to the old interface state.

**Result Return**

Request a bounded prerequisite when a decision needs an answer that warrants separate work. State the required answer, intended use, and receiving slot. A new request event is distinct from a work identity; an existing work may be called again. Use the identifiers and interfaces supplied by the runtime. A return may be complete, partial, or a supported limitation. Preserve scope, observed evidence, exact input versions, and material gaps. An existing result may be returned by exact reference where the interface permits it. On return, assess the answer against the caller's intended use and current record. Choose which supported findings to adopt and for what purpose. Keep other gaps open or request a bounded follow-up. Receiving an answer does not adopt it, complete its caller, or imply that an external artifact has been updated.

**Use Links**

When a result supports a judgment, an output, or an external change, explicitly declare its exact version and purpose. Keep the historical basis of that use. When new evidence changes the same underlying answer, identify that existing source and submit the supported revision. Distinct questions remain separate. You are responsible for this source correspondence; the runtime does not infer it from a matching word or supply a gold list of affected work. For a review, consider the original use, the old and new result, the work's latest record, and the current software evidence. Retain an outcome when its relevant conditions still hold, recording the reason. Otherwise make the required change through the permitted interface and inspect the resulting state before concluding that the edit succeeded. If evidence is insufficient, seek it and keep the review unresolved. An unrelated outcome should not be edited merely because a source has a new version. A later revision is a new review obligation.

**Completion And Evidence**

Keep availability, adoption, review, local outcome, and full-task completion separate. A tool receipt establishes only what it actually reports. Use observations of the affected artifact to support an edit outcome, not pre-edit evidence. Before a new dependent use or final delivery, address the relevant unresolved reviews. Preserve requirements not yet fulfilled. If the original task cannot be completed within the budget, report its incomplete status and remaining work. All interaction and model calls use the common accounting and delivery protocol.

# B Experimental Details

This appendix details the comparison mechanisms, RWD annotation, delayed probes, paired task outcomes, and resource accounting for the experiments in Section 6.

## B.1 Comparison mechanisms and ablation scope

Each configuration in Table 2 evaluates the full benchmark task set with one outcome per task. Core ablations use the same sets with Qwen3.8-27B. Strict scores divide successful task counts by the set size, while graded scores average native task scores.

Within each frozen configuration, methods share the model revision, chat template, reasoning settings, and native action interfaces. Each receives screenshot observations and can access its own interaction history. Step, token, model-call, and wall-time limits are matched, as are retry rules. The common delivery check gives each method the original requirement checklist, a final opportunity to inspect environment evidence, and the same summary and validation budget. It supplies neither source correspondences nor lists of affected consumers.

Persistent Memory stores requirements, results, unresolved questions, completed work, and failures, including free-text descriptions of use. It retrieves relevant memory on demand and supplies it to the current decision. Adaptive Replanning uses environment feedback to add, delete, reorder, or reopen items in a plan that retains active, blocked, and completed work. Feedback-based Correction uses tool feedback to critique and iteratively revise the current target, and can search earlier outputs to bring them into focus. Explicit Dependency retains completed nodes in its task/dataflow graph, revises the graph as execution proceeds, and activates downstream work when prerequisites become available. Local Intervention replaces its current result-use-feedback entry as execution moves to another use, while earlier content remains accessible in history.

All comparisons can revisit earlier work through retained information. DLE additionally indexes declared uses by exact version to retrieve historical consumers after a source update. The model judges whether an edit is needed. Table 5 specifies the automatic support removed by each ablation and the information retained for reconstruction.

Table 5: Interventions behind Table 3. Native interfaces, delivery checks, and resource accounting are shared with full DLE.

| Variant | Removed support | Retained information and operations |
|---|---|---|
| w/o carried work state | Automatic carrying of judgments, uncertainty, source coverage, remaining questions, next direction, and re-entry summary. | Original request, work identity, results, use links, and full history. |
| w/o saved return point | Saved intended use and receiving-slot cues for the unfinished judgment. | Caller identity, returned result, request, history, and legal return of control. |
| w/o automatic affected-work lookup | Automatic consumer enumeration and pending-review presentation after an update. | Source identities, exact versions, historical uses, update observations, and manual record/link queries. |

## B.2 RWD episode boundaries and aggregation

For the OSWorld 2.0 Vanilla trajectories with Qwen3.8-27B, an episode comprises one result transition and all its relevant uses. Revision concerns an adopted answer corrected by an external source update or later evidence. Visual refreshes, rephrasing, and changes to unadopted hypotheses are excluded.

The episode begins when the answer becomes available or is corrected. The original request and preceding decisions establish the intended uses, while observations and tool returns establish what answer or correction is available. Subsequent decisions and artifact observations determine whether those uses have been continued or reconsidered before renewed reliance or delivery. For a transition affecting several uses, the episode remains unresolved until all relevant uses have been addressed.

Seeking evidence for an insufficient partial answer is an appropriate response. Necessary information acquisition, legitimate independent work, and failures confined to native GUI actions are excluded from RWD attribution. Attributed decisions reacquire an already available answer, recover a lost connection to its previously established use, proceed from a superseded basis, or repair the resulting consequences.

GPT-5.6 performed the trajectory annotation for Table 1 and Figure 2, applying the criteria and exclusions above to label RWD episodes and identify decisions attributed to RWD and rework.

Table 1 counts each eligible episode once and deduplicates trajectories for run coverage. Revision coverage is conditioned on trajectories containing an eligible revision. Figure 2 pools RWD-related decisions and associated rework within each method's own successful or unsuccessful group and divides by all outer decisions in that group. A decision concerning several uses is counted once.

### B.3 Construction and scoring of delayed episodes

Each base episode in Section 6.3 specifies a work situation, a result transition, and the uses whose response will be scored. A trial applies one method at one delay to that episode. The 80 availability and 80 revision episodes are each evaluated at four delays, giving 320 trials per transition type and method. The per-delay denominator is the same 80 episodes throughout. A trial that fails to reach or complete the required result-use response does not count as successful.

The probes begin before the relation being tested is formed. An availability episode starts before the model has established the pending need. A revision episode starts before the first version is adopted in a judgment or artifact. Each method builds its own records through execution.

After establishing the pending need in availability episodes or adopting the first version in revision episodes, the method performs the specified independent work. After the chosen number of outer decisions, the available answer or correction enters through an ordinary observation or tool return. For each paired condition, methods share a starting checkpoint or deterministic prefix, intervening-work specification, transition trigger, remaining budget, and success criterion. Reusing the same base episode across methods and delays holds the underlying work and correction fixed while changing the intervening execution.

Scoring checks the effect on work. Availability requires the correct answer to advance its intended use without redundant acquisition or damage to other requirements. Revision requires all necessary changes to affected consumers, preservation of unaffected objects, and completion before renewed reliance or delivery. The target uses belong to the scoring criterion and are withheld from the acting methods. Environment outcomes or independent rubrics determine success, rather than the presence of an internal pending-review or resumable flag. The transition-type summary in Figure 5(b) assigns equal weight to each delay level.

### B.4 Paired task outcomes and resource accounting

Figure 4 compares the same tasks under DLE and the strongest comparison method for each metric. OSWorld and Odysseys use graded scores, so higher and lower refer to progress on the native metric. Table 6 separates shared MobileWorld outcomes from DLE-only and comparison-only successes. Their difference gives the net gain.

Table 6: Paired MobileWorld outcomes with Qwen3.8-27B. Explicit denotes Explicit Dependency.

| | Explicit succeeds | Explicit fails | Total |
|---|---|---|---|
| DLE succeeds | 38 | 25 | 63 |
| DLE fails | 13 | 125 | 138 |
| Total | 51 | 150 | 201 |

Differences use unrounded task scores or integer success counts and are rounded to one decimal place using ties-to-even.

Resource accounting covers decisions, model calls, tokens, and elapsed time for every attempted task. Auxiliary reasoning, returns, reviews, and retries contribute to the totals. A control-only decision consumes an outer decision. An action batch dispatched once counts as one native-action dispatch. Continued executions retain cumulative counters.

# C Execution Cases

These cases connect DLE's records to concrete decisions and observed software states. The first three follow native desktop execution: an audit answer returns to its saved use, timetable information guides a course selection, and new observations change the work's current judgment. The final controlled excerpt isolates how one revised result reaches two earlier uses.

## C.1 Returning an answer to its saved use

The course-planning request requires the agent to identify remaining compulsory courses and distribute them across two semesters. An initial text-extraction attempt cannot read the image-based degree audit. The caller therefore keeps the audit question open and requests further reading from the same prerequisite work, preserving the purpose of identifying which courses must be included in the plan.

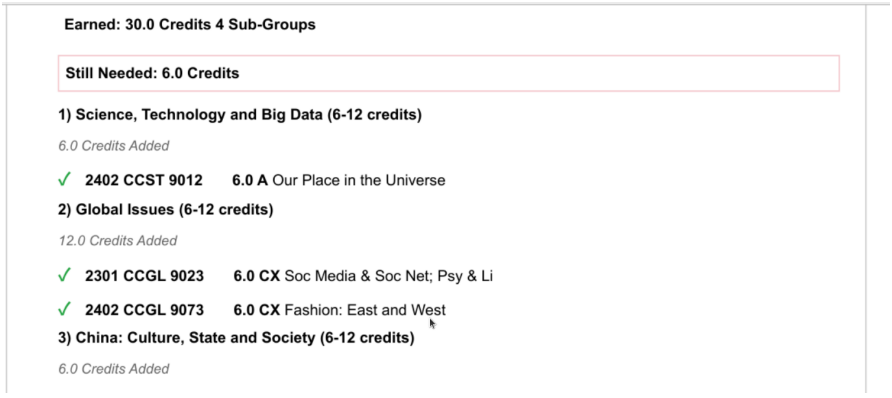



Figure 6: Evidence available when the audit answer returns. The visible report distinguishes completed credit from a remaining Common Core requirement. The execution records below connect the returned answer to the caller's next planning decision.

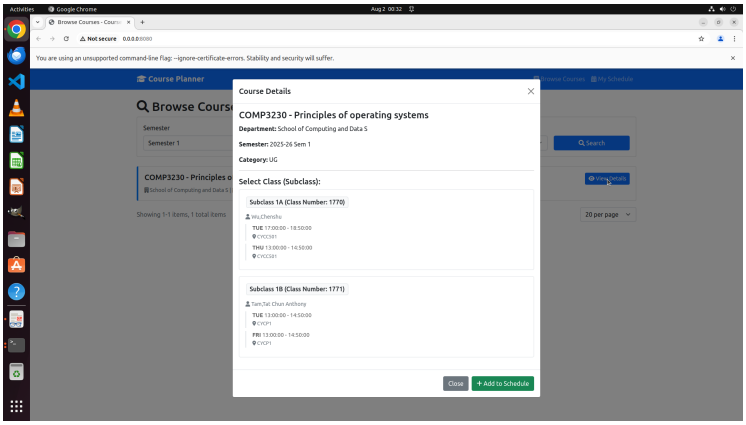
The prerequisite work reads the report visually and returns the required course list, completed courses, and remaining credit requirements. Figure 6 shows part of that source at return. The runtime delivers the answer to the saved audit-use slot. In its next decision, the caller explicitly adopts that exact result version to determine the courses still required and incorporates the answer into its current planning judgment.

| Stage | Recorded state or operation |
|---|---|
| Pending use | Identify remaining compulsory courses for the next academic year's plan. |
| Prerequisite work | Read the image-based audit after the unsuccessful text-extraction attempt. |
| Return | Deliver the completed audit answer to the original caller and its saved use. |
| Adoption | Select the returned version for identifying required courses and update the planning judgment. |
| Next request | Read the syllabus to establish course-load, elective, and semester constraints. |

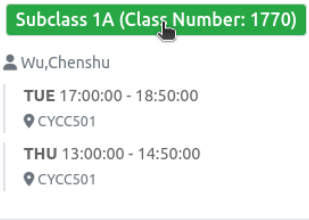
Adopting the audit answers one planning question, while syllabus constraints and section availability remain to be checked. The caller consequently requests the syllabus rules as the next prerequisite.

### C.2 Using a recorded finding to select a course section

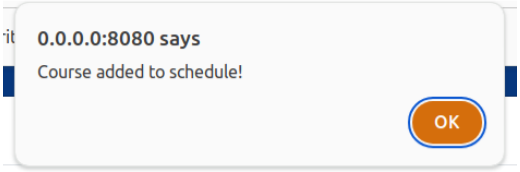
A further native sequence connects a recorded timetable finding to a visible enrolment action. The course-selection work inspects the available sections of COMP3230 and records that their meeting times avoid 09:00, as requested by the user. It explicitly adopts the section information for choosing a first-semester class, while keeping the other courses' timetable checks open.




(a) Inspecting the available sections and meeting times.




(b) Selecting Subclass 1A.




(c) Confirmation after adding the section.

Figure 7: Recorded information guiding a native action. The full view shows the course and its section options. The enlarged views show the selected section and the subsequent "Course added to schedule!" message.

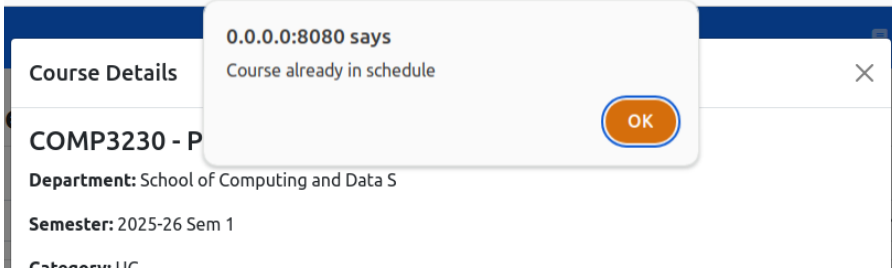
The model selects Subclass 1A and clicks *Add to Schedule*. The next observation contains the application's confirmation message. The model records this local outcome and updates the course-selection need, retaining the remaining courses and timetable checks as unfinished work.

| Stage | Recorded state or operation |
|---|---|
| Finding and use | Section times are recorded and adopted for choosing a class that avoids 09:00. |
| Native operation | Subclass 1A is selected and added through the course-planner interface. |
| Observed feedback | The application displays its confirmation message. |
| Work State update | The selected course is recorded as added, while other enrolment requirements remain open. |

The timetable supports the section choice, while the subsequent confirmation supports recording it as added.

### C.3 Updating work state from current evidence

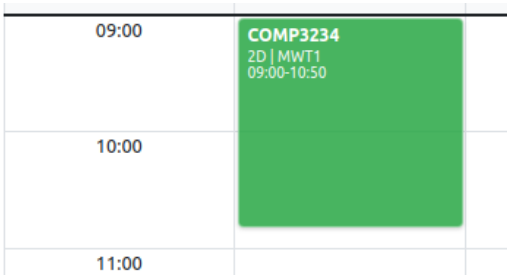
Later in the same native execution, empty timetable views lead the agent to suspect that earlier enrolments have been lost. It tries to add COMP3230 again. The application instead reports "Course already in schedule," prompting a different next step: inspect the existing schedule before attempting further enrolment.




(a) A duplicate-add response prompts reinspection.




(b) Earlier first-semester entries are still present.




(c) A second-semester morning meeting remains to be assessed.

Figure 8: New observations changing the current work judgment. The duplicate-add message prompts a check of the existing timetable. Visible courses support retaining the earlier enrolments, while the work record keeps remaining checks open.

The model records that the duplicate-add response conflicts with the apparently empty timetable and that the existing entries need verification. On reopening *My Schedule*, it observes the first-semester courses and updates the enrolment need accordingly. It then checks the second semester, recording the visible courses and a morning meeting that still needs assessment against the user's preference.

| Record field | Change supported by the observations |
|---|---|
| Current judgment | Replace suspected loss of enrolments with the observed presence of earlier course entries. |
| Next direction | Inspect both semester views before adding courses again. |
| Open conditions | Retain checks of unviewed timetable rows, remaining course requirements, and calendar export. |

Work State carries the revised interpretation into the next decision. Confirming that courses remain present resolves the immediate uncertainty without treating the overall enrolment and calendar request as complete.

### C.4 Revisiting the work that used a corrected result

A single recorded segment of the controlled course application follows a schedule result from initial adoption through review by its two consumers. The request separates schedule lookup, semester planning, and calendar creation into distinct work records. The excerpt ends after the planning and calendar works have each reviewed the correction.

The lookup result initially places CS201 in Spring at 09:00. The planning work adopts that version to assign the semester, and the calendar work adopts it to create the event. Both preserve their resulting judgments and exact input versions. Later work reveals a corrected meeting time of 10:00. The model registers a new version of the same schedule source, and the runtime marks its earlier uses for review. A correction work performs the calendar edit before the original consumers are resumed.

Table 7: Recorded continuation and review after a schedule correction. The two works use different properties of the same result.

| Recorded stage | Planning work | Calendar work |
|---|---|---|
| Original adoption | Uses Spring from the original schedule to create the plan. | Uses 09:00 from the same schedule to create the event. |
| Source correction | The schedule's semester stays Spring. The recorded use becomes available for review. | The schedule's time changes to 10:00. The recorded use becomes available for review. |
| Work revisited | Its current record supplies the Spring assignment. Review retains that judgment. | Its current record supplies the earlier creation. Review checks the event already edited by the correction work. |
| Observed outcome | Plan remains in Spring. | Calendar event is at 10:00. |

The planning work receives its current record and the revised schedule, checks that the semester is unchanged, and retains the assignment. The calendar work receives its own current record and checks the event now stored at 10:00. Its historical creation result continues to cite the earlier schedule version, while the review records why the current event is supported.

The separate historical and reviewed bases in Appendix A preserve both the original creation basis and the later assessment.