Qwen-Planner-Agent: A Closed-Loop AI-for-AI Framework for Real-World Mobile Planner Agents

MAI Team, Alibaba Token Hub, Alibaba Group

https://github.com/Tongyi-MAI/Qwen-Planner-Agent

Abstract

The rapid progression of large language models is extending AI from passive content generation into the active workflows of engineering and scientific discovery. This shift raises a compelling question: can AI be both the object of development and an active participant in building next-generation AI systems? We explore this question by building **Qwen-Planner-Agent** within a closed-loop AI-for-AI framework for scalable development and iterative improvement. Mobile planning offers a demanding test of this approach: complex, long-horizon tasks challenge agent reliability, while costly real-device interaction limits development scalability. The framework connects data production, model training, and deployment through a shared action-feedback-verification contract. (i) *AI for Data* builds a human-gated agentic data flywheel in which specialized agents construct tasks, collect interaction trajectories, curate and balance training data, and use training feedback to guide subsequent data generation. (ii) *AI for Training* combines a supervised planning cold start with hybrid-environment online agentic reinforcement learning, where we introduce Competence-Aware Reward-and-Advantage Engineering (CARE) to reduce reasoning and tool-use costs while preserving task performance. (iii) *AI* drives model–harness co-evolution through an execution-evidence-driven loop that orchestrates memory, skills, and tools at runtime and feeds structured action feedback and preserved failure traces back into coordinated model and harness adaptation. Qwen-Planner-Agent achieves the best overall performance among all evaluated models and systems on MobilePA-Bench, improving over its base model across tool use, memory, skills, and sub-agent coordination. Further evaluations of our model show improvements across non-mobile agentic benchmarks while largely preserving general capabilities.

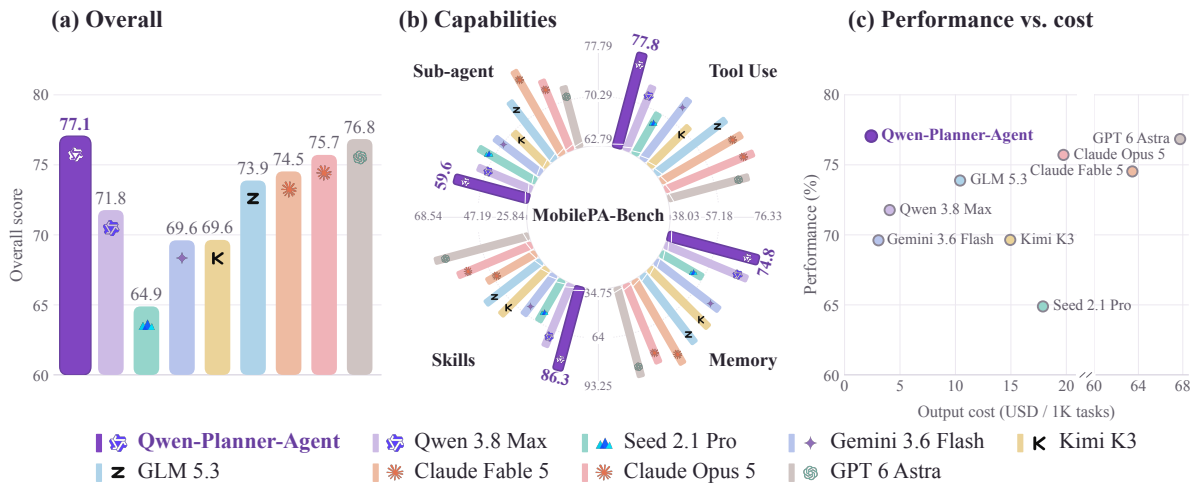

Figure 1: Qwen-Planner-Agent achieves frontier-level Overall performance on MobilePA-Bench (left), with strong tool-use, memory, skill-use, and sub-agent capabilities (center) at a low estimated output cost including thinking tokens (right).

Contents

1. Introduction

> *"Since the design of machines is one of these intellectual activities, an ultraintelligent machine could design even better machines; . . . "*
>
> — I. J. Good, *Speculations Concerning the First Ultraintelligent Machine*, 1965 [1].

Recent advances in language-model agents are extending AI beyond generating individual outputs toward executing multi-step workflows guided by interaction and feedback [2]. These advances make it increasingly practical to pursue a long-standing ambition: using AI to help build and improve AI. As early as 1950, Turing proposed developing machine intelligence by educating a "child machine" and iteratively refining its design through experimentation [3]. Today, language-model agents invite a further step: involving AI not only as the system being developed, but also as a participant in the development process. This raises a central question: how can these capabilities be organized into a scalable, feedback-driven development lifecycle? We study *AI for AI* from this perspective, using execution experience to guide coordinated improvements in data production, model training, and deployment.

We investigate this question by developing a mobile planner agent for complex, real-world tasks. Completing a high-level user goal requires coordinating actions across applications, maintaining context as states change, recovering from failures, and verifying that the intended outcome has been reached. Developing these capabilities calls for diverse interaction data, effective policy learning, and runtime support suited to changing tasks and resources. Yet real-device interaction is costly and difficult to parallelize, constraining the scale of development and evaluation [4, 5]. Task-specific verification provides a complementary opportunity: execution traces and observable outcomes can supply evidence for assessing progress and guiding targeted improvements. Mobile planning therefore provides a concrete setting for studying scalable, AI-assisted agent development.

In this report, we present an *AI-for-AI* framework for developing **Qwen-Planner-Agent**, a complete agent system that couples a trained Planner Model with a unified Harness. Within this framework, AI interprets execution feedback to identify capability gaps and guide coordinated updates to training data, learning strategies, and runtime support. The *AI for Data* stage combines AI-assisted task construction and failure diagnosis with automated trajectory collection and curation. Training and development-set feedback guides task generation and sampling adjustments. The *AI for Training* stage combines a planning-oriented cold start with hybrid-environment online agentic RL. Competence-Aware Reward-and-Advantage Engineering (CARE) adapts rewards and calibrates advantages, with a bounded LLM-based controller configuring predefined reward schedules from training and validation feedback. The *AI for Harness* stage integrates the Planner Model with a Harness that supplies tool-conditioned Skills, persistent Memory, and execution feedback. AI assists memory consolidation, while failure diagnosis guides data updates and LLM-based Harness revisions.

The development process adapts as the agent's capabilities evolve. Diagnosed failures guide new tasks and data sampling, while updated model behavior informs Harness revisions. In turn, revised Harness instructions shape the context and interaction experience used in subsequent model learning. This reciprocal adaptation links model improvement with runtime refinement across development rounds. Model parameters remain fixed during serving; offline updates are validated and versioned, with human review retained for ambiguous and release-critical decisions.

On MobilePA-Bench, Qwen-Planner-Agent 27B achieves the highest Overall score among the evaluated models and agent systems. Its estimated per-task output cost, including thinking tokens, is lower than that of the commercial LLMs in our cost comparison. Beyond mobile planning, our Planner Model, Qwen-Planner-Model, demonstrates broad planning and tool-use capabilities across general agentic

benchmarks. Ablation studies further support the effectiveness of model–Harness co-evolution in both mobile planning and general agentic settings.

In summary, our contributions are threefold:

- **A closed-loop AI-for-AI framework.** We present a practical exploration of AI-for-AI through the development of mobile planner agents. AI turns execution feedback into targeted improvements in training data, learning strategies, and runtime support. By adapting these development decisions to evolving agent capabilities and leveraging scalable hybrid environments, our framework provides a closed-loop approach to building and iteratively improving real-world agents.
- **Qwen-Planner-Agent.** We develop a unified Model–Harness agent system that couples generalizable planning with adaptive runtime support. The Planner Model learns task decomposition, grounded tool use, and failure recovery, while the Harness assembles tool-conditioned Skills, persistent Memory, and execution feedback to accommodate changing resources and user context. AI consolidates execution experience and diagnoses failures to guide model training and Harness refinement. These reviewed updates provide a pathway toward model–Harness co-evolution.
- **Performance, generalization, and efficiency.** Qwen-Planner-Agent achieves the highest Overall score among the evaluated frontier models and agent systems on MobilePA-Bench, demonstrating strong mobile planning and task-completion capabilities at a lower estimated per-task output cost than the commercial LLMs included in our cost comparison. The Planner Model also demonstrates broad competence across general agentic benchmarks, showing that its planning and tool-use capabilities extend beyond mobile environments.

2. Qwen-Planner-Agent

2.1. Overview

2.1.1. AI-for-AI Framework

Figure 2 summarizes the AI-for-AI lifecycle for developing Qwen-Planner-Agent, which comprises a Planner and a Harness. *AI for Data* (Section 2.3) combines AI-assisted task construction with automated interaction collection and curation to supply training tasks and trajectories. *AI for Training* (Section 2.4) learns the planner from these assets through a planning-oriented cold start and competence-adaptive online agentic RL. *AI for Harness* (Section 2.5) equips the planner with a Harness for Skills, Memory, and execution feedback, with AI assisting memory consolidation and Harness refinement.

Within the AI-for-AI lifecycle, intermediate evaluation uses a held-out development set rather than the final benchmark test set. Development tasks and their trajectories are excluded from direct training, while their evaluation results and diagnosed failure patterns guide subsequent data generation, training adjustments, and Harness refinement.

Training and development-set results, together with deployment traces, feed AI-assisted diagnosis that guides targeted tasks, data reweighting, and Harness revisions. Revised Harness instructions shape the context and trajectories used for subsequent model training, while updated model behavior informs further Harness refinement. This feedback connects the three stages across development rounds. Model and Harness updates are reviewed and versioned offline; model parameters remain fixed during serving.

2.1.2. Task Formulation

Given a user request x, Qwen-Planner-Agent interacts with a partially observed environment initialized at state s_0. The underlying state s_t includes the environment and execution-relevant runtime state and is

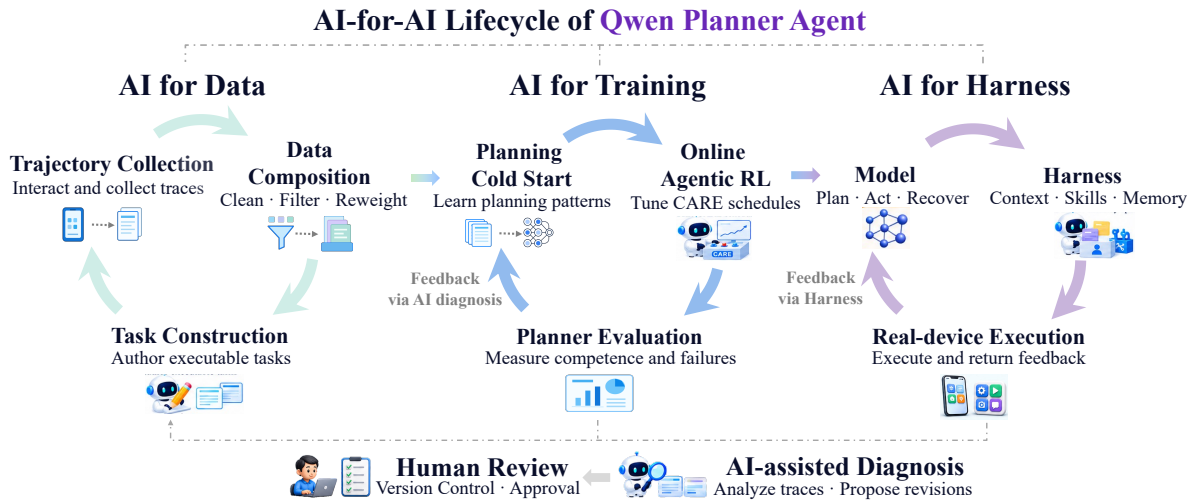

Figure 2: AI-for-AI lifecycle of Qwen-Planner-Agent, comprising three interconnected phases. (i) AI for Data combines AI-assisted task generation with automated trajectory collection and curation, using model feedback to refine subsequent tasks and training-data composition. (ii) AI for Training combines planning-oriented cold-start training with hybrid-environment online reinforcement learning, while CARE adapts reward scheduling and advantage signals to the model's evolving competence. (iii) AI for Harness equips the planner with a unified Harness that manages Skills, persistent Memory, and execution feedback, while AI-assisted diagnosis guides subsequent model and Harness refinement and feeds new requirements back into data production and training.

not directly exposed to the policy. Instead, the policy receives observations o_t through the environment interface. At step t, the Harness combines the action–observation history with retrieved memory m_t and loaded skills $\mathcal{K}_t$ to construct the model context. The policy selects an action from the currently available structured action set $\mathcal{A}_t$:

$$c_t = \mathcal{H}_\eta(x, m_t, \mathcal{K}_t, o_{\le t}, a_{<t}), \quad a_t \sim \pi_\theta(\cdot \mid c_t, \mathcal{A}_t). \tag{1}$$

Here $\mathcal{H}_\eta(\cdot)$ denotes the Harness context-construction function under instruction configuration η, c_t the resulting model context, and π_θ the policy parameterized by θ. The environment executes a_t, transitions to the next underlying state s_{t+1}, and returns the next observation o_{t+1}:

$$s_{t+1} \sim P_{\text{trans}}(\cdot \mid s_t, a_t), \quad o_{t+1} \sim P_{\text{obs}}(\cdot \mid s_{t+1}, a_t). \tag{2}$$

$P_{\text{trans}}(\cdot)$ models environment dynamics, while $P_{\text{obs}}(\cdot)$ models the structured feedback exposed to the policy, such as tool results, observable state changes, or execution errors. This formulation accommodates both deterministic and stochastic backends.

A task-specific verifier evaluates completion using the interaction history and available execution evidence:

$$v_{t+1} = \mathcal{V}(x, \xi_{t+1}, \tau_{\le t+1}), \quad \text{where } \tau_{\le t+1} = (o_{\le t+1}, a_{\le t}). \tag{3}$$

The verifier $\mathcal{V}(\cdot)$ produces the verification outcome v_{t+1} for request x from the interaction history $\tau_{\le t+1}$ and available execution evidence ξ_{t+1}. The evidence ξ_{t+1} includes relevant initial conditions and backend state records available to the verifier; this evidence need not be exposed to the policy. If execution continues, the returned observations enter the next model context. Interaction terminates when the verifier confirms task completion, the agent requests clarification or refuses an unsafe request, or the execution budget is exhausted. The action space covers typed tool calls, memory access and updates, skill selection

and loading, clarification or refusal, and task-completion declarations. The agent primarily acts through structured tools rather than pixel-coordinate GUI actions, although tools may return visual observations when needed. Backends may differ in their internal state representations and execution mechanisms while exposing compatible task, action, observation, and verification records.

2.2. Hybrid Mobile Environment Infrastructure

Mobile-agent development requires scalable interaction as well as feedback that reflects real execution. We combine programmatic sandboxes, LLM-simulated environments, and selected real-device sessions to support the task interactions formulated in Section 2.1.2. The infrastructure organizes these backends into shared data-collection, evaluation, and online-training workflows.

2.2.1. Complementary Environment Backends

The three backends trade off scalability, scenario coverage, and execution fidelity, making each suitable for different task requirements.

Programmatic sandbox environments execute typed tool calls through predefined program logic over structured application databases. Deterministic transitions, task-specific resets, and state-based verification support reproducible, high-throughput interaction. Their coverage is limited to implemented tools and state transitions, so new applications or exceptional behavior require additional engineering. They are therefore most suitable for repeatable tasks with explicit state and completion conditions.

LLM-simulated environments use language models to generate environment responses for long-tail interactions that are difficult to implement with fixed logic. They broaden scenario coverage without requiring a dedicated programmatic implementation for every case. However, responses may be inconsistent with prior actions or environment state, so trajectories require task-specific validation before use.

Real-device environments execute actions in live device sessions, capturing the effects of OS permissions, authenticated services, cross-application dependencies, and runtime changes. They provide direct evidence of behavior that simulation may miss, but incur higher interaction costs, limited parallelism, and difficult resets. They are therefore valuable for tasks whose completion depends on actual device or service behavior.

2.2.2. Hybrid Environment Strategy

We match tasks to backends according to their execution requirements. Tasks with well-defined, reproducible state changes primarily use programmatic sandboxes, while LLM-simulated environments supplement long-tail interactions without suitable fixed implementations. Real-device sessions are used selectively for tasks dependent on live device or service behavior. This allocation combines scalable simulated interaction with prioritized device access where simulation lacks sufficient execution fidelity.

A common agent-facing interface makes these experiences usable within the same data and training workflows. Tasks expose typed actions, structured responses, and task-specific completion criteria; their interaction records retain execution errors, observable state changes, and verification outcomes. Backend implementations and internal states remain distinct. Compatible records enter shared curation and rollout-processing pipelines, without assuming that simulated and real-device feedback have identical reliability.

2.2.3. Training Infrastructure

Our infrastructure separates model-side training from environment-side execution. The training layer coordinates distributed rollout generation, model updates, and training-resource scheduling. A client–server environment-management layer creates, schedules, and cleans up environment instances and real-device sessions, while the corresponding backends implement resets and state handling. This separation allows model computation and environment capacity to be managed independently.

During an online rollout, a policy worker interacts with a backend through the environment-management layer and receives execution feedback after each action. Task verifiers assess completion, while environment validation and trajectory checks determine which records are admitted to training. The training layer uses the admitted trajectories for policy updates. Task failure is distinct from an invalid execution record: unsuccessful interactions can still provide learning and diagnostic evidence. Detailed data-curation rules are described in Section 2.3.

The environment-management layer also supports data collection and evaluation. It remains distinct from the deployment-time Harness, which assembles the Planner Model's context from Skills, Memory, tools, and execution feedback. Further training and environment-management details are provided in Appendix B.

2.3. AI for Data: An Agent-Driven Data Flywheel

The data flywheel in Figure 3 turns capability requirements into executable tasks, collects interaction trajectories, and constructs training datasets that evolve with planner performance. AI assists task construction, failure diagnosis, and targeted data refinement, while automated workflows handle rollout collection and data processing. Training and development-set feedback informs which tasks to generate and how to adjust sampling in the next iteration. The resulting assets support both planning-oriented cold-start training and online reinforcement learning.

2.3.1. Task Construction

Task-construction agents translate target capabilities into executable task specifications. Initial objectives come from product requirements, available tool inventories, and representative user scenarios; later iterations also incorporate diagnosed capability gaps, as described in Section 2.3.4. Each specification contains a user goal, available resources, relevant initial conditions, target capabilities, and completion criteria. It defines what must be accomplished without prescribing a single reference trajectory, allowing different valid plans to satisfy the same objective.

Construction jointly considers *scenario coverage* and *capability coverage*. Scenario coverage spans application domains, tools, user intents, and interaction patterns. Capability coverage targets information acquisition, tool routing, argument grounding, multi-step dependency handling, state tracking, recovery, and verified task completion. This distinction helps introduce new behavioral requirements rather than merely adding more instances of familiar scenarios.

2.3.2. Interaction Trajectory Collection

Tasks are executed in backends matched to their interaction requirements. Mobile tasks follow the hybrid strategy in Section 2.2.2; general-agent tasks use executable service environments, while coding tasks use repository and code-execution environments.

An automated rollout workflow runs agent policies through multi-step environment interaction and records their trajectories. Each record preserves selected actions, environment feedback, intermediate outcomes,

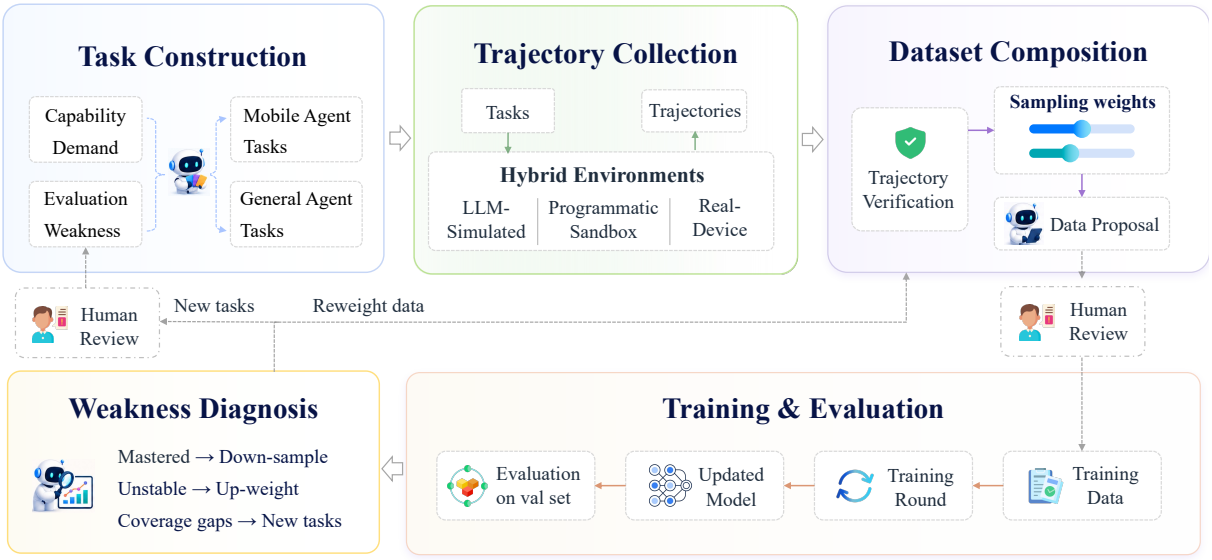

Figure 3: **AI-for-Data workflow.** AI-assisted task construction supplies executable tasks for automated interaction collection and curation. Verified trajectories and resettable tasks support planning-oriented cold-start training and online RL, respectively. AI-assisted analysis of rollout and development-set feedback guides targeted task generation and sampling adjustments for the next iteration, with human review before each data release. Solid blue arrows denote automated within-iteration flow; dashed orange arrows denote the human-gated transition between iterations.

execution errors, retries, recovery attempts, completion evidence, and the final outcome. Successful trajectories provide candidate supervision for planning, tool use, and task closure. Failed and incomplete trajectories are also retained so that diagnosis can examine where execution diverged, whether recovery was attempted, and why the task remained unfinished. Collection therefore preserves the process leading to an outcome, not only its final success label.

2.3.3. Training Dataset Composition

Task-completion verification and data-quality assessment are separate decisions. Environment verifiers determine whether completion criteria have been met, while the curation workflow checks whether the corresponding record is suitable for learning. Automated processing normalizes heterogeneous trajectories, removes malformed or duplicate examples, and checks schema consistency and support for reported outcomes in the execution evidence. Retained records are tagged by capability and linked to failure attributions from rollout analysis. Successful completion alone does not establish training suitability, and unsuccessful interactions may still provide useful diagnostic evidence.

Low-confidence, conflicting, safety-sensitive, or insufficiently supported cases are routed to human reviewers, who may approve, correct, reject, or quarantine them. Retained records preserve their task source, execution backend, generating policy, capability annotations, outcome, and failure attribution, making later sampling and repair decisions traceable.

The curated data are organized into mobile and non-mobile groups. Mobile data form the core of training and cover cross-application planning, tool execution, state tracking, Memory, Skills, sub-agent coordination, recovery, and task completion. Non-mobile data include general-agent trajectories, coding tasks, and reasoning and instruction-following examples. General-agent trajectories exercise structured tool use and multi-turn planning over services such as Model Context Protocol (MCP) servers [6]; coding tasks add repository understanding, iterative editing, and testing. These examples provide complementary

supervision for multi-step problem solving while helping preserve general reasoning and instruction following.

The pipeline maintains two training assets: verified interaction trajectories for planning-oriented cold-start training and resettable task instances with reliable completion criteria for online reinforcement learning. These support learning from recorded demonstrations and newly generated interactions, respectively. We configure sampling weights across mobile and non-mobile data, capabilities, difficulty levels, and sources rather than sampling in proportion to raw corpus size. These weights are revised using the feedback described below, keeping mobile planning as the primary objective while retaining complementary non-mobile supervision.

2.3.4. Feedback-Driven Data Refinement

As the planner improves, the value of individual tasks changes: mastered tasks may become redundant, whereas unstable behaviors and uncovered capabilities require additional training. AI-assisted analysis of task-level rollouts and capability-level development-set results informs three types of updates: reducing redundant examples, increasing coverage of unstable behaviors, and constructing tasks for missing capabilities.

At the task level, rollout-analysis agents examine completion outcomes and failure patterns. Reliably mastered tasks are down-sampled while retaining a small preservation set, and inconsistently completed tasks receive greater weight as stabilization data. Failures involving tool routing, argument grounding, state tracking, recovery, or premature termination are grouped into targeted repair sets. Ambiguous or weakly supported cases return to curation rather than entering the training pool as ordinary examples.

At the capability level, analysis agents combine held-out development-set results for Tool Use, Memory, Skills, and Sub-agent with trace-derived failure distributions to distinguish recurring gaps from isolated errors. If the existing pool contains suitable examples, their sampling proportions are adjusted. If coverage is insufficient, task-construction agents generate new tasks with targeted capability requirements, difficulty levels, tool combinations, or interaction patterns. Feedback thus determines both which data to select and which tasks to construct next. Development-set failure patterns guide the construction of new training tasks; the development tasks and their trajectories are not added to the training pool.

Proposed task additions and removals, sampling adjustments, repair and quarantine decisions, and quality indicators are recorded together for review. Human reviewers may approve, revise, or reject these changes; only an approved dataset and sampling configuration are frozen for the next training round. Each release preserves a versioned history linking diagnosed capability gaps to the data updates made in response. The feedback loop thus changes both the content and composition of subsequent training data, rather than simply expanding the corpus.

2.4. AI for Training: Competence-Adaptive Agentic Optimization

The training pipeline begins with a planning-oriented cold start that converts curated data from the AI-for-Data pipeline into an initial policy following the shared action–feedback–verification contract. Starting from this policy, we perform hybrid-environment online agentic reinforcement learning across programmatic sandboxes, LLM-simulated environments, and selected real-device sessions. Within the RL stage, we propose Competence-Aware Reward-and-Advantage Engineering (CARE), which adapts reward composition and advantage scaling to measured group competence to improve reasoning and tool-use efficiency while preserving task quality. Figure 4 summarizes this process.

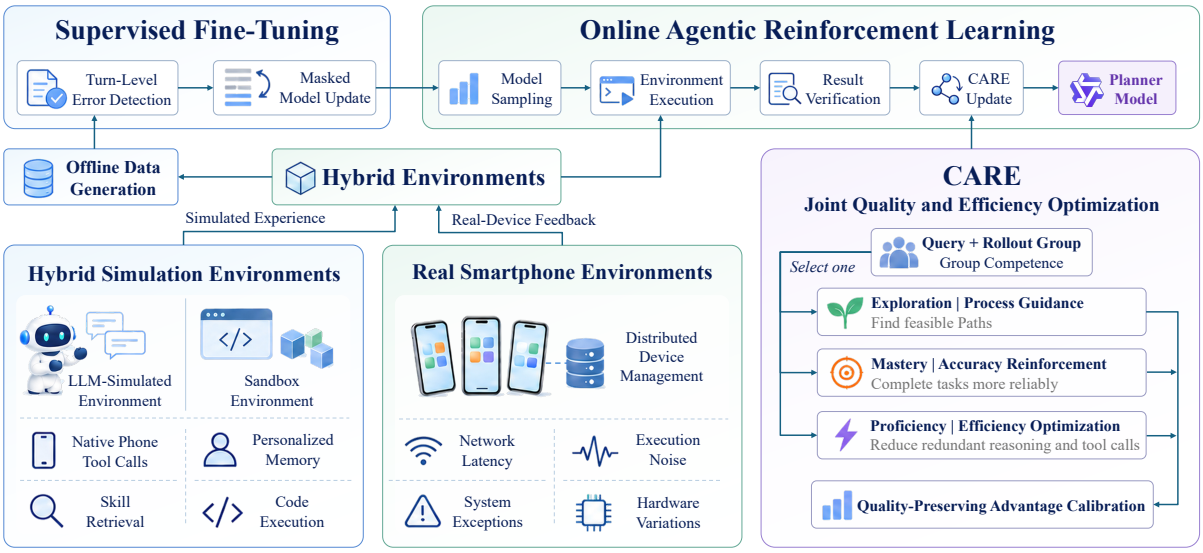

Figure 4: **Training framework for the Planner Model in Qwen-Planner-Agent.** Supervised fine-tuning with turn-level error detection and loss masking establishes a planning-oriented cold-start policy, followed by online agentic reinforcement learning in hybrid environments combining LLM simulation, programmatic sandboxes, and selected real-device interaction. Within the RL stage, CARE (Competence-Aware Reward-and-Advantage Engineering) selects process guidance, accuracy reinforcement, or efficiency optimization according to each trajectory group's competence. Quality-preserving advantage calibration limits the amplification of small efficiency differences, promoting efficient reasoning and tool use while retaining verified task success as the primary objective.

2.4.1. Planning-Oriented Cold Start

We initialize the policy through supervised fine-tuning on curated data from the AI-for-Data pipeline, learning task decomposition, action sequencing, tool use, state-conditioned replanning, and recovery under the shared action–feedback–verification contract. Training prioritizes verified mobile trajectories and includes non-mobile data for broader agentic capabilities, general reasoning, and instruction following, with controlled sampling weights across data groups. After sample-level filtering, we mask model-action turns identified as erroneous during trajectory curation. For each trajectory τ_i in a training batch, let $\mathcal{E}_i$ denote the set of erroneous turn indices and $\boldsymbol{y}_{i,j}$ the token sequence generated in turn j. We minimize the masked cross-entropy objective

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -Z^{-1} \sum_{i,j} \mathbf{1}\{j \notin \mathcal{E}_i\} \log \pi_\theta(\boldsymbol{y}_{i,j} \mid h_{i,j}), \tag{4}$$

where $h_{i,j}$ is the conditioning context preceding turn j and Z is the total number of unmasked model-generated tokens in the batch. Execution feedback is retained in the context, while verified recovery turns remain supervised.

2.4.2. Hybrid-Environment Online Agentic Reinforcement Learning

Starting from the cold-start policy, we use online agentic reinforcement learning to strengthen long-horizon planning, state tracking, and failure recovery. Curated demonstrations establish useful execution patterns but cannot cover all interaction states and failure modes encountered as the policy evolves. Online rollouts expose the planner to this policy-dependent distribution, with verifier feedback guiding improvements in task completion and robustness.

2.4.3. CARE: Competence-Aware Reward-and-Advantage Engineering

CARE adapts reward composition and advantage scaling to group competence to promote reliable task completion and efficient execution. Its central principle is to emphasize reasoning and tool-use efficiency as task success becomes reliable, while retaining verified success as the primary learning objective. A fixed reward design is poorly matched to groups at different competence levels: groups with sparse success benefit from verified progress signals, groups with mixed outcomes should consolidate task completion, and groups whose success has saturated can shift attention toward execution efficiency. Moreover, standard within-group normalization can over-amplify small efficiency differences once success has saturated. CARE addresses these two issues through competence-adaptive reward scheduling and quality-preserving advantage calibration. A bounded LLM-based controller periodically configures the predefined scheduling parameters from recent training statistics and held-out development-set feedback.

For each task, we sample a group of G trajectories, $\mathcal{G} = \{\tau_i\}_{i=1}^{G}$ using a rollout behavior policy. A task-specific rule-based verifier or rubric-guided generative reward model assigns each trajectory a binary success indicator $s_i \in \{0, 1\}$. The group success rate $\bar{s} = \frac{1}{G}\sum_i s_i$ serves as the competence signal for reward scheduling and advantage calibration.

Competence-Adaptive Reward Scheduling CARE defines three competence-dependent reward regimes: *progress shaping*, *outcome consolidation*, and *efficiency refinement*. Each trajectory group is assigned to a regime according to its current success rate $\bar{s}$, allowing different regimes to coexist within the same training batch.

$$R_i = s_i + \begin{cases} \lambda_{\text{prog}} R_{\text{prog},i}, & \bar{s} \in [0, p_{\text{low}}), \quad \text{progress shaping} \\ 0, & \bar{s} \in [p_{\text{low}}, p_{\text{high}}), \quad \text{outcome consolidation} \\ -\lambda_{\text{eff}} e_i, & \bar{s} \in [p_{\text{high}}, 1], \quad \text{efficiency refinement} \end{cases} \tag{5}$$

Here $R_{\text{prog},i}$ measures verified progress, and e_i is a normalized execution-cost penalty. The group-success thresholds satisfy $0 \le p_{\text{low}} < p_{\text{high}} < 1$, and the reward weights λ_{prog} and λ_{eff} are nonnegative.

AI-Assisted Schedule Configuration The competence regimes determine the auxiliary reward for each trajectory group, while their thresholds and reward weights can be adjusted as training evolves. An LLM-based controller configures reward-scheduling parameters $\boldsymbol{\phi} = \{\lambda_{\text{prog}}, \lambda_{\text{eff}}, p_{\text{low}}, p_{\text{high}}\}$ every N policy-update steps. Recent on-policy training statistics characterize the optimization state, while held-out development-set performance serves as the primary signal for selecting the configuration used over the next N steps:

$$\boldsymbol{\phi}_{t+1:t+N} = f_{\text{LLM}}\left(\mathcal{P}, \mathcal{S}^{\text{train}}_{\le t}, \mathcal{M}^{\text{val}}_{\le t}\right), \quad t \in \{0, N, 2N, \ldots\} \tag{6}$$

Here $\mathcal{P}$ specifies the optimization objective, parameter semantics, admissible ranges, and output format. $\mathcal{S}^{\text{train}}$ summarizes group success, progress, and efficiency statistics, and $\mathcal{M}^{\text{val}}$ records success and execution quality on the held-out development set. Each configuration is held fixed for the subsequent N policy updates and applied consistently in reward scheduling and advantage calibration; individual groups continue to select their competence regime according to their current success rate.

Quality-Preserving Advantage Calibration Reward scheduling controls reward composition, but within-group normalization can counteract the intended attenuation of efficiency-only learning signals. Standard GRPO-style [7] advantage computation centers rewards and divides by their within-group standard deviation, bringing groups with nonzero reward variation to approximately unit advantage scale. Once success saturates, small efficiency differences can therefore produce advantages comparable

in scale to those of groups with mixed success outcomes. To examine this effect, we consider the efficiency-refinement regime, where $R_i = s_i - \lambda_{\text{eff}} e_i$. Let $\overline{R} = \frac{1}{G}\sum_j R_j$ and $\overline{e} = \frac{1}{G}\sum_j e_j$. The standardized advantage and reward variance are

$$\begin{aligned} \widehat{A}_i^{\text{std}} &= \frac{R_i - \overline{R}}{\sigma_R + \epsilon} = \frac{(s_i - \overline{s}) - \lambda_{\text{eff}}(e_i - \overline{e})}{\sigma_R + \epsilon} \\ \text{where} \quad \sigma_R^2 &= \overline{s}(1-\overline{s}) + \lambda_{\text{eff}}^2 \sigma_e^2 - 2\lambda_{\text{eff}} \operatorname{Cov}(s, e) \end{aligned} \tag{7}$$

For a fully successful group ($s_i = 1$ for all i), the success variance and its covariance with e are zero. For $\lambda_{\text{eff}} > 0$ and $\sigma_e > 0$, the expressions simplify to

$$\begin{aligned} \lim_{\overline{s}\to 1}(R_i - \overline{R}) &= -\lambda_{\text{eff}}(e_i - \overline{e}), \quad \lim_{\overline{s}\to 1} \sigma_R = \lambda_{\text{eff}}\sigma_e \\ \lim_{\overline{s}\to 1} \widehat{A}_i^{\text{std}} &= -\frac{\lambda_{\text{eff}}(e_i - \overline{e})}{\lambda_{\text{eff}}\sigma_e + \epsilon} \approx -\frac{e_i - \overline{e}}{\sigma_e} \end{aligned} \tag{8}$$

When ϵ is negligible relative to $\lambda_{\text{eff}}\sigma_e$, the efficiency coefficient approximately cancels under normalization. Even with a small efficiency weight (e.g., $\lambda_{\text{eff}} = 0.1$), advantages in success-saturated groups therefore remain approximately unit scale. This gives efficiency-only groups an advantage scale comparable to that of groups with mixed success outcomes, weakening the intended emphasis on task success and potentially encouraging excessive trajectory compression.

An alternative is to normalize reward components separately before aggregation, as in GDPO [8]. However, scaling each component by its own within-group standard deviation changes its effective weight in the original reward space and can alter trajectory rankings under the composed reward. We instead preserve the composed reward and calibrate its group-level advantage scale by imposing a success-derived floor on the normalization denominator:

$$\sigma_{\text{anchor}} = \sqrt{p_{\text{high}}(1 - p_{\text{high}})}$$

$$\widehat{A}_i = \begin{cases} \dfrac{R_i - \overline{R}}{\sigma_R + \epsilon}, & \text{progress or outcome regime,} \\ \dfrac{R_i - \overline{R}}{\max(\sigma_R, \sigma_{\text{anchor}}) + \epsilon}, & \text{efficiency-refinement regime} \end{cases} \tag{9}$$

Calibration uses the same efficiency-refinement threshold p_{high} as reward scheduling, with the anchor set to the corresponding binary-success standard deviation. For fully successful groups where the floor is active, the within-group standard deviation of the calibrated advantages is $\lambda_{\text{eff}}\sigma_e/(\sigma_{\text{anchor}} + \epsilon)$, preserving its dependence on the efficiency weight. The shared denominator retains the relative weighting of reward components while limiting the amplification of small efficiency differences.

Given the calibrated advantages, we maximize the following clipped group-relative surrogate objective [9, 10]:

$$\mathcal{J}_{\text{RL}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{T_i}\sum_{t=1}^{T_i} \min\left(\rho_{i,t}(\theta)\widehat{A}_i, \operatorname{clip}\left(\rho_{i,t}(\theta), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}\right)\widehat{A}_i\right)\right] \tag{10}$$

where θ and θ_{old} denote the current and rollout behavior policy parameters, respectively; $y_{i,t}$ is the t-th generated token in trajectory τ_i; $h_{i,t}$ is the interaction history preceding it; $\rho_{i,t}(\theta) = \pi_\theta(y_{i,t} \mid h_{i,t})/\pi_{\theta_{\text{old}}}(y_{i,t} \mid h_{i,t})$; T_i is the number of generated tokens in τ_i; and ϵ_{low} and ϵ_{high} denote the lower and upper clipping ranges.

2.5. AI for Harness: Toward Model–Harness Co-Evolution

Deployment is not the endpoint of model development, but a continuing source of executable evidence. A trained Planner Model can acquire generalizable capabilities in task decomposition, tool selection,

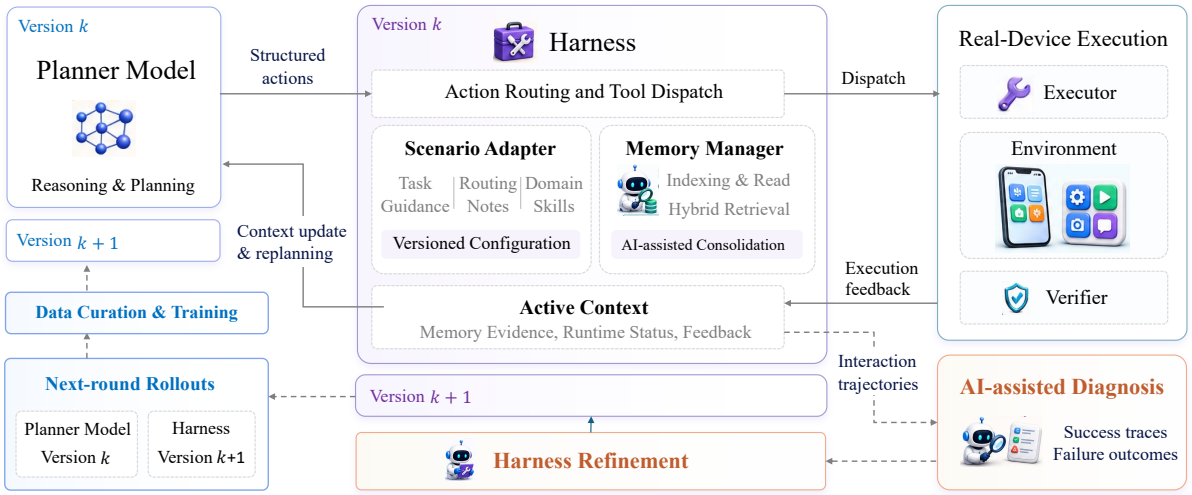

Figure 5: **Model–Harness Co-Evolution.** The Harness connects the Planner Model to deployment resources and preserves execution traces and failure evidence. Offline diagnosis of this evidence can inform subsequent data, model, and Harness refinement under a governed development process.

argument grounding, state tracking, and failure recovery. However, deployment-time tool inventories, operating rules, user information, and interaction histories continue to change. Encoding every change in model parameters would require repeated retraining, whereas placing all possible rules and historical records in the prompt would create long, noisy contexts.

We therefore place a unified *Harness* between the Planner Model and its deployment environment. At request time, the Harness connects the model to external executors, assembles an active context from Tools, Skills, persistent Memory, and runtime constraints, and relays structured feedback after each action. Across development cycles, it preserves deployment evidence for AI-assisted trajectory analysis, failure diagnosis, and candidate refinement of both the model and the Harness. AI for Harness thus operates at two timescales: online, the Harness adapts the model context to the current request; offline, accumulated evidence informs subsequent model and Harness updates. These updates remain reviewed and versioned rather than being applied autonomously during serving.

Agent improvement involves two coupled forms of adaptation: updating the policy and adapting the execution scaffold under which it operates. We develop a model–Harness co-evolution framework that connects these channels through an iterative feedback loop. The Harness plays a dual role: it shapes the agent's current execution behavior and the experience distribution underlying subsequent model training. Conversely, model updates change the behaviors and limitations that Harness adaptation must address. Each component therefore changes the conditions under which the other is optimized.

Let η denote the editable instruction configuration of the Harness introduced in Section 2.1, comprising general and domain-specific instructions. The policy π_θ and the configured context-construction function $\mathcal{H}_\eta$ induce the trajectory distribution $p_{\theta,\eta}(\tau \mid x)$ through environment interaction on task x. We seek to maximize the joint objective

$$J(\theta, \eta) = \mathbb{E}_{\substack{x \sim \mathcal{D} \\ \tau \sim p_{\theta,\eta}(\cdot|x)}} \left[R_x(\tau) \right], \tag{11}$$

where $\mathcal{D}$ spans heterogeneous task environments and R_x represents task-dependent feedback. Resource budgets and task-specific interaction requirements define the practical operating conditions of this optimization.

Our current implementation pursues this objective through alternating model updates and feedback-driven

Harness revisions:

$$\begin{aligned}\theta^{(k+1)} &= \mathcal{U}_M(\theta^{(k)}; \mathcal{B}^{(k)}, \eta^{(k)}),\\ \eta^{(k+1)} &= \mathcal{U}_H\Big(\eta^{(k)}; \mathcal{F}(\theta^{(k+1)}, \eta^{(k)})\Big).\end{aligned} \tag{12}$$

Here, k indexes co-evolution rounds, and $\mathcal{U}_M$ performs reinforcement learning using trajectories $\mathcal{B}^{(k)}$ generated under the fixed Harness instruction configuration $\eta^{(k)}$. The updated model is then evaluated on the held-out development set with the same configuration to obtain development-set feedback $\mathcal{F}(\theta^{(k+1)}, \eta^{(k)})$, which an LLM-based editor $\mathcal{U}_H$ uses to revise the execution instructions. The revised configuration $\eta^{(k+1)}$ is applied through $\mathcal{H}_{\eta^{(k+1)}}$ in the next training round, closing the loop between parameter learning and contextual adaptation. This coupling makes Harness revision consequential beyond its immediate execution effects: it also reshapes the experience from which the model subsequently learns. The framework thus provides a basis for examining how changes in execution behavior translate into learning and system-level progress over successive rounds.

2.5.1. Unified Model–Harness Runtime

The Harness defines the runtime boundary among the Planner Model, external resources, and the execution environment. Given a user request, it assembles the active context from conversation history, currently exposed tool specifications, operational guidance supplied by the Scenario Adapter, evidence retrieved by the Persistent Memory Manager, and applicable runtime constraints. The Planner Model then produces a structured action. A separate Executor applies the action to the application or device and returns a tool result, an observable state change, or an execution error. This feedback enters the next planning turn unless completion is verified, user clarification is required, or a safety or execution-budget condition stops the interaction.

The components have distinct responsibilities. The Planner Model interprets the request, decomposes the task, resolves conflicts, selects actions, fills arguments, and replans after feedback. The Scenario Adapter conditions procedural guidance on the tools available for the current request, whereas the Persistent Memory Manager stores, governs, and retrieves evidence across interactions. Actions that modify the target application or device state are carried out by the Executor. The Harness coordinates these interfaces and retains their joint trace without taking over the Planner Model's decision-making role. This separation keeps generalizable planning behavior in the model while allowing deployment-specific information to change outside its parameters.

2.5.2. Context Management with Skills and Memory

Skills and Memory provide complementary forms of deployment-time context. Skills describe *how* available resources should be used, including tool dependencies, operating procedures, and multi-turn interaction requirements. Memory provides evidence about *what* is already known or has previously occurred, including user preferences, past events, established decisions, and pending intentions. The Harness assembles tool-conditioned Skill guidance and retrieved Memory with tool specifications, interaction history, and runtime constraints. When explicit Skill-selection tools are exposed, the Planner Model selects and loads Skills through the structured action interface.

Scenario-conditioned Skill guidance. The Scenario Adapter handles variation in deployment-time tool inventories without modifying the Planner Model's parameters. Offline, it compiles tool schemas, a complete tool-to-Skill mapping, capability-domain Skills, routing notes, and guidance constraints into a versioned configuration. At request time, it treats the tools already exposed to the model as the candidate set, selects the corresponding Skill guidance, removes rules that depend exclusively on unavailable resources, and adds applicable routing and multi-turn instructions. The resulting guidance is incorporated

into the active context. The Scenario Adapter neither discovers additional tools nor executes actions; it provides procedural knowledge matched to resources already available to the Planner Model.

Evidence-grounded Persistent Memory. The Persistent Memory Manager handles information that must remain available beyond the active conversation context. Persistent state is organized by function rather than stored as an undifferentiated interaction history. A compact user model records stable preferences and behavioral requirements; episodic Memory preserves detailed observations and dialogue evidence; curated long-term Memory stores consolidated facts, decisions, and reusable lessons; and prospective Memory represents future intentions with explicit activation conditions, scope, expiry, and completion status. These Memory types have different access and trust boundaries, preventing all historical content from being treated as equally reliable or relevant.

Memory follows a controlled lifecycle of *capture*, *validation*, *indexing*, *retrieval*, *consolidation*, and *revision*. Each Memory unit links a concise structured summary to source metadata and original evidence, supporting efficient retrieval without sacrificing traceability. Bounded AI-assisted processing helps summarize long trajectories, merge duplicate observations, identify contradictions, and formulate candidate updates. These candidates do not automatically become authoritative Memory: provenance, trust, recency, and conflict checks precede consolidation. When new evidence changes an existing fact, the revised entry explicitly supersedes the obsolete record rather than leaving competing versions unresolved.

At inference time, the Persistent Memory Manager retrieves a compact evidence set for the current task. Retrieval combines semantic and keyword matching with relevance, recency, importance, and diversity controls. Evidence assembly also reflects the structure of the request: temporal questions require event ordering, cross-session aggregation requires entity and action deduplication, and questions about changing facts require explicit resolution of superseding evidence. Retrieved Memory is supplied to the Planner Model as evidence rather than as an executable instruction, and it does not acquire permission to invoke tools or alter external state.

The Scenario Adapter and Persistent Memory Manager therefore serve the same context-orchestration process rather than forming separate runtime systems. The former contributes procedural guidance matched to the current resources, while the latter contributes historical evidence matched to the current task. Together, they preserve a shared Planner Model across heterogeneous deployment scenarios while allowing dynamic information to be updated outside model parameters. In the experiments, we distinguish only between configurations with and without the unified Harness; persistent Memory remains an internal capability of that Harness, not a separately named runtime system. Further implementation and evaluation details are provided in Appendix A.

2.5.3. Privacy-Preserving Memory Safety

Protecting user privacy requires security controls across the entire memory lifecycle, including collection, extraction, storage, retrieval, sharing, and deletion. The system separates private user memories from shared agent experiences. User profiles, preferences, and personal facts are restricted to the corresponding workspace, user, and agent, while reusable experiences are shared only within the designated workspace and agent. Before entering shared memory, these experiences undergo processing to reduce user-specific and sensitive details. Each extracted memory is linked to its source events for traceability, and raw processing data is cleared after task completion to limit unnecessary retention. Context files are generated from stored memories, with version checks and coordinated updates helping prevent outdated or deleted information from being reused.

Access control determines who can read, create, modify, delete, or share memories. The current implementation uses service-level authentication tokens and separate administrator credentials, while user-

level access enforcement depends on trusted upstream services. For multi-user deployment, authorization should additionally verify the caller's identity, the owner of the requested memory, and the permitted operation. Role-based access control and user-specific permission rules should enforce least privilege, with access denied unless explicitly authorized. These checks should apply to both direct memory access and semantic retrieval, ensuring that search results contain only information the requesting user is allowed to access.

Users should have clear control over what is remembered, how long it is retained, and whether it can be reused across sessions or contribute to shared experiences. The system should support memory inspection, correction, deletion, and withdrawal of permission, with changes reflected in stored records, generated context files, and relevant caches. Removing personal details from shared experiences reduces privacy risks but does not guarantee anonymity. Additional safeguards, including sensitive-information detection, encryption during transmission and storage, protected audit logs, and explicit backup retention policies, should support accountable memory use while minimizing the collection and retention of personal information.

2.5.4. Model and Harness Refinement from Execution Feedback

Each runtime interaction produces a structured deployment trace that links the active Tool and Skill configuration, retrieved Memory and its provenance, the action and arguments selected by the Planner Model, the Executor response, observable state evidence, execution errors, and the final Verifier outcome. Preserving these relationships makes it possible to inspect how both model decisions and Harness configuration contribute to task outcomes rather than retaining only the final response.

The accumulated deployment traces and development-set feedback support AI-assisted offline diagnosis along two complementary paths. Model-side failures include incorrect task decomposition, route selection, argument grounding, state tracking, error recovery, and task closure. Harness-side failures include missing or conflicting Skill guidance, inappropriate tool exposure, retrieval errors, stale Memory, and incomplete feedback formatting.

The resulting diagnoses enter different update paths. Model-side evidence can guide subsequent task construction, data repair, and optimization objectives. Harness-side evidence can inform candidate revisions to tool-to-Skill mappings, procedural guidance, Memory policies, resource exposure, retrieval mechanisms, and feedback interfaces. Candidate changes remain subject to rule-based checks and version control, while low-confidence, conflicting, or release-critical decisions are routed for human review. No candidate update is applied autonomously to a serving system.

Updates to the model alter the planning behavior observed by the Harness, while updates to the Harness alter the context, resources, and action space encountered by the model. These coupled effects define a governed pathway for model–Harness co-adaptation and, over repeated validated development cycles, toward model–Harness co-evolution. We present this mechanism as a development pathway rather than claiming that sustained autonomous co-evolution has already been established.

3. Experiments

We evaluate the complete Qwen-Planner-Agent system and its Planner Model from three perspectives: mobile-planning performance and output cost, agentic capabilities beyond mobile tasks, and the contributions of training and Harness support. We develop two Planner Models based on Qwen backbones, with model sizes of 35B-A3B and 27B. In our experiments, Qwen-Planner-Model denotes the trained planner without our deployment-time Harness, while Qwen-Planner-Agent denotes the complete system with Harness support.

Table 1: **Cross-Model and System Comparison on MobilePA-Bench.** Qwen-Planner-Model reports the trained planner checkpoint without the Harness; Qwen-Planner-Agent includes deployment-time context, Memory, Skill, and feedback support. Bold denotes the highest observed score in each column.

Model / System	Access / Size	Overall	Tool Use	Memory	Skills	Sub-agent
Closed-Source Models						
GPT 6 Astra	Closed-Source	76.84	75.71	74.73	**93.25**	53.93
Claude Opus 5	Closed-Source	75.71	77.60	71.81	83.00	59.55
Claude Fable 5	Closed-Source	74.53	77.21	**76.33**	69.00	**68.54**
Gemini 3.6 Flash	Closed-Source	69.61	74.71	69.68	65.75	51.69
Gemini 3.1 Pro	Closed-Source	68.16	75.38	65.69	55.75	61.80
Claude Opus 4.8	Closed-Source	66.05	72.40	56.38	69.25	47.19
Seed 2.1 Pro	Closed-Source	64.89	70.29	57.18	64.00	55.06
Qwen 3.7 Max	Closed-Source	64.89	72.31	64.10	53.75	51.69
Open-Source Models						
GLM 5.3	744B-A40B	73.88	76.44	72.07	77.00	58.43
Qwen 3.8 Max	2.4T-A95B	71.77	73.65	73.14	75.75	51.69
Kimi K3	2.8T-A104B	69.64	71.54	71.01	74.75	47.19
GLM 5.3 Flash	320B-A18B	68.31	71.35	65.16	68.25	59.55
GLM 5.2	744B-A40B	66.06	73.75	60.37	58.00	55.06
Baselines and Qwen-Planner-Models/Agents						
Qwen Baseline	35B-A3B	54.90	64.04	44.41	47.50	44.94
Qwen-Planner-Model	35B-A3B	64.79	66.15	61.17	71.00	52.81
Qwen-Planner-Agent	35B-A3B	69.91	71.25	67.02	78.00	52.81
Qwen Baseline	27B	67.22	68.37	67.82	73.75	47.19
Qwen-Planner-Model	27B	71.90	72.79	70.74	79.25	55.06
Qwen-Planner-Agent	27B	**77.05**	**77.79**	74.76	86.25	59.55

3.1. Mobile Planning Performance and Efficiency

We assess end-to-end mobile planning on MobilePA-Bench [11], comprising **1,700+ executable tasks**, **200+ mobile tools**, and **13 query-task domains**. Tasks start from controlled initial states; completion is verified through tool calls, terminal state changes, or agent behavior. Table 1 compares our base models, trained planners, and complete agents with Claude Opus 5, Opus 4.8, and Fable 5 [12]; Gemini 3.6 Flash and 3.1 Pro [13]; GPT-6 Astra and GPT-5.6 Sol [14]; Seed 2.1 Pro [15]; Qwen 3.7 Max [16] and 3.8 Max [17]; GLM 5.3, 5.3 Flash, and 5.2 [18]; and Kimi K3 [19]. Final evaluation is separate from development evaluations in the AI-for-AI loop (Section 2.1).

Performance. Qwen-Planner-Agent 27B achieves an Overall score of **77.05%** on MobilePA-Bench, ranking first among the evaluated models and agent systems and outperforming GPT 6 Astra (76.84%) and Claude Opus 5 (75.71%). Both model sizes improve over their corresponding Qwen baselines: Overall performance rises from 67.22% to 77.05% for 27B and from 54.90% to 69.91% for 35B-A3B. The contributions of planner training and Harness support are examined separately in the ablation studies. Across individual capabilities, Qwen-Planner-Agent 27B improves over its baseline in Tool Use, Memory, Skills, and Sub-agent coordination, achieving the highest Tool Use score of 77.79%, with Memory and Skills scores of 74.76% and 86.25%, respectively.

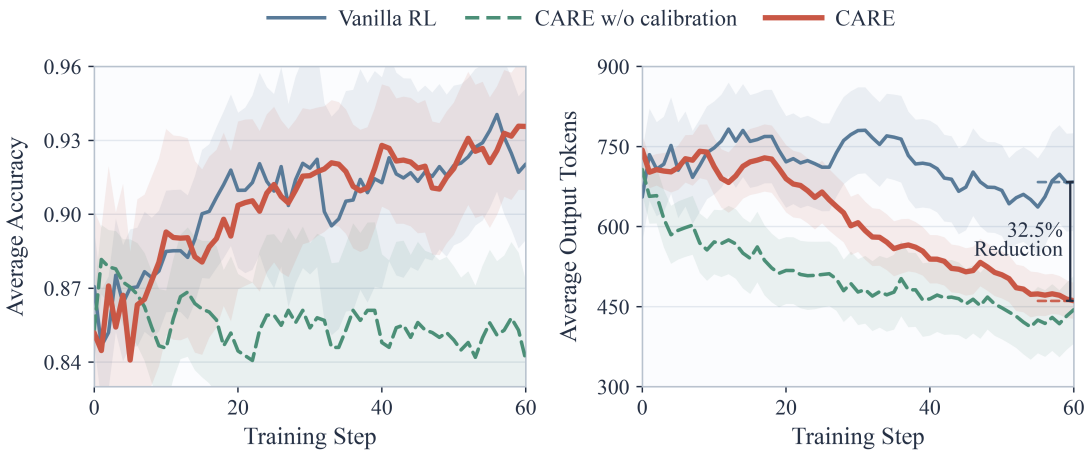

Figure 6: **Training dynamics of CARE and advantage calibration.** Under the same training setup, full CARE maintains accuracy comparable to Vanilla RL while using 32.5% fewer output tokens at the final plotted step. CARE without advantage calibration generates even shorter outputs but plateaus at lower accuracy. Lines show smoothed means over four training environments.

Efficiency. As shown in Figure 1, Qwen-Planner-Agent combines the highest Overall performance in the cost comparison with the lowest estimated output cost, at **$2.41 per 1,000 tasks**, compared with $3.06–$67.76 for the other evaluated models. Costs are estimated from mean output tokens per task, including thinking tokens, and the corresponding output-token rates. These results indicate a favorable performance–cost trade-off under the evaluated pricing assumptions. The estimates cover output-token charges only, excluding input tokens, external tool charges, device execution, and additional Harness processing.

3.2. Ablation Studies

We conduct three complementary analyses to examine training, runtime support, and iterative refinement. First, we evaluate how CARE and advantage calibration affect task success and output efficiency. Second, we measure the contribution of Harness support while holding planner parameters fixed. Finally, we assess whether alternating model training and Harness refinement improves performance on both mobile-planning and general tool-use tasks.

3.2.1. CARE Training Dynamics and Advantage Calibration

Using the 27B baseline from the main comparison, we compare Vanilla RL, full CARE, and CARE without quality-preserving advantage calibration under the same training setup across four training environments (Figure 6). The left panel reports average training accuracy, while the right panel reports average output length in tokens. The comparison with Vanilla RL evaluates CARE as a whole, while removing advantage calibration isolates its contribution under the same training setup.

CARE maintains average training accuracy comparable to Vanilla RL while reducing output length by **32.5%** at the final shared plotted step, based on the smoothed curves. Without advantage calibration, outputs become shorter but training accuracy remains substantially lower over the plotted interval. This pattern is consistent with the normalization effect analyzed in Section 2.4: small efficiency differences can acquire disproportionate advantage scale in success-saturated groups, while calibration limits their amplification.

Table 2: **Performance Comparison on MobilePA-Internal and MCPMark.** All scores are percentages; higher is better. Model-harness co-evolution improves the overall scores over the model-only baseline by 5.83 and 8.98 percentage points, respectively. Co-evolution results are reported after four iterations on MobilePA-Internal and three iterations on MCPMark.

Method	MobilePA-Internal				MCPMark
	Overall	Tool Use	Long Context	Personalized Planning	
Model Only	82.67	83.36	85.43	75.46	38.00
Model+Harness	84.23	85.52	86.18	75.46	42.26
Model–Harness Co-evolution	**88.50**	**89.76**	**88.69**	**82.42**	**46.98**

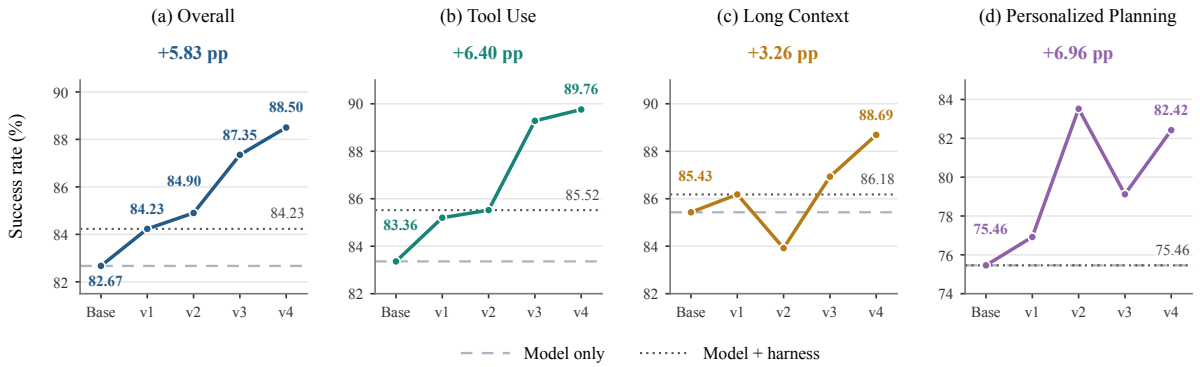

Figure 7: Model–harness co-evolution on MobilePA-Internal across four iterations (v1–v4). Base denotes the model-only baseline. Horizontal lines mark the two baselines and coincide in (d); panel annotations give v4 gains over Base in percentage points. Vertical ranges differ across panels.

3.2.2. Harness Contribution under Fixed Checkpoints

Table 1 provides paired evaluations before and after adding the Harness to each trained planner. Holding the 27B checkpoint fixed, the Harness raises Overall from 71.90% to 77.05%, with Tool Use increasing from 72.79% to 77.79%, Memory from 70.74% to 74.76%, Skills from 79.25% to 86.25%, and Sub-agent from 55.06% to 59.55%. For the 35B-A3B checkpoint, Overall rises from 64.79% to 69.91%, while Sub-agent remains unchanged at 52.81%. These comparisons identify the contribution of deployment-time context and execution support separately from changes in model parameters. The paired evaluations measure the combined effect of Harness support, with Section 3.3 further examining its behavior on long-history tasks.

3.2.3. Toward Model-Harness Co-Evolving

To evaluate the effectiveness of model–harness co-evolution, we use a separate 27B baseline checkpoint from the one used in the main comparison and conduct experiments on MobilePA-Internal and MCPMark. MobilePA-Internal is a comprehensive internal evaluation set that we construct to cover three task categories: Tool Use, Long Context, and Personalized Planning. MCPMark [20] is a general-purpose tool-use benchmark spanning five domains, allowing us to examine the framework's applicability beyond mobile-agent tasks. We compare co-evolution against two baselines: the model alone and the model equipped with harness. Both experiments organize model training into rounds with a fixed number of training steps. As in Eq. 12, each round trains the policy under a fixed harness. The updated policy is then evaluated under the same harness on a held-out evolve set, which serves as the development set for co-evolution rather than an independent final test set. Feedback from this development set guides harness revision for subsequent training; the evolve tasks and their trajectories are excluded from direct training.

As shown in Table 2, Model–Harness Co-evolution improves overall performance from 82.67% to 88.50% on MobilePA-Internal after four iterations and from 38.00% to 46.98% on MCPMark after three iterations. These final scores also exceed those of the harness, which achieves 84.23% on MobilePA-Internal and 42.26% on MCPMark. Figure 7 shows the iteration-wise progress on MobilePA-Internal: overall performance improves across successive iterations, with Tool Use, Long Context, and Personalized Planning all outperforming both baselines by the final iteration. These results support the effectiveness of Model–Harness Co-evolution beyond the gains provided by harness alone, across both mobile-agent and general-purpose tool-use tasks.

3.3. Long-History Memory Evaluation

3.3.1. Evaluation Protocol

We compare without-Harness and with-Harness inference to assess whether persistent memory helps when relevant evidence exceeds the active context. External baselines also include Seed 2.0 [21]. Without the Harness, the question and interaction history are placed directly in the model context; models are grouped by a 1M- or 256K-token evaluation budget, and the earliest history is truncated when that budget is exceeded. With the Harness, its Persistent Memory Manager processes the full history and retrieves task-relevant evidence for answering. Each matched Qwen pair uses the same planner checkpoint with different history-access mechanisms; input-token and processing costs vary between the two regimes.

Following the Mem0 evaluation protocol [22], LoCoMo [23] and LongMemEval [24] report binary LLM-judge accuracy, while BEAM [25] reports average rubric score. All scores are expressed as percentages, with higher values indicating better performance. Official Mem0 results are external references, not measurements produced with our Harness.

3.3.2. Results Across History Lengths

Direct-context performance as history grows. Direct-context inference remains strong on LoCoMo, LongMemEval, and BEAM-100K, but performance declines as history grows. Claude Opus 5 provides the strongest direct-context result through BEAM-1M, reaching 73.63% on BEAM-500K and 54.84% on BEAM-1M. Under the truncation protocol described above, no direct-context model exceeds 25.64 on BEAM-10M. Qwen-Planner-Model 27B remains competitive in the direct 256K setting, scoring 94.35% on LoCoMo and 95.00% on LongMemEval.

Matched Harness gains across history lengths. The benefit of explicit memory becomes pronounced on the longer BEAM settings. Across all four matched Qwen model pairs reported in Table 3, the Harness improves every result on BEAM-500K, BEAM-1M, and BEAM-10M, raising mean scores from 42.18% to 68.29%, from 40.41% to 70.27%, and from 23.42% to 61.66%, respectively. By contrast, the mean scores on LoCoMo, LongMemEval, and BEAM-100K change only from 93.90% to 94.12%, from 91.65% to 92.24%, and from 71.12% to 72.05%, respectively, and several individual pairs show small regressions. For Qwen-Planner-Model 27B, adding the Harness raises BEAM-500K from 42.01% to 72.20%, BEAM-1M from 40.12% to 73.71%, and BEAM-10M from 21.99% to 67.24%.

Comparison, trade-offs, and interpretation. The strongest with-Harness results are split across the longest settings: Qwen Baseline (27B) with the Harness achieves the best BEAM-1M score at 73.97%, while Qwen-Planner-Agent 27B achieves the best BEAM-10M score at 67.24% and scores 73.71% on BEAM-1M. For reference, the best direct-context results on BEAM-1M and BEAM-10M are 54.84% and 25.64%, and the officially reported Mem0 results are 64.10% and 48.60%. The Mem0 scores are

Table 3: **Long-history Memory Evaluation with and without Harness Support.** Models evaluated without the Harness are grouped by their context budget. Qwen-Planner-Model denotes planner-only inference; Qwen-Planner-Agent denotes the same planner with Harness support. LoCoMo and LongMemEval report binary LLM-judge accuracy, and BEAM reports average rubric score. All scores are reported as percentages (%). Bold denotes the best observed result in each column; † denotes externally reported Mem0 results, provided for reference rather than evaluated with our Harness. Avg. is the unweighted mean over available benchmark scores; for Mem0, it is computed over the four reported scores.

Model / System	Access / Size	LoCoMo	LongMem Eval	BEAM 100K	BEAM 500K	BEAM 1M	BEAM 10M	Avg.
w/o Memory Harness (1M Context Length Budget)								
Claude Opus 5	Closed-Source	**97.20**	**97.11**	**82.19**	**73.63**	54.84	20.49	70.91
GPT-5.6-Sol	Closed-Source	95.77	94.20	74.24	63.55	51.07	22.38	66.87
Gemini 3.6 Flash	Closed-Source	95.51	94.80	70.29	65.08	45.53	18.53	64.96
Claude Opus 4.8	Closed-Source	95.39	94.40	68.04	59.21	41.44	16.99	62.58
Qwen3.8-Max	2.4T-A95B	95.39	94.98	75.47	62.66	46.62	20.84	65.99
GLM-5.2	744B-A40B	95.06	93.80	71.33	64.23	48.47	25.09	66.33
w/o Memory Harness (256K Context Length Budget)								
Kimi K3	2.8T-A104B	95.38	95.40	77.11	44.59	40.93	21.82	62.54
Seed 2.0	Closed-Source	94.48	88.60	63.96	37.20	37.95	24.88	57.85
Qwen Baseline	35B-A3B	93.38	89.40	69.31	40.92	40.64	25.64	59.88
Qwen-Planner-Model	35B-A3B	93.44	87.80	66.58	42.09	40.14	22.96	58.84
Qwen Baseline	27B	94.41	94.40	76.26	43.71	40.75	23.07	62.10
Qwen-Planner-Model	27B	94.35	95.00	72.32	42.01	40.12	21.99	60.97
w/ Memory Harness								
Mem0 (Official)†	Closed-Source	92.50	94.40	–	–	64.10	48.60	74.90
Qwen Baseline	35B-A3B	93.18	90.00	64.93	63.87	65.17	56.68	72.31
Qwen-Planner-Agent	35B-A3B	93.44	86.55	69.93	65.65	68.21	57.23	73.50
Qwen Baseline	27B	94.93	95.80	78.00	71.42	**73.97**	65.50	79.94
Qwen-Planner-Agent	27B	94.93	96.60	75.35	72.20	73.71	**67.24**	**80.01**

external reference results. Across the matched Qwen comparisons, Harness gains are concentrated in the longer-history settings, while changes on shorter-history benchmarks vary across model pairs. Section 3.5 illustrates how evidence retrieval and reconciliation support long-history reasoning.

3.4. General Agentic Capability Evaluation

Qwen-Planner-Model improves agentic performance beyond mobile planning while largely preserving general capabilities. Table 4 compares planner-only inference with the corresponding Qwen baseline on Claw-Eval [26], BFCL-v4 [27], MCP-Atlas [28], Tau-3 [29], Toolathlon [30], SWE-Multilingual [31], and SWE-Pro [32]. The 35B-A3B and 27B variants improve on six and five of these seven agentic benchmarks, respectively, demonstrating stronger capabilities across function calling, multi-tool interaction, and coding tasks, although gains are not uniform across benchmarks. Meanwhile, performance on MMLU-Redux [33], C-Eval [34], and IFEval [35] remains close to the baselines, with small decreases. Together, these results show that our training recipe strengthens capabilities applicable to non-mobile agentic tasks without substantially compromising general reasoning and instruction following.

Table 4: **Performance Beyond Mobile Planning.** Benchmarks are grouped into agentic capability and general capability, with coding-agent evaluations included in the former. Each Qwen-Planner-Model checkpoint is compared with its corresponding base model. Avg. denotes the arithmetic average of the seven agentic or three general benchmarks within each group. All scores are percentages (%).

Model	Size	Agentic Capability								General Capability			
		Claw-Eval	BFCL-v4	MCP-Atlas	Tau-3	Tool-athlon	SWE-Multil.	SWE-Pro	**Avg.**	MMLU-Redux	C-Eval	IFEval	**Avg.**
Qwen Baseline	35B-A3B	72.83	60.63	59.40	67.94	29.30	65.67	48.43	57.74	89.67	90.45	90.94	90.35
Qwen-Planner-Model	35B-A3B	**74.51**	**66.99**	**68.69**	67.06	**37.15**	**71.67**	**50.07**	**62.31**	89.53	90.39	89.83	89.92
Qwen Baseline	27B	82.48	67.06	71.05	74.27	39.70	76.65	61.70	67.91	88.73	89.74	91.68	90.05
Qwen-Planner-Model	27B	79.58	**73.60**	**73.25**	**74.95**	**44.02**	74.89	**64.71**	**69.29**	88.67	89.30	91.13	89.70

3.5. Qualitative Analysis

To complement the quantitative evaluation in Section 3.1, we examine six execution traces that illustrate how Qwen-Planner-Agent coordinates planning and execution. In the unified runtime described in Section 2.5.1, the Harness supplies context and execution feedback, while the Planner Model selects and revises actions. The cases progress from reconciling persistent Memory and interaction history with current state to enforcing Skill-guided prerequisites, selecting conditional actions, and recovering from tool failures. Cross-application sub-agent coordination extends these dependencies across application boundaries. Together, the trajectories show how Qwen-Planner-Agent preserves task constraints while adapting execution to observed results.

Reconciling persistent memory with live state. Personalized execution requires reconciling remembered intent with the current application state. In Figure 8, Qwen-Planner-Agent retrieves the active flight monitors and saved itinerary, then confirms that the trip covers both Shenzhen and Guangzhou. Because Shenzhen is already monitored, it creates only the missing Guangzhou monitor, leaving the existing record and its settings intact. The case illustrates how memory and live state jointly determine the required action while preventing duplicate work.

Resuming tasks from verified execution state. After two hotspot requests, Qwen-Planner-Agent resumes casting from tool-confirmed state despite a misleading success acknowledgement (Figure 9). The history records an active wireless session and a rejected second start (H1–H2), establishing that the existing session must first be stopped. It stops the active session, waits for idle, and starts the bedroom cast while preserving wireless mode.

Enforcing execution prerequisites with Skills. An urgent shutdown request also requires a note to be saved, creating a dependency that must be resolved before the device becomes unavailable. In Figure 10, Qwen-Planner-Agent loads the note, power, and conflict-arbitration Skills and waits for the saved note's identifier before issuing power-off. Retrieved guidance thus becomes an execution prerequisite that preserves task completion before a blocking action.

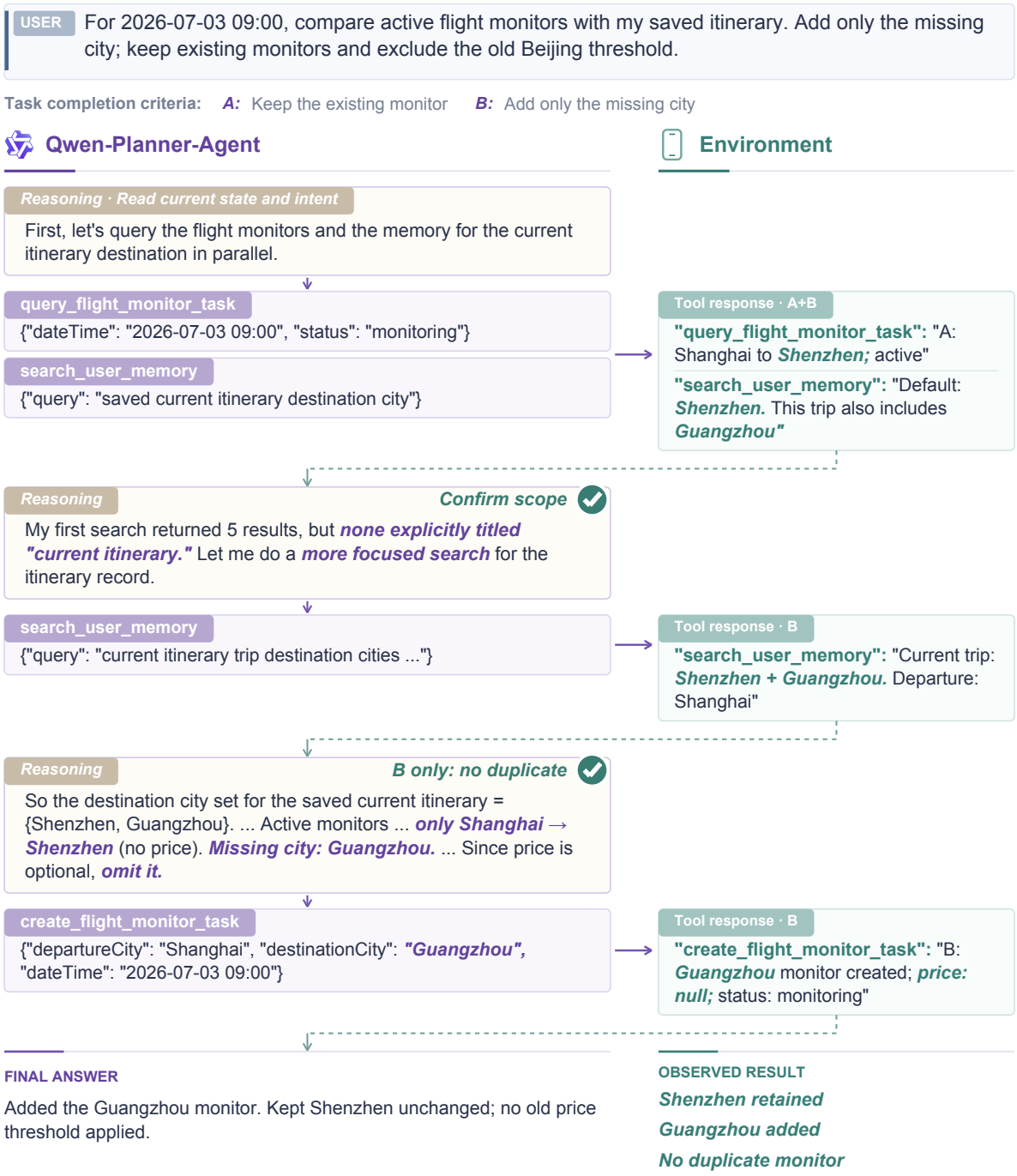

Figure 8: Reconciling persistent memory with active task state. The saved itinerary includes Shenzhen and Guangzhou, while an active monitor already covers Shenzhen (A). Qwen-Planner-Agent adds only Guangzhou (B), preserving the existing monitor and excluding the old Beijing price threshold.

Figure 9: State-grounded tool use across dialogue turns. H1 records the established wireless mode, while H2 records that a cast is still active. Qwen-Planner-Agent follows these states, waits for the active session to stop, then starts the bedroom cast. The final return confirms both the revised destination and the retained connection method.

Figure 10: Skill-guided prerequisite preservation before shutdown. Qwen-Planner-Agent first obtains a persistent identifier for the saved note (A) and only then issues power-off (B). The returned states confirm that the retrieved procedural constraint is realized in execution and that both parts of the request complete.

Conditioning tool execution on observed state. Conditional requests require the planner to resolve both whether an action is needed and when it may be issued. In Figure 11, Qwen-Planner-Agent queries the connected headphones and selects the branch corresponding to transparency mode. It then enables Do Not Disturb and waits for the successful return before switching to noise reduction. The trajectory combines state-dependent branching with an explicit ordering constraint, translating the user's conditional request into a sequence of native tool calls.

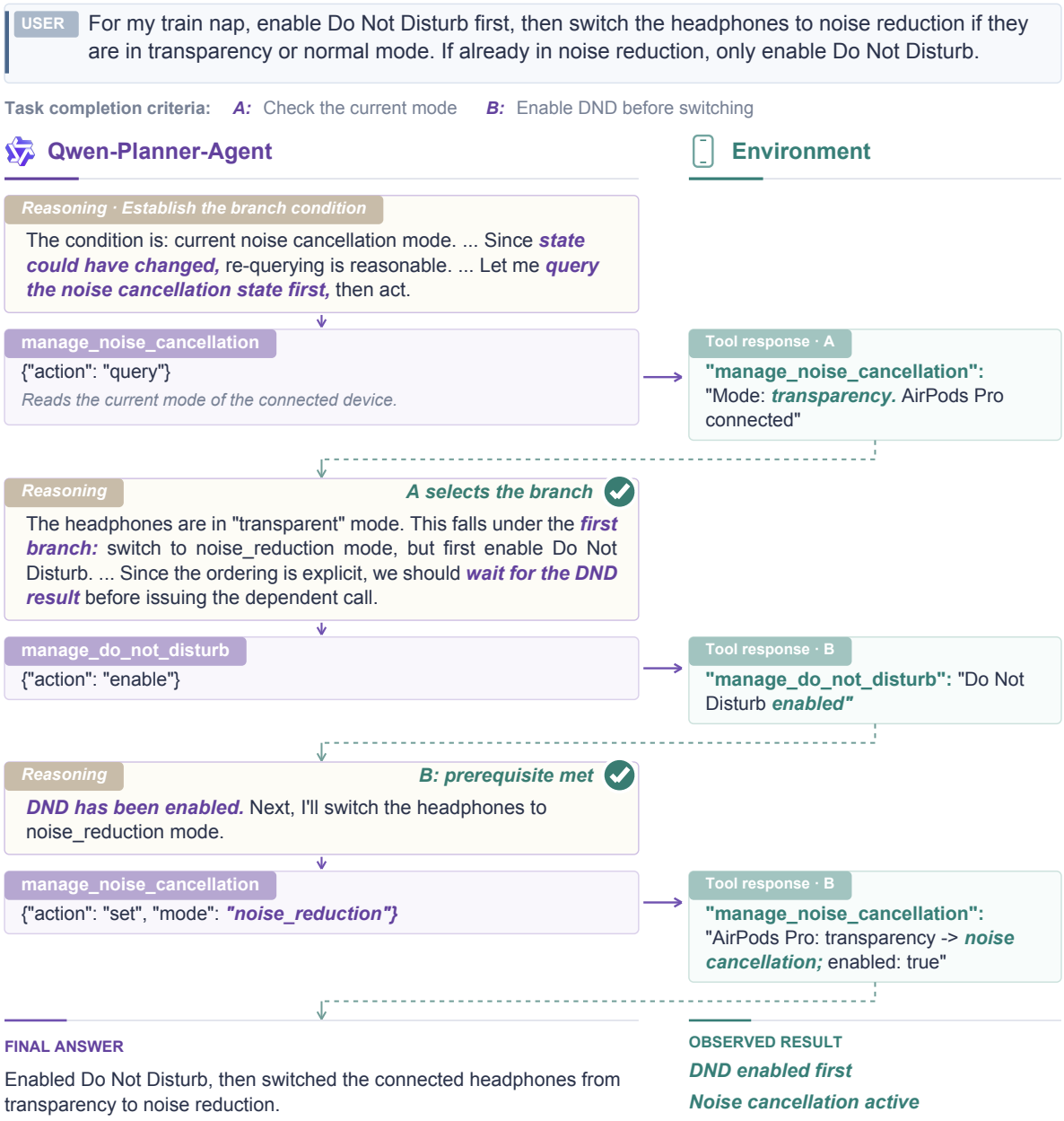

Figure 11: Conditional tool use under an ordering constraint. The observed transparency mode selects the branch that requires a headphone adjustment (A). Qwen-Planner-Agent first enables Do Not Disturb, then uses its successful return to proceed with noise reduction (B). Both requested settings are confirmed by the tool results.

Recovering execution through tool substitution. An alternative tool can sustain execution when the original operation fails. In Figure 12, the unit-conversion tool rejects both full unit names and their abbreviations. Qwen-Planner-Agent uses the standard conversion factor to compute 154.32 pounds from the observed 70 kilograms, then sends the result to the coach identified in the interaction history. This substitution preserves the plan's data dependency, allowing the remaining communication step to complete.

Figure 12: Tool substitution for workflow recovery. After two conversion attempts fail, Qwen-Planner-Agent uses the calculator to obtain 154.32 pounds (A) and sends the result to the coach (B). H1 links the earlier saved contact to the final message. All five current tool-call rounds are shown; the successful calculation and message return establish completion of the recovered workflow.

Preserving dependencies across sub-agent handoffs. Sub-agent orchestration requires handoffs to be conditioned on execution results and to preserve the artifacts needed by the next stage. In Figure 13, Qwen-Planner-Agent waits for the WhatsApp session to report three completed downloads before launching the File Manager session. It then transfers the returned filenames together with their subject-folder assignments into the second instruction. The A-to-B transition is therefore grounded in a completed result rather than the order of two prompts. Both session returns and the recorded task state confirm that the cross-application workflow reaches the requested organization.

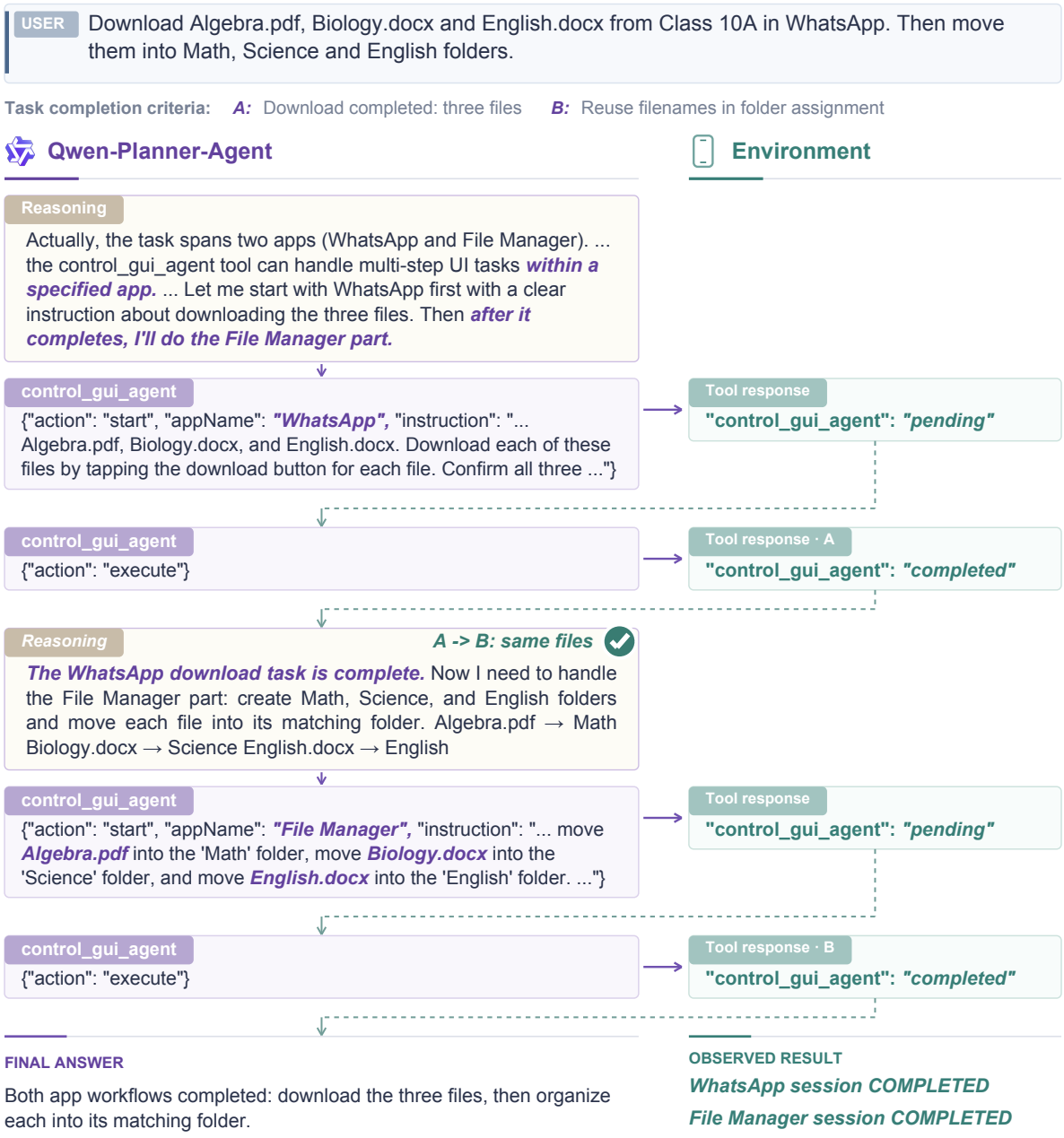

Figure 13: Dependency-preserving sub-agent coordination. Qwen-Planner-Agent waits for the WhatsApp download result (A), then passes the same three filenames and their destination folders to the File Manager session (B). The two session returns and the task-state check confirm completion across the application boundary.

4. Related Work

4.1. AI-Assisted AI Development

AI-assisted data production provides one route for improving subsequent models. Self-Instruct generates instruction-tuning examples, while Cosmopedia extends synthetic generation to pretraining corpora [36, 37]. Feedback can make this process more targeted: LLM2LLM augments examples that a student model fails on, and AutoIF uses code-based verification and execution feedback to filter instruction-following data [38, 39]. Our data pipeline applies feedback-guided refinement to interactive planning. AI-assisted task construction and failure diagnosis guide new tasks and sampling adjustments, while automated collection and curation produce executable training experience.

AI also participates in optimizing the artifacts and workflows used to develop models. OPRO, TextGrad, and DSPy use evaluation signals to improve candidate solutions, textual components, or language-model programs [40, 41, 42]. Reward-generation methods such as Eureka and CARD extend this assistance to reinforcement learning [43, 44], while AutoML-Agent and the AI Scientist address broader development and research workflows [45, 46]. These directions motivate our systems-level investigation: how can execution experience guide connected improvements in data, training, and deployment? We study this question through the development of Qwen-Planner-Agent, using diagnosed failures to inform training tasks, learning configurations, and runtime revisions within one iterative lifecycle.

4.2. Mobile Agents and Interactive Environments

Mobile-agent systems connect high-level requests to application operations through different interfaces. AppAgent learns application-specific interaction knowledge from exploration or demonstrations, while Mobile-Agent-E separates planning from visual execution and retains reusable experience [47, 48]. AppWorld instead exposes structured application APIs and evaluates tasks through environment state [49]. GUI interaction and structured tools offer complementary access to applications; neither interface alone determines whether an agent can track dependencies, maintain context, recover from errors, and verify a complex task's completion.

Qwen-Planner-Agent focuses on this planning problem rather than introducing a new screen-grounding architecture. MobilePA-Bench measures relevant capabilities through stateful execution across Tool Use, Memory, Skills, and Sub-agent [11]. To develop these capabilities at scale, we combine reproducible programmatic sandboxes, LLM-simulated long-tail interactions, and selected real-device sessions. The emphasis is on using their complementary feedback to support data collection and online learning, while the Planner Model and Harness coordinate execution under changing tasks and resources.

4.3. Agentic Reinforcement Learning

Verifiable-reward optimization provides a foundation for training interactive planners. GRPO normalizes rewards within sampled groups without a learned critic, and DAPO and GSPO further develop group-based policy optimization [7, 10, 50]. In interactive tasks, actions also change the environment and determine subsequent observations, complicating exploration and credit assignment [51]. Process supervision, partial rollouts, and branching exploration address aspects of these challenges by providing intermediate learning signals or reusing interaction experience [52, 53, 54].

Mobile RL additionally faces costly resets and variable execution conditions. DigiRL combines offline initialization with online Android interaction; MobileRL uses difficulty-adaptive replay and curriculum filtering; Mobile-R1 trains through multi-turn interaction; and PhoneBuddy combines resettable mock applications with real-app training [4, 55, 56, 5]. Our approach combines hybrid-environment online

agentic RL with competence-adaptive optimization. CARE changes the reward emphasis as task success improves and calibrates advantages to prevent small efficiency differences from dominating learning. A bounded LLM-based controller configures predefined scheduling parameters using training and development-set feedback. This separates feedback-driven configuration from the rule-based reward and advantage computations applied during training.

4.4. Agent Harnesses and Model–Harness Co-evolution

Memory and skills provide complementary runtime resources for long-horizon agents. External-memory methods retain episodes, reflections, managed context, or multimodal evidence for later retrieval [57, 58, 59, 60, 61]. Skill-oriented methods retain executable procedures, induced workflows, and reusable task knowledge [2, 62, 63, 48, 64]. These resources motivate a Harness that selects relevant historical evidence and procedural guidance for the current task, rather than requiring the Planner Model to encode every changing fact or operating rule in its parameters.

Harness optimization makes this runtime support itself an object of improvement. AutoHarness synthesizes code harnesses from environment feedback, Meta-Harness searches over Harness code using prior scores and traces, Natural-Language Agent Harnesses expresses control logic as editable instructions, and MemoHarness uses prior execution experience to adapt Harness control dimensions [65, 66, 67, 68]. Our focus is the interaction between Harness revision and model learning: a revised Harness changes the context and experience used in subsequent policy training, while the updated model's behavior informs the next revision. We implement this coupling through alternating reinforcement learning and LLM-based instruction editing, using held-out development feedback. The multi-round experiments examine this process in mobile-planning and general tool-use settings, providing evidence for iterative model–Harness improvement within the evaluated scope.

5. Conclusion

In this report, we present Qwen-Planner-Agent, a unified Planner Model–Harness agentic system developed through a closed-loop feedback-driven AI-for-AI framework. Execution experience guides AI-assisted improvements in data production, training, and deployment. Qwen-Planner-Agent achieves the highest Overall score on MobilePA-Bench among the evaluated systems at a low estimated output cost, while Qwen-Planner-Model improves on most matched non-mobile agentic benchmarks and largely preserves general capabilities. Multi-round experiments further support alternating model training and Harness refinement within the evaluated settings. Together, these results provide a concrete practice of using AI to help develop more capable agents. Future work will focus on reducing manual intervention in data refinement, training adaptation, and Harness updates, enabling more scalable and efficient iterations of agent capabilities. We aim to automate more of this development lifecycle while retaining human oversight at safety-critical decision points.

References

[1] I. J. Good. Speculations concerning the first ultraintelligent machine. In Franz L. Alt and Morris Rubinoff, editors, *Advances in Computers*, volume 6, pages 31–88. Academic Press, 1965.

[2] 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. *arXiv preprint arXiv:2305.16291*, 2023.

[3] Alan M. Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950.

[4] Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

[5] Zhengyang Tang, Xin Lai, Pengyuan Lyu, Xinyuan Wang, Tianyi Bai, Chenxin Li, Yiduo Guo, Huawen Shen, Yuxuan Liu, Junyi Li, et al. Phonebuddy: Training open models for agentic phone use. *arXiv e-prints*, pages arXiv–2606, 2026.

[6] Anthropic. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol, November 2024.

[7] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[8] Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, et al. Gdpo: Group reward-decoupled normalization policy optimization for multi-reward rl optimization. *arXiv preprint arXiv:2601.05242*, 2026.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[10] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

[11] Yi Zhu, Xiongwei Wu, Qiyi Wang, Tingyu Qu, Jiajun Liu, Sihan Cao, Long Chen, Weigao Sun, Feida Zhu, Yiran Zhong, et al. Mobilepa-bench: Benchmarking mobile planner agents on complex real-world tasks. *arXiv preprint arXiv:2608.23035*, 2026.

[12] Anthropic. Anthropic's transparency hub. https://www.anthropic.com/transparency, 2026. Model reports and system cards for Claude Opus 5, Fable 5, and Opus 4.8. Accessed 2026-09-16.

[13] Google. Gemini API: Models. https://ai.google.dev/gemini-api/docs/models, 2026. Official model documentation, including Gemini 3.6 Flash and Gemini 3.1 Pro. Accessed 2026-09-16.

[14] OpenAI. OpenAI API: Models. https://developers.openai.com/api/docs/models, 2026. Official model documentation, including GPT-6 Astra and GPT-5.6 Sol. Accessed 2026-09-16.

[15] ByteDance Seed Team. Seed2.1 officially released: Advancing AI productivity. https://seed.bytedance.com/en/blog/seed2-1-officially-released-advancing-ai-productivity, June 2026.

[16] Alibaba Cloud. Qwen3.7-Max. https://docs.modelstudio.console.alibabacloud.com/en/model-studio/qwen3-7-max, 2026. Official model documentation. Accessed 2026-09-16.

[17] Qwen Team. Qwen3.8-Max: A new bar for coding and cowork. https://qwen.ai/blog?id=qwen3.8, August 2026.

[18] GLM-5 Team et al. GLM-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

[19] Kimi Team et al. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

[20] Zijian Wu, Xiangyan Liu, Xinyuan Zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, Jinjie Ni, Yufan Yang, Arvin Xu, and Michael Qizhe Shieh. MCPMark: A benchmark for stress-testing realistic and comprehensive MCP use. In *International Conference on Learning Representations*, volume 2026, pages 79852–79895, 2026.

[21] ByteDance Seed Team. Seed 2.0 official launch. https://seed.bytedance.com/blog/seed-2-0-official-launch, February 2026.

[22] Mem0. Mem0 research: LoCoMo, LongMemEval and BEAM benchmarks. https://mem0.ai/research, 2026. Official evaluation protocols and reported results. Accessed 2026-09-16.

[23] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, 2024.

[24] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

[25] Mohammad Tavakoli, Alireza Salemi, Carrie Ye, Mohamed Abdalla, Hamed Zamani, and J Ross Mitchell. Beyond a million tokens: Benchmarking and enhancing long-term memory in LLMs. *arXiv preprint arXiv:2510.27246*, 2025.

[26] Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Towards trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026.

[27] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392. PMLR, 2025.

[28] Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, Manasi Sharma, Andrew Park, Martin Dimakis, Ernesto Gabriel Hernandez Montoya, Dan Rambado, Ivan Salazar, Rafael Cruz, MohammadHossein Rezaei, Chetan Rane, Ben Levin, Daniel Yue Zhang, Brad Kenstler, and Bing Liu. MCP-Atlas: A large-scale benchmark for tool-use competency with real MCP servers. *arXiv preprint arXiv:2602.00933*, 2026.

[29] Sierra Research. τ^3-bench: Tool-agent-user interaction benchmark. https://github.com/sierra-research/tau2-bench, 2026. Official repository and release documentation for the third benchmark version. Accessed 2026-09-16.

[30] Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *arXiv preprint arXiv:2510.25726*, 2025.

[31] Kabir Khandpur, Kilian Lieret, Carlos E. Jimenez, Ofir Press, and John Yang. SWE-bench Multilingual. https://www.swebench.com/multilingual.html, 2025. Official benchmark release.

[32] Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

[33] Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile van Krieken, and Pasquale Minervini. Are we done with MMLU? *arXiv preprint arXiv:2406.04127*, 2024.

[34] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. C-Eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*, 2023.

[35] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[36] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 13484–13508, 2023.

[37] Moritz Günther et al. Cosmopedia: How to create large-scale synthetic pretraining data. *Hugging Face Blog*, 2024.

[38] Nicholas Lee, Thanakul Wattanawong, Sehoon Kim, Karttikeya Mangalam, Sheng Shen, Gopala Anumanchipalli, Michael Mahoney, Kurt Keutzer, and Amir Gholami. Llm2llm: Boosting llms with novel iterative data enhancement. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 6498–6526, 2024.

[39] Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. Self-play with execution feedback: Improving instruction-following capabilities of large language models. In *International Conference on Learning Representations*, volume 2025, pages 39286–39313, 2025.

[40] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pages 12028–12068, 2024.

[41] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

[42] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

[43] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, et al. Eureka: Human-level reward design via coding large language models. In *International conference on learning Representations*, volume 2024, pages 26516–26560, 2024.

[44] Shengjie Sun, Runze Liu, Jiafei Lyu, Jing-Wen Yang, Liangpeng Zhang, and Xiu Li. A large language model-driven reward design framework via dynamic feedback for reinforcement learning. *Knowledge-Based Systems*, 326:114065, 2025.

[45] Patara Trirat, Woomin Jeong, et al. Automl-agent: A multi-agent llm framework for full-pipeline automl. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

[46] Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of ai research. *Nature*, 651(8107):914–919, 2026.

[47] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI conference on human factors in computing systems*, pages 1–20, 2025.

[48] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025.

[49] 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)*, pages 16022–16076, 2024.

[50] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025.

[51] Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, et al. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*, 2025.

[52] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601, 2024.

[53] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

[54] Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. In *International Conference on Learning Representations*, volume 2026, pages 16981–17017, 2026.

[55] Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Jiayu Huang, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Yuxiao Dong, et al. Mobilerl: Online agentic reinforcement learning for mobile gui agents. In *International Conference on Learning Representations*, volume 2026, pages 35282–35315, 2026.

[56] Jihao Gu, Qihang Ai, Yingyao Wang, Pi Bu, Jingxuan Xing, Zekun Zhu, Wei Jiang, Ziming Wang, Yingxiu Zhao, Ming-Liang Zhang, et al. Mobile-r1: Towards interactive reinforcement learning for vlm-based mobile agent via task-level rewards. *arXiv e-prints*, pages arXiv–2506, 2025.

[57] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[58] 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.

[59] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

[60] Chunliang Chen, Ming Guan, Xiao Lin, Jiaxu Li, Luxi Lin, Qiyi Wang, Xiangyu Chen, Jixiang Luo, Changzhi Sun, Dell Zhang, et al. Telemem: Building long-term and multimodal memory for agentic ai. *arXiv preprint arXiv:2601.06037*, 2025.

[61] Jusen Du, Weigao Sun, Disen Lan, Jiaxi Hu, Zhang Tao, and Yu Cheng. Mom: Linear sequence modeling with mixture-of-memories. In *International Conference on Learning Representations*, volume 2026, pages 106613–106631, 2026.

[62] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[63] Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steven Y Ko, Sangeun Oh, and Insik Shin. Explore, select, derive, and recall: Augmenting llm with human-like memory for mobile task automation. *arXiv preprint arXiv:2312.03003*, 2023.

[64] Qijia Chen, Andrea Bellucci, Zhida Sun, and Giulio Jacucci. Skilldroid: Compile once, reuse forever. *arXiv preprint arXiv:2604.14872*, 2026.

[65] Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P Murphy. Autoharness: improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026.

[66] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

[67] Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. Natural-language agent harnesses. *arXiv preprint arXiv:2603.25723*, 2026.

[68] Yue Huang, Wenjie Wang, Han Bao, Yuchen Ma, Xiaonan Luo, Yi Nian, Haomin Zhuang, Zheyuan Liu, Yue Zhao, and Xiangliang Zhang. Memoharness: Agent harnesses that learn from experience. *arXiv preprint arXiv:2607.14159*, 2026.

[69] Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, et al. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library. *arXiv preprint arXiv:2506.06122*, 2025.

[70] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[71] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[72] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577. USENIX Association, 2018.

Appendix

A. Harness Configuration and Memory Details

This appendix expands the deployment mechanisms summarized in Section 2.5. The Harness is the common runtime that assembles context, connects the Planner Model to external resources, relays structured execution feedback, and retains traces for diagnosis. The Scenario Adapter and Persistent Memory Manager are internal components of this runtime rather than independent runtime systems.

A.1. Scenario-Adaptation Configuration

The Scenario Adapter converts deployment-time tool availability into compact operational guidance. Offline, it compiles tool schemas, capability-domain Skill definitions, a complete tool-to-Skill mapping, and guidance constraints into a versioned configuration. Each build records a digest of the target tool schemas, checks coverage of the deployed tool set, and resolves ambiguous mappings through explicit overrides. The resulting artifact contains a tool-to-Skill index, reusable domain guidance, global execution rules, candidate-specific routing notes, multi-turn instructions, priority rules, and configurable limits on injected content.

At request time, the Adapter receives the tools already exposed to the planner; it does not discover additional resources. It maps these candidate tools to the corresponding Skills and filters each guidance block against the request-visible tool set. General rules are retained, instructions tied exclusively to unavailable tools are removed, and tool-family lists are rewritten to include only reachable resources. Candidate-specific routing notes and conditional multi-turn guidance are then added before the result is incorporated into the active context. Requests that explicitly expose Skill tools receive a generic Skill-use protocol instead of candidate-derived guidance, preventing the Adapter from revealing or bypassing the intended Skill-selection process.

The planner remains responsible for interpreting the request, resolving conflicts, selecting actions, and replanning after feedback. The Scenario Adapter neither executes actions nor persists device state. Agent-backed capabilities and atomic tools enter the same structured action interface, while Skills provide procedural guidance for composing their use rather than acting as opaque commands.

A.2. Persistent Memory Lifecycle

The Persistent Memory Manager stores cross-interaction information outside the model in an inspectable workspace. It distinguishes a compact user model for stable preferences and behavioral requirements, episodic records for detailed observations and dialogue evidence, curated long-term memory for consolidated facts and lessons, prospective memory for future intentions with explicit activation conditions, and review records that document maintenance without becoming authoritative memories. These layers differ in both function and access: compact validated information may be made readily available, whereas detailed or less trusted evidence requires retrieval under the current task.

Memory formation begins with observations and dialogue evidence captured during interaction. Before a long context is compressed, a persistence step records important unstored requirements, decisions, and state changes. Each memory unit links a structured summary to source metadata and a lossless dialogue record. The summary supports efficient retrieval; metadata records provenance, speaker, turn, and order; and the original record permits direct verification of numbers, dates, quotations, contradictions, and changing facts. Invalid summaries are treated as ingestion failures rather than silently replaced, keeping ingestion success, summary quality, and source fidelity separately observable.

Information remains episodic until a staged process establishes that it is useful and sufficiently trustworthy for longer-term use. Deterministic checks first enforce eligibility, provenance, and trust requirements; bounded model-assisted processing is then used to organize related observations, remove duplicates, identify recurring evidence, and express accepted information concisely. Consolidation merges duplicates with existing memory, preserves source links, and marks obsolete entries as superseded when newer evidence changes the current state. Recalled material is tagged so that it is not re-ingested as new evidence, preventing repeated retrieval from artificially increasing its apparent importance.

Retrieval combines semantic and keyword matching with relevance, recency, importance, and diversity controls. Query structure determines how evidence is assembled: temporal questions require event ordering, cross-session aggregation requires entity and action deduplication, and questions about updated facts require explicit resolution of superseding evidence. For long histories, retrieval may proceed through high-recall search, construction of a query-specific evidence ledger, source verification, and focused re-examination. Retrieved Memory remains evidence supplied to the planner; it does not authorize or execute an action.

Provenance and lifecycle controls bound the authority of stored content. User-provided, agent-derived, externally sourced, and system-generated information remain distinguishable, and content cannot promote its own trust level. Prospective items carry activation conditions, scope, expiry, and completion status. Consolidation reports record additions, merges, and superseded entries, while earlier versions are retained before substantial revisions. If semantic search or model-assisted consolidation is unavailable, the system can fall back to simpler retrieval, retain information in episodic form, or postpone maintenance without blocking the primary task.

A.3. Memory Evaluation and Reproducibility

LoCoMo, LongMemEval, and BEAM are evaluated through a common hierarchy of samples, sessions, dialogue turns, and questions while retaining their original task semantics and scoring rules. We compare models paired with the Persistent Memory Manager against their corresponding model-only settings. Memory ingestion, answer generation, aggregation, and error handling follow the same pipeline, and answer generation, retrieval, and automated judging are assigned separate roles to reduce coupled model bias.

Retrieval integrity is part of a valid execution. If the planner invokes a retrieval operation, the trace must contain at least one successful result. A failed attempt followed by a successful one is recorded as recovery; traces in which every attempt fails remain eligible for retry. Timeouts, context overflows, embedding failures, and retrieval-service errors are recorded as infrastructure failures rather than model errors. The evaluation pipeline records protocol accuracy over valid executions, end-to-end accuracy over all samples, and valid-execution coverage separately so that model quality is not conflated with runtime reliability; the aggregate scores in Table 3 follow each benchmark's stated scoring protocol.

Reproducibility is supported through sample-level isolation, configuration fingerprints, ingestion snapshots, and per-question checkpoints. Each evaluation unit has an independent session, index, and result space, enabling interrupted runs to resume from a known state. The retained artifact chain includes dataset manifests, memory records, retrieval traces, model calls, judge outputs, configurations, and aggregate metrics. These records make it possible to distinguish failures caused by memory construction, missing retrieval evidence, answer generation, automated scoring, or infrastructure.

B. Infrastructure Details

Our training infrastructure supports the main stages of mobile-agent development, including supervised fine-tuning, long-context training, reinforcement learning, and on-policy distillation. It combines distributed model training and rollout generation with a shared environment management layer. The same system can support both dense and Mixture-of-Experts (MoE) models, as well as tasks running in sandboxes, simulators, and real mobile devices.

Agentic RL. We build our agentic reinforcement learning system on Roll [69]. Megatron-Core [70] is used for distributed policy training, while vLLM [71] provides efficient rollout inference. Ray [72] coordinates training, inference, environment interaction, and reward computation across the cluster. This design allows each part of the pipeline to use resources according to its own workload.

Training and rollout generation run asynchronously on separate groups of GPUs. Rollout workers continue to interact with environments and generate trajectories while learner workers update the policy. This separation reduces idle time and improves system efficiency compared with a fully synchronous pipeline. It also makes the allocation of training and inference resources easier to adjust for different models and tasks.

The system supports long-context reinforcement learning for large MoE models through several forms of distributed parallelism. Training configurations are selected based on model size, context length, memory use, and communication cost. For example, context parallelism distributes long sequences across devices, while expert parallelism spreads MoE experts across the training cluster. Group-based policy optimization is used to compare multiple trajectories for each task and provide a more stable learning signal.

The system also checks that parallel groups, rollout batches, and model partitions are compatible before training begins. These checks help distribute samples and tokens evenly across devices and prevent invalid distributed configurations.

Environment Management. Section 2.2.3 introduces the division of responsibilities between Roll and ROCK. At the implementation level, the same environment layer is reused for evaluation, supervised data collection, and reinforcement learning, and it supports general-agent and code-agent tasks through backend-specific adapters. This reuse allows sandboxed tasks, simulated mobile tasks, and real-device interactions to enter a shared training pipeline without requiring identical execution engines.

During online training, Roll manages distributed rollout generation, model updates, and training-resource scheduling, while ROCK manages environment instances, device sessions, and their execution lifecycles. Environment validation and trajectory filtering are applied before samples are admitted to training. These checks preserve the separation between distributed model computation and backend-specific execution while keeping the resulting trajectories compatible with the common data and optimization pipeline.

C. Additional Mobile-Planning Case Studies

These six trajectories extend Section 3.5 in a progression from procedural dependencies, through scoped revisions, to context use across user turns. They show Qwen-Planner-Agent recovering complete procedures, sequencing actions with interacting effects, limiting device and memory revisions to their intended scope, and reusing information across dialogue turns. Each task is completed with supporting tool returns or stored state. Each figure retains all tool-call rounds of the current interaction. Earlier turns in the two dialogue examples provide the interaction history, and repeated letter and H indexes connect prior requirements or observations to the highlighted decisions.

C.1. Procedural Dependencies

Recovering complete procedures through targeted retrieval. Figure 14 examines whether the planner can detect that a retrieved memory is insufficient for the requested procedure. The first result covers input sharing but omits the complete coding setup. Qwen-Planner-Agent identifies the missing information, retrieves the full workflow, and preserves its order through editor launch, input sharing, and USB debugging. The A/B markers separate the setup prerequisites from the final debugging action, and each returned state is incorporated before the next dependent call.

Composing Skills under execution dependencies. Figure 15 separates parallelizable preparation from state-dependent execution. The connectivity and Bluetooth Skills can be loaded together, but their device actions cannot be applied in arbitrary order because enabling airplane mode disables Bluetooth. Qwen-Planner-Agent therefore waits for the airplane-mode transition (A) before restoring Bluetooth (B). The final returns verify both requested settings, showing that the planner accounts for cross-tool side effects rather than treating the calls as independent.

C.2. Scoped Device and Memory Revisions

Correcting device state within explicit constraints. Figure 16 tests whether a stored preference can be translated into the smallest valid state change. The observed configuration has TalkBack enabled and Select to Speak disabled, while the preference requires the reverse and explicitly preserves the font setting. Qwen-Planner-Agent queries the relevant states, applies the two necessary accessibility changes (A), and leaves the font configuration untouched (B). This observation–delta–verification pattern grounds memory in device state while respecting an explicit non-interference constraint.

Updating persistent memory within the requested scope. Related memories may require different treatment under the same request. In Figure 17, Qwen-Planner-Agent identifies three hotel-networking records, updates the usual Wi-Fi and the fallback for unstable connections, and retains the existing large-attachment rule. It issues the two independent replacements together using their original record identifiers. The final memory state confirms both revisions and the preserved third record, showing how a compound preference change can be applied without duplicating memories or broadening the edit.

Figure 14: Memory-guided recovery of an ordered procedure. Qwen-Planner-Agent recognizes that the first memory is incomplete and retrieves the missing workflow. It then preserves the recovered dependency order: editor launch and input sharing (A) precede USB debugging (B), with successful returns confirming all three actions.

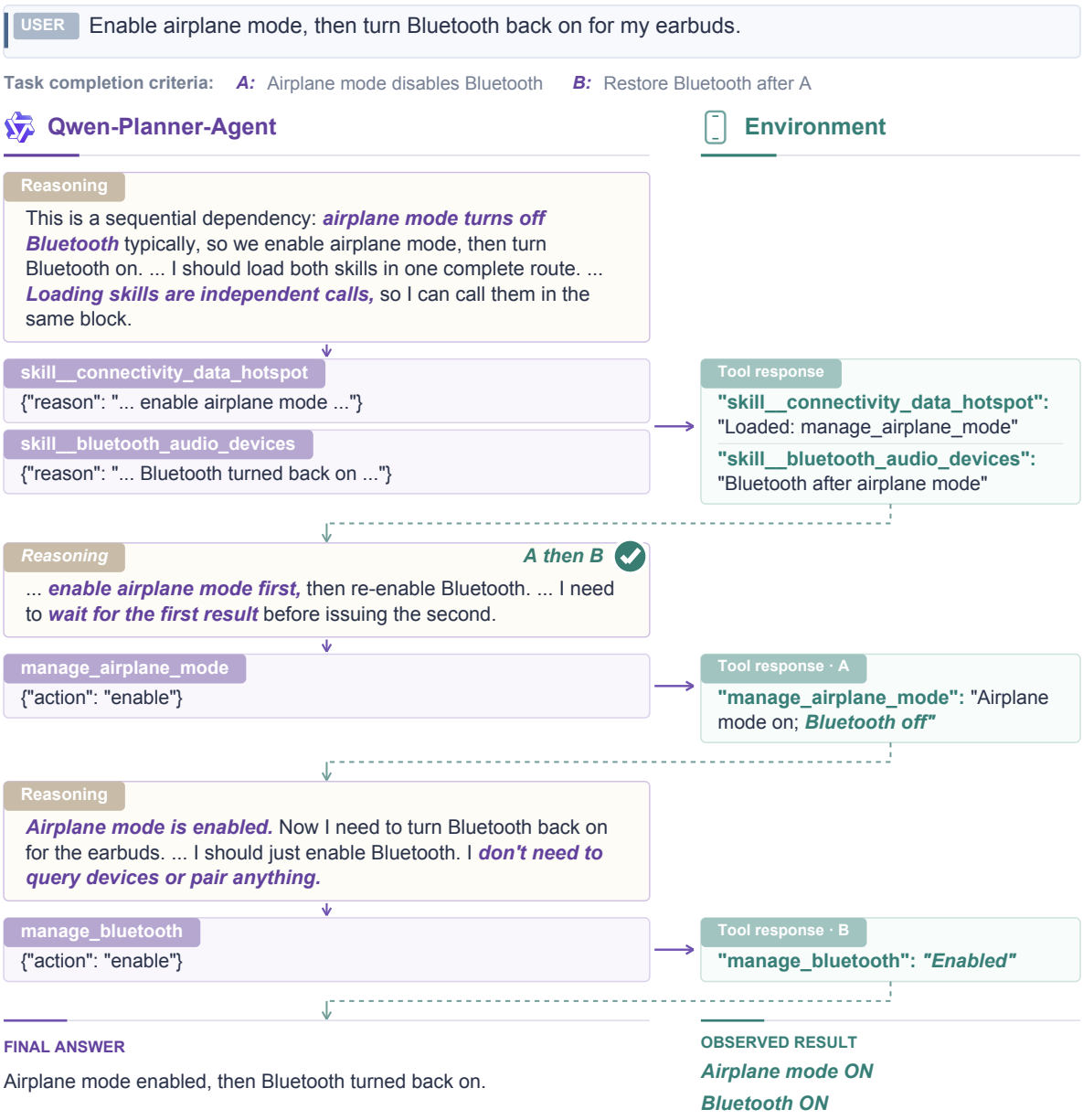

Figure 15: Skill-guided planning over interacting state changes. Qwen-Planner-Agent loads the relevant Skills together, then enables airplane mode (A) before restoring Bluetooth (B). This order accounts for the first action's side effect and leaves both requested settings enabled.

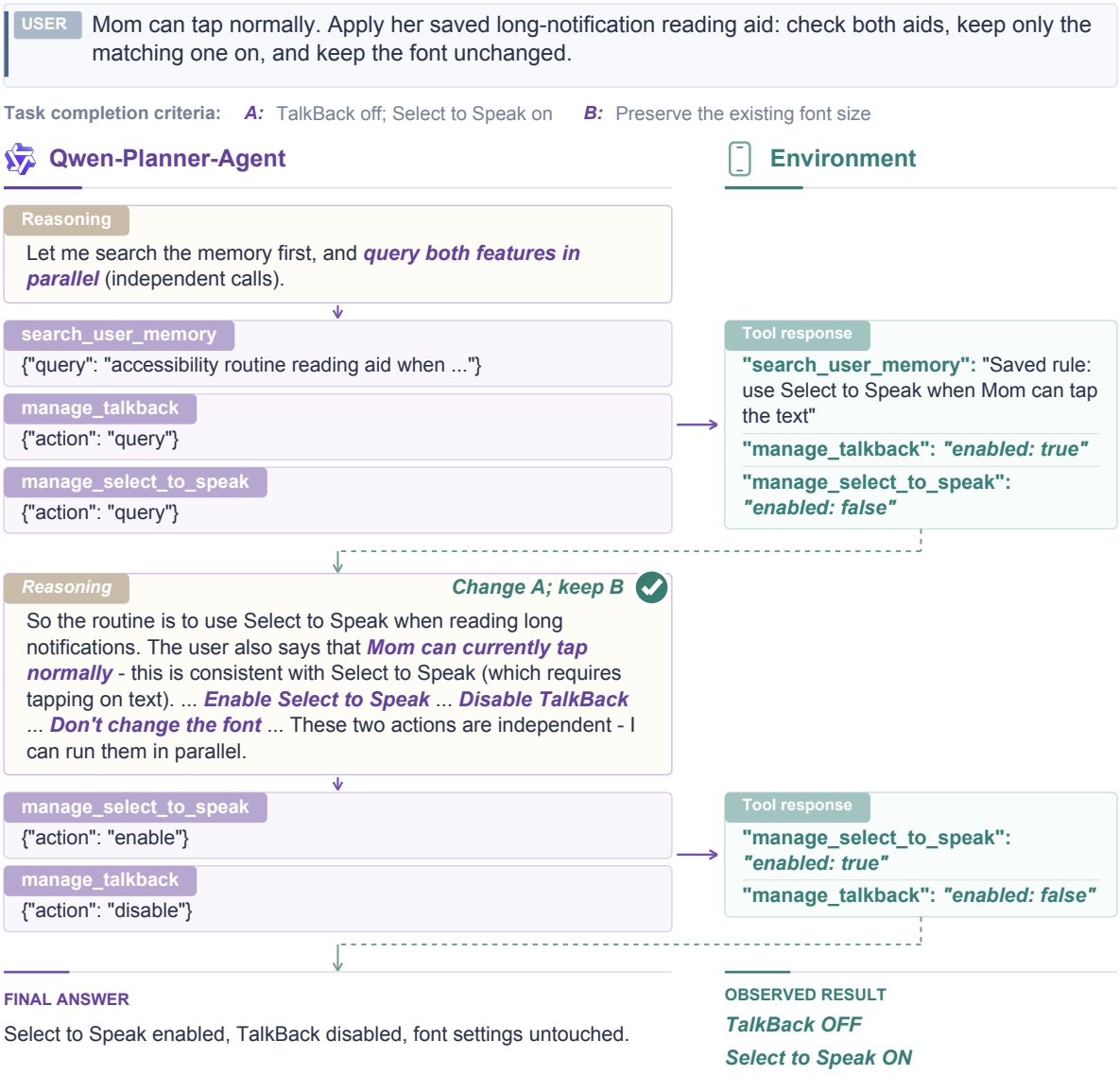

Figure 16: Memory-grounded correction within an explicit scope. Qwen-Planner-Agent observes both reading-aid states, enables Select to Speak, and disables TalkBack (A). It leaves the font setting unchanged (B), realizing the stored preference with only the required state changes.

Figure 17: Coordinated revision of persistent memory. Qwen-Planner-Agent updates records 0427 and 0429 for the new Wi-Fi and USB fallback (A), while retaining the large-attachment rule in 0428 (B). The same record identifiers connect retrieval to the two targeted writes; the final memory state confirms that the third record is unchanged.

C.3. Constraint Use Across Dialogue Turns

Retaining task constraints across topic changes. Figure 18 places four intervening turns between a shared weekday-only alarm requirement and a later request specifying just two times. Qwen-Planner-Agent applies the earlier constraint (H1) to both creation calls. The returned records confirm weekday repetition, showing that the constraint remains effective across topic changes without being restated.

Figure 18: Constraint-preserving tool use across dialogue turns. H1 links the earlier weekday-only requirement to two alarm requests after four intervening topics. Qwen-Planner-Agent carries the retained constraint into both tool calls, and both returned records confirm weekday repetition.

Resolving dialogue references for targeted state revisions. In Figure 19, the user asks to disable "the middle one" after establishing three interception rules. Qwen-Planner-Agent combines interaction order with the international-delivery explanation to identify international-call interception (H2). The final state confirms that only this rule is disabled, preserving one-ring-call and marketing-message interception (H1 and H3).

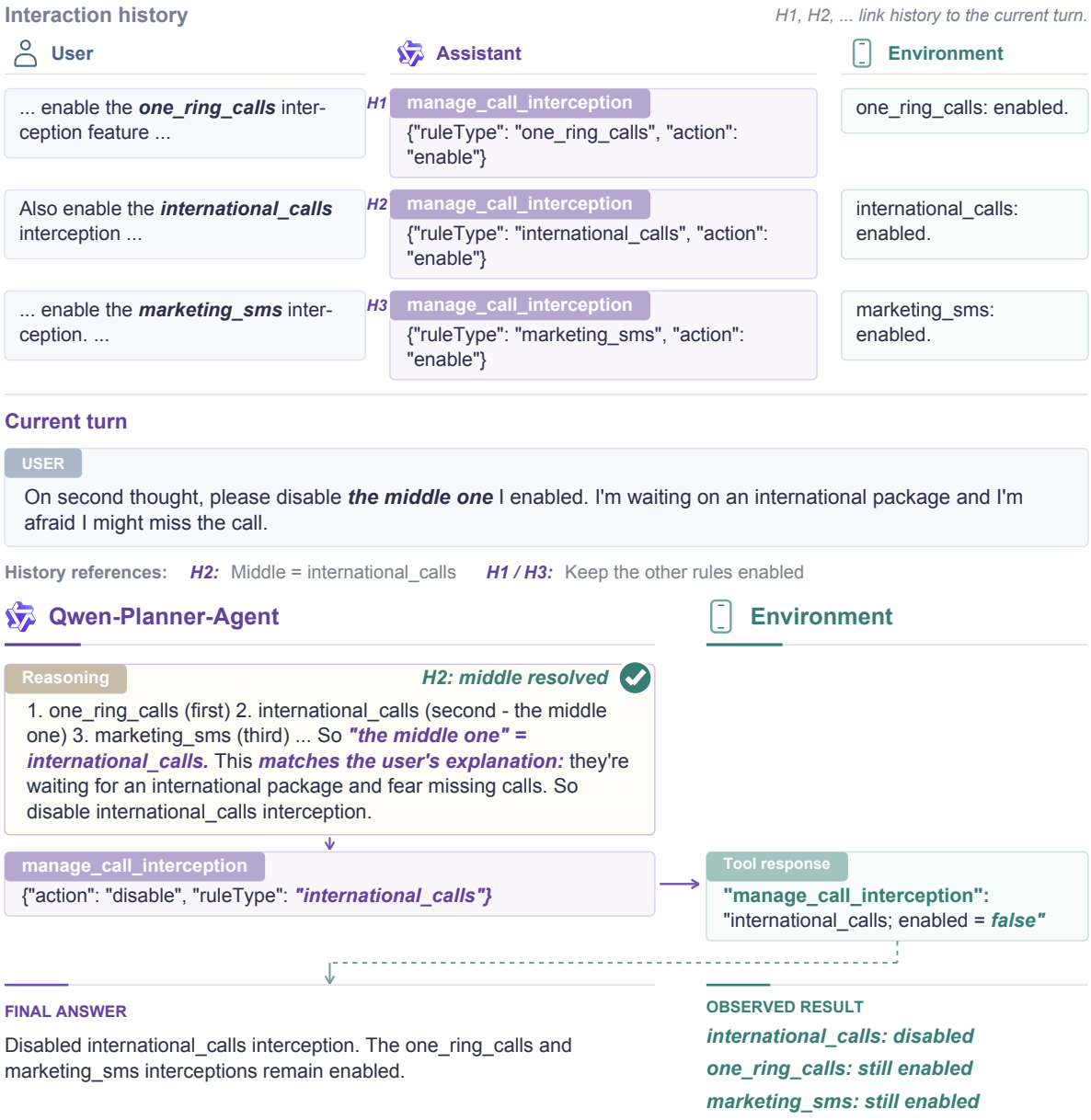

Figure 19: Reference-grounded tool use over accumulated device state. Qwen-Planner-Agent resolves "the middle one" as international-call interception (H2), consistent with the user's explanation. It disables only that rule while preserving one-ring-call and marketing-message interception (H1 and H3).