BENCHMARKING GENERAL MOBILE ASSISTANTS IN CHALLENGING REAL-WORLD SCENARIOS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graphical user interfaces provide a challenging setting for evaluating agents that must perceive, reason, and act over interaction trajectories. Existing mobile benchmarks have established reproducible Android evaluation environments, but there remains room to broaden application coverage and capture more demanding user workflows. We present GMA, a benchmark for evaluating general mobile assistants in challenging real-world scenarios across seven custom applications adapted from open-source projects and 300 tasks spanning four progressive difficulty tiers, from atomic interactions to realistic long-horizon workflows. Evaluation of nine frontier models under a shared agent setup reveals a sharp decline in task completion as execution demands increase, with agents increasingly making partial progress without completing the full workflow. We further find that preserving interaction history substantially improves performance, especially on demanding tasks, while retaining increasingly more raw context eventually yields diminishing returns. Motivated by these results, we use GMA as a controlled testbed for mobile harness design, covering context management, within-task memory, planning, and verification. Our experiments show that harness effectiveness is highly conditional. The effect of a given design varies with task demands, alternative implementations within the same module can yield substantially different outcomes, composition can diminish or even reverse marginal gains, and the same design can affect foundation models differently. Overall, GMA provides a reproducible testbed for mobile agent evaluation and a controlled setting for understanding how foundation models and their surrounding harness jointly shape reliable execution in complex mobile workflows.

1 INTRODUCTION

Recent advances in large language models (LLMs) have enabled agents to move beyond static language tasks and interact autonomously with external environments (Wei et al., 2025; Merrill et al., 2026; Huang & Jiang, 2026). Graphical user interfaces (GUIs) (Xie et al., 2024; Yuan et al., 2026), and mobile devices in particular (Rawles et al., 2024; Kong et al., 2026), provide an important setting for evaluating such capabilities. Mobile agents must perceive visual interfaces, translate user intentions into grounded actions, and adapt to changing application states. Realistic workflows further require long-horizon planning, information retrieval, constraint satisfaction, and the preservation of intermediate information and task progress (Li et al., 2025; Shi et al., 2026). As trajectories grow longer, local mistakes, forgotten information, and incomplete progress can accumulate and compromise the overall workflow. These characteristics make mobile use a challenging testbed not only for basic interaction capability, but also for reliable execution over extended tasks.

Existing benchmarks have established strong foundations for executable mobile agent evaluation. AndroidWorld (Rawles et al., 2024) provides a reproducible Android environment with dynamically instantiated tasks and state-based verification, while MobileWorld (Kong et al., 2026) extends this framework with additional applications, longer cross-application workflows, agent-user interaction, and MCP-augmented tasks. Building on this progress, we explore two directions in which mobile agent evaluation can be further expanded: broader application coverage spanning more functional domains and interaction patterns, and more demanding tasks that combine multi-step execution, cross-application dependencies, information synthesis, and realistic user constraints. Importantly,

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

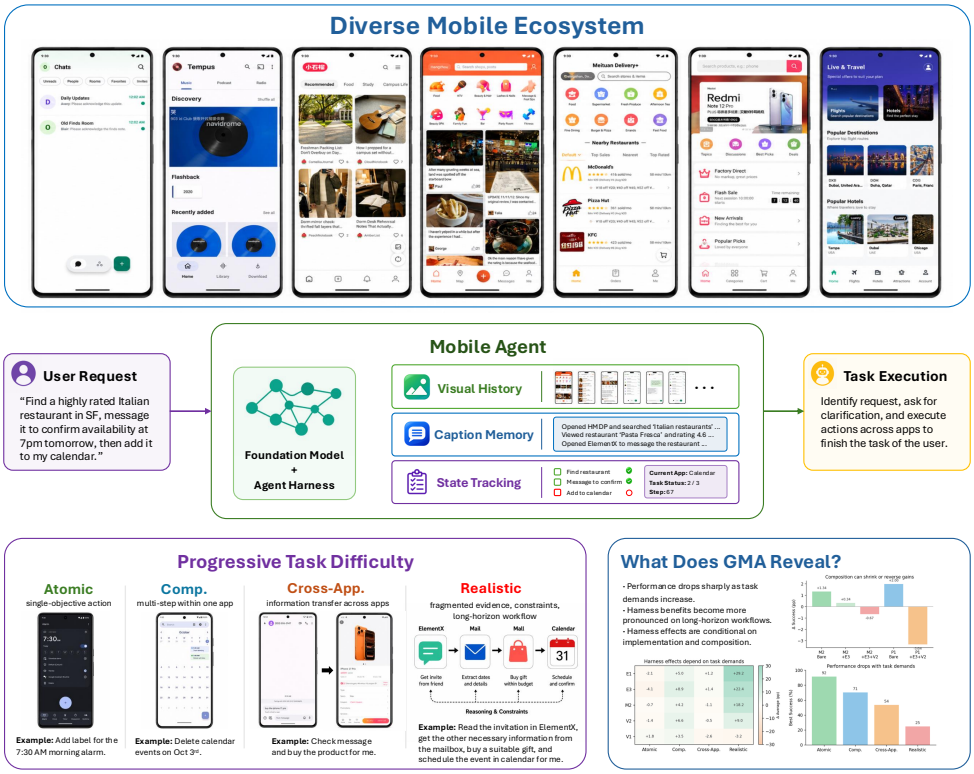


Figure 1: **Overview of GMA**. GMA provides a diverse and reproducible Android ecosystem with seven custom applications and 300 tasks across four progressively demanding tiers, from Atomic interactions to Realistic workflows. We evaluate nine frontier models under a shared agent setup and further use GMA as a controlled testbed for studying harness design across context management, memory, planning, and verification. Results reveal substantial performance degradation as task demands increase and show that harness effects are highly conditional on task demands, implementation choices, composition, and the underlying foundation model.

this progression provides a common axis for examining how agent behavior changes as execution demands increase, rather than assessing complex workflows in isolation from simpler tasks.

To this end, we present GMA, a benchmark for evaluating general mobile assistants in challenging scenarios. We develop seven custom applications from open-source projects, covering messaging, media management, lifestyle sharing, local services, delivery, e-commerce, and travel, and integrate them into a reproducible self-hosted Android environment. Across these applications, we construct 300 tasks organized into four progressive difficulty tiers: Atomic, Compositional, Cross-Application, and Realistic. The tasks progress from basic interface operations to long-horizon workflows involving fragmented information, multiple applications, and interdependent requirements. Each task is decomposed into independently verifiable subtasks, enabling both strict task-level success measurement and fine-grained analysis of partial progress when the workflow is unfinished.

We evaluate nine frontier models on GMA under a shared agent setup. The results reveal a systematic degradation as task demands increase. Models that perform strongly on Atomic and Compositional tasks become substantially less reliable on Cross-Application and Realistic workflows, while the gap between partial progress and full task success also grows. Relative model performance also shifts across difficulty tiers, indicating that strong performance on simpler interactions does not fully characterize reliability on longer, more interdependent workflows. We additionally find that access to interaction history substantially affects performance, especially on demanding tasks, although simply retaining more raw context eventually exhibits diminishing returns.

Building on these observations, we use GMA to study how harness design affects mobile agent execution. Rather than comparing complete agent systems, we organize recurring mobile agent designs

into four functional modules: context management, within-task memory, planning, and verification, and evaluate alternatives under a fixed model and environment across task demands. The resulting analysis shows that harness effectiveness is highly conditional. A given design can vary with task demands, alternative implementations within the same module can yield substantially different outcomes, and marginal gains can shrink or reverse under composition. We further find that the same harness design can produce different effects across underlying models, both in overall magnitude and across task demands. Together, these results position GMA as both a broader benchmark for evaluating general mobile assistants and a controlled testbed for studying how foundation models and their surrounding harness jointly shape reliable execution in complex mobile workflows.

2 Related Work

2.1 GUI and Mobile Evaluation

Agent evaluation has expanded from static prediction toward interactive execution across web navigation (Yao et al., 2022a; Zhou et al., 2023), software engineering (Jimenez et al., 2024; Deng et al., 2025; Yang et al., 2026), information research (Wei et al., 2025; Du et al., 2025), and workspace automation (Tang et al., 2026; Shepard & Salimans, 2026). GUI environments further require agents to perceive visual content, ground instructions to interface elements, and adapt their actions to changing application states. Early benchmarks such as ScreenSpot (Cheng et al., 2024) and Android in the Wild (Rawles et al., 2023) focus on visual grounding or action prediction from offline trajectories, while OSWorld (Xie et al., 2024) and AndroidWorld (Rawles et al., 2024) evaluate end-to-end interaction in executable desktop and mobile environments with programmatic verification. This shift towards closed-loop evaluation makes it possible to study not only individual action accuracy, but also how errors and intermediate state accumulate over extended execution.

Recent mobile benchmarks extend this setting along complementary dimensions. MobileWorld (Kong et al., 2026) introduces longer cross-application workflows, agent-user interaction, and MCP-augmented tasks. MobileBench-OL (Wu et al., 2026b) studies online execution and robustness across Chinese applications. PSPA-Bench (Nie et al., 2026) focuses on personalized assistance. AndroidDaily (Sui et al., 2026) evaluates closed-source commercial applications. MobileGym (Wu et al., 2026a) supports scalable evaluation and training in simulated mobile environments. iOSWorld (Jang et al., 2026) extends interactive evaluation to iOS. These benchmarks broaden evaluation along dimensions such as application coverage, personalization, realism, and scalability. GMA complements these efforts by expanding the reproducible Android application ecosystem and organizing tasks into progressively more demanding tiers, providing a shared setting for examining not only overall capability but also how failure patterns emerge as execution complexity increases.

2.2 Agent Harness

Agent systems have long relied on mechanisms surrounding the foundation model to organize interaction, manage context, and connect models with external environments, and these mechanisms have recently been discussed collectively as an agent harness (Anthropic, 2025a;b). Earlier systems such as ReAct (Yao et al., 2022b), Reflexion (Shinn et al., 2023), SWE-agent (Yang et al., 2024), and OpenHands (Wang et al., 2025a) already incorporated different forms of reasoning, memory, feedback, and environment interfaces. More recent work explicitly studies harness design and optimization (OpenAI, 2026; Lin et al., 2026; Team, 2026), highlighting that agent performance depends on both the foundation model and the surrounding execution system.

For GUI agents, existing harnesses incorporate mechanisms for context management, planning, memory, and verification. HiconAgent (Zhou et al., 2026) compresses interaction history. MobileAgent-v3.5 (Xu et al., 2026) distributes planning, execution, verification, and memory across specialized roles. PhoneHarness (Li et al., 2026) coordinates GUI and structured tool actions within a unified execution loop. TSR (Zheng et al., 2026) explicitly maintains task requirements, progress, and execution state over long trajectories. These works demonstrate the value of different harness mechanisms, but they are typically evaluated within complete systems that also differ in foundation models, interfaces, and execution settings. Our analysis instead organizes recurring mechanisms into a shared functional design space and compares alternative implementations under a fixed model and environment across progressively increasing task demands.

3 GMA ENVIRONMENT

3.1 APPLICATIONS AND INFRASTRUCTURE

GMA extends the reproducible Android evaluation environments established by AndroidWorld (Rawles et al., 2024) and MobileWorld (Kong et al., 2026) with seven custom applications adapted from fully functional open-source projects. They cover instant messaging (**ElementX**), music management (**Tempus**), lifestyle sharing (**XiaoShiLiu**), local discovery (**HMDP**), food delivery (**Meituan**), e-commerce (**Mall**), and travel planning (**Travel**). These applications introduce substantially different interfaces and interaction patterns, ranging from threaded communication and social content to multi-stage checkout, product comparison, and constraint-heavy travel booking. We additionally retain standard Android system applications together with Mattermost and Mastodon from MobileWorld, forming a heterogeneous ecosystem in which tasks can transfer information and state across functionally distinct applications.

Rather than relying on simplified mock interfaces or external production services, we preserve the original interaction flows of these applications while decoupling them from their public APIs and self-hosting the required backend services and datasets. This design provides realistic interfaces while allowing application state to be deterministically initialized and reproduced across runs. Building on the containerized emulator and snapshot infrastructure of AndroidWorld and MobileWorld, we coordinate emulator restoration with backend reset, cache clearance, session invalidation, and task-specific data injection so that both frontend and backend state are restored consistently.

To support task construction and evaluation across heterogeneous applications, we introduce a unified declarative *asset* abstraction. An asset represents a typed piece of task state, such as a contact, message, calendar event, social post, e-commerce order, or travel booking. Tasks specify their initial conditions as collections of assets, which are materialized through application-specific state adapters, while desired outcomes are expressed using the same representation. This separates task semantics from application-specific storage schemas and allows state from different applications to be composed within a single workflow. Based on these assets, we implement reusable programmatic verification criteria for checking whether relevant entities exist, are modified, or are removed. Multiple criteria can be combined to evaluate both complete task success and intermediate progress. For text-entry and answer-submission tasks, we use deterministic matching rules specified by the task contract rather than LLM-based judging.

3.2 TASK DESIGN AND ANNOTATION

GMA contains 300 tasks organized into four tiers with progressively increasing execution demands. **Atomic** tasks isolate single-objective operations with explicitly defined target states, primarily testing GUI perception and control. **Compositional** tasks chain multiple operations, typically within one application, and introduce short-horizon planning, information retrieval, and state tracking. **Cross-Application** tasks introduce explicit dependencies in which information obtained from one application determines actions in another, requiring reliable information transfer across applications. **Realistic** tasks further combine long-horizon execution with fragmented evidence, competing constraints, distractors, and underspecified requirements. For example, one workflow requires the agent to infer a product category from an ElementX poll, retrieve a budget constraint from Mail, compare eligible products in Mall, and finally schedule the associated event in Calendar. The final benchmark contains 73 Atomic, 129 Compositional, 78 Cross-Application, and 20 Realistic tasks.

Each task is decomposed into independently verifiable subtasks corresponding to concrete milestones in the user request, such as retrieving a required piece of information, completing an order, or creating a calendar event. This decomposition allows GMA to distinguish complete success from partial progress when an agent completes only part of a long workflow, which is particularly important for diagnosing failures on complex tasks. Task quality is controlled through a dual-annotator review process conducted on our annotation platform. Annotators directly operate the Android emulator to execute each task and verify that the intended workflow is feasible under the specified initial state. They additionally check the semantic clarity of the user instruction, the appropriateness of its assigned difficulty tier, and the consistency between the intended requirement, its subtask decomposition, and the corresponding programmatic verification criteria. This process helps ensure that tasks are both executable in practice and correctly specified for deterministic evaluation.

Table 1: Overall and difficulty-specific performance on GMA. We report strict task success rate (*Success*) and the average proportion of completed subtasks (*Average*). The best result is highlighted in bold and the second-best is underlined.

Models	Overall		Atomic		Compositional		Cross-Application		Realistic	
	Success	*Average*	*Success*	*Average*	*Success*	*Average*	*Success*	*Average*	*Success*	*Average*
Doubao-Seed-2.1-pro	61.33	75.10	**91.78**	**92.47**	63.57	79.61	43.59	64.17	5.00	25.31
Qwen3.7-Plus	49.33	63.78	78.08	80.37	51.94	69.78	30.77	51.87	0.00	11.00
Qwen3.8-Max	**67.33**	**79.43**	**91.78**	**92.47**	**70.54**	<u>83.09</u>	**53.85**	**71.15**	10.00	40.64
Gemini 3.5 Flash	51.33	63.53	79.45	80.14	58.91	73.62	25.64	44.27	0.00	12.97
Gemini 3.7 Flash	<u>65.33</u>	<u>79.17</u>	<u>90.41</u>	<u>91.10</u>	<u>69.77</u>	**83.74**	<u>47.44</u>	<u>68.62</u>	<u>15.00</u>	<u>47.31</u>
GPT-5.5	55.67	68.52	82.19	82.19	62.79	77.13	33.33	56.44	0.00	10.14
GPT-5.6 Sol	61.33	75.12	87.67	88.36	<u>69.77</u>	81.19	35.90	59.98	10.00	46.73
Claude Opus 4.7	58.67	71.42	83.56	84.25	65.12	77.43	33.33	55.42	**25.00**	**48.31**
Claude Opus 4.8	57.67	70.87	86.30	86.99	65.12	78.94	30.77	49.81	10.00	42.10

4 Evaluation on GMA

4.1 Experimental Settings

Agent Setup. We evaluate each foundation model using the same zero-shot, vision-only autoregressive agent framework. At each interaction step, the model observes the current raw screenshot together with the retained interaction context, without access to XML accessibility trees or other structural UI information. The surrounding baseline harness preserves the complete textual history of previous reasoning and actions and retains the two most recent screenshots, without introducing explicit task-state memory, planning, or verification mechanisms.

Interaction Setup. The action space is adapted from MobileWorld (Kong et al., 2026), with `call_user` for requesting clarification, `answer` for submitting textual responses, and `status` for explicitly declaring task completion. All evaluated models interact with the same environment and underlying action space, while model-specific wrappers accommodate differences in tool-call format. Each episode has a maximum budget of 150 interaction steps and terminates when the agent issues `status` or `answer`, or when the step limit is reached.

User Simulation and Evaluation. We use Qwen3.7-Plus (Qwen Team, 2026a) as an automated user simulator under a task-specific contract that restricts it to information assigned to the simulated user and prevents unsolicited operational guidance. We report two complementary metrics based on programmatic subtask verification: *Success* requires all subtasks of a task to be satisfied, while *Average* measures the proportion of completed subtasks and therefore captures partial progress when the full workflow is not completed.

4.2 Main Results

We evaluate nine frontier models from five model families: ByteDance Seed (ByteDance Seed, 2026), Alibaba Qwen (Qwen Team, 2026a;b), Google DeepMind (Google DeepMind, 2026a;b), OpenAI (OpenAI, 2026a;b), and Anthropic (Anthropic, 2026a;b). Table 1 reports performance overall and across the four difficulty tiers.

Performance degrades systematically as task demands increase. The best achieved *Success* rate decreases from 91.78 on Atomic tasks to 70.54 on Compositional tasks, 53.85 on Cross-Application tasks, and only 25.00 on Realistic tasks. The corresponding best *Average* scores decline from 92.47 to 83.74, 71.15, and 48.31. The gap between *Success* and *Average* also becomes more pronounced as tasks grow more complex, indicating that agents increasingly make partial progress without sat-

isfying the complete set of requirements. These results reveal a systematic reliability degradation as mobile tasks move from local interactions toward longer workflows involving information transfer, cross-application dependencies, and multiple constraints.

Model strengths diverge as task demands change. No single model consistently dominates across all difficulty tiers. Qwen3.8-Max achieves the strongest overall performance and the highest *Success* on Compositional and Cross-Application tasks, while Claude Opus 4.7 performs best on Realistic workflows, reaching 25.00 *Success* and 48.31 *Average*. Gemini 3.7 Flash also remains competitive on Realistic tasks with 15.00 *Success* and 47.31 *Average*, despite trailing Qwen3.8-Max on Cross-Application tasks. These shifts indicate that strong performance on shorter or moderately complex mobile interactions does not fully characterize how models behave when workflows require sustained reasoning, coordination, and state maintenance over longer horizons.

4.3 Task Completion Awareness

A reliable mobile agent must determine not only how to complete a task, but also whether the requested workflow has actually been completed. Because GMA verifies task state after every interaction step, we can distinguish whether an agent fails to recognize genuine success from whether it incorrectly declares completion before all requirements are satisfied. We define an episode as *Ever-Pass* if it reaches full success at any point during execution.

Table 2: Termination behavior of episodes that reach full success at least once. *Explicit* denotes termination through an explicit completion or answer action, while *Cap* denotes reaching the maximum step limit.

Model	*Ever-Pass*	*Final*	*Explicit*	*Cap*
Doubao-2.1	185	184	183	2
Qwen3.7	148	148	147	1
Qwen3.8	204	202	204	0
Gemini-3.5	157	154	157	0
Gemini-3.7	197	196	197	0
GPT-5.5	168	167	167	1
GPT-5.6	189	184	189	0
Claude 4.7	181	176	180	1
Claude 4.8	176	173	175	1
Total	**1,605**	**1,584**	**1,599**	**6**

Agents are effective at recognizing success once it has been reached. Of the 1,605 *Ever-Pass* episodes, 1,584 remain successful at final termination, and the median delay from first success to explicit termination is only one step. However, the opposite direction is substantially less reliable. Among 1,899 explicit completion declarations, 428 occur before all task requirements are satisfied. Thus, the dominant completion-related failure is not continuing unnecessarily after success, but declaring completion prematurely before the full workflow has been satisfied.

4.4 Performance Across Applications

Figure 2 shows substantial performance variation across applications. Travel is particularly challenging, with an *Average* score of only 24.38, while XiaoShiLiu, Mall, and ElementX also fall toward the lower end of the range. In contrast, other newly introduced applications such as Meituan, Tempus and HMDP achieve considerably stronger results. Thus, broader application coverage does not uniformly increase difficulty, but several of the weakest-performing domains arise from applications newly covered by GMA, indicating that current agents do not yet generalize reliably across diverse mobile interfaces and workflows.

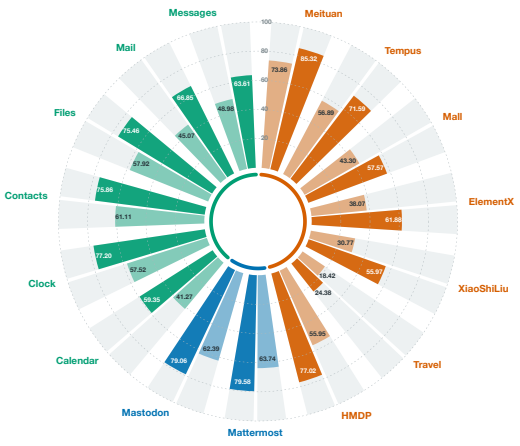


Figure 2: Performance across applications averaged over evaluations. Light and dark bars denote *Success* and *Average*, respectively. Orange denotes applications introduced in GMA, blue denotes applications inherited from MobileWorld, and green denotes Android system applications.

4.5 Effect of Context Retention

The baseline harness preserves the full textual trajectory but only the two most recent screenshots. We further examine how sensitive agent

performance is to these context retention choices. As shown in Table 3, expanding the retained visual history from one to eight screenshots increases *Success* from 43.33% to 52.00% and *Average* from 57.20 to 67.10. The benefit becomes particularly pronounced on more demanding tasks, with the Realistic *Average* increasing from 9.97 to 21.44. Retaining sixteen screenshots provides no further overall gain, with *Average* decreasing to 65.52, although Realistic performance continues to increase slightly to 23.81. Effects of textual trajectory history is also clear that under the two-screenshot setting, removing previous reasoning traces decreases *Success* from 49.33% to 41.67% and *Average* from 63.78 to 55.77. These results show that retaining interaction history can substantially affect model performance, especially on demanding workflows, while simply increasing the amount of raw context eventually exhibits diminishing returns.

5 Analyzing Harness Effects with GMA

Section 4 shows that preserving interaction history can substantially improve model performance, especially on more demanding workflows, but that simply retaining more raw context eventually yields diminishing returns. This motivates us to look beyond raw history retention to harness mechanisms that explicitly organize task-relevant information, track progress, guide future actions, and verify execution. We therefore use GMA to study how these mechanisms affect mobile agent performance and how their effects vary with task demands.

Table 3: Effect of context retention on Qwen3.7-Plus. **Images** denotes the number of most recent screenshots retained, and **Thought** indicates whether previous reasoning traces are preserved.

Images	Thought	Overall		Difficulty			
		Success	*Average*	*Atomic*	*Comp.*	*Cross-App.*	*Realistic*
1	✓	43.33	57.20	72.83	61.67	47.30	9.97
2	✓	49.33	63.78	80.37	69.78	51.87	11.00
2	✗	41.67	55.77	80.37	57.13	43.03	6.89
4	✓	51.00	65.53	80.37	71.89	54.64	12.86
8	✓	52.00	67.10	83.56	72.32	54.77	21.44
16	✓	52.00	65.52	79.68	70.95	53.97	23.81

GMA is particularly suitable for this analysis because its four task tiers span progressively increasing execution demands within the same reproducible application ecosystem and evaluation protocol, while fine-grained subtask verification reveals partial progress even when the full task fails. This provides a controlled axis for examining how the effects of harness mechanisms change as task demands increase. To isolate these effects, we keep the foundation model, environment, prompting interface, and action space fixed rather than comparing end-to-end agent systems, whose performance differences may arise from many factors simultaneously.

5.1 A Controlled Harness Design Space

To enable controlled comparisons, we organize recurring mechanisms in existing mobile agents into four modules: context management (E), within-task memory (M), planning (P), and verification (V). For context management, E_0 retains only a recent visual window, while E_1, E_2, and E_3 preserve older interactions through interface captions, action-linked state summaries, and recursive trajectory compaction, respectively. For memory, M_1 maintains free-form task state while M_2 organizes requirements, progress, intermediate information, and the current subgoal into structured fields. Planning ranges from a fixed initial plan (P_1), to a revisable rolling plan (P_2), to hierarchical planning (P_3), while verification operates either locally after individual transitions (V_1) or periodically at the trajectory level (V_2). Table 4 shows how representative mobile harnesses instantiate different subsets and implementations of these four functions.

Table 4: Mapping representative mobile harnesses onto the four functional modules. "–" indicates no corresponding mechanism.

Method	Context (E)	Memory (M)	Planning (P)	Verification (V)
Mobile-Agent-v2 (Wang et al., 2024)	–	M_1	P_2	V_1
Mobile-Agent-E (Wang et al., 2025b)	–	M_1	P_3	V_1
HiconAgent (Zhou et al., 2026)	E_3	–	–	–
MobileUse (Li et al., 2025)	–	M_1	–	$V_1 + V_2$
TSR (Zheng et al., 2026)	–	M_2	–	V_1
ReInAgent (Jia et al., 2025)	–	M_2	P_2	V_2
AndroTMem (Shi et al., 2026)	–	M_2	–	–

We evaluate these designs with Qwen3.7-Plus while keeping the foundation model, environment, prompting interface, and action space fixed. The basic harness (E_0, M_0, P_0, V_0) is a lightly adapted variant of the baseline agent setup used in Section 4, preserving the same core pipeline while making small adjustments to fit the modular design space. Consequently, its absolute results differ slightly from those of the main experiment. We use this configuration as the common reference point for controlled comparisons. Complete results for individual designs and evaluated compositions are reported in Appendix D.

5.2 Harness Effects Depend on Task Demands

The effect of a harness design varies substantially with the execution demands of the task, and this dependence is especially pronounced on Realistic workflows. As shown in Figure 3, replacing discarded screenshots with interface captions E_1 decreases *Average* by 2.06 on Atomic tasks, but improves it by 4.99, 1.23 and 29.25 respectively on the other three tiers. Other designs show a similar pattern. Structured task-state memory M_2 provides limited benefit on the first three tiers but improves Realistic *Average* by 18.20, while recursive trajectory compaction E_3 yields a 22.42-point gain on the same tier. These results indicate that certain harness designs become particularly valuable when the task requires maintaining information and progress across longer trajectories, fragmented evidence, and interdependent requirements.

Increasing task demands can also change the direction of an effect rather than merely its magnitude. Transition-level verification V_1 improves *Average* by 1.82 on Atomic tasks and 3.52 on Compositional tasks, yet reduces performance by 2.57 and 3.16 on Cross-Application and Realistic tasks, respectively. The opposite pattern appears for recursive trajectory compaction E_3, which decreases Atomic performance by 4.11 but improves all three more demanding tiers, with the largest gain of 22.42 on Realistic tasks. These results suggest that effects observed on simpler tasks do not necessarily predict how the same design will behave on more demanding workflows. By evaluating the same harness designs across progressively harder tasks within a shared environment, GMA makes such shifts directly observable rather than collapsing them into a single aggregate score.

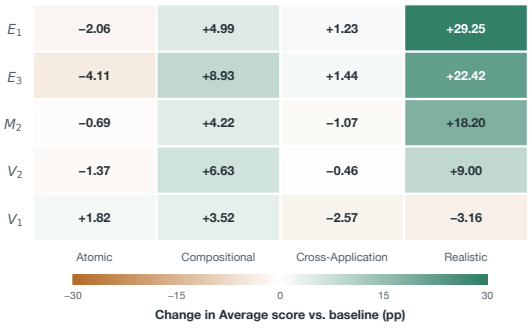


Figure 3: Difficulty-dependent effects of representative harness designs. Each cell reports the change in *Average* relative to the baseline.

5.3 Harness Effects Are Conditional

Different designs within the same harness module can produce substantially different effects. As already shown in Figure 3, the two verification designs exhibit different behaviors across task demands. Transition-level verification V_1 improves Atomic and Compositional performance but degrades Cross-Application and Realistic performance, while trajectory-level verification V_2 provides its largest gains on Compositional and Realistic tasks. Similar differences appear for other modules. Structured memory M_2 improves the overall *Average* by 2.58, while free-form memory M_1 decreases it by 1.91. Recursive context compaction E_3 improves *Average* by 4.71, compared with a 1.22-point gain from action-linked summaries E_2. These results show that the module category alone is insufficient to characterize harness effectiveness, as different designs within the same module can lead to substantially different outcomes.

Harness mechanisms also interact rather than contributing independently. As shown in Figure 4(a), the marginal gains of individual designs can diminish as other forms of harness support are introduced. Adding M_2 to the bare harness improves *Success* by 1.34, but the effect falls to 0.34 when E_3 is already present and to -0.67 after $E_3 + V_2$. Similarly, the gain from V_2 decreases from 4.00 on the bare harness to 1.34 after E_3 and 0.33 after $E_3 + M_2$. Composition can also reverse the direction of an effect. As shown in Figure 4(b), P_1 improves *Success* by 2.00 on the bare harness but reduces it by 3.34 after $E_3 + V_2$, while V_1 changes from a 2.00-point gain in isolation to a 1.34-point

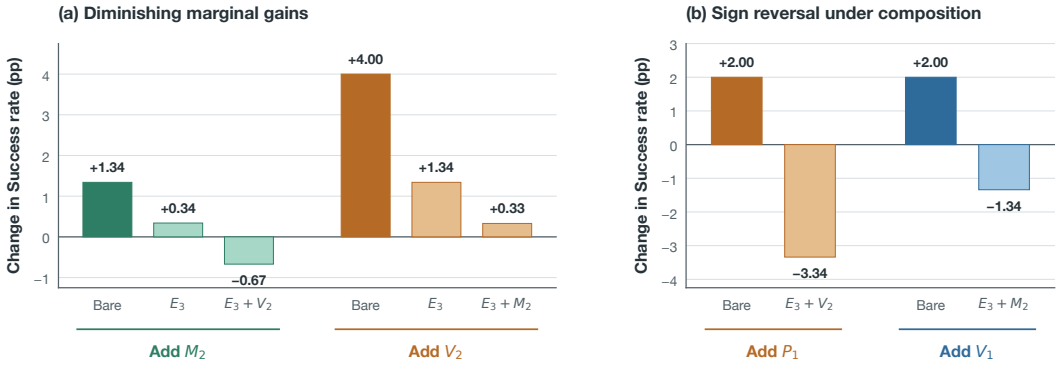


Figure 4: Composition effects under different surrounding harness configurations. (a) Change in *Success* when adding M_2 and V_2 to progressively stronger configurations. (b) Change in *Success* when adding P_1 or V_1 to the bare harness and stronger configurations.

decrease after $E_3 + M_2$. Together, these results show that the marginal effect of a harness design depends on the surrounding configuration, and its behavior in isolation does not necessarily predict its contribution under composition.

Finally, we examine whether these harness effects generalize across foundation models by additionally evaluating M_1 and M_2 with GPT-5.6 Sol. As shown in Table 5, M_2 improves overall performance for both models, but its effects are distributed differently across task demands. For Qwen3.7-Plus, M_2 increases *Average* from 62.56 to 65.14, with the largest gain on Realistic tasks from 7.72 to 25.92. For GPT-5.6 Sol, the overall improvement is smaller, from 74.56 to 75.11 on the *Average* score. Conversely, M_1 decreases overall *Average* for both models, but its effect is considerably stronger for GPT-5.6 Sol, whose overall *Average* score drops from 74.56 to 70.91. These results indicate that the same harness design can produce different trade-offs across task demands depending on the underlying foundation model.

Table 5: Effect of M_1 and M_2 across different foundation models.

Model	Memory	Overall		Difficulty			
		Success	*Average*	*Atomic*	*Comp.*	*Cross-App.*	*Realistic*
Qwen	None	47.33	62.56	83.11	65.24	52.96	7.72
	M_1	47.67	60.65	78.31	66.67	46.06	14.19
	M_2	48.67	65.14	82.42	69.46	51.89	25.92
GPT	None	62.33	74.56	89.95	82.68	59.38	25.14
	M_1	57.67	70.91	83.11	79.07	52.13	47.02
	M_2	64.33	75.11	87.67	80.79	60.10	51.20

Taken together, these analyses show that harness effectiveness is highly conditional. Different designs within the same module can produce substantially different outcomes, and the marginal contribution of a mechanism can shrink or even reverse once other forms of support are present. The same design can also affect foundation models differently, both in overall magnitude and in how gains and losses are distributed across task demands. By exposing these variations within a shared environment, GMA provides a controlled setting for studying when and how harness design contributes to reliable execution of mobile agents.

6 Conclusion

We present GMA, a benchmark for evaluating general mobile assistants across seven custom applications and 300 tasks with progressively increasing execution demands. Evaluation of nine frontier models under a shared agent setup reveals a systematic decline from simple interactions to complex workflows, with agents increasingly making partial progress without completing all requirements. We further show that preserving interaction history is especially important on demanding tasks, while retaining more raw context eventually yields diminishing returns. Using GMA as a controlled testbed for harness analysis, we find that harness effectiveness is highly conditional. Effects vary with task demands and implementation choices, can shrink or reverse under composition, and differ across foundation models. Together, these results establish GMA as both a broader reproducible benchmark and a controlled setting for understanding how foundation models and their surrounding harness jointly shape reliable execution in complex mobile workflows.

AI USE STATEMENT

AI tools are used in this work in three limited roles: assisting task construction, retrieving related work, and polishing the manuscript. For task construction, the authors design the task requirements and annotation criteria, and use AI to assist in generating task code according to these specifications. AI-generated tasks are not directly included in the benchmark. Their design and implementation are reviewed by the authors, and the finalized tasks further undergo the dual-annotator quality control process described in the paper to verify feasibility, semantic clarity, difficulty assignment, and consistency between task requirements, subtask decomposition, and programmatic verification.

AI is also used to assist in retrieving potentially relevant prior work. All retrieved references are manually reviewed by the authors to verify their existence, relevance, and accurate characterization before inclusion in the manuscript. Finally, AI tools are used for language polishing and improving the clarity of the presentation. The authors remain responsible for the substantive writing, technical claims, interpretation of results, and final wording of the paper. In all cases, AI serves as an assistive tool, while the authors retain responsibility for reviewing and validating the resulting content.

ETHICS STATEMENT

GMA is designed for evaluating mobile agents in a controlled and reproducible environment. The applications used in the benchmark are adapted from open-source projects and decoupled from external production services, with the required backend services and datasets hosted locally. Consequently, benchmark execution does not interact with real users or live commercial services, and actions such as messaging or booking are performed only within the evaluation environment.

Mobile agents may eventually operate in real user-facing applications and perform consequential actions involving communication, personal information, purchases, or reservations. Failures in such settings could lead to unintended actions or privacy and security risks. GMA evaluates task execution capability in a controlled environment and should therefore not be interpreted as establishing the safety or readiness of an agent for real-world deployment. We encourage future work to consider appropriate safeguards, user confirmation, privacy protection, and other deployment-specific safety requirements when transferring mobile agent systems to real services.

REPRODUCIBILITY STATEMENT

We provide detailed information necessary to reproduce the benchmark construction, evaluation protocol, and harness experiments. Appendix A describes the seven custom applications, self-hosted infrastructure, task construction process, state initialization, and programmatic verification. Appendix B provides the complete user-simulator and agent prompts, together with the interaction protocol and execution settings. Appendix C specifies the modular ablation agent, including the context management, memory, planning, and verification mechanisms and their corresponding prompts. We additionally document the model settings, interaction budget, context-retention policy, task decomposition, and evaluation metrics used throughout the experiments.

REFERENCES

Anthropic. Building agents with the claude agent sdk. https://claude.com/blog/building-agents-with-the-claude-agent-sdk, September 2025a.

Anthropic. Effective harnesses for long-running agents. https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents, November 2025b.

Anthropic. System Card: Claude Opus 4.7. https://www-cdn.anthropic.com/037f06850df7fbe871e206dad004c3db5fd50340/Claude%20Opus%204.7%20System%20Card.pdf, April 2026a.

Anthropic. System Card: Claude Opus 4.8. https://www-cdn.anthropic.com/0b4915911bb0d19eca5b5ee635c80fef830a37ea.pdf, May 2026b.

ByteDance Seed. Seed2.1 Officially Released: Advancing AI Productivity. https://seed.bytedance.com/en/blog/seed2-1-officially-released-advancing-ai-productivity, June 2026.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.505.

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?, 2025. URL https://arxiv.org/abs/2509.16941.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. *arXiv preprint*, 2025.

Google DeepMind. Gemini 3.5 Flash Model Card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-5-Flash-Model-Card.pdf, May 2026a.

Google DeepMind. Gemini 3.7 Flash Model Card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-7-Flash-Model-Card.pdf, August 2026b.

Wenqi Huang and Peter Jiang. Deepswe v1.1: a cleaner, more reproducible benchmark for frontier coding agents, 2026. URL https://github.com/datacurve-ai/deep-swe.

Lawrence Keunho Jang, Mareks Woodside, Geronimo Carom, Andrew Keunwoo Jang, Jing Yu Koh, and Ruslan Salakhutdinov. iosworld: A benchmark for personally intelligent phone agents, 2026. URL https://arxiv.org/abs/2606.09764.

Haitao Jia, Ming He, Zimo Yin, Likang Wu, Jianping Fan, and Jitao Sang. Reinagent: A context-aware gui agent enabling human-in-the-loop mobile task navigation, 2025. URL https://arxiv.org/abs/2510.07988.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. Mobile-world: Benchmarking autonomous mobile agents in agent-user interactive, and mcp-augmented environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://arxiv.org/abs/2512.19432.

Chenxin Li, Zhengyao Fang, Zhengyang Tang, Pengyuan Lyu, Xingran Zhou, Xin Lai, Fei Tang, Liang Wu, Yiduo Guo, Weinong Wang, Junyi Li, Yi Zhang, Yang Ding, Huawen Shen, Sunqi Fan, Shangpin Peng, Zheng Ruan, Anran Zhang, Benyou Wang, Chengquan Zhang, and Han Hu. Phoneharness: Harnessing phone-use agents through mixed gui, cli, and tool actions, 2026. URL https://arxiv.org/abs/2606.14832.

Ning Li, Xiangmou Qu, Jiamu Zhou, Jun Wang, Muning Wen, Kounianhua Du, Xingyu Lou, Qiuying Peng, Jun Wang, and Weinan Zhang. Mobileuse: A hierarchical reflection-driven GUI agent for autonomous mobile operation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=KR6tnkb6h4.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026. URL https://arxiv.org/abs/2604.25850.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Hongyi Nie, Xunyuan Liu, Yudong Bai, Yaqing Wang, Yang Liu, Quanming Yao, and Zhen Wang. Pspa-bench: A personalized benchmark for smartphone gui agent, 2026. URL https://arxiv.org/abs/2603.29318.

OpenAI. GPT-5.5 System Card. https://deploymentsafety.openai.com/gpt-5-5/gpt-5-5.pdf, April 2026a.

OpenAI. GPT-5.6 System Card. https://deploymentsafety.openai.com/gpt-5-6/gpt-5-6.pdf, July 2026b.

OpenAI. Harness engineering: Leveraging Codex in an agent-first world. https://openai.com/index/harness-engineering/, February 2026.

Qwen Team. Qwen3.7-Plus: Multimodal agent intelligence, May 2026a. URL https://qwen.ai/blog?id=qwen3.7-plus.

Qwen Team. Qwen3.8: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control, 2023. URL https://arxiv.org/abs/2307.10088.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2024. URL https://arxiv.org/abs/2405.14573.

Daniel Shepard and Robin Salimans. Automationbench, 2026. URL https://arxiv.org/abs/2604.18934.

Yibo Shi, Jungang Li, Linghao Zhang, Zihao Dongfang, Biao Wu, Sicheng Tao, Yibo Yan, Chenxi Qin, Weiting Liu, Zhixin Lin, Hanqian Li, Yu Huang, Song Dai, Yonghua Hei, Yue Ding, Xiang Li, Shikang Wang, Chengdong Xu, Jingqi Liu, Xueying Ma, Zhiwen Zheng, Xiaofei Zhang, Bincheng Wang, Nichen Yang, Jie Wu, Lihua Tian, Chen Li, and Xuming Hu. Androtmem: From interaction trajectories to anchored memory in long-horizon gui agents, 2026. URL https://arxiv.org/abs/2603.18429.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Yifan Sui, Xin Huang, Hongbing Li, Fang Xu, Jiahe Lv, Haolong Yan, Yeqing Shen, Litao Liu, Zhimin Fan, Ziyang Meng, Jia Wang, Junbo Qi, Kaijun Tan, Zheng Ge, Xiangyu Zhang, Daxin Jiang, and Osamu Yoshie. Androiddaily: A verifiable benchmark for mobile gui agents on real-world closed-source applications, 2026. URL https://arxiv.org/abs/2605.27761.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Weizheng Wang, Hongzhang Huang, Jun Zhou, Jiachen Song, Shaoli Yu, Jinqi Wang, Zihang Zhou, Hongyi Zhou, Yuting Lv, Jinyang Li, Jiashuo Liu, Ruoyu Chen, Chunwei Liu, GuoLiang Li, Jihua Kang, and Fan Wu. Workspace-bench 1.0: Benchmarking ai agents on workspace tasks with large-scale file dependencies, 2026. URL https://arxiv.org/abs/2605.03596.

Harness Bench Team. Harness bench: Measuring harness effects in realistic agent workflows, 2026. URL https://arxiv.org/abs/2605.27922.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*, 2024.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=OJd3ayDDoF.

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*, 2025b.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents, 2025. URL https://arxiv.org/abs/2504.12516.

Dingbang Wu, Rui Hao, Haiyang Wang, Shuzhe Wu, Han Xiao, Zhenghong Li, Bojiang Zhou, Zheng Ju, Zichen Liu, Lue Fan, and Zhaoxiang Zhang. Mobilegym: A verifiable and highly parallel simulation platform for mobile gui agent research, 2026a. URL https://arxiv.org/abs/2605.26114.

Qinzhuo Wu, Zhizhuo Yang, Hanhao Li, Pengzhi Gao, Wei Liu, and Jian Luan. Mobilebench-ol: A comprehensive chinese benchmark for evaluating mobile gui agents in real-world environment. *arXiv preprint arXiv:2601.20335*, 2026b.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, et al. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. Programbench: Can language models rebuild programs from scratch?, 2026. URL https://arxiv.org/abs/2605.03546.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757. Curran Associates, Inc., 2022a. doi: 10.52202/068431-1508. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/82ad13ec01f9fe44c01cb91814fd7b8c-Paper-Conference.pdf.

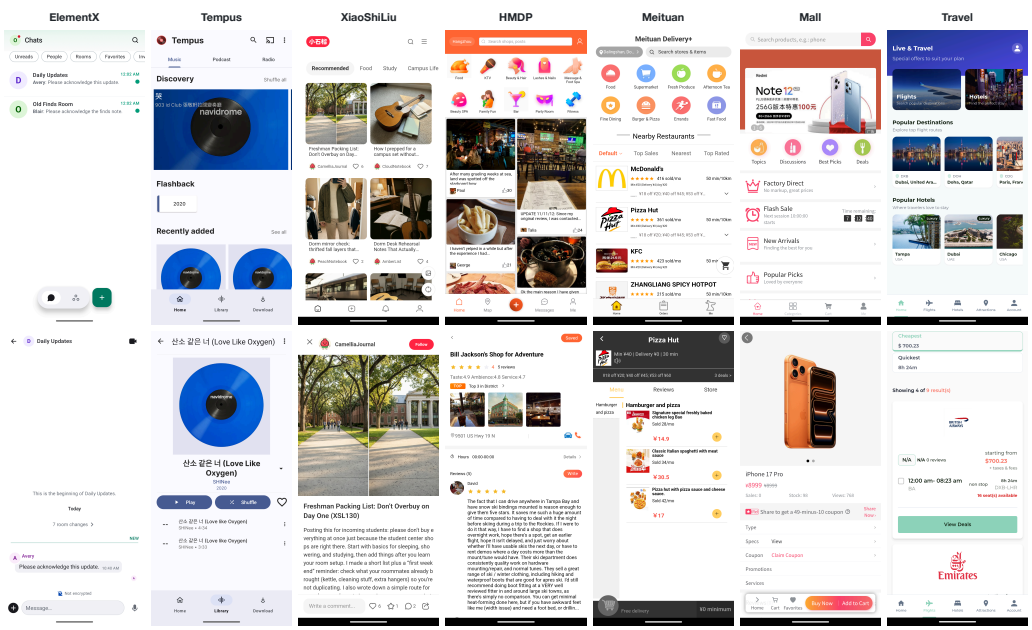


Figure 5: Screenshots of the seven custom applications developed for GMA. From left to right, the applications cover instant messaging, music library, lifestyle sharing, local discovery, food delivery, e-commerce, and travel planning, providing diverse interfaces, interaction patterns, and application-specific workflows for mobile agent evaluation.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, Dunjie Lu, Haikong Lu, Qi Zhen, Xinyuan Wang, Jiaqi Deng, Yuhao Yang, Cheng Chen, Boyuan Zheng, Alex Su, Xiao Yu, Hao Zou, Saaket Agashe, Xing Han Lu, Manpreet Kaur, Zhengyang Qi, Vincent Sunn Chen, Frederic Sala, Dayiheng Liu, Junyang Lin, Zhou Yu, Yu Su, Siva Reddy, Xin Eric Wang, Peng Qi, Tianbao Xie, and Tao Yu. Osworld2.0: Benchmarking computer use agents on long-horizon real-world tasks, 2026. URL https://arxiv.org/abs/2606.29537.

Yujie Zheng, Zikang Liu, Xin Zhao, and Ji-Rong Wen. A task-state representation for long-horizon mobile gui agents, 2026. URL https://arxiv.org/abs/2607.00502.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Xurui Zhou, Gongwei Chen, Yuquan Xie, Zaijing Li, Kaiwen Zhou, Shuai Wang, Shuo Yang, Zhuotao Tian, and Rui Shao. Hiconagent: History context-aware policy optimization for gui agents, 2026. URL https://arxiv.org/abs/2512.01763.

A Additional Details of the GMA Environment

A.1 Applications

To broaden the functional coverage of the evaluation environment, we introduce seven custom applications based on open-source projects. Each represents a common category of mobile service and provides distinct interfaces, interaction patterns, and application-specific workflows.

- **ElementX**[1] serves as an instant messaging client built upon the Matrix ecosystem. It supports direct and group conversations, public and private rooms, message editing, file sharing, and interactive polls, enabling tasks involving interpersonal communication, information aggregation, and collaborative coordination.
- **Tempus**[2] is a music library application linked to a self-hosted Navidrome server. Agents can navigate catalogs of songs, albums, and artists, curate playlists, and manage favorites, supporting tasks involving media retrieval, comparison, and organization of digital collections.
- **XiaoShiLiu**[3] models a multimodal lifestyle-sharing platform. It includes categorized images, nested comment threads, likes, content collections, and social following, requiring agents to interpret unstructured social content and perform multi-stage social interactions.
- **HMDP**[4] provides a crowdsourced local discovery and review environment. Agents can search for businesses, inspect user reviews, publish experience posts, favorite shops, follow users, and manage vouchers, supporting workflows involving local information retrieval and preference-based decisions.
- **Meituan**[5] replicates an on-demand food delivery service with restaurant and menu browsing, address management, cart configuration, ordering workflows, and post-order feedback. These interactions require agents to complete transactional sequences and multi-stage checkout procedures.
- **Mall**[6] provides a general e-commerce environment supporting product search, category browsing, variant selection, shopping-cart management, checkout, and order tracking. It enables tasks involving hierarchical navigation, user constraints, and commercial transactions.
- **Travel**[7] provides a multi-domain travel service covering flights, hotels, and tourist attractions. It supports search and comparison, passenger information entry, booking, and payment, introducing constraint-heavy planning, scheduling dependencies, dense information retrieval, and extensive multi-field form completion.

Rather than relying on simplified mock interfaces, we adapt these applications from fully functional open-source projects to preserve realistic interaction flows. To remove the variability and unreliability of external production services, we decouple the applications from their original APIs and self-host the necessary backend services and databases with locally managed datasets. We additionally provide programmatic interfaces for initializing, querying, and verifying application state.

Beyond the seven custom applications, GMA incorporates standard Android system utilities and retains Mattermost and Mastodon from MobileWorld (Kong et al., 2026). Together, these applications form a heterogeneous mobile ecosystem that supports cross-application workflows requiring information transfer, state tracking, and sequential execution across distinct functional domains.

A.2 Infrastructure

Inherited Foundation. We build GMA on the containerized evaluation infrastructure established by AndroidWorld (Rawles et al., 2024) and MobileWorld (Kong et al., 2026). We inherit emulator snapshots for reproducible starting environments and objective state-based verification for evaluating task completion, while extending this infrastructure to support a substantially broader collection of self-hosted applications and cross-application tasks.

Unified State Abstraction. To coordinate heterogeneous application state, we introduce a unified declarative asset abstraction. An asset represents a typed and modular piece of task state, such as

[1] Android client source: `https://github.com/element-hq/element-x-android`. Server source: `https://github.com/element-hq/dendrite`.

[2] Android client source: `https://github.com/eddyizm/tempus`. Music server source: `https://github.com/navidrome/navidrome`.

[3] Project source: `https://github.com/ZTMYO/XiaoShiLiu`.

[4] Project source: `https://github.com/java-up-up/hmdp-plus`.

[5] Web client source: `https://github.com/zwStar/vue-meituan`. Server source: `https://github.com/zwStar/meituan-backend`.

[6] Web client source: `https://github.com/macrozheng/mall-app-web`. Server source: `https://github.com/macrozheng/mall`.

[7] Project source: `https://github.com/mojahidhasan/fullstack-nextjs-golobe-travel-agency`.

a user contact, message thread, calendar event, social post, e-commerce order, or travel booking. A task specifies its initial conditions as a collection of assets, which are dynamically materialized after environment reset through application-specific adapters. The same representation is used to describe expected final states, providing a common interface for task construction and verification across native Android applications and self-hosted services. By separating high-level task semantics from application-specific storage schemas, this abstraction makes state from different applications composable within a single task.

Application State Adapters. We develop dedicated state adapters for each of the custom application domains. Each adapter provides standardized programmatic operations for creating, querying, and resetting evaluation-relevant entities. Because the emulator frontend and self-hosted backend services maintain state independently, restoring an emulator snapshot alone is insufficient to reproduce the full environment state. Our initialization pipeline therefore coordinates snapshot restoration with backend resets, cache clearance, session invalidation, and deterministic asset injection.

Composable State Verification. Based on the unified asset representation, we implement reusable verification criteria that test whether particular assets exist, have been modified, or have been deleted. Multiple criteria can be combined within a task, enabling both a strict binary success signal and a fine-grained score reflecting partial completion. The same verification interface therefore applies from atomic single-application operations to long-horizon workflows involving several applications.

Deterministic Text Verification. For tasks requiring text entry or explicit answer submission, we avoid non-deterministic mechanisms such as semantic matching or LLM-as-a-Judge. Instead, formatting requirements are explicitly specified in the task instruction and evaluated using deterministic verification rules. This provides consistent evaluation across models and runs while requiring agents to satisfy the requested output contract precisely.

A.3 Task Construction and Annotation

The four difficulty tiers are designed to progressively increase the structural demands of task execution. **Atomic** tasks isolate single-objective operations with explicitly defined target states, primarily evaluating basic GUI perception and control. **Compositional** tasks require agents to chain multiple operations, typically within one application, introducing short-horizon planning, information retrieval, and state tracking. **Cross-Application** tasks introduce explicit inter-application dependencies in which information obtained in one environment determines actions in another, requiring longer-horizon planning and precise information transfer. **Realistic** tasks inherit these cross-application dependencies while further introducing underspecified requirements, fragmented evidence, distractors, and competing constraints across multiple applications. For example, a task may require an agent to infer a product category from an ElementX poll, retrieve a budget constraint from Mail, compare eligible products in Mall, and subsequently schedule an associated event in Calendar.

To support fine-grained evaluation of complex workflows, each task is decomposed into one or more discrete subtasks. A subtask corresponds to an independently verifiable milestone within the overall user requirement, such as drafting an email, completing a cart checkout, or extracting a specific piece of information. Evaluating these milestones separately makes it possible to measure partial progress even when the complete task is not successfully finished.

In addition to the four difficulty tiers, we consider five non-exclusive dimensions as lightweight guidance during task construction. These dimensions are intended to diversify interaction and reasoning requirements rather than serve as mandatory per-task labels. A task may exhibit multiple dimensions, while some tasks are not explicitly associated with any of them.

- **Multi-Step Workflow** captures tasks that require completing an overarching objective through a coordinated sequence of actions, potentially across multiple applications, where information or outcomes from earlier steps are required later.
- **Selection & Optimization** captures tasks in which agents must compare multiple feasible alternatives and select an option that best satisfies a set of constraints or preferences.
- **Information Gathering** captures tasks that require agents to actively locate, interpret, and synthesize information from the environment rather than receiving all necessary parameters directly in the user instruction.

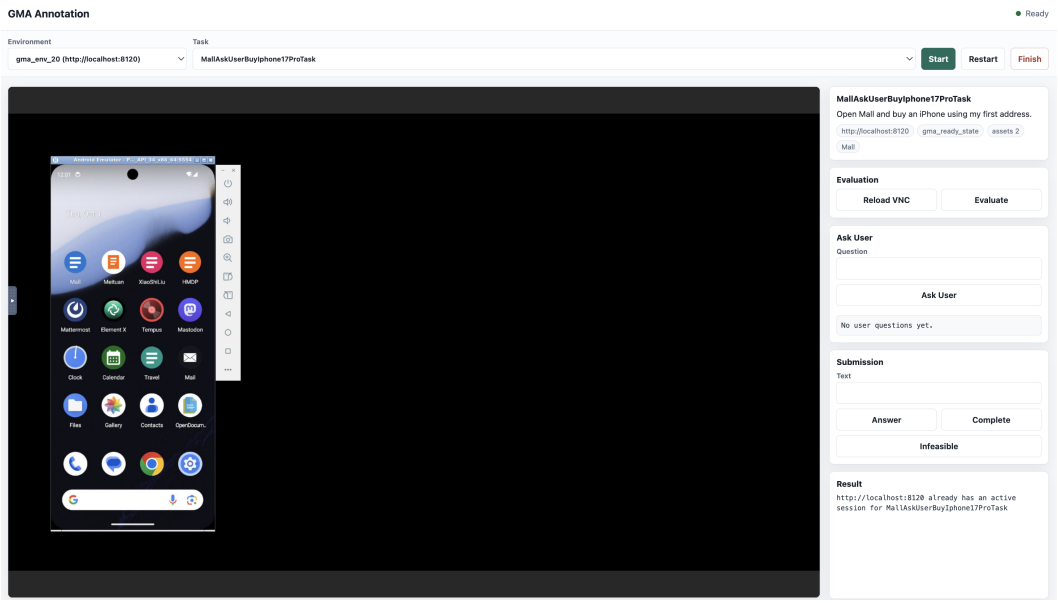


Figure 6: Annotation platform used for task construction and quality control. Annotators execute tasks directly in the Android emulator and review task specifications, subtask decomposition, and verification criteria.

- **Conditional** captures workflows whose correct execution depends on an observed condition, requiring the agent to select, modify, or terminate a workflow according to the applicable branch.
- **Invalid Instruction** captures requests that cannot be completed as stated because they are infeasible, unsupported, or missing essential information, requiring the agent to recognize the limitation and respond appropriately, for example by requesting clarification or proposing a valid alternative.

The finalized benchmark contains 300 tasks: 73 Atomic, 129 Compositional, 78 Cross-Application, and 20 Realistic. Each task undergoes a dual-annotator quality-control process on our annotation platform. Annotators directly execute the task in the Android emulator to verify that the intended workflow is feasible under the specified initial state. They further check the semantic clarity of the instruction, the appropriateness of its assigned difficulty and auxiliary dimensions, and the consistency between the intended user requirement, subtask decomposition, and programmatic verification criteria. Figure 6 shows the annotation interface used during this process.

B User Simulator and Agent Prompts

This section provides the prompts used in our main evaluation setup. We first describe the prompt for the user simulator used in tasks that require user-agent interaction. Then we provide the system prompt and interaction protocol for the agent evaluated in the main experiments.

B.1 User Simulator

As described in Section 4.1, we use Qwen3.7-Plus as an automated user simulator for tasks that permit agent-user interaction. Whenever the agent invokes `call_user`, the simulator receives the question together with a task-specific interaction contract. The contract specifies what information the simulated user is allowed to reveal, ensuring that the simulator does not provide information that the agent is expected to discover from the environment.

The simulator is instructed to respond only when the requested information is permitted by the task contract. Otherwise, it returns no response. The complete clarification history is included in subsequent calls so that multi-turn interactions remain consistent. We allow at most five simulated-user turns per task. The simulator uses temperature 0.2 and a maximum output length of 256 tokens.

User Simulator System Prompt

You are the simulated human user in a mobile benchmark task. Use only the task contract provided by the benchmark. Decide whether the user should respond to the agent question. Do not solve the task, do not operate the phone, do not mention policies, and do not say you are simulated. If the question asks for information the contract does not allow the user to provide, set should_respond to false. Return JSON only, with keys: should_respond (boolean), response (string), and reason (short string).

High-level examples:
- If the contract says the user knows the target room and the agent asks which room to use, answer with that room.
- If the contract says the agent must discover a verification code from Messages and the agent asks the user for the code directly, set should_respond=false.
- If the agent asks for information unrelated to the task contract, set should_respond=false.

At each clarification turn, the simulator receives the following user message:

User Simulator Input Template

Task user-simulation contract:
{contract_json}

Agent question:
{question}

Decide whether the user should respond, and if so what the user says.

The task-specific contract contains natural-language interaction instructions and, when applicable, private facts that the simulated user may reveal and an explicit response policy. Previous question-answer exchanges are also retained in the contract. The expected output is a JSON object of the form {`"should_respond": bool, "response": str, "reason": str`}. If `should_respond` is false, the agent receives no response.

B.2 Main Experiment Agent

The main experiments use a vision-only autoregressive agent. At each step, the agent receives the current screenshot together with the retained interaction history and predicts one action. The default harness preserves the complete textual history of previous reasoning and actions while retaining the two most recent screenshots. Older screenshots are removed from the model context, but their corresponding textual interaction history remains available. No explicit task-state memory, planning, verification, or screenshot-captioning module is used in this setup.

All actor calls use temperature 0 and a maximum output length of 1024 tokens. Each episode is limited to 150 interaction steps.

Main Agent System Prompt

You are a mobile GUI agent operating an Android phone from screenshots.

You must output exactly:
Thought: <one concise sentence>
Action: <one JSON object>

Allowed actions:
- “action_type”:“click”,“coordinate”:[x,y]
- “action_type”:“double_tap”,“coordinate”:[x,y]
- “action_type”:“long_press”,“coordinate”:[x,y]

- “action_type”:“drag”,“start_coordinate”:[x1,y1],“end_coordinate”:[x2,y2]
- “action_type”:“scroll”,“direction”:“up | down | left | right”
- “action_type”:“input_text”,“text”:“...”
- “action_type”:“keyboard_enter”
- “action_type”:“navigate_home”
- “action_type”:“navigate_back”
- “action_type”:“open_app”,“app_name”:“exact app name from the available apps list”
- “action_type”:“call_user”,“text”:“question for the user”
- “action_type”:“wait”
- “action_type”:“answer”,“text”:“final answer”
- “action_type”:“status”,“goal_status”:“complete | infeasible”

Rules:
- Screen coordinates are relative on a 0-1000 scale, where [0,0] is top-left and [1000,1000] is bottom-right.
- Only output one action each turn.
- Use status complete only when the task is done.
- Use status infeasible only when the task cannot be completed.
- When you believe the task is fully completed, your next action must be “action_type”:“status”,“goal_status”:“complete”.
- For tasks that ask you to report an answer rather than change the UI, use “action_type”:“answer”,“text”:“...” as the final action.
- Do not continue interacting with the phone after the task is complete.
- If you need to type text, assume the field is already focused only if it clearly is; otherwise click it first.
- If the screen is loading or transitioning, use wait.
- Use call_user when the goal is ambiguous or missing necessary information. Do not guess required user preferences that are not in the task goal.
- Prefer the shortest reliable path.

The system prompt is followed by the list of available applications:

Available Applications

Available apps:
- Calendar
- Camera
- Chrome
- Clock
- Contacts
- ElementX
- Files
- Gallery
- HMDP
- Mail
- Mall
- Maps
- Mastodon
- Mattermost
- Meituan
- Messages
- Settings
- Tempus
- Travel
- XiaoShiLiu

Open an app using its exact name from this list with open_app.
Do not use package names or URLs, and do not invent other app names.

At the first interaction step, the screenshot is accompanied by the original task instruction:

Initial Agent Input

Current task goal: goal

Current screenshot dimensions: WxH pixels. Choose the next single action.

At subsequent steps, the model receives the new screenshot and:

Subsequent Agent Input

Current screenshot dimensions: WxH pixels. Choose the next single action.

When the agent has previously called the user, the next turn additionally contains:

User Interaction Injection

New user interaction information:
- Agent asked: {question}
- User answered: {response or '[no response]'}
Treat these answers as authoritative task information.

The complete previous actor response, including both `Thought` and `Action`, is retained in the textual history. No accessibility tree, element list, or other structured representation of the interface is provided. Coordinates predicted on the normalized 0-1000 scale are rescaled to the current screenshot dimensions before execution. The parser also normalizes a small number of equivalent action aliases. If an actor output cannot be parsed, the main agent executes `wait` and preserves the original output in the history rather than retrying the generation.

C Ablation Agent Configurations

For the controlled experiments in Section 5, we use a composable ablation agent that preserves the actor protocol, action space, coordinate representation, and system prompt of the main agent, while adding independently configurable context management (E), within-task memory (M), planning (P), and verification (V) modules.

The available configurations are:

- **Context management:** E_0 removes evicted screenshots, E_1 replaces them with interface captions, E_2 replaces them with action-linked event summaries, and E_3 recursively compacts older trajectory prefixes.
- **Within-task memory:** M_0 uses no explicit memory, M_1 maintains a free-form task-state record, and M_2 maintains a structured task-state checklist.
- **Planning:** P_0 uses no explicit plan, P_1 generates one fixed initial plan, P_2 maintains an adaptive flat plan, and P_3 maintains an adaptive hierarchical plan.
- **Verification:** V_0 performs no explicit verification, V_1 verifies each executed transition, and V_2 periodically audits the complete trajectory.

The all-zero configuration (E_0, M_0, P_0, V_0) is a light adaptation of the main experimental harness. It preserves the same underlying actor protocol and two-screenshot visual context, but uses the modular ablation-agent message assembly and execution loop. We therefore use it as the common reference point within the controlled harness experiments rather than treating its absolute score as identical to the baseline in Section 4.

The order in which the modules are applied is fixed. At each step, verification first evaluates the result of the preceding action. The memory module is then updated with the new observation and any verification result. Context management incorporates new screenshot and optionally performs

compaction, after which the planning module updates the current plan. The resulting memory, plan, and verification records are then injected into the actor context before the next action is selected.

Ablation Agent Execution Loop

```
Input: task goal g, modes (e, m, p, v), maximum steps T = 150
Initialize context C, trajectory tau, memory state s, plan pi, verification record r

for t = 1, ..., T:
    observe current screenshot

    if a previous action exists:
        if v = V1:
            verify the previous transition
        if v = V2 and five actions have been executed:
            audit the recent trajectory
        if m != M0:
            update memory from the latest observation and verification

    update context with the new screenshot
    if e = E3 and the compaction threshold is reached:
        compact the older trajectory prefix

    if p = P1 and no plan exists:
        generate the initial plan
    if p in P2, P3:
        update the plan

    inject memory, plan, verification, and user information
    generate exactly one actor action
    execute the action

    terminate on status/answer or after T steps
```

The ablation agent retains two raw screenshots in all configurations. For E_3, the compaction threshold is eight active screenshots. V_2 performs an audit every five executed actions. M_1 and M_2 are updated after every executed action. P_1 is generated once at the beginning of a task, whereas P_2 and P_3 are updated at every interaction step. Actor and module calls use temperature 0. Invalid structured outputs are retried up to three times with a corrective parsing message.

C.1 Context Management

E_1: Interface Captioning. When an older screenshot leaves the retained visual window, E_1 replaces it with a concise task-relevant caption. The captioner is explicitly instructed to describe only visually supported information and not to infer task success or recommend subsequent actions.

E_1 Captioner Prompt

You caption an earlier GUI observation for a computer-use agent. This is a memory-compression call, not an action-selection call.

Describe only task-relevant facts visibly supported by the screenshot being captioned. When a preceding screenshot and executed agent output are supplied, use them only to identify the observed result of that action. Preserve the app or page, important visible values and selections, results, dialogs, errors, confirmations, and unresolved UI state that could matter later.

Do not recommend a next action, expose chain-of-thought, repeat click coordinates, infer hidden success, or describe decorative layout. Do not treat the preceding agent's expectation as evidence. Return one concise paragraph of at most 80 words and no heading, JSON, XML, bullets, or preamble.

The actor is additionally informed that captions are fallible historical memory:

E_1 Actor Context Note

Historical screenshot captions

Some older screenshots in the conversation may be replaced by explicitly labelled historical captions. Use them as fallible memory of earlier GUI states. Never ground a click or other coordinate action against a historical caption or an image described by one; ground actions only against the latest current screenshot. The current screenshot and latest user response are authoritative when they conflict with historical memory.

E_2: Action-Linked Event Summaries. E_2 summarizes individual historical transitions using the screenshots before and after an executed action.

E_2 Event Summarization Prompt

You compress one historical GUI transition for a mobile agent. Given the task, the screenshot before an action, the executed action, and the screenshot after the action, state only the task-relevant observed event. Separate the attempted action from its visible result. Use uncertain language when success is not visually supported. Do not recommend a next action, reveal chain-of-thought, or repeat coordinates. Return one concise paragraph of at most 80 words with no heading, list, JSON, XML, or preamble.

E_3: Recursive Trajectory Compaction. E_3 periodically replaces an older trajectory prefix with a recursively maintained free-form summary. A previous compacted summary, when present, is merged with the newly compacted evidence.

E_3 Trajectory Compaction Prompt

You compact an older prefix of a mobile GUI agent trajectory into one concise free-form handoff summary. Preserve task-relevant authoritative user answers and corrections, attempted actions and their visibly supported results, stable GUI facts, failures, unresolved uncertainty, and the observed GUI state at the compaction boundary. If a previous compacted summary is supplied, merge it with the new evidence instead of referring to it as a separate summary.

Do not recommend a next action, create a plan or checklist, name a current subgoal, reveal chain-of-thought, invent success, or include coordinates. Later user corrections replace earlier conflicting information. Return only plain free-form text of at most 500 words, with no heading, JSON, XML, or preamble.

C.2 Within-Task Memory

Both M_1 and M_2 are updated after every executed action. Their update call receives the original task, previous memory state, executed action, latest screenshot, authoritative user information, and any available verifier result.

Memory Update Input Template

Original Task
{goal}

Previous Memory
{previous state}

Executed Action
{action}

Actor Thought
{thought or '[none]'}

Separate Verifier Diagnosis
{verification, if available}

Authoritative User Information
- Agent asked: {question}
- User answered: {response or '[no response]'}

Update memory using the latest screenshot and return only the required output.

M_1: Free-Form State.

M_1 Free-Form Memory Prompt

You maintain a short task-progress record for a GUI agent. This is a state-update call, not an action-selection call. The current screenshot and any new user response show what happened after the previous action was executed.

Write one concise paragraph of at most 150 words containing only durable, task-level facts: requirements visibly confirmed as completed, requirements still unresolved, important user-provided information, and genuine blockers or uncertainties. Treat the previous action as successful only when the screenshot or user response provides unambiguous evidence. If its result is ambiguous, record the uncertainty instead of assuming success. Do not infer the semantic meaning of an unlabeled icon or a visual change alone.

Preserve useful facts from the previous record, but correct them when current evidence contradicts them. Do not record click coordinates, detailed navigation history, transient UI operations, the agent's reasoning, or recommendations for the next action. Do not invent hidden results and do not propose or output a GUI action.

Return only the paragraph. Do not use JSON, XML tags, headings, bullet points, numbered lists, or any other structured format.

M_2: Structured State.

M_2 Structured Memory Prompt

You maintain a structured latest task-state record for a mobile GUI agent. This is a memory-update call, not action selection or a separate verification call. Use the task, previous state, executed action, latest screenshot, user facts, and any supplied verifier diagnosis to update a compact checklist. Preserve useful facts and explicitly retain uncertainty. Do not recommend coordinates or output a GUI action. Return strict JSON only with this exact shape:
{
"task_summary": "brief task and current situation",

```
  "completed": ["requirements believed complete"],
  "remaining": ["requirements not complete"],
  "facts": ["durable task or user facts"],
  "failures": ["important failed attempts or blockers"],
  "uncertainties": ["unresolved uncertainty"],
  "current_subgoal": "current objective or empty string"
}
Do not put the same item in completed and remaining.
```

The resulting memory record is inserted into the actor context as fallible information, and the current screenshot and authoritative user responses always take precedence.

C.3 PLANNING

All planning variants share the following system prompt:

Planning Module Prompt

You are the planning module of a mobile GUI agent. Planning is separate from action selection, memory, and verification. Use the task, current screenshot, trajectory, and any supplied module records to return only the requested strict JSON plan. Do not emit coordinates or a GUI action. Plans must be concise and task-specific. Never claim a requirement is complete without supplied evidence.

P_1 and P_2 use a flat plan:

P_1/P_2 Output Schema

```
{"steps":["ordered high-level step"]}
```

while P_3 uses a hierarchical representation:

P_3 Output Schema

```
{
  "subgoals":[
    "name":"subgoal","status":"pending | active | done"
  ],
  "active_subgoal":"name or empty string",
  "local_steps":["next local step"]
}
```

P_1 is generated only once at the first interaction step, whereas P_2 and P_3 are recomputed at every step using the most recent trajectory, memory, and verification records.

C.4 VERIFICATION

V_1: Transition-Level Verification. V_1 compares the screenshots immediately before and after every executed action.

V_1 Transition Verifier Prompt

You are a separate GUI action verifier. Compare screenshots before and after the executed action and diagnose only its visible outcome. Do not plan or recommend the next action. Return strict JSON

Table 6: Full results for individual harness module designs on Qwen3.7-Plus. Each configuration changes one module from the baseline (E_0, M_0, P_0, V_0).

Module	Design	Overall		Atomic		Compositional		Cross-Application		Realistic	
		Success	*Average*	*Success*	*Average*	*Success*	*Average*	*Success*	*Average*	*Success*	*Average*
Baseline (E_0, M_0, P_0, V_0)		47.33	62.56	80.82	83.11	44.96	65.24	32.05	52.96	0.00	7.72
Context	E_1	52.00	66.48	79.45	81.05	54.26	70.23	32.05	54.19	15.00	36.97
	E_2	49.00	63.78	79.45	80.14	48.84	67.73	33.33	53.52	0.00	18.61
	E_3	52.33	67.27	76.71	79.00	57.36	74.17	32.05	54.40	10.00	30.14
Memory	M_1	47.67	60.65	76.71	78.31	51.94	66.67	25.64	46.06	0.00	14.19
	M_2	48.67	65.14	80.82	82.42	51.16	69.46	26.92	51.89	0.00	25.92
Planning	P_1	49.33	63.71	79.45	80.14	47.29	67.36	37.18	55.87	0.00	10.83
	P_2	49.33	64.09	82.19	84.47	49.61	67.19	30.77	51.94	0.00	17.08
	P_3	47.00	60.86	78.08	78.77	48.84	66.76	26.92	47.35	0.00	10.08
Verification	V_1	49.33	63.64	83.56	84.93	50.39	68.76	28.21	50.39	0.00	4.56
	V_2	51.33	65.56	79.45	81.74	55.81	71.87	30.77	52.50	0.00	16.72

only:{"outcome":"success | partial_success | wrong_result | no_effect | uncertain","evidence": "concise visible evidence"}

V_2: Trajectory-Level Verification. V_2 is invoked every five executed actions and audits progress against the original task using the recent trajectory with the current memory and plan when available.

V_2 Trajectory Verifier Prompt

You are a separate periodic trajectory verifier. Audit progress against the original requirements using the current screenshot and supplied records. Do not select or recommend a next action. Return strict JSON only:{"trajectory_status":"on_track | off_track | uncertain","completed_subgoals":[], "omitted_requirements":[],"problems":[],"summary":"concise evidence-grounded audit"}

The latest verification record is exposed to the actor as a fallible diagnosis rather than as an action instruction. It is also available to the memory and planning modules on subsequent updates.

D Full Harness Ablation Results

We provide the complete results of the harness decomposition experiments in Tables 6 and 7. We report the absolute *Success* and *Average* scores for all evaluated configurations across the four difficulty tiers. All values are percentages.

E Additional Analyses of Harness Effects

In Section 5 we analyze how harness effects vary with task demands, implementation choices, module composition, and the underlying foundation model. In this section, we provide two complementary analyses that further characterize when different harness configurations are useful. We first study harness behavior as a function of the interaction length required for successful execution, providing an execution-level view beyond the difficulty tiers used in the main analysis. We then examine the potential benefit of selecting different harness configurations across tasks rather than applying a single fixed configuration throughout the benchmark.

Table 7: Full results for the evaluated compositions of harness modules on Qwen3.7-Plus. The baseline is included for reference.

Configuration	Overall		Atomic		Compositional		Cross-Application		Realistic	
	Success	*Average*	*Success*	*Average*	*Success*	*Average*	*Success*	*Average*	*Success*	*Average*
(E_0, M_0, P_0, V_0)	47.33	62.56	80.82	83.11	44.96	65.24	32.05	52.96	0.00	7.72
(E_0, M_0, P_1, V_1)	46.67	61.64	82.19	83.79	45.74	64.98	26.92	49.58	0.00	6.22
(E_0, M_2, P_0, V_2)	53.00	67.85	82.19	83.56	57.36	72.85	30.77	53.30	5.00	35.00
(E_0, M_2, P_3, V_0)	50.33	66.00	80.82	82.42	55.04	72.21	25.64	49.07	5.00	32.08
(E_1, M_0, P_0, V_2)	52.00	66.73	78.08	80.37	55.04	72.22	33.33	53.69	10.00	32.31
(E_1, M_2, P_0, V_0)	51.00	67.67	82.19	83.79	53.49	73.71	29.49	53.14	5.00	26.56
(E_1, M_2, P_0, V_2)	52.00	67.95	79.45	81.05	57.36	73.42	30.77	54.54	0.00	37.14
(E_3, M_0, P_0, V_2)	53.67	68.02	82.19	84.47	58.14	72.63	30.77	53.06	10.00	36.46
(E_3, M_0, P_1, V_2)	50.33	66.37	80.82	81.51	52.71	70.41	28.21	53.50	10.00	35.22
(E_3, M_2, P_0, V_0)	52.67	69.04	82.19	83.56	55.04	73.05	32.05	56.64	10.00	38.47
(E_3, M_2, P_0, V_1)	51.33	68.67	80.82	82.42	54.26	73.60	32.05	57.02	0.00	32.06
(E_3, M_2, P_0, V_2)	53.00	68.89	82.19	83.56	55.81	75.09	33.33	53.38	5.00	35.89

E.1 Harness Effects across Execution Length

The four GMA difficulty tiers characterize progressively increasing structural demands, but tasks within the same tier can still exhibit substantially different execution lengths. We therefore complement the established difficulty-based analysis with an execution-level view of harness effectiveness.

For each task, we define its *baseline trajectory length* as the number of interaction steps taken by the baseline configuration, regardless of whether the resulting episode succeeds. We partition the 300 tasks into four groups according to this observed length: 59 tasks have baseline trajectories of at most 10 steps, 92 have 11-25 steps, 44 have 26-60 steps, and 105 have more than 60 steps. We then evaluate the baseline and representative harness configurations E_3, V_2, and $E_3 + V_2$ on these same task groups.

As shown in Figure 7(a), differences between configurations are relatively small for tasks associated with shorter baseline trajectories. In the $\leq$10-step group, *Success* ranges from 79.66% to 81.36% across the four configurations. A similar pattern holds for the 11-25-step group, where the baseline achieves 71.74% *Success*, compared with 67.39% for E_3, 70.65% for V_2, and 69.57% for $E_3 + V_2$. Thus, on tasks for which the baseline terminates after relatively few interactions, the evaluated harness mechanisms provide no consistent advantage.

The pattern changes for tasks associated with longer baseline trajectories. In the 26-60-step group, the baseline achieves 54.55% *Success*, while $E_3 + V_2$ reaches 61.36%. The differences become substantially larger among the 105 tasks whose baseline trajectories exceed 60 steps. Baseline *Success* falls to 3.81%, whereas E_3, V_2, and $E_3 + V_2$ reach 21.90%, 16.19%, and 20.95%, respectively. Although the relative ordering of configurations varies across trajectory-length groups, all three harness configurations substantially outperform the baseline in the longest group. The largest gains over the baseline therefore occur on tasks for which baseline execution extends over many interactions.

We further examine how often episodes continue until the interaction budget is exhausted. As shown in Figure 7(b), 29.67% of baseline episodes reach the 150-step limit. This proportion decreases to 17.00% under E_3 and to 14.33% under $E_3 + M_2 + V_2$. Thus, these stronger harness configurations are substantially less likely to consume the full interaction budget, complementing the *Success* rate improvements observed among tasks with long baseline trajectories.

Taken together, these results provide an execution-level perspective complementary to the difficulty-based analysis in Section 4. Harness effects are limited and sometimes negative among tasks with short baseline trajectories, but become substantially larger when baseline execution extends over many interactions. This pattern is consistent with the intuition that longer running executions place greater demands on preserving task-relevant information, maintaining intermediate progress, and coordinating decisions across distant parts of the trajectory.

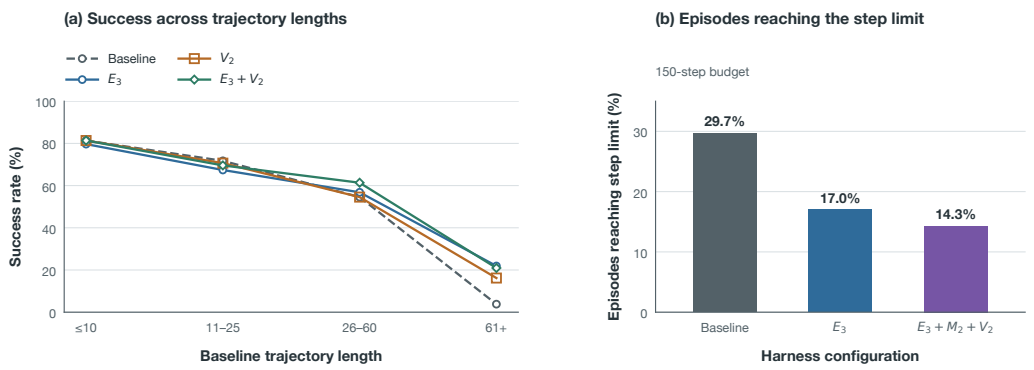


Figure 7: Execution dynamics under representative harness configurations. **(a)** Success grouped by baseline trajectory length, defined as the number of interaction steps taken by the baseline configuration. **(b)** Fraction of episodes reaching the 150-step interaction limit.

E.2 Harness Selection across Tasks

The previous analyses evaluate harness designs using configurations shared across all tasks. However, the observed dependence of harness effects on task demands suggests that no single configuration need be preferable for every task. We therefore examine how performance changes when harness configurations are selected at different levels of granularity.

Figure 8 compares four selection strategies over all evaluated configurations. The first two use a single fixed configuration for the complete benchmark, selecting the configuration with either the highest overall *Success* or the highest overall *Average*. A difficulty-specific strategy selects, for each difficulty tier, the configuration with the highest tier-level *Average*, and then pools the outcomes across all tasks. Finally, a per-task oracle selects the configuration obtaining the highest score independently for every task. The oracle is not intended as a practical selection strategy, but provides an upper bound on the variation exposed by the evaluated configuration set.

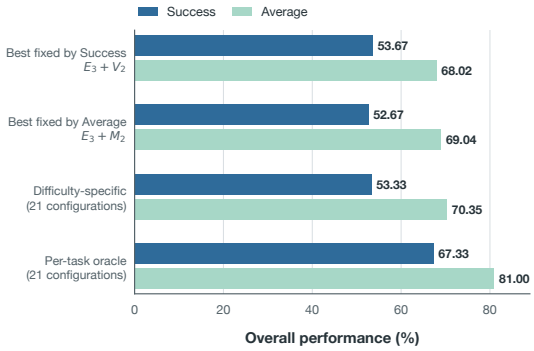
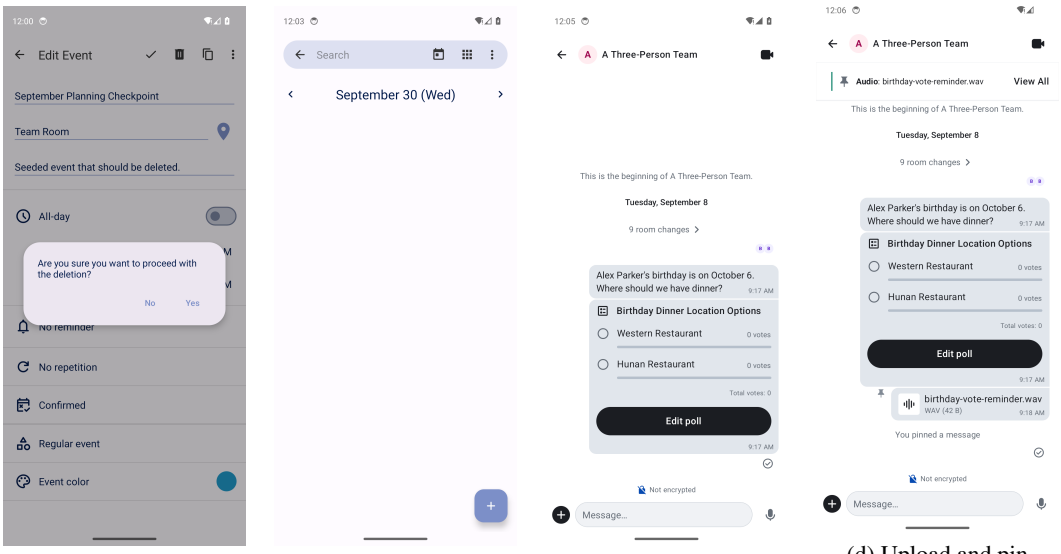


Figure 8: Performance under different harness selection strategies. Fixed settings use one configuration for all tasks, difficulty-specific selection chooses the strongest evaluated configuration independently for each difficulty tier, and the per task oracle selects the strongest evaluated configuration separately for each task.

Among fixed configurations, $E_3 + V_2$ achieves the highest overall *Success*, reaching 53.67% *Success* and 68.02 *Average*. Selecting the fixed configuration according to *Average* instead chooses $E_3 + M_2$, which reaches 52.67% *Success* and 69.04 *Average*. Thus, even when a single configuration must be used globally, the preferred harness differs depending on whether evaluation emphasizes strict task completion or partial progress.

Difficulty-specific selection reaches 53.33% *Success* and 70.35 *Average*. Under this strategy, the selected configurations are V_1 for Atomic tasks, $E_3 + M_2 + V_2$ for Compositional tasks, $E_3 + M_2 + V_1$ for Cross-Application tasks, and $E_3 + M_2$ for Realistic tasks. Compared with the best fixed configuration selected by *Average*, difficulty-specific selection improves *Average* to 70.35, while its *Success* changes only slightly. This suggests that task difficulty provides some useful signal for selecting harness configurations, but the gain from tier-level adaptation remains limited.

A much larger difference appears with task-level selection. The per-task oracle reaches 67.33% *Success* and 81.00 *Average*, corresponding to improvements of 14.00 and 10.65 points over difficulty-



(a) Delete existing event (b) Create birthday event (c) Message and poll (d) Upload and pin reminder

Figure 9: Successful long-horizon Cross-Application execution. The agent completes the required Calendar operations and subsequently sends the message, creates the poll, uploads the audio reminder, and pins it in ElementX.

specific selection, respectively. The substantially larger gain from per-task selection indicates considerable heterogeneity among tasks even within the same difficulty tier.

These results further illustrate the conditional nature of harness effectiveness. Difficulty captures an important source of variation, but does not fully determine which configuration is most effective for an individual task. Tasks assigned to the same tier can still differ in interaction structure, information-retention requirements, cross-step dependencies, and failure modes. The large gap between difficulty-specific selection and the per-task oracle therefore suggests substantial headroom for adaptive harnesses that choose or activate mechanisms according to the requirements of individual tasks rather than relying on a globally fixed design.

F Task Examples of GMA

Table 8 presents representative examples from the four difficulty tiers of GMA. We include five examples for each tier to illustrate the diversity of tasks within each difficulty level. For readability, the task descriptions are shortened while preserving their core requirements.

G Case Studies

To complement the aggregate results, we present four representative trajectories from Qwen3.7-Plus on GMA. The selected cases illustrate successful long-horizon cross-application execution, an exact-format failure, inconsistent state propagation in a realistic workflow, and successful clarification through the user simulator. Rather than reproducing complete trajectories, we focus on the task requirements, key execution outcomes, and representative screenshots that highlight the behavior of interest in each case.

G.1 Case Study 1: Successful Long-Horizon Cross-Application Execution

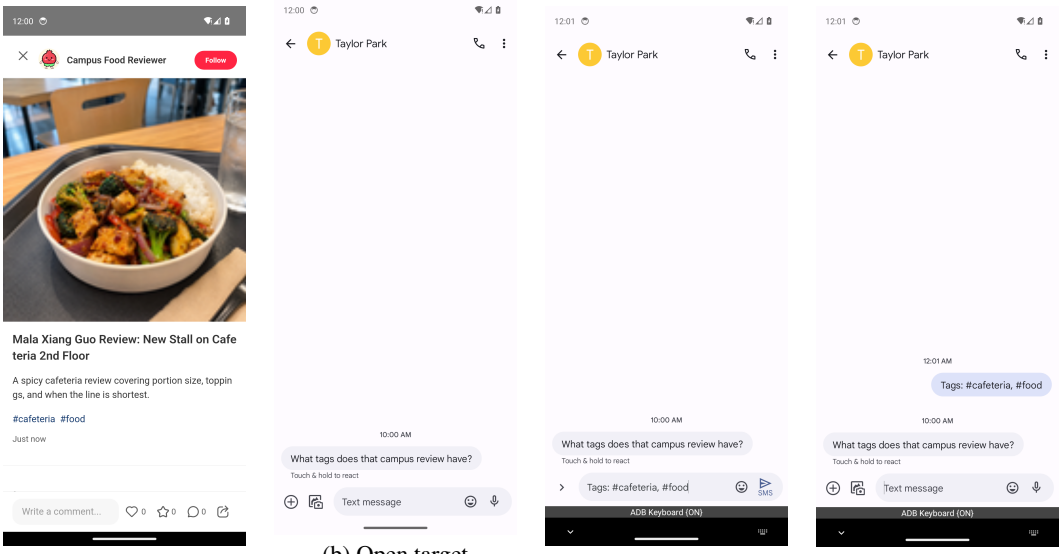
Figure 9 shows a successful Cross-Application trajectory that requires coordinated operations in Calendar and ElementX. The task consists of five independently verified requirements: deleting an existing Calendar event, creating a new birthday event for Alex Parker, sending the required message in ElementX, creating a dinner location poll, and uploading and pinning an audio reminder.

Table 8: Representative task examples from the four difficulty tiers of GMA. Task descriptions are shortened for presentation while preserving their essential requirements.

Tier	Application(s)	Example Task
Atomic	Clock	Create a 6:00 AM alarm labeled Morning Alarm.
	Messages	Send Taylor Morgan exactly I know in Messages.
	Mall	Browse the HUAWEI brand and favorite the HUAWEI Mate 80 Pro.
	Tempus	Delete the playlist named My Favorite Playlist.
	Travel	Purchase one adult and one child ticket for Ubud Monkey Forest on October 6, 2026, using the specified visitor names, and complete payment.
Compositional	Clock	Create a recurring Tuesday alarm with vibration enabled, delete the Get Up alarm, and modify alarm A's time, repeat day, and vibration setting.
	Clock	Check whether an 8:00 AM alarm exists and enable it only if tomorrow is not Wednesday.
	Calendar	Create a Daily Music Practice event with a requested subtitle. Since Calendar does not support subtitles, obtain user guidance and place the text in the event description instead.
	Mall	Compare the available Lenovo laptops, buy the cheapest one using the first saved address, and complete payment with Alipay.
	Contacts	Modify multiple fields for Alex Parker and Morgan Blake, including names, phone labels, phone number, and websites, requesting the missing phone-label preference when needed.
Cross-App.	Calendar, ElementX, Messages	Create an Old Friends Dinner Calendar event and send its date, time, and location through ElementX and Messages using the required exact format.
	Calendar, ElementX, Files	Delete an old Calendar event, create Alex Parker's birthday event, then send a message, create a dinner-location poll, upload an audio reminder, and pin it in ElementX.
	Meituan, ElementX	Check a Pizza Hut store's rating and address. If the rating meets the threshold, send the address to Riley Park in ElementX using the specified format.
	XiaoShiLiu, Messages	Find and like/save a specified XiaoShiLiu post, read its tags, and send them in the same order to Taylor Park through Messages using the required format.
	Travel, Calendar	Book and pay for a stay at Jordaan Boutique Inn in Amsterdam, then create a Calendar event for the business trip containing the booked hotel information.
Realistic	ElementX, Mail, Calendar, Messages	Reconcile the latest dinner polls, email constraints, calendar availability, and message-based tie-breaking instructions to select a final dinner plan, then create and communicate it in the required formats.
	Messages, Mail, Contacts, Files, Gallery, Calendar, Clock	Identify an urgent home-repair issue from fragmented evidence, send a formatted maintenance email with matching attachments, schedule the access window, create two alarms, and notify the roommate.
	Mail, Messages, Contacts, Mall, Calendar, Clock	Coordinate moving day by reconciling landlord, mover, and roommate information. Create the delivery address, purchase the matching wardrobe, schedule building access, set an alarm, and send the required confirmations.
	Mail, Travel, Calendar, ElementX, Messages, Clock	Determine the approved London itinerary from multiple emails, book and pay for the required flight and hotel, then propagate the confirmed itinerary into Calendar, ElementX, Messages, and a Clock alarm.
	Mail, XiaoShiLiu, Mall, Contacts, Messages	Read a tablet purchase request, research XiaoShiLiu posts and comments to identify a product satisfying the constraints, purchase it in Mall, and communicate the choice with the required supporting evidence.

Qwen3.7-Plus completes all five requirements in 64 steps. The trajectory first performs required Calendar modifications and switches to ElementX to complete the communication and file-related operations. The agent successfully preserves the relevant task information across applications and finishes the message, poll, file upload, and pinning operations without omitting any requirement.

This example illustrates that a long trajectory does not necessarily imply failure. Even when the high-level task structure is relatively clear, execution can require many low-level GUI interactions, particularly for operations such as manipulating dates, times, files, and room-level controls. At the same time, strict task success depends on completing every required subtask, and omission of even the final pinning operation would result in partial rather than full success.



(a) Read post tags (b) Open target conversation (c) Compose message (d) Send incorrect format

Figure 10: Exact-format failure in a Cross-Application task. The agent correctly identifies the tags and reaches the target conversation, but retains the hash symbols in the final message, causing the exact-message verification criterion to fail.

G.2 Case Study 2: Exact-Format Failure and Premature Completion

Figure 10 shows a Cross-Application trajectory in which the agent successfully completes most of the requested workflow but fails the final verification because of a small formatting error. The task requires the agent to find a specified post in XiaoShiLiu, like and save it, read its tags, and send those tags to Taylor Park through Messages using an exact output format.

Qwen3.7-Plus correctly identifies the relevant tags as `#cafeteria` and `#food`, and successfully completes the like and save operations. However, when composing the message, it sends `Tags: #cafeteria, #food` instead of the required `Tags: cafeteria, food`. As a result, the first two verification criteria pass, while the exact-message criterion fails.

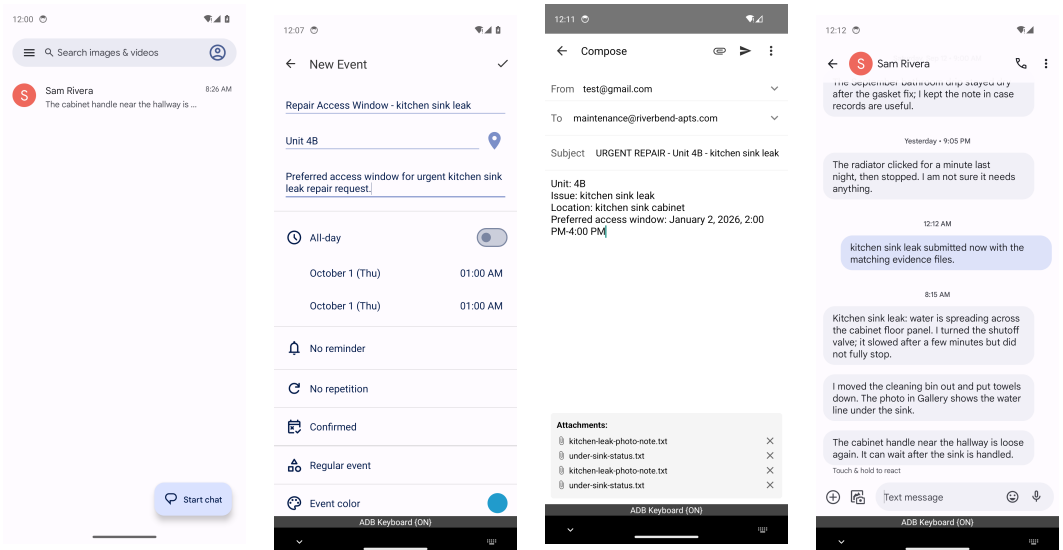
Immediately after sending the message, the agent terminates the trajectory and treats the task as successfully completed. This case illustrates that successful navigation and correct high-level reasoning are not sufficient for strict task success: small errors in the final artifact can invalidate an otherwise correct workflow. It also shows how premature termination can prevent the agent from detecting and correcting such errors.

G.3 Case Study 3: Inconsistent State Propagation in a Realistic Workflow

Figure 11 shows a Realistic task that requires the agent to integrate information across Messages, Mail, Files, Gallery, Calendar, and Clock. The agent must identify the urgent repair issue, send a formatted maintenance email with the correct attachments, create a Calendar event for the selected access window, create two alarms, and notify the roommate.

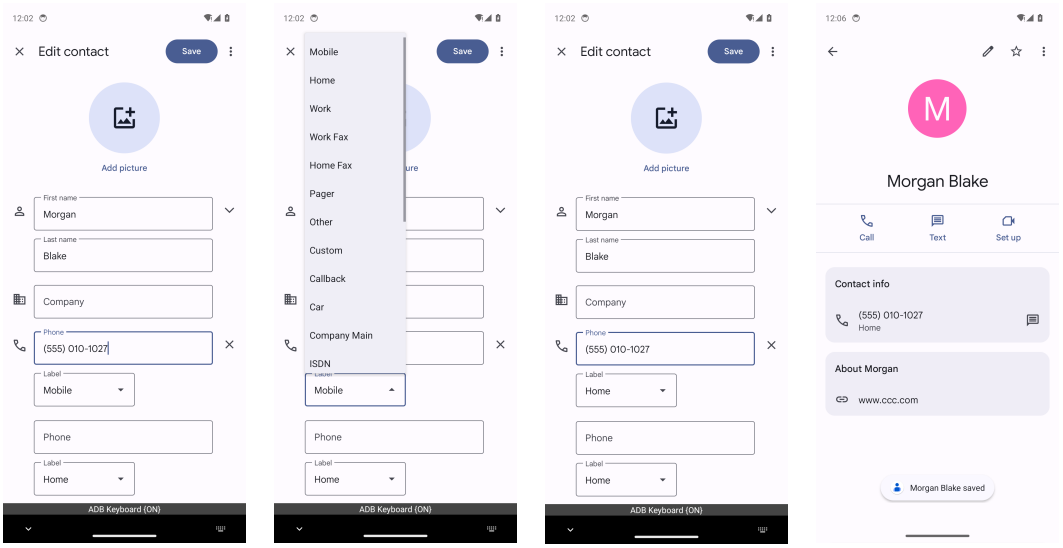
Qwen3.7-Plus completes three of the five verification criteria. It correctly creates the Calendar event, sets the maintenance-reply alarm, and sends the roommate confirmation message. However, two outputs contain inconsistent task state. In the maintenance email, the agent uses an example date from the instruction rather than the selected October 2 access window. In addition, the repair-access alarm is created at the correct time but on October 1 instead of October 2.

Notably, the intended October 2 access window is represented correctly in Calendar but incorrectly in Mail and Clock. The failure therefore does not come from selecting the wrong plan initially. Instead, the same task variable is propagated inconsistently across applications. The agent eventually terminates the trajectory as complete despite these two unsatisfied requirements.



(a) Gather repair information (b) Create access event (c) Compose maintenance email (d) Terminate workflow

Figure 11: Partial completion of a Realistic home-repair workflow. The agent gathers the required information and completes several outputs, but propagates task state inconsistently across applications and terminates with unsatisfied verification criteria.



(a) Request clarification (b) Open label menu (c) Select Home label (d) Complete task

Figure 12: Successful clarification through the user simulator. The agent recognizes that the required phone label is unspecified, requests the missing information, applies the returned preference, and continues to successful task completion.

This case illustrates a central challenge of Realistic tasks: an agent may successfully complete most of a long workflow while a small number of state-maintenance errors still prevent strict task success. It also helps explain the gap between *Success* and *Average* observed on the Realistic tier.

G.4 Case Study 4: Clarification Through the User Simulator

Figure 12 shows a Compositional Contacts task in which the instruction does not fully specify the required edits. The agent is asked to modify several fields for Alex Parker and Morgan Blake, including the phone label for Morgan Blake, but the target label itself is not provided in the instruction.

Rather than selecting a label arbitrarily, Qwen3.7-Plus explicitly asks the user which label should be used. The user simulator returns the task-authorized response, *Home*, after which the agent continues the contact editing workflow and successfully satisfies the corresponding verification criteria.

This case illustrates the intended role of the user simulator in GMA. Information that is intentionally omitted from the task is not provided to the agent in advance. Instead, it becomes available only when the agent recognizes the ambiguity and requests relevant clarification. Such tasks therefore test not only GUI execution, but also whether the agent can identify when additional user input is necessary rather than making an unsupported assumption.