MOBILE SKILLS BENCH: BENCHMARKING LONG-HORIZON INTERACTIVE MOBILE AGENTS WITH MULTIMODAL SKILLS AND HYBRID EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Mobile agents are increasingly expected to carry out everyday workflows that span multiple applications, changing interfaces, and evolving user requests. However, such long-horizon tasks remain challenging for current agents. Agent skills offer a promising remedy, because they package reusable procedural knowledge, and multimodal skills further add visual grounding through reference images. Yet their benefit remains unclear, since no benchmark compares skill designs under controlled conditions on long, interactive mobile tasks with verifiable outcomes and hybrid execution. To address this gap, we introduce Mobile Skills Bench (MSB), a controlled testbed for evaluating agent skills, which comprises 145 long-horizon tasks across 24 simulated applications with a verifiable backend, provides a user simulator, and supports both Pure GUI and Hybrid (GUI, MCP, and CLI) execution. MSB further includes a multimodal skill library of 73 procedures with 166 reference images, organized by function, by function–app pair, and by app. We then evaluate nine models in three agent harnesses. Without skills, the nine models achieve an average task success rate of only 28.9% under Hybrid and 12.5% under Pure GUI execution. The models often fail to ask for missing information, claim completion prematurely, or stall without recovery. With textual skills, the two averages rise to 34.2% and 15.0%, although the gain varies with the model and execution mode. However, adding the tested static reference images yields no consistent further gain. Finally, the three library organizations reach similar task success, but organizing the library by app uses 31% fewer input tokens on average than organizing it by function or by function–app pair, which suggests a simple principle for designing skill libraries in mobile scenarios.

1 INTRODUCTION

Mobile agents are increasingly expected to automate complex everyday workflows, as illustrated by the Doubao AI Assistant on the nubia M153 and the announced preview of Gemini's multi-step task automation on Android (ZTE Corporation, 2026; Google, 2026d). In these workflows, agents must adapt to changing interfaces, coordinate across applications, preserve intermediate results, and respond to evolving user requests (Kong et al., 2026). Meeting these demands requires not only correct individual actions but also coherent procedures that remain applicable across multiple steps and user turns. Sustaining such procedures, however, remains challenging for current agents. Reusable agent skills target this gap by supplying procedural guidance (Li et al., 2026b; Zhang et al., 2026a;b). Specifically, a skill packages instructions and supporting resources for a class of tasks, allowing agents to follow established procedures rather than derive each procedure anew. In particular, multimodal skills complement these procedures with reference images for recognizing interface states, grounding actions, and verifying progress (Zhang et al., 2026b; Jiang et al., 2026). However, the effectiveness and design of these skills for long-horizon, interactive mobile agents remain insufficiently understood, motivating our central research question: *To what extent do reusable skills, particularly multimodal skills, improve long-horizon, interactive mobile task completion, and how should they be designed and used to improve performance across execution modes?*

Answering this question requires benchmarks that capture how mobile agents apply and reuse skills. Because skills encode multi-step procedures, evaluating their sustained utility calls for long tasks

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.

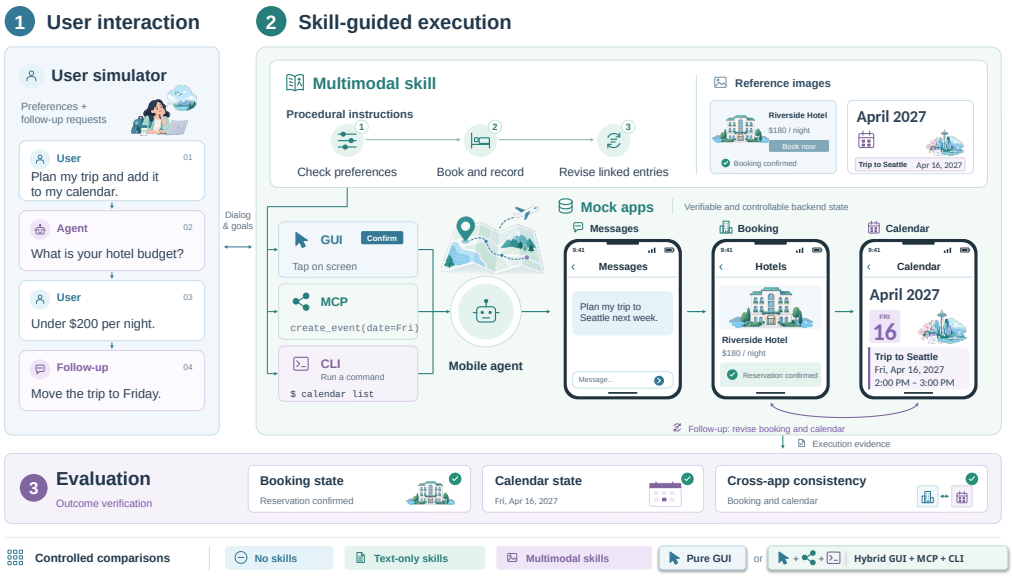


Fig. 1: Overview of MSB. A simulated user supplies preferences and follow-up requests as an agent executes cross-app workflows with skill guidance through GUI, MCP, or CLI interfaces. Controllable backend state supports verification of booking outcomes, calendar updates, and cross-app consistency. Controlled comparisons evaluate no skills, textual skills, and multimodal skills under Pure GUI or Hybrid execution.

with interdependent stages that exercise repeated skill selection, composition, and reuse. A user simulator is also needed to reflect user interaction throughout these workflows, providing clarification and introducing follow-up requests as execution progresses. Meanwhile, verifiable backend state is necessary to determine whether the resulting actions satisfy task goals and constraints. The evaluation should further support hybrid execution through graphical user interface (GUI) operations, Model Context Protocol (MCP) tools, and command-line interface (CLI) commands, allowing skill utility to be examined across execution modes. Although existing benchmarks provide parts of this setting, none offers a controlled setting that holds tasks, user interaction, and verification fixed while varying the skills and execution interface, so none can compare skill designs systematically (Rawles et al., 2025; Xu et al., 2025; Wu et al., 2026; Li et al., 2026a; Kong et al., 2026).

Early benchmarks such as AndroidWorld and AndroidLab established reproducible mobile-agent evaluation, with programmatic task initialization and verification (Rawles et al., 2025; Xu et al., 2025). To make verification possible, however, they build on open-source, offline apps for note-taking, calendars, and expenses. Their tasks thus rarely involve the online services people use daily, such as shopping and ticket booking, and they remain short, with AndroidWorld episodes averaging 14.3 steps (Tab. 1). Such short tasks seldom need the multi-step procedures that skills package, so they are ill-suited for evaluating skills. Adding online services, in turn, is hard because their backend state is **unreadable**, **unforkable** for independent trials from identical starting conditions, and changed **irreversibly** by actions such as payments. To overcome these constraints, MobileGym replaces these services with browser-hosted mock apps whose state is inspectable, controllable, and replicable (Wu et al., 2026), thereby enabling repeated evaluation without real-world side effects.

Recent work also broadens execution and user interaction. PhoneHarness combines GUI operations, CLI commands, and host-side tools with progressive loading of skill instructions (Li et al., 2026a). MobileWorld combines long-horizon, cross-application workflows with simulated-user clarification, MCP-augmented tasks, and deterministic outcome verification (Kong et al., 2026). None of them, however, compares skill designs as requests evolve and interfaces change (Tab. 1).

To address this gap, we introduce Mobile Skills Bench (MSB, Fig. 1), a controlled testbed for comparing skill designs. It comprises 145 tasks in a simulation environment with 24 applications. Its task contracts specify dependent subgoals, user-held information, staged follow-up requests, and private verification conditions. A user simulator supports clarification and subsequent requests, while re-

Tab. 1: Comparison of mobile-agent evaluation settings.

Benchmark	User Simulator	Hybrid Execution	Avg. Steps	Det. Eval.	Multimodal Skill
AndroidArena (Xing et al., 2024)	×	×	6.6	×	×
A3 (Chai et al., 2026)	×	×	—	×	×
Pro-Bench (Yang et al., 2026)	×	×	—	×	×
AndroidDaily (Sui et al., 2026)	×	×	—	×	×
SPA-Bench (Chen et al., 2025)	×	×	—	×	×
MobileAgentBench (Wang et al., 2024)	×	×	—	✓	×
AndroidWorld (Rawles et al., 2025)	×	◐	14.3	✓	×
AndroidLab (Xu et al., 2025)	×	×	—	◐	×
MobileGym (Wu et al., 2026)	×	×	17.1	✓	×
MobileWorld (Kong et al., 2026)	✓	✓	27.8	✓	×
PhoneHarness (Li et al., 2026a)	×	✓	23.0	◐	×
MSB (ours)	✓	✓	**51.2**	✓	✓

Avg. Steps: average steps per episode (MSB: without skills under Pure GUI, over all nine models); Det. Eval.: deterministic evaluation. ✓/×: documented/not documented in the cited setting; ◐: partial support; —: not reported.

settable application state enables repeatable starting conditions. Moreover, the same task instances can be evaluated through Pure GUI or Hybrid execution. To study skill guidance in this setting, we construct a shared procedure library with Function-grouped, Function-atomic, and App-oriented indexes, rather than separate task-specific solutions. Within each execution mode, textual and multimodal skills share their catalogue and procedures and differ only in access to reference images.

We then evaluate nine advanced models from Google, Anthropic, OpenAI, Moonshot AI, and Alibaba in three agent harnesses to examine how current models perform without skills, whether skills improve the completion of long-horizon interactive tasks, whether reference images add value, how skills affect interaction efficiency, and how a skill library should be organized (Sec. 4). We find that ① without skills, no evaluated model exceeds 48.5% task success, and the failures differ by model family. The models with the highest task success still often fail to ask for the information they lack, the Gemini models often claim completion before the task is done, and the Claude models often stall without recovering. ② Skills improve task success on average, but the gains are uneven. They are largest under Hybrid execution and for models with higher baseline success that read the skills, whereas models with lower baseline success and models that rarely read the skills, such as Gemini 3.1 Pro, gain little. Under Pure GUI, only the two Gemini Flash models gain substantially. ③ Static reference images, however, add no consistent benefit beyond the textual procedures. ④ Moreover, the gains come from converting failed attempts into successes rather than from shortening episodes, so skills enable agents to complete more tasks at a comparable step cost. ⑤ Finally, among the three organizations of the skill library, the App-oriented organization achieves comparable success at the lowest input-token cost.

In summary, our contributions are threefold: ① *A controlled testbed for skill research on mobile agents.* On a resettable simulator with verifiable backend state, MSB poses the same 145 long-horizon tasks, with a shared user simulator and private verifiers, under both Pure GUI and Hybrid execution, so that skill modality, library organization, and execution interface can be varied while everything else stays fixed. ② *A multimodal skill library and its systematic evaluation.* We build a shared library that pairs each textual procedure with reference images of the relevant screens and organizes the same procedures in three ways: Function-atomic, with one package per function–app pair; Function-grouped, with one package per function; and App-oriented, with one package per app. Using this library, we evaluate nine models in three agent harnesses without skills, with textual skills, and with multimodal skills, and we compare the three library organizations (Sec. 4). ③ *Model-specific diagnostics from trajectories.* We analyze the recorded trajectories and trace the failures of each model family to measurable behaviors, such as unasked questions, premature completion claims, and unrecovered stalls (Appendix A.1).

2 Related Work

Mobile Benchmarks. Tab. 1 compares mobile benchmarks by user simulation, hybrid execution, reported execution length, deterministic outcome evaluation, and multimodal skill support. AndroidWorld and AndroidLab establish controlled environments and standardized protocols for mobile-

agent evaluation (Rawles et al., 2025; Xu et al., 2025). Extending evaluation to everyday services, however, introduces opaque backend state, difficult resets, and potentially irreversible actions. MobileGym addresses these obstacles with controllable mock applications and state-based verification, but its benchmark remains centered on GUI execution (Wu et al., 2026). PhoneHarness expands execution to GUI operations, CLI commands, and host-side tools with progressive skill disclosure, although its documented evaluation does not specify a dedicated user simulator (Li et al., 2026a). To incorporate user interaction, MobileWorld combines simulated-user clarification with MCP-augmented cross-app tasks, averaging 27.8 steps (Kong et al., 2026). None of these benchmarks systematically compares textual and multimodal skills, which MSB does through controlled comparisons on verifiable long-horizon tasks under matched Pure GUI and Hybrid execution.

Agent Skills. SkillsBench and SWE-Skills-Bench evaluate skill utility across general and software-engineering tasks, but do not address visual grounding in mobile execution (Li et al., 2026b; Han et al., 2026). MMSkills couples procedures with state-linked reference images and selective reading (Zhang et al., 2026b). VISUALSKILL uses text-only controls derived from the same source material to assess the added value of reference images (Jiang et al., 2026). However, both studies evaluate skills outside the mobile-agent setting, so they do not establish which multimodal skill designs remain effective for long-horizon mobile workflows with evolving user requests and hybrid execution. MSB addresses this gap through controlled comparisons of skill modality, organization, and reading on verifiable, long-horizon interactive mobile tasks.

3 Mobile Skills Bench

3.1 Benchmark Overview

MSB is a controlled benchmark for testing how textual and multimodal skills support long-horizon, interactive mobile agents (Fig. 1). It comprises 145 tasks in an environment with 24 simulated apps (Appendix B, Tab. 5), including cross-app workflows and evolving user requests. A user simulator supplies user-held facts and follow-up requests (Fig. 2). Resettable app state and a simulated clock provide repeatable starting conditions, while private verifiers check task outcomes. Each task runs under two execution modes, Pure GUI and Hybrid. Pure GUI restricts environment actions to GUI operations, whereas Hybrid additionally permits MCP tools and CLI commands. Because both modes share the same tasks, user simulator, and verifier, skill conditions can be compared under controlled conditions. Appendix B describes task and environment construction, and Appendix B.4 lists the MCP-style tools and CLI commands available under Hybrid execution.

3.2 Task Definition

We specify each benchmark task q by a contract

$$\mathcal{C}_q = \big(\{(x_q^m, \mathcal{I}_q^m)\}_{m=0}^{M_q}, V_q\big), \tag{1}$$

which fixes what the user requests, what the user knows, and what counts as success. Stage m pairs a user request x_q^m with the set $\mathcal{I}_q^m$ of user-held facts omitted from that request. The initial request is x_q^0, and the follow-up requests $x_q^1, \ldots, x_q^{M_q}$ arrive as execution progresses. The private verifier V_q encodes the success criteria, and the agent fulfills the contract exactly when $V_q(\tau) = 1$ on the episode trajectory τ defined below. Because completing all stages requires a sequence of task-relevant state changes, each task is long-horizon.

The contract fixes what the user knows and asks for, whereas a single large language model (LLM) user simulator $\mathcal{U}$, shared across tasks, plays the user during execution. Following the user-simulation guidelines of τ^2-bench (Barres et al., 2026), $\mathcal{U}$ answers the agent's questions in its own words, but draws only on the facts of the stages reached so far, keeps every specific value exact, and says that it knows nothing beyond them. Because it sees neither the screen nor the app state, its replies depend only on the dialogue. The follow-up requests, in contrast, are fixed by the contract. When the agent reports the current request as done, $\mathcal{U}$ raises the next one verbatim without grading the work, and the episode ends once no request is left. Consequently, completion is judged only by V_q.

We model task execution as a partially observable Markov decision process (POMDP). The latent state $s_t = (e_t, u_t)$ combines the environment state e_t (app backends, device files, and simulated

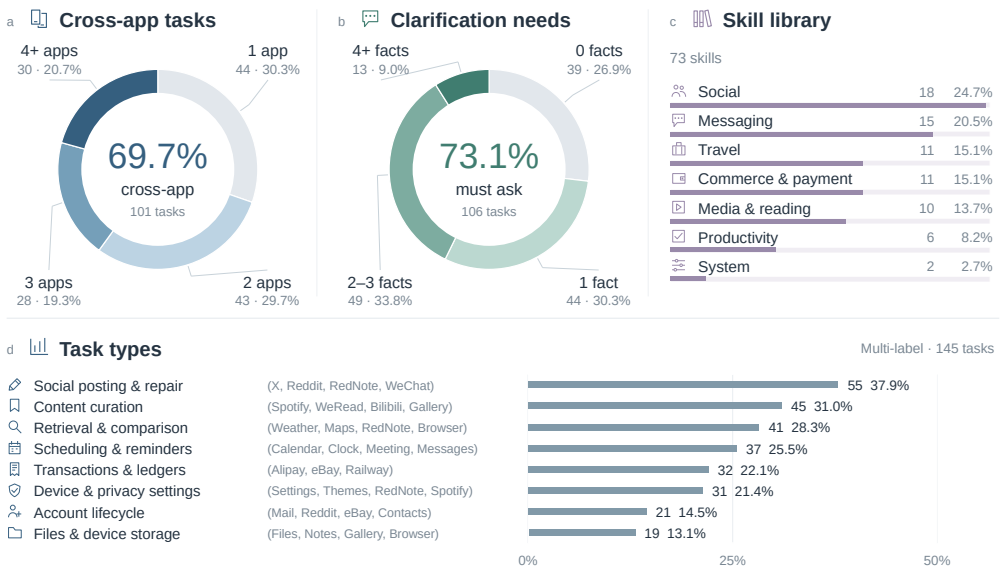


Fig. 2: Task composition and skill coverage in MSB. (a) Apps involved per task, with cross-app tasks requiring at least two apps. (b) User-held facts omitted from the initial instruction and available through clarification. (c) Skill categories of the 73 function–app procedures, which the Function-atomic, Function-grouped, and App-oriented organizations package into 73, 48, and 24 packages. (d) Task activities with representative apps, where a task may carry multiple labels. Task percentages in (a), (b), and (d) use 145 tasks as the denominator.

clock) with the user state u_t (the current stage, disclosed facts, and dialogue history). The agent observes only o_t, comprising screenshots, user messages, and permitted tool outputs, and acts through mobile operations and user communication. Each action induces $s_{t+1} \sim \mathcal{T}(\cdot \mid s_t, a_t)$ and $o_{t+1} \sim \Omega(\cdot \mid s_{t+1}, a_t)$. An episode of T steps yields the trajectory $\tau = (s_0, o_0, a_0, \ldots, a_{T-1}, s_T, o_T)$, from which the verifier deterministically computes $V_q(\tau) \in \{0, 1\}$.

The agent has access to a library of N skills. A multimodal skill $k_i = (d_i, p_i, v_i)$ consists of a description d_i of its purpose and applicability, instructions p_i, and reference images v_i. The three skill conditions share the same N skills and differ only in the components the agent can access:

$$K_\varnothing = \varnothing, \qquad K_{\text{text}} = \{(d_i, p_i)\}_{i=1}^{N}, \qquad K_{\text{MM}} = \{(d_i, p_i, v_i)\}_{i=1}^{N}. \tag{2}$$

At each step, the agent samples an action $a_t \sim \pi_\theta(\cdot \mid h_t, K)$, where h_t contains the initial instruction and the observation–action history, including any skill content that it has read. Conditioning on K denotes access to these resources rather than loading the entire library into context, and the agent observes the live screenshot in every condition.

We compare the conditions by the task success rate under a fixed interaction budget,

$$J(\theta, K) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathbb{E}_{\tau \sim P_{\theta,K}(\cdot|q)}[V_q(\tau)], \tag{3}$$

where $\mathcal{Q}$ is the set of 145 tasks and $P_{\theta,K}$ is the trajectory distribution induced by the agent, environment, and user simulator. We estimate J by averaging V_q over independent runs of each task and compare it across skill conditions while holding the model, tasks, execution mode, and budget fixed.

3.3 Benchmark Composition

Task distributions. MSB is designed to evaluate sustained task completion across applications and user turns, rather than only isolated GUI operations. The 145 tasks span diverse mobile activities, including content curation, social posting and repair, scheduling and reminders, transactions and ledgers, and device and privacy settings (Fig. 2). Of these tasks, 69.7% involve multiple applications, requiring agents to carry intermediate results and constraints across app boundaries. Meanwhile,

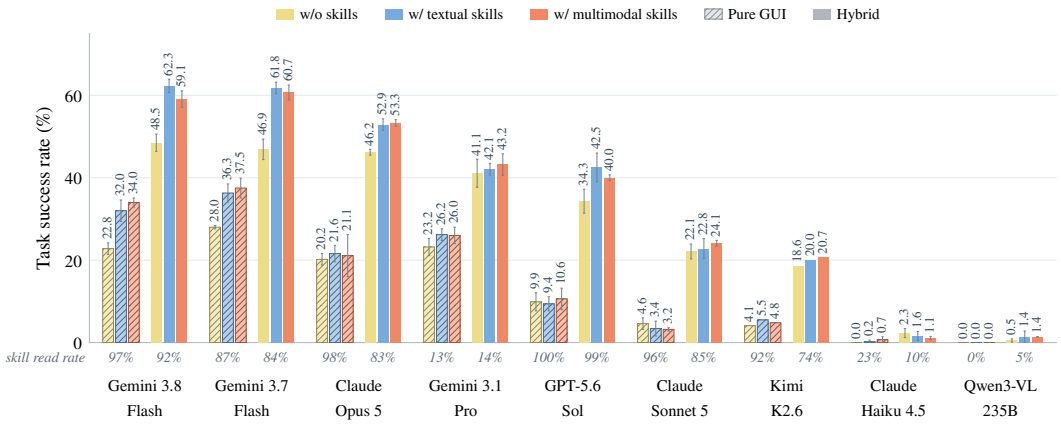


Fig. 3: Task success rate across nine mobile-agent models. Each model has a left triplet for Pure GUI execution (hatched) and a right triplet for Hybrid execution with GUI, MCP, and CLI (solid). Yellow, blue, and orange indicate no skills, textual skills, and multimodal skills, respectively. Bars show the reported mean and error bars show one SD across runs where available. The row labeled "skill read rate" gives, under each triplet, the share of textual-skill episodes in which the agent reads a skill document. Appendix A.2 lists the means and SDs, and Appendix A.4 reports steps.

73.1% require user-held information omitted from the initial instruction, making the recognition and resolution of missing information part of successful execution. Staged follow-up requests further require agents to adapt their plans while preserving relevant results from earlier stages.

Skill library. We organize one shared procedure library in three ways to study how the organization of skills affects task success and token cost. ① **Function-atomic** is a flat catalogue of 73 packages, one for each function in each app, such as `message-send-wechat`. ② **Function-grouped** merges the procedures of the same function into 48 packages, such as `message-send`, whose `SKILL.md` (Zhang et al., 2025) routes the agent to the procedure of each app. ③ **App-oriented** groups the procedures into 24 packages, one per app, whose `SKILL.md` lists the procedures of that app. The procedure documents are byte-identical across the three organizations, and Appendix A.5 shows how each organization routes the agent to them (Fig. 13). Within each organization, multimodal skills interleave the same procedural text with reference images, so textual and multimodal skills share the same catalogue.

4 Experiments

4.1 Experimental Setup

Models. We evaluate nine mobile-agent models from Google, Anthropic, OpenAI, Moonshot AI, and Alibaba. Tab. 6 in Appendix B.3 lists each model with its reference and the harnesses it runs through. Gemini 3.8 Flash also serves as the user simulator for clarification and staged follow-up requests, and replacing it with Claude Haiku 4.5 leaves the main conclusions unchanged (Appendix B.3.3). Each model is evaluated on the 145-task suite under Pure GUI and Hybrid execution, with reset initial states, a 100-step budget, and a 15-step no-progress stopping rule.

Agent harness. We evaluate every model in a minimal ReAct-style observe–act loop (Yao et al., 2023). At each step the model receives the task instruction, the live screenshot, the textual result of its previous action, and, when skills are enabled, the skill catalogue. It then returns one action in JSON, which the harness executes. Unless stated otherwise, skills use the Function-grouped organization (Sec. 3.3), and RQ5 compares the three organizations. The harness enforces the 100-step budget and the 15-step no-progress stopping rule, and uses the same user simulator for every model. Appendix B.3 details the observation, action space, skill access, user simulator, and scoring. To check whether our conclusions depend on the harness, Appendix B.3.2 also runs Claude Opus 5 and GPT-5.6 Sol in the command-line agents of their vendors, Claude Code and Codex.

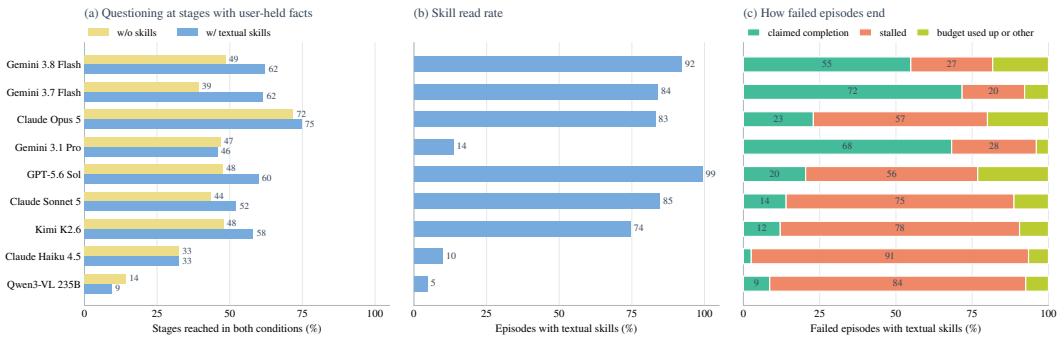


Fig. 4: Behavior of the nine models under Hybrid execution. (a) Share of stages with user-held facts where the model asks a question, without skills (yellow) and with textual skills (blue). (b) Share of textual-skill episodes in which the agent reads a skill document. (c) How failed textual-skill episodes end: the agent claims completion but the verifier rejects it, the harness stops the episode after 15 steps without progress, or the agent uses up the 100-step budget or gives up.

Metrics. The primary metric is success rate: the percentage of episodes in which a private verifier confirms every required outcome and constraint from the app states (Appendix B.3.1). We also report the mean number of steps per episode over all episodes (Appendix A.4). Each setting uses three independent runs unless noted otherwise (Appendix B.3.1). We report the mean and run-to-run standard deviation (SD) of the success rate. Differences are given in percentage points (pp).

4.2 Experimental Results

We examine five research questions (RQs): model performance and limitations without skills (RQ1), skill effectiveness across models and execution modes (RQ2), the added value of reference images (RQ3), the effect of skills on interaction steps (RQ4), and skill organization for mobile agents (RQ5). Appendix C walks through one skill-guided Hybrid episode end to end.

RQ1: How well do current models perform on long-horizon mobile tasks without skills, and what limitations do they exhibit?

Takeaway 1: Without skills, the long-horizon tasks are challenging for every evaluated model, and no model exceeds a task success rate of 48.5%. Models frequently fail to request missing information, and under Hybrid execution the characteristic failure endings differ across model families: the Gemini models tend to claim completion prematurely, the Claude models stall after repeating the same action, and GPT-5.6 Sol stalls despite varied actions. The three models with the lowest success rates, Kimi K2.6, Claude Haiku 4.5, and Qwen3-VL 235B, end the majority of failed episodes in stalls.

Tab. 2 and Fig. 4 show that without skills the tasks are challenging for every evaluated model, and that models often fail to ask for missing information and each has its own bottleneck. Appendix A.1 details the per-model results and the behavioral evidence.

- *Limited success without skills.* The highest success rates are 48.5% under Hybrid execution (Gemini 3.8 Flash) and 28.0% under Pure GUI (Gemini 3.7 Flash) (Tab. 2). Averaged over the two execution modes, the Gemini models complete 32.2–37.5% of the tasks, the Claude models complete between 1.2% (Claude Haiku 4.5) and 33.2% (Claude Opus 5), GPT-5.6 Sol completes 22.1%, and the open-weight models complete 11.4% (Kimi K2.6) and 0.3% (Qwen3-VL 235B).
- *Insufficient questioning.* No model asks a question at more than 75% of the stages with user-held facts, even with textual skills, so the agent often lacks information that the task requires (Fig. 4a).
- *Bottlenecks by model family.* A failed episode ends in a *claimed completion* when the agent reports the task as done but the verifier rejects it, and in a *stall* when the harness stops it after 15 steps without new pages, screens, or information. The Gemini models ask for missing information less

often than Claude Opus 5, at 39–49% against 72% of the stages without skills (Fig. 4a), and often claim completion before the task is done. In addition, Gemini 3.1 Pro seldom reads the skills. The Claude models instead often stall without recovering once the GUI path fails, and Kimi K2.6 and Qwen3-VL 235B stall in most of their failed episodes (Fig. 4b and c). GPT-5.6 Sol also stalls most often, but it repeats the same action less often before the stall (Appendix A.1).

RQ2: To what extent do skills improve long-horizon mobile task success, and how does the improvement vary across models and execution modes?

Takeaway 2: Skills improve task success on average, with larger gains under Hybrid execution and for models with higher baseline success, whereas models with lower baseline success gain little. Under Pure GUI, only the two Gemini Flash models gain substantially, while the other models, including Claude Opus 5 and GPT-5.6 Sol, show little or no gain.

Fig. 3 shows that **skills improve the task success rate on average, but the gain depends on the model and the execution mode**. On average, textual and multimodal skills each add 3.8 pp. Appendix A.2 lists the model-level results (Tab. 3) and read rates (Fig. 5).

- *Models with higher baseline success gain more.* For the five models with the highest success without skills, textual skills add 8.9 pp under Hybrid execution and 4.3 pp under Pure GUI. For the four models with the lowest baseline success, which solve at most 22% of the tasks under Hybrid execution and 5% under Pure GUI without skills, textual skills add only 0.6 and 0.1 pp.
- *A model must also read the skills.* Gemini 3.1 Pro has high baseline success but reads a skill in only 14% of episodes (Fig. 3), so it gains only 1.0 pp under Hybrid execution, although, in the episodes where it does read, its success is 20.3 pp higher than in the no-skill episodes of the same tasks and runs. Conversely, Claude Sonnet 5 and Kimi K2.6 read in 74–96% of episodes yet gain less than 1.5 pp, so reading benefits only models that can execute the procedure.
- *Hybrid execution gains more than Pure GUI.* Averaged over all nine models, textual skills add 5.2 pp under Hybrid execution and 2.4 pp under Pure GUI. This suggests that MCP and CLI actions give the model a more direct way to carry out the procedures.

RQ3: Do reference images in multimodal skills add value beyond textual procedures?

Takeaway 3: Augmenting textual skills with static reference images yields no consistent improvement in task success, whether or not tool and command-line actions are available.

Fig. 6 shows that **adding static reference images to textual skills does not consistently change the task success rate in either execution mode**. Appendix A.3 gives the per-model changes, a trajectory analysis, and an example of a reference image.

- *Absence of a consistent effect.* Across the 18 model–mode settings, adding images raises success in eight, lowers it in eight, and leaves it unchanged in two, with changes between -3.2 and $+2.0$ pp (Fig. 6). The direction even reverses within a model, as Gemini 3.8 Flash gains 2.0 pp under Pure GUI but loses 3.2 pp under Hybrid execution.
- *Unchanged failure modes.* The rate of repeated GUI actions on an identical screen and the distribution of failure endings remain essentially unchanged (Appendix A.3).

RQ4: How do skills affect the number of interaction steps required to complete a task?

Takeaway 4: Skills increase the number of completed tasks at a comparable step cost but seldom reduce the length of individual episodes.

Figs. 3 and 9 show that **skills add completed tasks at a similar step cost but rarely shorten episodes**. For example, textual skills raise the success of Gemini 3.8 Flash under Hybrid execution from 48.5% to 62.3%, while its mean episode grows only from 42 to 47 steps. Across the 18 model–mode settings, however, textual skills shorten the mean episode in only four and the mean episode time in only seven. Appendix A.4 compares paired runs of each task with and without skills and finds two reasons why episodes do not become shorter.

- *Reading offsets the saved operations.* On tasks solved in both conditions, skills save few operations, while the agent spends a few extra steps reading skill documents. On tasks that fail in both conditions, the same reading lengthens the episode without leading to success, for example from 45 to 54 steps, including 7 skill reads, for Gemini 3.8 Flash under Hybrid execution (Fig. 12a,c).
- *Newly solved tasks replace long failures.* Without skills, the agent usually goes through most of the workflow and declares completion with about half of the requirements unmet. With skills, it visits largely the same pages and also completes the missing requirements, so the success takes about as many steps as the failure it replaces. In the same setting, these tasks take 54 steps instead of 51 on average (Fig. 12b).

RQ5: Which organization of a skill library is most suitable for mobile agents?

Takeaway 5: Organizing a skill library by function, by function–app pair, or by app yields comparable task success, whereas organizing the library by app incurs the lowest inference cost and is therefore the most suitable for mobile agents.

Since apps share functions such as messaging, the same procedures can be indexed by function–app pair, function, or app (Sec. 3.3). Fig. 14 compares these three organizations and shows that **they reach similar task success, whereas the App-oriented organization costs the least**. We therefore recommend organizing a mobile skill library by app and placing its catalogue in a cacheable prefix. Appendix A.5 provides the full comparison.

- *Similar success.* The mean success of each organization lies between 43.4% and 44.5%, and in every setting the three organizations differ by at most 4.8 pp.
- *Shortest catalogue.* The catalogue is part of the prompt that the model receives at every step, so a longer catalogue adds input tokens to every call. App-oriented lists one short entry per app and moves each procedure's description into the app's router, which enters the context only after the agent reads it. Function-atomic instead repeats a function's shared description in every app's entry, so its catalogue is about seven times as long.
- *Lowest cost.* App-oriented therefore uses 6–34% fewer input tokens than Function-grouped and 28–63% fewer than Function-atomic. Over the full evaluation of the three Gemini models, this saves an estimated $888 (15%) relative to Function-grouped and $3,342 (41%) relative to Function-atomic at public list prices without caching. Because caching depends on the harness and the serving infrastructure, these figures are an upper bound on the saving (Appendix A.5).

5 Conclusion

We introduce MSB, a controlled and verifiable benchmark for long-horizon, interactive mobile agents with a multi-turn user simulator and matched Pure GUI and Hybrid execution. Our evaluation shows that current models remain far from reliable on such tasks, and that skills contribute mainly through their procedural text when a model reads and follows them, while static reference images add little. Skills yield more successful episodes rather than shorter ones, and the organization of a skill library matters more for cost than for success. These results suggest that future skills should target what agents most often lack, namely knowing when to ask the user, when a task is complete, and how to recover from a dead end. In future work, we will add more simulated apps to broaden the tasks and interaction patterns of MSB, and extend it beyond static interfaces with video demonstrations, such as screen recordings of app workflows, and audio-driven interaction.

USE OF AI TOOLS

We used large language models to polish the writing of this paper.

REFERENCES

Anthropic. Claude Code. Software, 2025a. URL https://github.com/anthropics/claude-code. Accessed September 18, 2026.

Anthropic. Claude Haiku 4.5 System Card. System card, Anthropic, October 2025b. URL https://www.anthropic.com/claude-haiku-4-5-system-card.

Anthropic. Claude Opus 5 System Card. System card, Anthropic, July 2026a. URL https://www.anthropic.com/claude-opus-5-system-card.

Anthropic. Claude Sonnet 5 System Card. System card, Anthropic, June 2026b. URL https://www.anthropic.com/claude-sonnet-5-system-card.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL Technical Report. arXiv preprint arXiv:2511.21631, 2025. URL https://arxiv.org/abs/2511.21631.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2506.07982.

Yuxiang Chai, Shunye Tang, Han Xiao, Weifeng Lin, Hanhao Li, Jiayu Zhang, Liang Liu, Pengxiang Zhao, Guangyi Liu, Guozhi Wang, Shuai Ren, Rongduo Han, Haining Zhang, Siyuan Huang, and Hongsheng Li. A3: Android Agent Arena for Mobile GUI Agents with Essential-State Procedural Evaluation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 3774–3789. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.findings-acl.184. URL https://aclanthology.org/2026.findings-acl.184/.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Yixing Li, Xurui Zhou, Weiwen Liu, Shuai Wang, Kaiwen Zhou, Rui Shao, Liqiang Nie, Yasheng Wang, Jianye Hao, Jun Wang, and Kun Shao. SPA-Bench: A Comprehensive Benchmark for SmartPhone Agent Evaluation. In *International Conference on Learning Representations (ICLR)*, pp. 61769–61806, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/9a75f4dd9679aa4ff80a4e6f0a1dc700-Abstract-Conference.html.

Google. Gemini 3.1 Pro Model Card. Model card, February 2026a. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/. Accessed September 18, 2026.

Google. Gemini 3.7 Flash Model Card. Model card, August 2026b. URL https://deepmind.google/models/model-cards/gemini-3-7-flash/. Accessed September 18, 2026.

Google. Gemini 3.8 Flash Model Card. Model card, September 2026c. URL https://deepmind.google/models/model-cards/gemini-3-8-flash/. Accessed September 18, 2026.

Google. Let Gemini handle your multi-step daily tasks on Android. The Keyword, February 2026d. URL https://blog.google/innovation-and-ai/products/gemini-app/android-multi-step-tasks/. February 25, 2026. Accessed September 18, 2026.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. SWE-Skills-Bench: Do Agent Skills Actually Help in Real-World Software Engineering? arXiv preprint arXiv:2603.15401, 2026. URL https://arxiv.org/abs/2603.15401.

Ziyan Jiang, Li An, Yujian Liu, Jiabao Ji, Qiucheng Wu, Jacob Andreas, Yang Zhang, and Shiyu Chang. VISUALSKILL: Multimodal Skills for Computer-Use Agents. arXiv preprint arXiv:2606.18448, 2026. URL https://arxiv.org/abs/2606.18448.

Kimi Team. Kimi K2.5: Visual Agentic Intelligence. Technical report, Moonshot AI, 2026. URL https://arxiv.org/abs/2602.02276v2.

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

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. arXiv preprint arXiv:2606.14832, 2026a. URL https://arxiv.org/abs/2606.14832.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv preprint arXiv:2602.12670, 2026b. URL https://arxiv.org/abs/2602.12670.

Moonshot AI. Kimi K2.6 Model Card. Official model card, April 2026. URL https://huggingface.co/moonshotai/Kimi-K2.6. Accessed September 18, 2026.

OpenAI. Codex CLI. Software, 2025. URL https://github.com/openai/codex. Accessed September 22, 2026.

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

Chris 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. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/01a83bc2f2732a58e6aa731e659e7101-Abstract-Conference.html.

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. arXiv preprint arXiv:2605.27761, 2026. URL https://arxiv.org/abs/2605.27761.

Luyuan Wang, Yongyu Deng, Yiwei Zha, Guodong Mao, Qinmin Wang, Tianchen Min, Wei Chen, and Shoufa Chen. MobileAgentBench: An Efficient and User-Friendly Benchmark for Mobile LLM Agents. arXiv preprint arXiv:2406.08184, 2024. URL https://arxiv.org/abs/2406.08184.

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. arXiv preprint arXiv:2605.26114, 2026. URL https://arxiv.org/abs/2605.26114. Accepted to EMNLP 2026, main conference.

Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. Understanding the Weakness of Large Language Model Agents within a Complex Android Environment. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 6061–6072. Association for Computing Machinery, 2024. doi: 10.1145/3637528.3671650. URL https://doi.org/10.1145/3637528.3671650.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. AndroidLab: Training and Systematic Benchmarking of Android Autonomous Agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2144–2166. Association for Computational Linguistics, July 2025. doi: 10.18653/v1/2025.acl-long.107. URL https://aclanthology.org/2025.acl-long.107/.

Leyang Yang, Ziwei Wang, Xiaoxuan Tang, Sheng Zhou, Dajun Chen, Wei Jiang, and Yong Li. ProBench: Benchmarking GUI Agents with Accurate Process Information. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(32):27547–27555, 2026. doi: 10.1609/aaai.v40i32.39974. URL https://ojs.aaai.org/index.php/AAAI/article/view/39974.

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

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping Agents for the Real World with Agent Skills. Anthropic Engineering Blog, 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Accessed September 25, 2026.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue (Steve) Liu, Xiaoxiao Li, and Philip S. Yu. CoEvoSkills: Self-Evolving Agent Skills via Co-Evolutionary Verification. In *Proceedings of the Conference on Language Modeling (COLM)*, 2026a. URL https://arxiv.org/abs/2604.01687.

Kangning Zhang, Shuai Shao, Qingyao Li, Jianghao Lin, Lingyue Fu, Shijian Wang, Wenxiang Jiao, Yuan Lu, Weiwen Liu, Weinan Zhang, and Yong Yu. MMSkills: Towards Multimodal Skills for General Visual Agents. arXiv preprint arXiv:2605.13527, 2026b. URL https://arxiv.org/abs/2605.13527.

ZTE Corporation. As an AI-Native Phone Pioneer, nubia Reshapes the Paradigm of Human-Device Interaction at MWC Barcelona 2026. ZTE News, March 2026. URL https://www.zte.com.cn/global/about/news/As-an-AI-Native-Phone-Pioneer-nubia-Reshapes-the-Paradigm-of-Human-Device-Interaction-at-MWC-Barcelona-2026.html. March 2, 2026. Accessed September 18, 2026.

CONTENTS OF THE APPENDIX

A Evidence for the Research Questions **13**

A.1 RQ1: Model Performance and Limitations 13

A.2 RQ2: Skill Effectiveness Across Models and Execution Modes 15

A.3 RQ3: Reference Images in Multimodal Skills 15

A.4 RQ4: Interaction Efficiency 16

A.5 RQ5: Skill Library Organization 19

B Benchmark Construction **22**

B.1 Task Construction 22

B.2 Environment Construction 22

B.3 Agent Harness 23

B.4 Execution Interfaces: MCP and CLI Surfaces 28

C A Skill-Guided Hybrid Execution Case Study **33**

Appendix A gives the evidence for the five research questions in order. Appendix B describes how the benchmark is built, namely the tasks, the environment, the agent harness, and the MCP and CLI interfaces. Appendix C walks through one skill-guided episode end to end.

A EVIDENCE FOR THE RESEARCH QUESTIONS

This appendix gives the evidence for the five research questions in the order of Sec. 4.2. Each subsection starts from the corresponding takeaway and presents only the data needed to support it.

A.1 RQ1: MODEL PERFORMANCE AND LIMITATIONS

This subsection gives the evidence for Takeaway 1. It first compares the success of the models without skills (Tab. 2) and then profiles the nine models under Hybrid execution with three measures, namely whether a model asks the user a question at a stage with user-held facts, whether it reads a skill document, and how its failed episodes end. We show Hybrid execution, in which models succeed most often and gain most from skills, because under Pure GUI execution stalls are the most common failure ending for every model and narrow the differences between models. Fig. 4 shows one measure per panel. The first two measures count actions, such as questions, not their content, so a question may still be irrelevant or incomplete, and a read document may be misapplied.

Success without skills. Tab. 2 shows that **no model exceeds 48.5% success without skills, and that success differs widely across models**. The highest success rates are 48.5% under Hybrid execution (Gemini 3.8 Flash) and 28.0% under Pure GUI (Gemini 3.7 Flash). The Gemini models and Claude Opus 5 lead with a mean of 32.2–37.5% over the two execution modes, followed by GPT-5.6 Sol at 22.1%. Claude Sonnet 5 and Kimi K2.6 reach only 13.4% and 11.4%, and Claude Haiku 4.5 and Qwen3-VL 235B only 1.2% and 0.3%, respectively. Within the Claude family, success falls from Claude Opus 5 to Claude Sonnet 5 to Claude Haiku 4.5, whereas Gemini 3.8 Flash and Gemini 3.7 Flash match or exceed Gemini 3.1 Pro. Restricting agents to GUI actions lowers the mean over all models from 28.9% to 12.5%. Tab. 3 (Appendix A.2) gives the run-to-run SDs.

Questioning. We pair the no-skill and textual-skill episodes of each model on the same task and run. We then keep the stages that have user-held facts and are reached in both episodes. We say that the model asks at a stage if it issues at least one `ASK_USER` action while the stage is active. Fig. 4a shows that **no model asks at more than three quarters of these stages, even with textual**

Tab. 2: Task success (%) of each model without skills under Hybrid and Pure GUI execution, and their mean. Small numbers are run-to-run SDs, Kimi K2.6 has a single run, and the last column averages over the nine models, none of which exceeds 48.5%.

	Gemini 3.7 Flash	Gemini 3.8 Flash	Claude Opus 5	Gemini 3.1 Pro	GPT-5.6 Sol	Claude Sonnet 5	Kimi K2.6	Claude Haiku 4.5	Qwen3-VL 235B	All models
Hybrid	46.9 ±2.5	48.5 ±2.1	46.2 ±0.7	41.1 ±3.4	34.3 ±2.9	22.1 ±1.8	18.6	2.3 ±1.1	0.5 ±0.4	28.9
Pure GUI	28.0 ±0.4	22.8 ±1.4	20.2 ±1.4	23.2 ±2.1	9.9 ±2.2	4.6 ±1.4	4.1	0.0 ±0.0	0.0 ±0.0	12.5
Mean	37.5 ±1.1	35.7 ±0.4	33.2 ±0.9	32.2 ±0.7	22.1 ±1.2	13.4 ±1.6	11.4	1.2 ±0.5	0.3 ±0.2	20.7

skills. Claude Opus 5 asks most often, at 74.8%, Gemini 3.8 Flash and Gemini 3.7 Flash ask at only 62.2% and 61.5%, respectively, and the other six models ask at 9.4–59.9%. The same holds under Pure GUI execution, where Claude Opus 5 again asks most often, at 70.3%. Since these facts are disclosed only on request, each unasked stage leaves the agent without information the task requires.

Skill reads. An episode counts as reading the skills when at least one skill document is served in it. Fig. 4b shows that **Gemini 3.1 Pro rarely reads the skills**. It reads a skill in only 13.6% of Hybrid episodes, far below the 83–92% of the other three models with the highest task success. Only Claude Haiku 4.5 and Qwen3-VL 235B, the two models with the lowest task success, read less often. Under Pure GUI execution, the gap is similar, at 13.1% against 87–98%. As a result, Gemini 3.1 Pro's Hybrid questioning rate stays at 46.9% without skills and 45.8% with them (Fig. 4a).

Failure endings. We classify each failed episode into one of the three endings defined in the caption of Fig. 4, using only the records of the harness and the verifier, without any model judgment (Appendix B.3.1). Fig. 4c shows that **models with similar success fail in different ways, and the way they fail follows the model family**.

- *Gemini models stop too early.* The Gemini models often declare a task finished while much of it is still undone. They end 54.9–71.7% of their failures with a claimed completion, and at that point 43–51% of the initially unmet verifier checks are still unmet. Their main weakness is therefore judging when a task is complete.
- *Claude models rarely recover from stalls.* The Claude models declare completion less often, but once the GUI path fails they keep trying it instead of changing strategy. Claude Opus 5 and Claude Sonnet 5 end 57.1% and 74.7% of their failures in a stall, respectively, although Claude Opus 5 succeeds more often than Gemini 3.1 Pro. In the last 15 steps before a stall, at least 97% are GUI actions and about 60% repeat an earlier action on an identical screen, while tool calls, skill reads, and questions to the user almost never occur. Their main weakness is therefore recovering from a dead end, even though tools, skills, and the user remain available.
- *GPT-5.6 Sol stalls more than it claims completion.* It ends 56.4% of its failures in a stall and 20.4% with a claimed completion. In the last 15 steps before a stall, 98.9% are GUI actions, but only 29.1% repeat an earlier action on the same screen, about half the share of the Claude models.
- *Models with the lowest success stall by default.* Kimi K2.6, Claude Haiku 4.5, and Qwen3-VL 235B end 78.4–90.9% of their failures in a stall, and they solve at most 21% of tasks under Hybrid execution (Tab. 3). For these models, stalling mainly reflects how rarely they make progress, so the ending separates model families only among the models with higher success.
- *A property of the model, not the setting.* The contrast holds with and without textual skills in both execution modes, so it reflects the models rather than the skills or the interface. Under Pure GUI execution with textual skills, the Gemini models still end 19.5–30.7% of their failures with a claimed completion, against at most 3.2% for the two Claude models.

A.2 RQ2: Skill Effectiveness Across Models and Execution Modes

This subsection gives the evidence for Takeaway 2 in two parts. Tab. 3 lists the success of every model together with the change Δ that each skill condition brings over no skills, which shows that skills help in both execution modes and more for models with higher baseline success and under Hybrid execution. Fig. 5 then relates each model's skill read rate to its gain when it reads, which shows that a model benefits only if it both reads the skills and can carry out the procedure.

Tab. 3: Task success (%) for all models, execution modes, and skill conditions: mean over r independent runs of all 145 tasks with the run-to-run SD in small type (single-run cells have no SD). Δ is the change from no skills in pp, and the last column averages the six conditions of each model, by which the rows are sorted. Bold marks the best skill condition within each execution mode, and the last row averages over the nine models. Values are the data behind Fig. 3.

		Pure GUI execution					Hybrid execution					
Model	r	w/o skills	w/ text	Δ	w/ multimodal	Δ	w/o skills	w/ text	Δ	w/ multimodal	Δ	**Average**
Gemini 3.7 Flash	3	28.0 $\pm$0.4	36.3 $\pm$2.2	+8.3	**37.5** $\pm$2.4	+9.5	46.9 $\pm$2.5	**61.8** $\pm$1.4	+14.9	60.7 $\pm$1.8	+13.8	45.2
Gemini 3.8 Flash	3	22.8 $\pm$1.4	32.0 $\pm$2.6	+9.2	**34.0** $\pm$1.1	+11.2	48.5 $\pm$2.1	**62.3** $\pm$1.6	+13.8	59.1 $\pm$2.0	+10.6	43.1
Claude Opus 5	3	20.2 $\pm$1.4	**21.6** $\pm$2.0	+1.4	21.1 $\pm$5.1	+0.9	46.2 $\pm$0.7	52.9 $\pm$1.4	+6.7	**53.3** $\pm$0.8	+7.1	35.9
Gemini 3.1 Pro	3	23.2 $\pm$2.1	**26.2** $\pm$1.4	+3.0	26.0 $\pm$2.0	+2.8	41.1 $\pm$3.4	42.1 $\pm$1.4	+1.0	**43.2** $\pm$2.6	+2.1	33.6
GPT-5.6 Sol	3	9.9 $\pm$2.2	9.4 $\pm$1.7	−0.5	**10.6** $\pm$2.6	+0.7	34.3 $\pm$2.9	**42.5** $\pm$3.5	+8.2	40.0 $\pm$0.7	+5.7	24.4
Claude Sonnet 5	3	**4.6** $\pm$1.4	3.4 $\pm$1.8	−1.2	3.2 $\pm$0.4	−1.4	22.1 $\pm$1.8	22.8 $\pm$2.4	+0.7	**24.1** $\pm$0.7	+2.0	13.4
Kimi K2.6	1	4.1	**5.5**	+1.4	4.8	+0.7	18.6	20.0	+1.4	**20.7**	+2.1	12.3
Claude Haiku 4.5	3	0.0 $\pm$0.0	0.2 $\pm$0.4	+0.2	**0.7** $\pm$0.7	+0.7	**2.3** $\pm$1.1	1.6 $\pm$1.1	−0.7	1.1 $\pm$0.4	−1.2	1.0
Qwen3-VL 235B	3	0.0 $\pm$0.0	0.0 $\pm$0.0	0.0	0.0 $\pm$0.0	0.0	0.5 $\pm$0.4	**1.4** $\pm$1.4	+0.9	**1.4** $\pm$0.0	+0.9	0.5
Average		12.5	15.0	+2.4	15.3	+2.8	28.9	34.2	+5.2	33.7	+4.8	23.3

Gain when a skill is read. Fig. 5 shows that a skill raises success only when a model both reads it and is able to carry out the procedure.

- *Read often and benefit.* Gemini 3.8 Flash, Gemini 3.7 Flash, Claude Opus 5, and GPT-5.6 Sol read skills in 83–99% of Hybrid episodes and gain 8–18 pp in the episodes where they read, so their overall gains in Tab. 3 are among the largest.
- *Read rarely but benefit most.* Gemini 3.1 Pro gains 20.3 pp under Hybrid and 14.0 pp under Pure GUI when it reads, but reads in only 14% of episodes, so its overall gain is small.
- *Gain little.* Claude Sonnet 5 and Kimi K2.6 read in 74–96% of episodes but gain at most 2.2 pp when they read, because they cannot carry out the procedure. Claude Haiku 4.5 and Qwen3-VL 235B read in at most 23% of episodes and gain at most 4.7 pp when they do.

The gain when reading is also smaller under Pure GUI, for example 1.4 pp for Claude Opus 5 against 9.1 pp under Hybrid execution, which matches the smaller overall gain in this mode. Because each model decides when to read, these gains are associations rather than causal effects.

A.3 RQ3: Reference Images in Multimodal Skills

This subsection gives the evidence for Takeaway 3 in two parts that follow the main text. First, Fig. 6 shows how adding reference images changes success for every model and execution mode. Then Fig. 7 compares the failure modes with and without images, and Fig. 8 shows a reference image.

Absence of a consistent effect. Fig. 6 shows that adding reference images changes task success in no consistent direction. Across the 18 model–mode settings, the change ranges from -3.2 to $+2.0$ pp, with eight increases, eight decreases, and two ties. Moreover, the direction can differ between execution modes for the same model. For example, Gemini 3.8 Flash gains 2.0 pp under Pure GUI but loses 3.2 pp under Hybrid execution. All of these changes are also small compared with the gains from textual skills, which reach 14.9 pp (Tab. 3).

Failure modes. We further compare the textual-skill and multimodal-skill episodes of the six models that usually read skills, namely Gemini 3.8 Flash, Gemini 3.7 Flash, Claude Opus 5, Claude

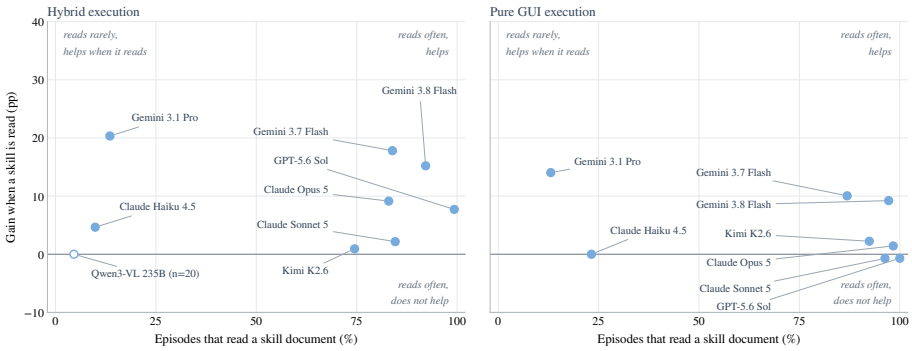


Fig. 5: Skill read rate against the gain when a skill is read, with textual skills. The horizontal axis is the share of episodes in which the agent reads a skill document. The vertical axis is their success minus that of the no-skill episode on the same task and run. The hollow marker is based on fewer than 30 episodes. Skills raise success only for models that read them and can carry out the procedure.

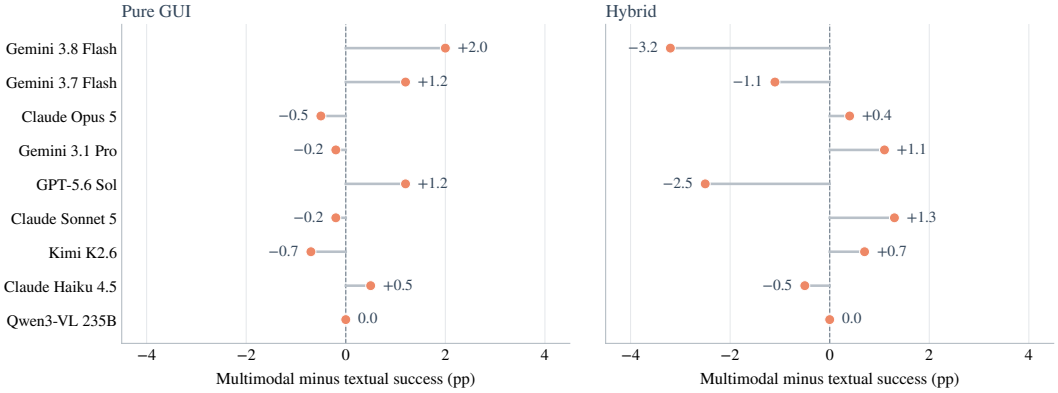


Fig. 6: Change in task success when static reference images are added to textual skills, for all nine models under Pure GUI (left) and Hybrid (right) execution. Each value is the multimodal minus the textual mean in Tab. 3, in pp. The change has no consistent direction across models or modes.

Sonnet 5, Kimi K2.6, and GPT-5.6 Sol. Fig. 7 shows that **the images leave the failure modes unchanged**. The images do enter GUI decisions, as the agent issues a GUI action at 88% and 82% of the steps that receive one. Nevertheless, about 18% of the GUI actions under Pure GUI and 16% under Hybrid execution repeat an earlier action on an identical screen regardless of the skill modality. Likewise, the shares of failed episodes that end in a stall or in a claimed completion differ by at most 1.4 pp between textual and multimodal skills.

A.4 RQ4: Interaction Efficiency

This subsection presents the evidence for Takeaway 4. It first shows that skills do not consistently shorten episodes and change their time in no consistent direction (Figs. 9 and 10). It then explains why, using the added steps (Fig. 11) and paired runs by task outcome (Fig. 12). Finally, it reports each model's cost per run (Tab. 4).

Episode length and time. Figs. 9 and 10 show that **skills increase the number of completed tasks, whereas the number of steps does not consistently decrease and the episode time changes in no consistent direction**. For the Gemini Flash models and Claude Opus 5, the mean number of steps changes by -5% to $+12\%$, whereas the number of completed tasks increases by 6–40% (Fig. 3). An episode takes 3.3 to 13.8 minutes on average, and model inference accounts for 77–92% of this time. Textual skills shorten the mean episode time in seven of the 18 settings and lengthen it in the other 11, with changes from -34% to $+38\%$. These changes mainly reflect differences in model-call speed between runs rather than in the number of steps. We exclude six runs in which

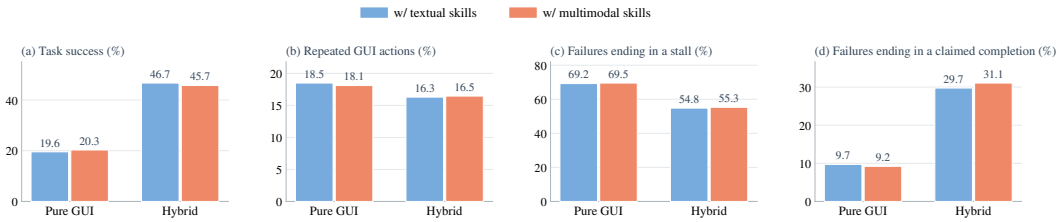


Fig. 7: Task success (a), GUI actions that repeat an earlier action on an identical screen (b), and failed episodes that end in a stall (c) or in a completion claim that the verifier rejects (d), with textual and multimodal skills for the six models that usually read skills. Reference images change neither repeated GUI actions nor failure endings.

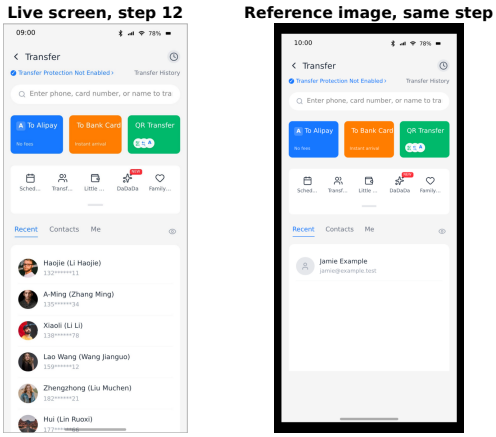


Fig. 8: A live screen and the reference image attached for the same Alipay transfer page. Both show the same layout and transfer options.

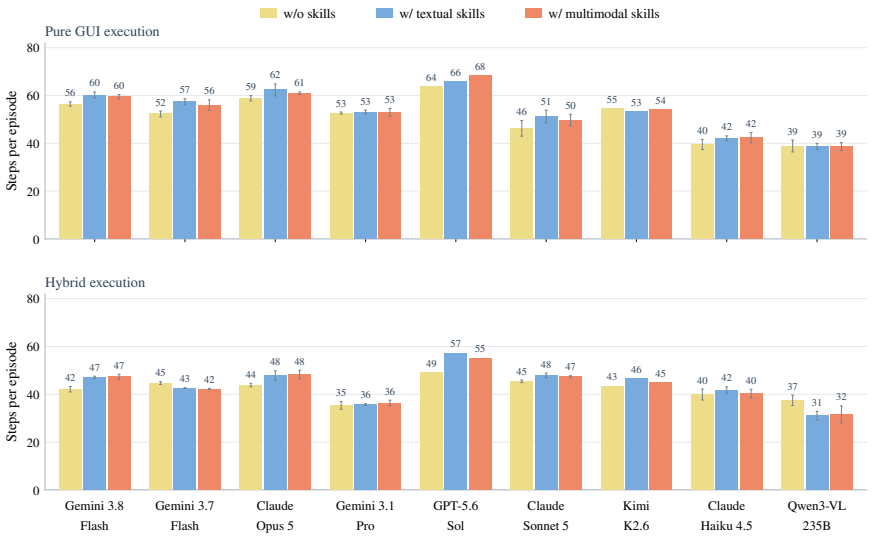


Fig. 9: Mean number of agent steps per episode for the nine models of Fig. 3 under Pure GUI (top) and Hybrid (bottom) execution, averaged over all valid episodes including failures. Yellow, blue, and orange denote no skills, textual skills, and multimodal skills, and whiskers show the SD across runs. Skills leave the mean episode length nearly unchanged, and textual skills shorten it in only four of the 18 settings.

the median model call takes more than twice as long as usual for that model, because this delay originates in the API rather than in the agent.

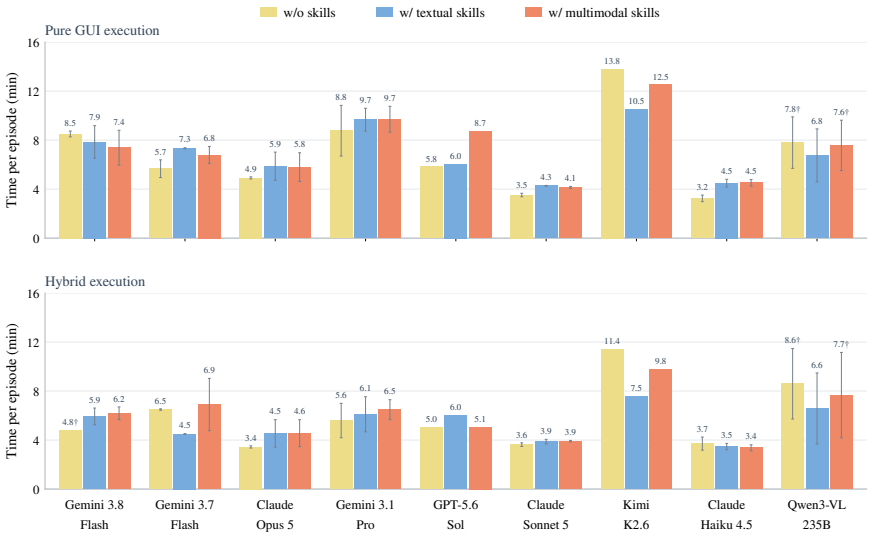


Fig. 10: Mean wall-clock time per episode in minutes for the nine models of Fig. 3 under Pure GUI (top) and Hybrid (bottom) execution, averaged over all valid episodes including failures. The time covers the agent's interaction from the first to the last step and excludes environment setup and verification. Yellow, blue, and orange denote no skills, textual skills, and multimodal skills, and whiskers show the SD across runs. A dagger marks cells from which runs slowed by the API are removed. Textual skills shorten the mean episode time in only seven of the 18 settings.

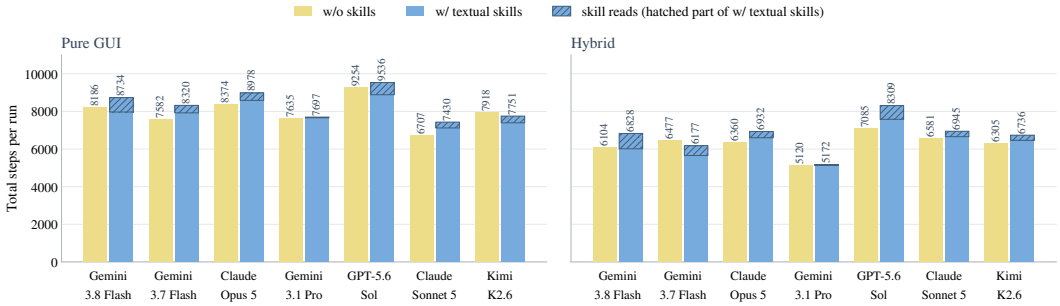


Fig. 11: Total steps per run of 145 tasks without skills (yellow) and with textual skills, split into skill-document reads (hatched) and all other steps (blue), for the seven models that complete at least five tasks per run without skills. For the Gemini Flash models and Claude Opus 5, the added steps consist mainly of skill reads.

Why episodes do not become shorter. Pairing the runs of each task with and without skills shows that no outcome group becomes consistently shorter with skills, for two reasons (Fig. 12).

- *Reading offsets the saved operations.* Fig. 11 shows that the added steps are mostly skill reads. For the Gemini Flash models and Claude Opus 5, reading adds roughly 2 to 6 steps per episode, while the other steps change by less than 5% except for Gemini 3.7 Flash under Hybrid execution. On tasks solved in both conditions, the other steps decrease by 5–10% in three of the four Gemini Flash settings, but the reads outweigh this saving everywhere except Gemini 3.7 Flash under Hybrid execution (Fig. 12a). On tasks that fail in both conditions, which form the largest group, episodes become longer by 1.5 to 8.9 steps in seven of the ten settings, and reading accounts for about half or more of each increase (Fig. 12c).
- *Newly solved tasks replace long failures.* Successful episodes on these tasks take 47–71 steps, comparable to the 44–62 steps of the failed attempts that they replace (Fig. 12b). The failed

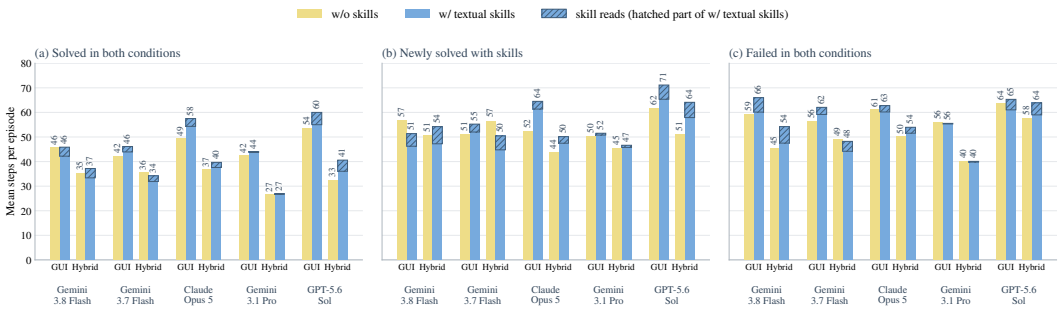


Fig. 12: Mean steps on paired runs of the same task without skills (yellow) and with textual skills (blue), grouped by outcome: (a) solved in both conditions, (b) newly solved with skills, and (c) failed in both conditions. The hatched segment of each textual-skill bar denotes skill reads. The five models shown newly solve at least 28 tasks in each execution mode, counting each of the three runs separately. Reading offsets the operations saved on tasks solved in both conditions (a), newly solved tasks require about as many steps as the failed attempts they replace (b), and failed episodes lengthen mainly through reading (c). No outcome group becomes consistently shorter with skills.

Tab. 4: Inference cost in US dollars of one run of the 145 tasks with the Function-grouped library. Each cost is 145 times the mean episode cost over runs, at public list prices with every input token at the full rate. As in Appendix A.5, Gemini Flash uses its regular rate, not the introductory one.

	Pure GUI			Hybrid		
Model	**No skills**	**Textual skills**	**Multimodal skills**	**No skills**	**Textual skills**	**Multimodal skills**
Gemini 3.8 Flash	47	139	138	118	198	201
Gemini 3.7 Flash	46	122	119	142	180	180
Claude Opus 5	241	649	631	559	886	901
Gemini 3.1 Pro	77	145	144	127	170	174
Claude Sonnet 5	80	223	216	252	369	368
Kimi K2.6	82	110	112	101	130	129
Claude Haiku 4.5	33	92	92	96	110	104
Qwen3-VL 235B	14	22	22	31	28	29

attempts do not end early, since 59% of them end with a claimed completion and 31% in a stall. They also follow largely the same workflow. On average, they visit 76% of the app pages of the later success, whereas two successful runs of the same task share 92–96%. However, they meet only 37–54% of the checks that are unmet at the start. Skills therefore lead the agent to complete the missing requirements along the same workflow rather than to take a shorter route. For example, in a task that prepares the phone for a morning flight, Gemini 3.8 Flash under Hybrid execution without skills makes every other required change but turns Bluetooth off, although the task requires it to stay on, and declares completion after 27 steps. With skills, it reads the device-configuration skill, keeps Bluetooth on, and succeeds in 30 steps, 5 of which are reads.

Inference cost. Beyond steps and time, Tab. 4 gives the cost of one 145-task run per model.

- *Skills raise the cost.* Textual skills make a run 1.3–2.9 times as expensive under Pure GUI and at most 1.7 times under Hybrid, and multimodal-skill runs cost within 5% of textual-skill runs.
- *Claude Opus 5 costs the most.* A run of Claude Opus 5 costs $241–901, which is 3.1–4.5 times the cost of the most expensive Gemini model in the same setting.
- *Qwen3-VL 235B costs the least.* A run costs $14–31 for it and $82–130 for Kimi K2.6.

A.5 RQ5: Skill Library Organization

This subsection gives the evidence for Takeaway 5 on the three Gemini models. It first describes how the three libraries differ (Fig. 13) and then compares their success and inference cost (Fig. 14).

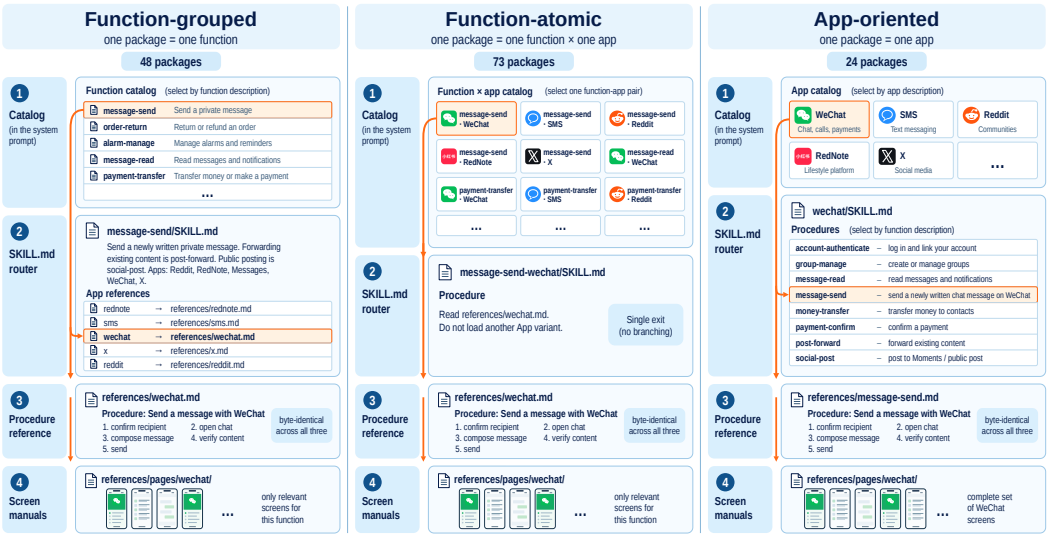


Fig. 13: Three catalogue structures for the same mobile procedures. Function-grouped (left) selects a function and then an app, Function-atomic (middle) selects a function–app pair, and App-oriented (right) selects an app and then a function. Each path retains a `SKILL.md` router and reaches byte-identical instructions, shown here for WeChat messaging. The function-based variants provide function-relevant screen manuals, whereas App-oriented provides all manuals of the app.

Library structures. Fig. 13 compares how an agent reaches a procedure in each organization, with WeChat messaging as the example. It shows that **the three organizations lead to byte-identical procedures but differ in the catalogue, the router, and the scope of the screen manuals**. The grouped organizations exploit the structure of mobile apps, which share functions such as registration, login, and messaging while each app supports several functions, and both aim to shorten the catalogue in every action prompt. In every organization, a procedure is reached through four layers.

- *Catalogue.* The system prompt lists one entry per package, and the agent selects a package by its description. Function-grouped lists 48 functions, such as `message-send`. Function-atomic lists 73 function–app pairs, such as `message-send` for WeChat, so the agent chooses the function and the app in one step. App-oriented lists 24 apps, such as WeChat. With the fewest entries, the App-oriented catalogue is the shortest, and the Function-atomic catalogue is the longest.
- *Router.* Each package opens with a `SKILL.md` file that routes the agent to a procedure. In Function-grouped, it describes the function, separates it from related functions, and maps each app that supports the function to a reference file, for example five apps for `message-send`. In Function-atomic, it points to the single reference of its app and forbids loading the variant of another app. In App-oriented, it lists all procedures of the app with one-line descriptions, for example eight for WeChat, and the agent selects the one it needs.
- *Procedure reference.* For a given function and app, all three paths end in the same procedure document. Its text is byte-identical across the three organizations, and only its file path differs. For WeChat messaging, it is a five-step procedure that confirms the recipient, opens the chat, composes the message, verifies the content, and sends it.
- *Screen manuals.* Each package also carries screen manuals, which describe in text the controls on each of the app's screens. The two function-based packages keep only the screens that the function uses, whereas the App-oriented package keeps all screens of the app.

Across the whole library, however, the three organizations provide the same content, namely the same 73 procedure documents and the same 353 screen manuals, whose text differs only in the file paths of cross-references. The organizations differ only in how this content is divided into packages and found, and in every organization the agent can read any document of the library.

Success and cost. Fig. 14 shows that the three organizations differ little in task success, and the App-oriented organization achieves the best balance between success and cost.

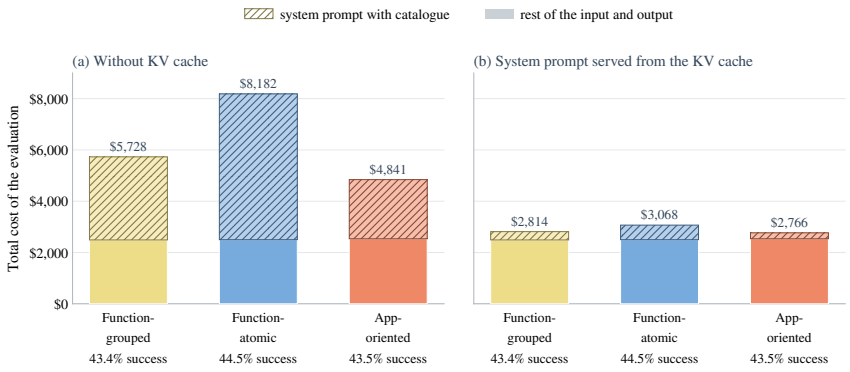


Fig. 14: Total inference cost of the organization comparison, without key–value (KV) cache (a) and with the system prompt served from the KV cache at a tenth of the input price (b). Each bar sums the 5,220 episodes of one organization, which cover the three Gemini models, both execution modes, both skill modalities, and three runs of the 145 tasks, at the standard Gemini API list prices. The hatched segment is the cost of the system prompt, which holds the catalogue and is resent at every step, and the solid segment covers the rest of the input and the output. The label under each bar gives the mean task success. The three organizations reach similar success, and App-oriented costs the least in both cases, although caching reduces its saving over Function-grouped to 2%.

- *Similar success.* The three organizations reach nearly the same success. Averaged over models, execution modes, and modalities, Function-grouped, Function-atomic, and App-oriented reach 43.4%, 44.5%, and 43.5%, respectively, and in every setting they lie within 4.8 pp of each other. The largest gap occurs for Gemini 3.1 Pro under Hybrid execution, where Function-atomic exceeds App-oriented by 3.3 and 4.8 pp with textual and multimodal skills, respectively, but with 62–66% more input tokens.
- *Lowest cost without caching.* Without caching, App-oriented is the cheapest organization, because it has the shortest catalogue and every action prompt resends the catalogue. It uses the fewest input tokens in every setting, 6–34% fewer than Function-grouped and 28–63% fewer than Function-atomic, and on average 31% fewer than the other two organizations. At the standard Gemini API list prices, the full evaluation costs $4,841 with App-oriented, against $5,728 with Function-grouped and $8,182 with Function-atomic (Fig. 14a). App-oriented therefore saves $888 (15%) and $3,342 (41%), or about $25 and $93 per run.
- *Smaller saving with KV cache.* KV caching removes most of the saving over Function-grouped but keeps a clear saving over Function-atomic. The system prompt, which holds the catalogue, is identical across tasks and steps, so the provider can reuse it from the KV cache and charge a tenth of the input price for it. The remaining cost, which covers the action log, the screenshots, and the output, is about $2,500 for every organization (solid segments in Fig. 14). With the system prompt cached, the evaluation therefore costs $2,766 with App-oriented, $2,814 with Function-grouped, and $3,068 with Function-atomic (Fig. 14b). App-oriented then saves only $48 (2%) over Function-grouped but still $302 (10%) over Function-atomic.

Recommendations. These results suggest a simple default for building a mobile skill library, namely to index it by app and to keep the catalogue short and in a fixed prefix.

- *Organize the library by app.* App-oriented matches the success of the other two organizations at the lowest cost, so it is a good default. It saves tokens because every action prompt resends the catalogue, about 49 times per episode, whereas a router document enters the context only after the agent reads it, and only for the apps that the task uses. App-oriented keeps one short entry per app in the catalogue, which names the app and its functions, and moves the description of each procedure into the router of the app. Its catalogue therefore grows with the number of apps rather than with the number of functions or function–app pairs.
- *Avoid a flat function–app catalogue.* Function-atomic lists every function–app pair and repeats the shared description of a function in the entry of each app, adding only a short app-specific note. Its catalogue is therefore about seven times as long as the App-oriented one. As a result, it costs

69% more than App-oriented without caching and 11% more with caching, while it improves the average success by only 1 pp. It is preferable only when success matters more than cost and the model benefits clearly from it, as Gemini 3.1 Pro does under Hybrid execution.

- *Keep the catalogue cacheable.* The catalogue should sit in the fixed system prompt, ahead of any content that changes between steps, so that the provider can serve it from the KV cache. It then costs a tenth as much, and App-oriented and Function-grouped cost almost the same.

B Benchmark Construction

This appendix describes how MSB is built. Appendix B.1 describes how each task is built in five steps, and Appendix B.2 how the simulated environment with its 24 apps and two execution modes is built. Appendix B.3 describes the agent harness and compares it with vendor CLI harnesses and the user simulator with another user simulator LLM, and Appendix B.4 lists the MCP tools and CLI commands that are available under Hybrid execution.

B.1 Task Construction

Task construction proceeds in five steps (Fig. 15), and each task is specified as a contract (Sec. 3.2) before any user-facing text is written.

- *User scenario.* We start from a realistic user goal and the app workflow that achieves it.
- *Task contract.* We specify the app-state changes that define success and the constraints the agent must respect. We then divide the task into stages and decide which facts each request states and which remain with the user simulator. The simulator discloses these facts only when asked and delivers the next follow-up request once the agent reports the current stage as done.
- *Instantiation.* We assign concrete entities and values, set the simulated clock, and prepare matching initial app states that keep shared information consistent across apps.
- *Natural language.* We rewrite the requests as natural user messages without changing their requirements or revealing user-held facts early. For example, a request can leave out the user's budget until the agent asks for it.
- *Validation.* We check that the initial states match the contract, that a constructed correct end state passes the verifier while the initial state fails, and that the verifier tests the intended outcomes and constraints. In addition, the authors manually review each task to confirm that it can be completed through the apps' interfaces. We then freeze the task for evaluation.

B.2 Environment Construction

Simulated environment. MSB builds on the browser-hosted simulator of MobileGym (Wu et al., 2026) and extends it with multi-turn task interaction and hybrid execution interfaces. The environment contains 24 simulated apps whose backend state can be inspected and reset, which allows repeatable trials and outcome verification without real-world side effects. Tab. 5 lists the apps, their main functions, and the number of tasks that involve each app. Fig. 16 shows representative pages. Beyond the pages of MobileGym, MSB adds a Mail app and 35 pages to the other apps, mainly for account sign-up, sign-in, and recovery and for orders, payments, and ticket changes.

Skill library. In addition to the apps, the environment includes the skill library used in the skill conditions. Its documents are written and revised with Claude Fable 5 from the simulator's source code, namely its page components, navigation declarations, and interface strings, and the authors then review every document. Throughout, no skill contains task answers or private verifier state, and the library is frozen before any skill-condition run.

Pure GUI and Hybrid execution. The two execution modes differ only in the interfaces they expose. Pure GUI exposes screenshots and GUI actions, whereas Hybrid additionally permits MCP-style tools and CLI commands. Both modes share the same user simulator and, when skills are enabled, the same access to skills. GUI actions and structured tools operate on the same simulated app state, so the agent can choose freely among the interfaces its mode permits. The tools are backed by the simulator and cannot access the private verifier state (Appendix B.4).

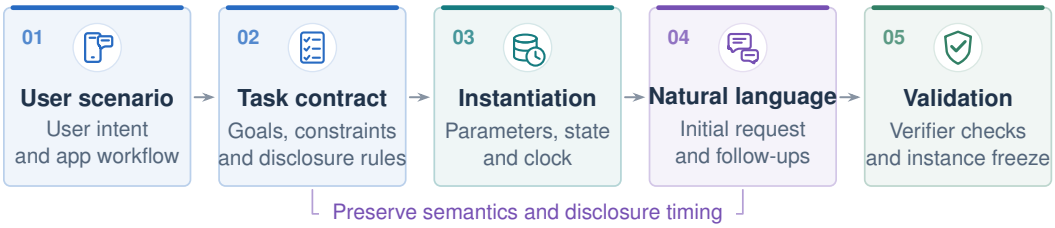


Fig. 15: Contract-first task construction in five steps, from a user scenario to a validated and frozen task. Natural-language rewriting changes the wording of a request but neither its requirements nor when the user discloses information.

Tab. 5: The 24 simulated apps grouped by category, with their main functions and the number of the 145 tasks that involve each app. The device also provides three utility apps that no task uses.

Category	App	Main functions	Tasks
Social and messaging	WeChat	Chats, contacts, Moments, transfers, red packets	21
	X	Posts, replies, bookmarks, follows, messages, privacy settings	28
	Reddit	Communities, posts, comments, saved posts, chat, registration	27
	RedNote	Notes, likes, collections, follows, chats, privacy settings	22
	Messages	SMS conversations	14
	Mail	Inbox, compose, reply, and forward	12
Shopping and payment	eBay	Search, cart and checkout, orders, returns and refunds, listings	16
	Alipay	Balance, bills, bank cards, transfers, payments and refunds	20
Travel and navigation	Railway 12306	Train search, booking, passengers, ticket changes, invoices	13
	Maps	Place search, routes, saved places, navigation settings	12
	Weather	City management, forecasts, air quality, alert settings	14
Media and reading	Spotify	Search, playlists, liked songs, queue, playback modes, sign-up	19
	Bilibili	Video search, favorite folders, follows, likes, profile	12
	WeRead	Bookshelf, reading, book lists, reading settings	14
	Gallery	Albums, photo viewing, and sharing	4
Productivity	Calendar	Events, search, and calendar subscriptions	24
	Clock	Alarms, timers, stopwatch, world clock	16
	Notes	Notes, folders, and to-do lists	16
	Tencent Meeting	Scheduling and editing meetings, recurring meetings, attendees	9
	Contacts	Contacts, call log, and dialing	10
	Files	Folders, search, move, rename, and preview	10
	Browser	Web browsing and search	6
System and utilities	Settings	Display, notification, and privacy settings	7
	Themes	Browsing and applying themes and wallpapers	2

B.3 Agent Harness

This subsection lists the evaluated models and describes the agent harness, namely what the model observes, which actions it can take, how it reaches skills and the user, and how an episode is scored (Appendix B.3.1). It then checks whether our conclusions depend on the harness by comparing it with two vendor CLI harnesses (Appendix B.3.2), and whether they depend on the user simulator by replacing it with Claude Haiku 4.5 (Appendix B.3.3).

B.3.1 Models and Harness Design

Because MSB is difficult, we mainly evaluate the most advanced models available (Tab. 6), namely Gemini 3.8 Flash, Gemini 3.7 Flash, Gemini 3.1 Pro, Claude Opus 5, Claude Sonnet 5, and GPT-5.6 Sol. We add Claude Haiku 4.5 and the open-weight models Kimi K2.6 and Qwen3-VL 235B to cover smaller and openly available models. Because of budget constraints, Kimi K2.6 is evaluated in a single run per setting. GPT-5.6 Sol is evaluated separately on the same tasks under the same protocol, with the same Gemini 3.8 Flash user simulator served through a different Google API endpoint, and Appendix A analyzes its episodes in the same way as those of the other models.

Every model in Sec. 4 runs in the same harness, which connects the phone simulator, the user simulator, the skill file system, and the verifier and is held fixed across models. Each step corresponds to

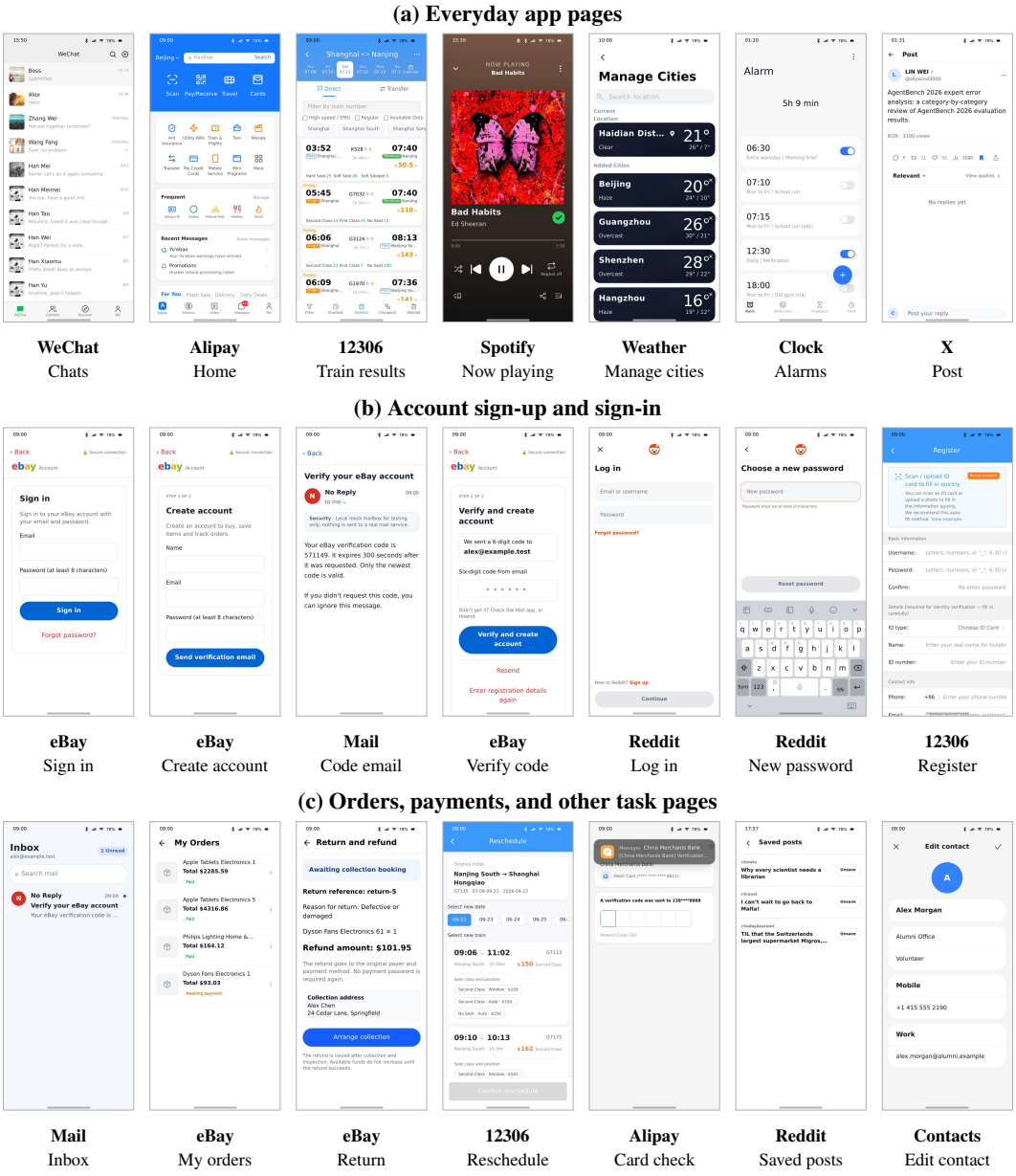


Fig. 16: Representative pages of the simulated apps, taken from evaluation episodes. (a) Everyday app pages. (b) Account sign-up and sign-in. To create an eBay account, for example, the agent must read the verification code sent to the Mail app. (c) Orders, payments, and other task pages.

one model call. The harness assembles the observation, the model returns one action in JSON, and the harness executes the action and records the step.

Observation. Each request contains the task instruction, the current screenshot, and the textual result of the previous action. This result is the return value of a tool call, the output of a CLI command, the user's reply, or, for a GUI gesture, whether the screen changed. The conversation history also keeps the model's own previous actions and their textual results. Screen coordinates are normalized to a 1000×1000 space. The harness reports an error only for an invalid action, such as a malformed action, an unknown tool, or a tool call that fails with an error code. It neither confirms successful actions nor describes the screen.

Action space. Under Hybrid execution, the model selects one of 15 actions in three groups.

Tab. 6: Models evaluated on MSB. All models run in our harness. Claude Opus 5 and GPT-5.6 Sol are additionally run through Claude Code (Anthropic, 2025a) and Codex (OpenAI, 2025) for the harness comparison in Appendix B.3.2. Gemini 3.8 Flash also serves as the user simulator, and Claude Haiku 4.5 replaces it in Appendix B.3.3.

Developer	Model	Harness
Google	Gemini 3.8 Flash (Google, 2026c)	Ours
Google	Gemini 3.7 Flash (Google, 2026b)	Ours
Google	Gemini 3.1 Pro (Google, 2026a)	Ours
Anthropic	Claude Opus 5 (Anthropic, 2026a)	Ours; Claude Code
Anthropic	Claude Sonnet 5 (Anthropic, 2026b)	Ours
Anthropic	Claude Haiku 4.5 (Anthropic, 2025b)	Ours
OpenAI	GPT-5.6 Sol (OpenAI, 2026)	Ours; Codex
Moonshot AI	Kimi K2.6 (Moonshot AI, 2026; Kimi Team, 2026)	Ours
Alibaba	Qwen3-VL 235B (Bai et al., 2025)	Ours

- *Screen actions.* Eight actions operate the screen, namely `CLICK`, `LONG_PRESS`, `SWIPE`, `TYPE`, `BACK`, `HOME`, `RECENT`, and `WAIT`.
- *Programmatic actions.* `TOOL_CALL` invokes an app's MCP tool with named arguments, and `SHELL_EXEC` runs a CLI command in the phone's shell, which is also how skill documents are read. Write tools that change an app's records require a two-step confirmation. The first call returns a preview and a confirmation token, and repeating the call with the token executes the write.
- *User and completion.* The remaining five actions involve the user or end a request. `ASK_USER` asks the user simulator a question, and `REQUEST_USER_INTERVENTION` hands a focused credential field to the user. In addition, `ANSWER` submits a structured answer, `COMPLETE` reports the current request as done, and `ABORT` gives up. After `COMPLETE`, the next request is raised or, if none remains, the episode ends.
- *Execution modes.* Compared with Hybrid execution, Pure GUI execution removes `TOOL_CALL` and limits `SHELL_EXEC` to the read-only documentation commands (`cat` and `ls` under `/system/skills`), so skills remain accessible in both modes. Pure GUI execution offers 14 actions with skills and 13 without. The harness is otherwise identical across modes.
- *Format adaptation.* Model outputs are adapted only when a model emits actions in a format that the protocol does not define. The only such adaptation in the reported runs rescales Kimi K2.6's fractional coordinates to the protocol's space, and the harness records it in every affected step.

Skill access. Skill access follows the progressive disclosure of Agent Skills (Zhang et al., 2025): the prompt lists only the catalogue, and the agent reads documents on demand. Skills are read through `SHELL_EXEC`, and each read counts as one step within the same 100-step budget.

- *Catalogue.* When skills are enabled, the system prompt lists the skill catalogue of the organization under evaluation (Sec. 3.3), with one line per package, giving its name and a short description.
- *Shared library.* All tasks are evaluated against the same catalogue, and the harness neither pre-selects nor injects the skills that a task requires. This design mirrors realistic deployment, in which an agent selects among the installed skills without prior knowledge of which ones a task needs. The skill conditions therefore quantify the benefit of skills under autonomous selection and retrieval, rather than an upper bound obtained by supplying the required skills. Consequently, whether a model reads the relevant skills (Fig. 4b) constitutes part of the measured effect.
- *Reference images.* With multimodal skills, reading a document also unlocks its reference images. After a read, the harness attaches these references to later requests, marked as skill references rather than as screenshots of the live phone.

User simulator. For every agent model, the user is simulated by Gemini 3.8 Flash following the user-simulation guidelines of τ^2-bench (Barres et al., 2026). Within this role, it answers `ASK_USER` questions in its own words from the user-held facts of the stages reached so far, keeps every specific value exact, and replies that it does not know when a question goes beyond them. It also sees neither the screen nor the app state, and it neither volunteers information nor plans for the agent. The

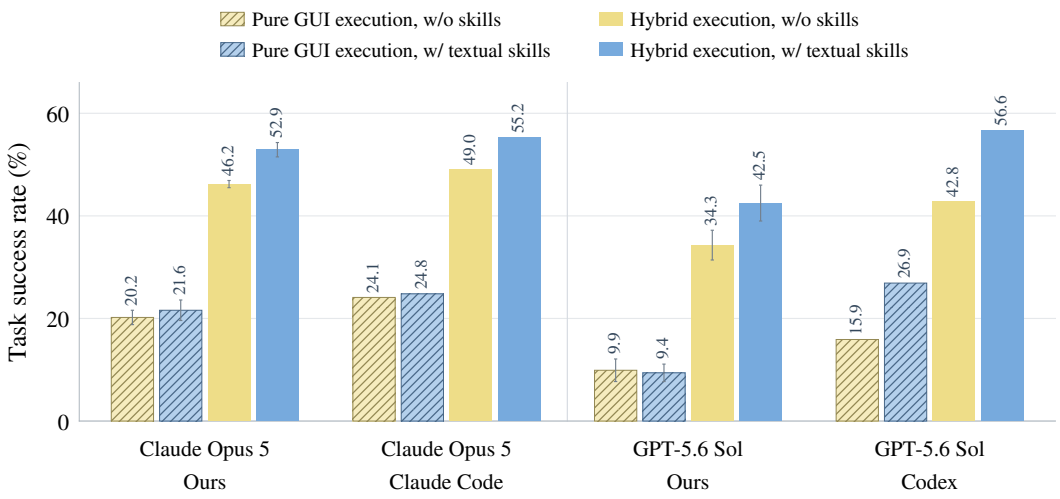


Fig. 17: Same model, different harness: task success rate of Claude Opus 5 and GPT-5.6 Sol under our harness (mean of three runs, SD whiskers) and under the vendor CLI harness (Claude Code and Codex, one run). Hatched bars are Pure GUI, and solid bars are Hybrid. Yellow is without skills, and blue is with textual skills. Task success remains limited under the vendor CLI harnesses, and the comparisons between execution modes, skill conditions, and models are largely preserved.

follow-up requests, by contrast, are fixed by the task: when the agent reports the current request as done, the simulator raises the next one in its authored wording without grading the work. Moreover, the simulator never operates the phone. Instead, the only device input made on the user's behalf is a scripted credential entry. When the agent issues `REQUEST_USER_INTERVENTION` while a password, PIN, or card field is open, the harness enters the task's credential into that field without revealing it, and the agent receives only a confirmation. If no such field is open, the request is refused, and in either case it counts as one step. Calls to the user simulator are recorded separately from those of the agent and do not count as agent steps. Appendix B.3.3 repeats the evaluation of Gemini 3.8 Flash with Claude Haiku 4.5 as the simulator.

Termination. Taken together, an episode ends in one of five ways: the model issues `COMPLETE` with no request left; the model issues `ABORT`; the episode reaches 100 steps; 15 consecutive steps visit no new page, show no new screen, and return no new information from the user, a tool, or a shell command; or five consecutive actions are refused.

Verification. After an episode ends, a private verifier decides success from the app records alone.

- *Inputs.* The verifier compares each app's backend records at the end of the episode with those before the first action. It ignores screenshots and the agent's final message, so a claimed completion counts only when the records confirm it.
- *Outcome checks.* The 145 tasks define 873 state checks, and 634 of them test a change that the requests ask for, such as a sent message or post; an order, payment, or refund; a calendar event or alarm; an app setting; or a new follow, save, or playlist entry. Most of these checks compare the change exactly, for example exactly one new order with the requested items, so both missing and extra changes fail. Values known only to the user, such as the return quantity in Appendix C, are part of the expected state.
- *Scope checks.* The remaining 239 checks, in 112 tasks, confirm that records outside the request stay unchanged, for example other conversations, unrelated orders, and other profile fields.
- *Scoring.* An episode succeeds only when every check passes, and its progress is the share of checks that pass. The verifier also runs on each task's initial state to record which checks are already satisfied before the first action.

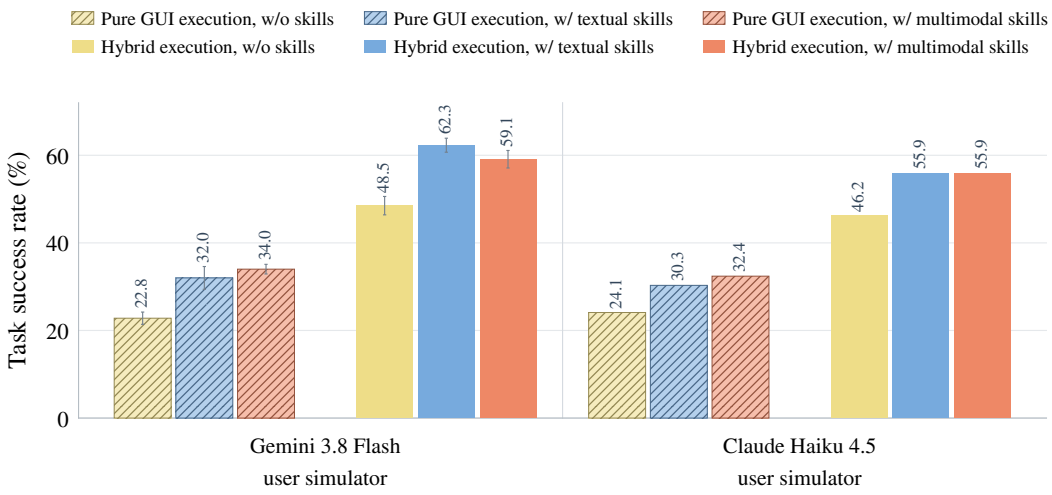


Fig. 18: Same agent, different user simulator: task success rate of Gemini 3.8 Flash with Gemini 3.8 Flash (mean of three runs, SD whiskers) and Claude Haiku 4.5 (one run) as the user simulator. Hatched bars are Pure GUI, and solid bars are Hybrid. Yellow, blue, and orange denote no skills, textual skills, and multimodal skills. Task success changes little with the simulator, and the comparisons between execution modes and skill conditions are preserved.

B.3.2 Harness Comparison

We check whether the conclusions of Sec. 4 depend on our harness. We run the same 145 tasks, protocol, user simulator, and verifier through two third-party harnesses, Claude Code for Claude Opus 5 and Codex for GPT-5.6 Sol, and compare each model with itself across harnesses (Fig. 17). Because of budget constraints, each setting under the vendor harnesses uses a single run. Codex concatenates the system prompt with the step text because its CLI has no system-prompt option.

Results. Fig. 17 shows that task success remains limited under both vendor harnesses and that the main conclusions of Sec. 4 hold under them.

- *Higher levels.* Both CLI harnesses score higher than ours, by 2–4 pp for Claude Opus 5 and, for GPT-5.6 Sol, by 6–8.5 pp without skills and 14–18 pp with textual skills.
- *Stable comparisons.* Hybrid execution outperforms Pure GUI under every harness. The skill gain for Claude Opus 5 is also similar through Claude Code (+6.2 pp under Hybrid and +0.7 pp under Pure GUI) and through our harness (+6.7 and +1.4 pp). Without skills, Claude Opus 5 outperforms GPT-5.6 Sol under both kinds of harness.
- *One harness-dependent result.* For GPT-5.6 Sol, textual skills change success by +8.2 and -0.5 pp under Hybrid and Pure GUI, respectively, through our harness, but by +13.8 and +11.0 pp through Codex. Codex retains what it has read, having opened `SKILL.md` and a procedure document in every episode and a screen manual in 27–41% of episodes. With textual skills, GPT-5.6 Sol therefore slightly exceeds Claude Opus 5 under the CLI harnesses, and its Pure GUI skill gain is the only conclusion that depends on the harness.

B.3.3 User Simulator Comparison

We check whether the conclusions of Sec. 4 depend on the user simulator. We rerun Gemini 3.8 Flash on the 145 tasks in all six settings with Claude Haiku 4.5 as the user simulator, and keep the agent, the harness, the simulator instructions, the skill library, and the budget unchanged (Fig. 18). Because of budget constraints, each setting uses a single run. We pair each task with the mean of its three runs under the Gemini 3.8 Flash simulator and report 95% bootstrap intervals over tasks.

Results. Fig. 18 shows that task success changes little with the user simulator and that the main conclusions of Sec. 4 hold with either simulator.

- *Small differences.* In five of the six settings, success differs by at most 3.2 pp between the two simulators, and the 95% interval of the paired difference includes zero.
- *Stable comparisons.* Hybrid execution outperforms Pure GUI with both simulators. With the Claude Haiku 4.5 simulator, textual skills add 9.7 pp under Hybrid and 6.2 pp under Pure GUI (13.8 and 9.2 pp with Gemini 3.8 Flash), and reference images again add no consistent gain (0.0 pp under Hybrid and +2.1 pp under Pure GUI).

B.4 Execution Interfaces: MCP and CLI Surfaces

This subsection lists the tools and commands available under Hybrid execution, namely the MCP-style tools by service (Tabs. 7 and 9) and the `adb`-style shell commands (Tab. 8), and it then reports how often the models use them (Fig. 19). Both surfaces read and write the same simulated app state as the GUI, which the verifier also inspects.

MCP surface. The MCP surface is organized as follows.

- *Inventory.* The registry contains 107 tools across 12 service groups. Each tool has a name, a description, JSON-schema parameters, and provenance metadata, and it returns a structured envelope (`ok`, `result`, `error`, `code`). Consequential writes additionally require the two-step confirmation described in Appendix B.3.1.
- *Provenance.* Of these tools, 84 are analogues of official REST APIs, 12 of published MCP servers, 7 are local or on-device operations, and 2 are weather tools adapted from an MCP tutorial. The remaining 2, `list_apps` and `get_state`, are benchmark inspection tools with no public counterpart. These categories describe design references, not interoperability with external services.
- *Access control.* Registration in the inventory does not by itself grant access, because two mocked controls apply. First, 63 tools carry the entitlements of their real interfaces, such as a Spotify Premium account, an X API tier, a Reddit authorization scope, Tencent Meeting enterprise credentials, eBay partner approval, and Alipay agent-payment authorization. These entitlements are granted at the most permissive tier by default, and lowering one produces a structured refusal. Second, 70 tools return `AUTH_REQUIRED` while their app is signed out, so register-and-recover tasks require a GUI sign-in before these tools become available.
- *State inspection.* Unlike the other tools, `get_state` returns the business records of one signed-in app through a per-app field allowlist, after removing credentials, identity numbers, undelivered messages, and fields used only by the harness. It can therefore reveal internal record identifiers and rows that the GUI shows only on secondary screens. It cannot, however, access the verifier's checks or expected state, the simulator's user-held facts, or future requests, which reside only in the host-side harness. Hybrid results should thus be read under this access model rather than as performance with public APIs alone, and we do not isolate the contribution of individual tools.

Tab. 7: MCP-style tool inventory by service group. Reference interfaces are design analogues, and local and benchmark-specific tools are identified separately.

Service group	Reference interface	Tools	Read	Write
Spotify	Spotify Web API	24	8	16
X	X API v2; XMCP analogue	20	8	12
Reddit	Reddit API (OAuth app)	17	4	13
eBay	eBay Sell / Buy / Post-Order APIs	13	5	8
Tencent Meeting	Tencent Meeting REST API	12	5	7
Map	Places/Routes MCP; Amap MCP; GPS	5	2	3
Calendar	Android CalendarContract; Workspace MCP	5	2	3
Alipay	Alipay MCP analogue	4	2	2
Weather	NWS (MCP tutorial sample)	2	2	0
Mail	Local mock inbox	2	1	1
Contacts	People API / ContactsContract	1	1	0
Generic	Benchmark harness	2	2	0
Total		107	42	65

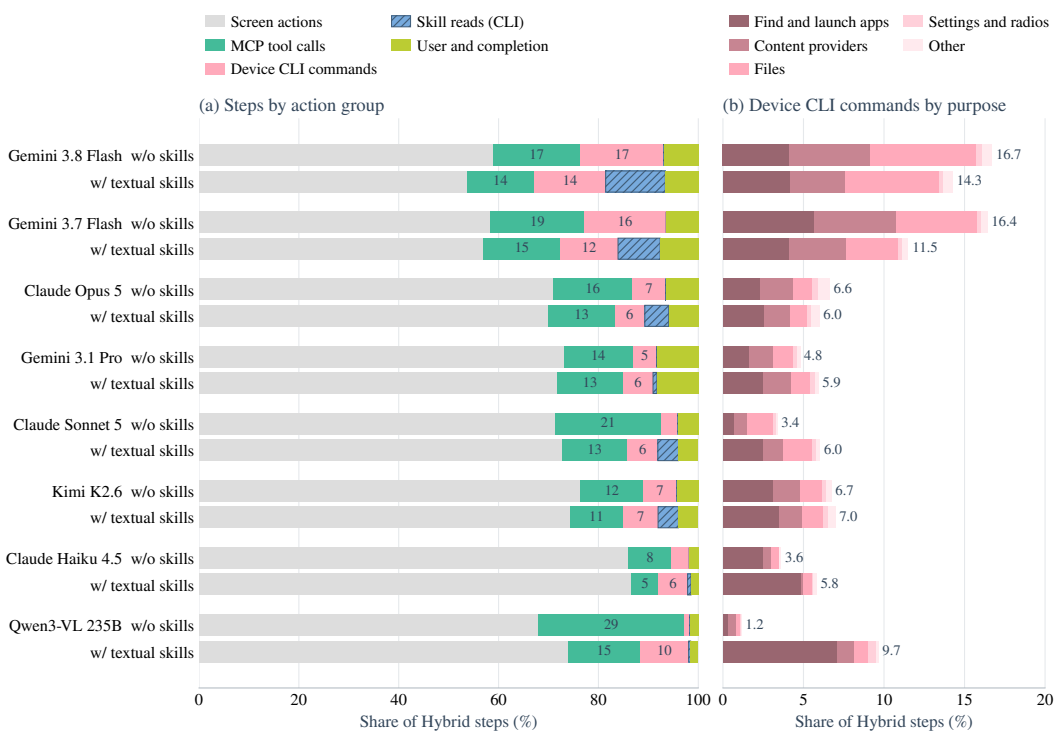


Fig. 19: Interface usage under Hybrid execution with the Function-grouped library. (a) Share of steps by action group without skills and with textual skills, where the numbers in the bars give the shares of MCP calls and device CLI commands in percent, and skill reads are CLI commands that open skill documents. (b) Share of steps spent on device CLI commands by purpose, with the total at the end of each bar. Content providers store contacts, messages, media, calendars, and notes.

CLI surface. The CLI surface complements these tools with device-level access.

- *Scope.* The shell exposes 30 commands in nine functional groups (Tab. 8). It mocks `adb shell`: the commands keep familiar names and selected flags but implement only the documented subset. Moreover, it runs as an in-browser facade over the simulated device and cannot read host files.
- *Device operations.* The `content` commands (`query`, `insert`, `update`, and `delete`) read and write contacts, messages, media, calendar, and notes through the same providers that the apps use. Likewise, `settings put` routes through the owning manager to preserve device constraints, `svc` and `cmd connectivity` toggle radios with Android's airplane-mode cascade, and `am start` resolves launcher, `SET_ALARM`, `VIEW`, and `DIAL` intents against the app manifests. A toybox-style file system also implements `ls`, `cat`, `find`, `mv`, and similar commands.
- *Declared simplifications.* The command metadata declares where the mock departs from Android: `stat` prints JSON rather than POSIX text, `find` matches substrings, `settings put` rejects unknown keys, `dumpsys alarm` renders the Clock alarm list rather than AlarmManager internals, and the battery model collapses charging status. The `input` command injects the same gestures as the GUI actions and is listed only for completeness.
- *Realism.* A production on-device agent would not hold `adb` access, and an ordinary phone does not expose a shell. This surface therefore models programmatic device access under the assumption that it has been granted.

Interface usage. We next examine how much each model uses the two surfaces (Fig. 19).

- *Screen actions dominate.* Under Hybrid execution, screen actions account for 54–87% of the steps of every model, whereas MCP tool calls account for 5–29% and device CLI commands for 1–17%.

- *CLI use varies by model.* The two Gemini Flash models spend 12–17% of their steps on device commands, while the other models spend 1–10%. Nevertheless, Claude Opus 5 and the Gemini Flash models issue at least one device command in 79–90% of their Hybrid episodes.
- *Few CLI purposes.* Most device commands find and launch apps, read or write records through content providers, or manage files, while settings and radio commands stay below 0.5% of steps.
- *Effect of skills.* With textual skills, every model's MCP share drops, and skill reads take up to 12% of steps. Multimodal skills change each share by at most 2.3 pp relative to textual skills.

Tab. 8: CLI command inventory. The 30 simulated `adb shell`-style commands are grouped by function; * marks required parameters, and R/W denotes read/write access.

Command	Parameters	Description	R/W
File system			
`ls`	`path`	List a directory; a file path prints itself	R
`cat`	`path*`	Print the contents of one or more files	R
`stat`	`path*`	Print a node's metadata (name, type, size, path) as JSON	R
`find`	`path*`, `-type`, `-name`	Search a subtree, filtered by node type and filename glob	R
`mkdir`	`path*`	Create a directory, creating parents as needed	W
`rm`	`-r`, `-f`, `path*`	Delete files, or a directory tree with -r	W
`cp`	`src*`, `dst*`	Copy file(s) to a path or into an existing directory	W
`mv`	`src*`, `dst*`	Move or rename file(s); overwrites an existing destination file	W
`echo`	`-e`, `text*`, `path`	Print text, or write (>) / append (>>) it to a file	W
Content providers			
`content query`	`--uri*`, `--projection`, `--where`	Read rows from a content provider, with optional projection and equality filter	R
`content insert`	`--uri*`, `--bind*`	Insert a row into a provider from typed --bind column values	W
`content update`	`--uri*`, `--where`, `--bind*`	Update rows addressed by row uniform resource identifier (URI) or --where selector	W
`content delete`	`--uri*`, `--where`	Delete rows addressed by row URI or --where selector	W
System settings			
`settings get`	`namespace*`, `key*`	Read one setting key from a namespace (unknown key prints null)	R
`settings put`	`namespace*`, `key*`, `value*`	Write one setting key; the write is routed through the owning manager	W
`settings list`	`namespace*`	Dump every supported key in a namespace as key=value lines	R
Connectivity			
`svc wifi`	`op*`	Turn Wi-Fi on or off	W
`svc data`	`op*`	Turn mobile data on or off	W
`svc bluetooth`	`op*`	Turn Bluetooth on or off	W
`cmd connectivity airplane-mode`	`op`	Read or set airplane mode; enabling it also turns off the radios	W
App launch / intents			
`am start`	`-a`, `-n`, `-p`, `-c`, `-d`, `-t`, `-e|--es`, `--ei`, `--ez`, `--eia`	Start an activity by explicit component, launcher intent, or data URI (SET_ALARM / VIEW / DIAL)	W
`monkey`	`-p*`, `-c`, `event-count*`	Launch a package through the launcher-intent smoke path	W
Package manager			
`pm list packages`	`FILTER`	List installed package names, optionally substring-filtered	R
GUI input injection			
`input`	`source`, `tap`, `swipe`, `draganddrop`, `text`, `keyevent`	Inject a GUI event: tap, swipe, draganddrop, text, or keyevent	W
Device / hardware			
`wm size`	—	Print the physical screen size in pixels	R
`wm density`	—	Print the screen density in dpi	R
`dumpsys battery`	`set level`, `set ac`, `set usb`, `set status`, `-f`, `unplug`, `reset`	Dump battery state; set level / charger / status, unplug, or reset	W
`dumpsys alarm`	—	Dump the alarm list under the AlarmManager header	R
`date`	`+FORMAT`	Print the device clock, optionally strftime-formatted	R
Meta			
`help`	—	Print the command overview	R

Tab. 9: Complete MCP-style tool inventory by service; * marks a required parameter.

Tool	Parameters	Description
Spotify (24)		

continued on next page

Tab. 9 continued

Tool	Parameters	Description
spotify_create_playlist	name*	Create a user-owned playlist
spotify_follow_artist	artist_name*, state	Follow or unfollow an artist (explicit state)
spotify_search_tracks	q*	Search tracks by title or artist and record the search in history
spotify_like_song	track_id*, state	Like or unlike a track by track identifier (ID) (explicit state)
spotify_add_to_playlist	playlist, playlist_id, track_id*, create_if_missing	Ensure a track is present in an owned playlist addressed by playlist_id
spotify_play	track_id*	Play a track by ID
spotify_add_to_queue	track_id*	Add a track by ID to the playback queue
spotify_remove_from_playlist	playlist, playlist_id, track_id*	Remove a track from a specified playlist
spotify_skip_next	—	Skip to the next track
spotify_skip_prev	—	Skip to the previous track
spotify_toggle_play	state	Set play or pause
spotify_toggle_shuffle	state	Set shuffle
spotify_toggle_repeat	mode*	Set repeat mode explicitly
spotify_search_albums	q*	Search albums by title or artist in the catalog the Album pages use
spotify_album_tracks	album_id*	Read the tracks of one album in album order (the same list the Album page shows)
spotify_artist_top_tracks	artist_name*, limit	Read an artist's popular tracks in the order the Artist page lists them (the same offline catalog the GUI Artist page renders)
spotify_query_playlists	—	Query playlists, including user-created playlists and their tracks, plus liked tracks
spotify_now_playing	—	Query current playback state, including track, position, playing status, shuffle, repeat, and queue
spotify_seek	position_ms*	Seek the current track to position_ms in milliseconds
spotify_set_volume	volume_percent*	Set playback-device media volume from 0 to 100
spotify_recently_played	limit	Query recently played history in newest-first order
spotify_query_liked_songs	—	Query liked tracks
spotify_query_followed_artists	—	Query followed artists
spotify_update_playlist_details	playlist_id*, name, description	Update the name or description of a user-created playlist; provide at least one field
X (20)		
x_post	text*, image	Create a post with text of at most 280 characters
x_reply	post_id*, text*	Reply to a post
x_like	post_id*, state	Like or unlike a post (explicit state)
x_retweet	post_id*, state	Repost or undo a repost (explicit state)
x_bookmark	post_id*, state	Bookmark or remove a bookmark from a post (explicit state)
x_follow	user_id*, state	Follow or unfollow a user (explicit state)
x_send_dm	conversation_id*, text*	Send a message in an existing direct-message conversation
x_query_conversations	—	Query direct-message conversations
x_update_profile	field*, value*	Update one profile field
x_quote	post_id*, text*	Quote a post and add commentary in a new post
x_query_bookmarks	—	Query bookmarked post IDs
x_query_follows	—	Query followed user IDs
x_delete_post	post_id*	Delete a post or reply created by the current user
x_search	query*	Search recent posts by keyword against text and author ID, case-insensitively
x_query_timeline	user_id	Query a user's post timeline in newest-first order
x_mute	user_id*, state	Mute or unmute a user (explicit state)
x_block	user_id*, state	Block or unblock a user (explicit state)
x_query_mentions	limit	List posts that mention the signed-in account (@handle), newest first, up to 'limit' (default 20)
x_query_trends	—	List current trending topics as shown on the Explore tab
x_query_user	handle*	Look up a public profile by handle (with or without @)
Reddit (17)		
reddit_vote_post	post_id*, dir*	Vote up or down on a post
reddit_comment	post_id*, body*	Comment on a post
reddit_save	post_id*, state	Save or unsave a post (explicit state)
reddit_join	community_id*, state	Join or leave a community (explicit state)
reddit_post	community*, title*, body, flair	Create a post in a specified community with a title, body and optional community flair
reddit_reply	post_id*, parent_comment_id*, body*	Reply to a comment in a nested thread
reddit_query_comments	post_id*	List the comments of one post as the post page shows them: seeded comments plus your own, each with {comment_id, parent_comment_id, author, body, score, created_utc}
reddit_vote_comment	comment_key*, dir*	Vote up or down on a comment (toggle)
reddit_edit_post	post_id*, title, content	Edit the title or body of the current user's post

continued on next page

Tab. 9 continued

Tool	Parameters	Description
reddit_delete_post	post_id*	Delete one of the current user's posts
reddit_edit_comment	comment_id*, body*	Edit one of the current user's comments
reddit_delete_comment	comment_id*	Delete one of the current user's comments
reddit_add_flair	post_id*, flair*	Set the flair on one of your own posts (replaces any existing flair)
reddit_update_profile	bio*	Update the profile bio
reddit_search	query*, subreddit	Search posts by matching a keyword against titles and bodies without changing state
reddit_query_saved	—	Query saved post IDs
reddit_query_joined	—	Query joined community IDs
eBay (13)		
ebay_list_item	title*, price*, condition, category_id	Create a listing for an item
ebay_query_listings	—	Query listings created by the current user with ebay_list_item
ebay_update_offer	listing_id*, price*	Update the price of one of the current user's active listings
ebay_end_item	listing_id*	End one of the current user's active listings (status becomes ended)
ebay_search	q*, sort, condition, location, free_shipping, max_total	Search items by keyword and record a search snapshot
ebay_place_order	item_ids, quantities, recipient_name, delivery_address	Start a checkout session for the signed-in buyer
ebay_query_inbox	—	Query the marketplace message inbox
ebay_query_orders	—	Query orders created by ebay_place_order
ebay_get_order	order_id*	Read one owned order, including exact line quantities, total cents and recipient
ebay_cancel_order	order_id*	Cancel an owned unpaid reservation and release inventory/holds through the existing backend
ebay_request_return	order_id*, request_id*, reason*, item_ids*, quantities*	Request return of specified owned order items
ebay_schedule_return	return_id*	Schedule pickup for an owned return using the existing GUI business action
ebay_query_returns	order_id*	Read returns and refunds for an owned order, including cents and status
Tencent Meeting (12)		
meeting_schedule	title*, start, start_time, duration*, note, password, invitees, confirm_token	Schedule a meeting
meeting_quick_start	—	Start an instant meeting as host
meeting_cancel_meeting	id*, confirm_token	Cancel a scheduled meeting
meeting_query_history	—	Query meeting history
meeting_add_contact	id*, name*, avatar	Add a meeting contact
meeting_query_contacts	—	Query the meeting contact list
meeting_update_meeting	id*, title, start, start_time, duration, note, password, invitees, confirm_token	Update a scheduled meeting; omitted fields retain their current values
meeting_end_meeting	confirm_token	End or leave the current meeting
meeting_mute_all	muted	Mute or unmute all other participants without affecting the current user
meeting_query_scheduled	—	Query scheduled meetings and their status
meeting_query_participants	meeting_id*	Query meeting participants
meeting_query_recordings	start, end	Query cloud recordings
Map (5)		
map_query_location	—	Query the map app's current location coordinates
map_search_poi	q*	Search nearby places of interest from an offline snapshot
map_place_details	place_id*	Query place details by place_id from the read-only offline snapshot
map_plan_route	to*, mode, from	Plan a route from the current location, or a location specified by from, to a destination using the offline snapshot
map_launch_navigation	to*, mode, from	Plan a route and hand it off to the Map app: the route becomes the active route with the route sheet open on screen, exactly as if the user had tapped Directions
Calendar (5)		

continued on next page

Tab. 9 continued

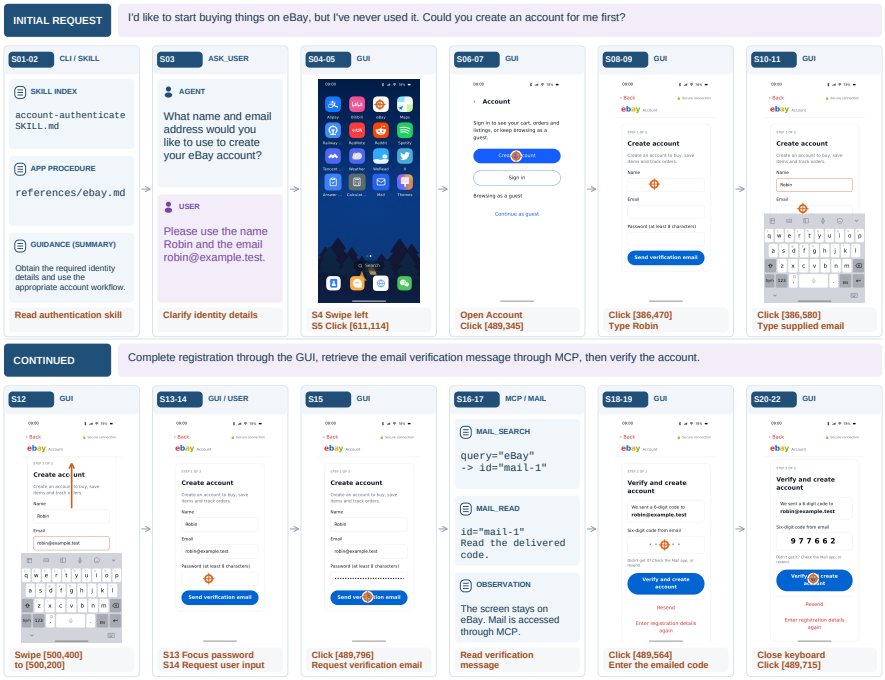
Tool	Parameters	Description
calendar_update_event	event_id*, title, start, end, duration_minutes, description, location, reminder_minutes, alarm_enabled, all_day	Update an existing event by event_id
calendar_delete_event	event_id*	Delete only the event identified by event_id
calendar_create_event	title*, start, start_ts, end, end_ts, duration_minutes, description, location, reminder_minutes, alarm_enabled, all_day	Create a Calendar event
calendar_query_events	from*, to*, query	List calendar events overlapping a time window, optionally filtered by a case-insensitive title/location/description substring
calendar_find_free_time	date*, duration_minutes*, window_start, window_end	Find free slots of at least duration_minutes on one day, between window_start and window_end (default 09:00-18:00), excluding existing non-all-day events
Alipay (4)		
alipay_create_payment	out_trade_no*, subject*, total_amount*, merchant_name*	Create a merchant payment order and open the Alipay cashier for it (official Alipay MCP mobilePay)
alipay_query_payment	out_trade_no*	Query a merchant payment by out_trade_no (official Alipay MCP queryPay)
alipay_refund_payment	out_trade_no*, refund_amount*, out_request_no, refund_reason, confirm_token	Refund all or part of a successful merchant payment (official Alipay MCP refundPay)
alipay_query_refund	out_trade_no*, out_request_no	Query one refund of a merchant payment by out_trade_no and out_request_no (official Alipay MCP refundQuery)
Weather (2)		
weather_get_forecast	city, day	Query a city weather forecast
weather_get_alerts	city	Query active severe-weather alerts for a city
Mail (2)		
mail_search	query	Search messages delivered to the current local Mock Mail inbox
mail_read	id*	Read a delivered message in the current local inbox and mark it read
Contacts (1)		
contacts_search	query*	Search the device contacts by name, phone digits or email (case-insensitive substring)
Generic (2)		
list_apps	—	List the IDs of all currently registered and operable apps
get_state	app_id	Read an app business-state snapshot plus deviceTime (the simulated device clock)

C A Skill-Guided Hybrid Execution Case Study

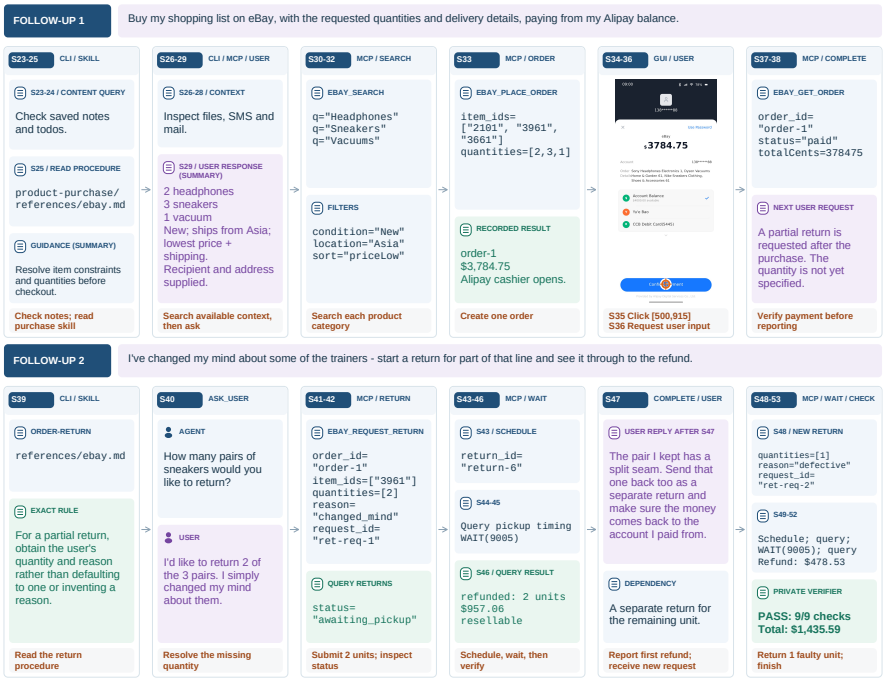
This appendix follows one eBay task through a skill-guided Hybrid episode and contrasts it with the no-skill episode on the same task, to show how skills guide single decisions within a long task.

Task and trajectory. Fig. 20 follows an eBay registration, purchase, and staged-return task executed by Gemini 3.7 Flash under Hybrid execution with Function-grouped multimodal skills. The episode contains **53 logged actions** across eBay, Mail, and Alipay. Only registration is requested initially, and the purchase and two returns arrive as staged follow-ups. The case thus combines long-horizon execution, cross-app coordination, user interaction, and GUI–MCP–CLI execution.

Registration and purchase. In Fig. 20(a), the agent reads `account-authenticate`, asks for the required identity details, completes the registration form through the GUI, and requests user intervention for password entry. It retrieves the verification message through `mail_search` and `mail_read` while staying on eBay. After the purchase follow-up arrives (Fig. 20(b)), it reads

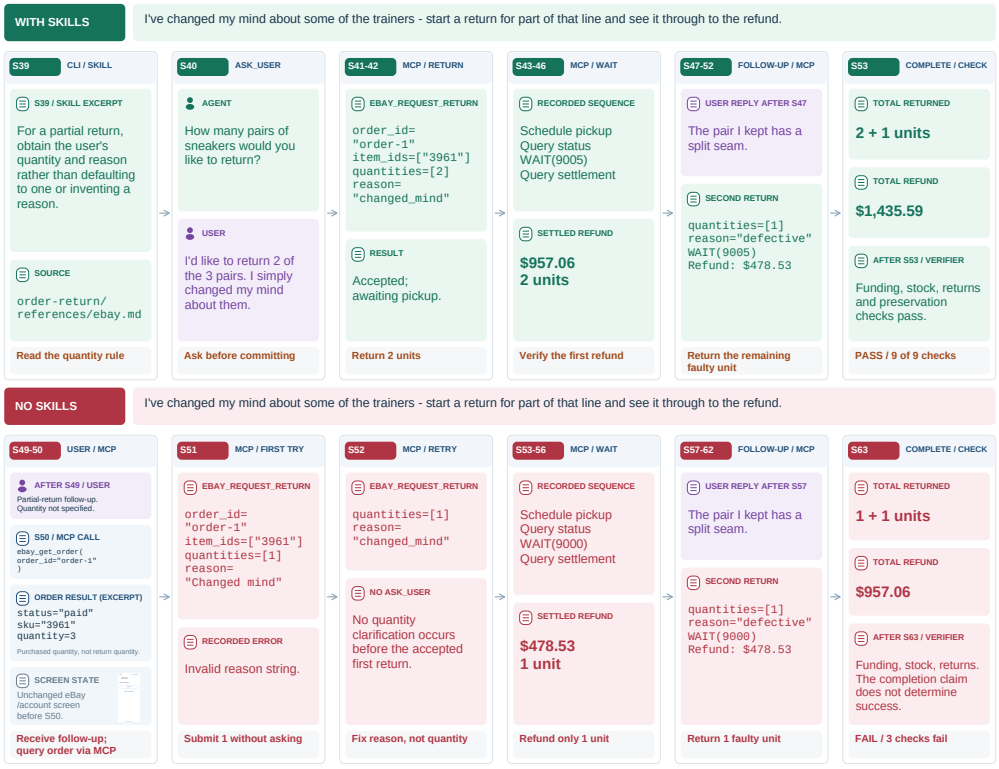


(a) Registration and verification



(b) Purchase and staged returns

Fig. 20: A skill-guided Hybrid workflow with staged user requests. (a) Registration combines skill reads, GUI interaction, user intervention, and Mail access through MCP. (b) Purchase and staged returns combine MCP, GUI payment, clarification, and simulated waiting. Together, the panels summarize all 53 actions. Phone images are pre-action screenshots, and cards show calls, replies, or labeled summaries. Orange marks denote GUI gestures. Panel (c) continues on the next page.



(c) Skill-guided and no-skill returns

Fig. 20: A skill-guided Hybrid workflow (continued). (c) With skills, the agent clarifies the return quantity and returns $2+1$ units. Without skills, it returns $1+1$ units and fails the funding, stock, and return checks. The eBay thumbnail shows the unchanged `/account` screen before step 50, and MCP provides the order details. The verifier checks the outcome after the agent reports completion.

`product-purchase`, obtains the missing shopping and delivery requirements, creates the order through MCP, pays through the Alipay GUI, and verifies the paid order before reporting completion.

Skill-guided return. The key decision occurs when the user requests a partial return but omits the quantity. At step 39, the agent reads `order-return`, which instructs it to obtain the user's quantity and reason rather than assume defaults. At step 40, it asks the user and learns that **two of the three pairs** should be returned for `changed_mind`. It then submits `quantities=[2]`, schedules pickup, waits for the simulated return process, and verifies a settled refund of **$957.06**. A later follow-up identifies the remaining pair as defective, and the agent creates a separate one-unit return and verifies another $478.53 refund. The two returns total **$1,435.59**, and all **9/9** backend checks of the verifier pass.

No-skill contrast and interpretation. Fig. 20(c) contrasts the skill-guided return with the corresponding no-skill episode. After the partial-return follow-up, step 50 queries the paid order through `ebay_get_order`, while the GUI remains on the unchanged eBay `/account` screen. The returned `quantity=3` is the number originally purchased, not the quantity the user wants to return. The agent never asks for the missing return quantity. Its first one-unit request is rejected because the reason string is invalid. The retry fixes the reason but still submits `quantities=[1]`. After a second one-unit defective return, only **1+1 units** are returned and the total refund is **$957.06**. The agent reports completion, but the private verifier rejects the outcome because the funding, stock, and return checks all fail. The two episodes illustrate skills as reusable **decision guidance**: they specify what to clarify, how to instantiate task-dependent arguments, and what outcomes to verify. The no-skill episode has access to the same Hybrid interfaces, so the decisive difference is not tool availability but whether the agent resolves the user's intent before acting. For this task and model, all three textual-skill runs and all three multimodal-skill runs succeed, whereas all three no-skill runs fail.