iOSWorld: A Benchmark for Personally Intelligent Phone Agents

Lawrence Keunho Jang, Mareks Woodside[*], Geronimo Carom[*], Andrew Jang[*]
Jing Yu Koh, Ruslan Salakhutdinov
Carnegie Mellon University

Abstract

A useful phone agent will have to be personally intelligent. It must reason over the user's identity, history, and preferences as they exist on their device, not just instructions in an impersonal sandbox. Existing phone-agent benchmarks evaluate the latter and largely ignore the former. We introduce iOSWorld, the first interactive native iOS simulator benchmark built around a persistent user identity that spans 26 newly-built iOS apps. These apps contain interconnected data including transaction histories, messaging threads, travel records, social relationships, and financial activity. iOSWorld includes 133 tasks across three categories of increasing difficulty: single-app tasks (27), multi-app tasks (60), and memory and personalization tasks (46). We evaluate leading frontier and open-source computer-use models under both vision-only and privileged vision+XML settings. The best configuration reaches 51% overall but only 36% on multi-app tasks. Privileged vision+XML access improves the stronger frontier models by up to 26%, while we see that smaller models do not benefit from the added accessibility-tree input. We release iOSWorld as an open-source benchmark, including all apps, seed data, tasks, rubrics, and evaluation code, to support reproducible research on personally intelligent phone agents.

1 Introduction

A person's phone is not a blank slate. Transactions, messages, social connections, and financial records accumulate across many applications, forming a record that any useful assistant would need to understand and navigate. We use the term *personally intelligent* to describe the corresponding agent capability: reasoning over a user's identity, history, and preferences as they actually exist on the device, rather than executing turn-by-turn instructions in isolation. Current benchmarks that we use to evaluate phone agents ignore this dimension. Tasks are issued against stock app states without persistent user data, no cross-app continuity, and no real notion of a true user. An agent that can tap the right button on a settings screen but cannot figure out how to use its owner's most common commute route has not demonstrated useful capability.

Existing benchmarks evaluate digital agents on Android (Rawles et al., 2025), web (Zhou et al., 2024; Koh et al., 2024), and desktop (Xie et al., 2024; Yang et al., 2025; Bonatti et al., 2025) environments. iOS serves over 2.5 billion active devices[1] and accounts for roughly 58–60% of U.S. mobile OS usage[2], but existing interactive phone-agent benchmarks target Android rather than native iOS simulator environments. Nor does any existing benchmark populate apps with a persistent user identity. We deliberately exclude tasks that primarily involve web browsing, as web-based workflows are already well-served by existing benchmarks (Zhou et al., 2024; Koh et al., 2024; He et al., 2024), and a phone agent with access to a web browser

[*]Equal contribution.

[1]Apple reported an installed base of over 2.5 billion active devices in 2026: https://finance.yahoo.com/news/apple-installed-tops-2-5-170414353.html

[2]StatCounter Global Stats (accessed March 2026): https://gs.statcounter.com/os-market-share/mobile/united-states

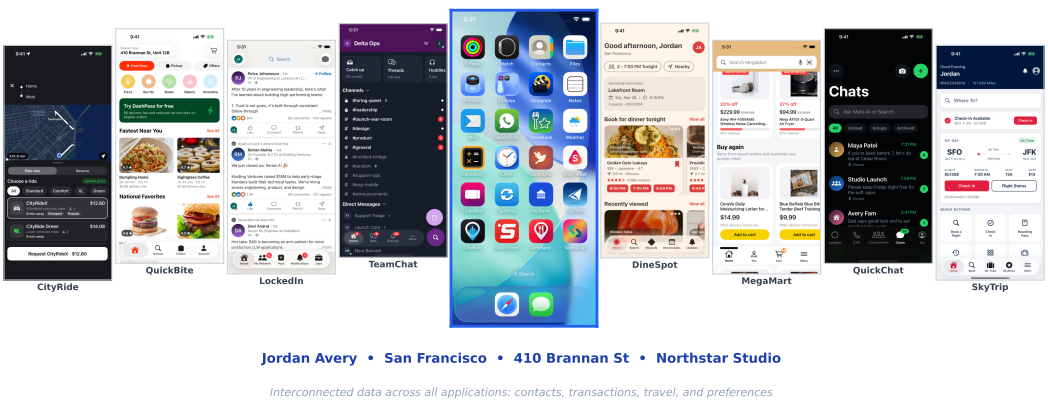

Figure 1: Overview of iOSWorld. 26 purpose-built iOS applications sharing a single user identity (Jordan Avery) with interconnected data across apps. The benchmark includes 133 tasks across three categories: (1) single-app, (2) multi-app, and (3) memory/personalization.

can be evaluated with those tools without the need to use the phone's native browser. Our focus is on native iOS apps and the personal data they contain. This is the primary domain where phone-specific agents are necessary, and no relevant interactive benchmark currently exists.

iOSWorld is the first interactive native iOS simulator benchmark centered on a user's personal identity, designed to measure how personally intelligent today's phone agents are. We built 26 purpose-built native iOS applications and populated them with interconnected data for a single persona, Jordan Avery. The same contacts appear across messaging, payment, and email apps. A food order on one app produces a bank charge and a receipt email in others; an upcoming flight aligns with a hotel reservation and a calendar reminder. We release **133 tasks** in three categories: **single-app** (27) tasks test basic interaction within one app; **multi-app** (60) tasks require carrying information across two to eight applications; and **memory and personalization** (46) tasks ask agents to discover implicit patterns from data living inside apps without being told where to look. The release also includes a schema for adding future tasks as agents improve. Our contributions are summarized as follows:

- The first interactive native iOS simulator benchmark with a coherent user identity spanning 26 purpose-built applications containing interconnected personal data.
- 133 tasks across three categories, evaluated with an LLM-as-a-judge pipeline validated against human annotators (κ=0.77).
- A systematic comparison of five frontier models and one open-source baseline (Qwen3.5 35B-A3B) under vision-only and privileged vision+XML settings. The best overall configuration achieves 81% overall ($\sim$82% single-app, 54% memory, 36% multi-app), while the strongest single-app configuration reaches $\sim$93%. Privileged vision+XML access improves the stronger frontier models by up to 26%, and smaller models do not show the same gain.
- We release all apps, seed data, tasks, rubrics, and evaluation code as open source to support reproducible research.

2 Related Work

2.1 GUI Agent Benchmarks

GUI Agents have largely been centered on the web and desktop. Foundational benchmarks such as MiniWoB (Shi et al., 2017), MiniWoB++ (Liu et al., 2018), WebShop, WebArena, and VisualWebArena (Yao et al., 2022; Zhou et al., 2024; Koh et al., 2024) emulated tasks on the web. Mind2Web (Deng et al., 2023) and WebVoyager (He et al., 2024) extended to real websites, and Xue et al. (2025) studied how evaluations transfer to live conditions.

At the OS and Desktop level, OSWorld (Xie et al., 2024) covers Linux, Windows Agent Arena (Bonatti et al., 2025) targets Windows, MacOSWorld (Yang et al., 2025) covers macOS, and WorkArena (Drouin et al., 2024) benchmarks enterprise knowledge work. GAIA (Mialon et al., 2024), TheAgentCompany (Xu et al., 2025a), and τ-bench (Yao et al., 2025) probe multi-step and multi-tool reasoning. All of these benchmarks present agents with predominantly impersonal or single user environments and explicit instructions.

2.2 Mobile Device Agents

Existing interactive mobile-agent benchmarks target Android. AndroidEnv (Toyama et al., 2021) provides an RL interface to the emulator, Android-in-the-Wild (Rawles et al., 2023) provides human demonstrations on Android, and AndroidWorld (Rawles et al., 2025) offers dynamic tasks with programmatic verification on a live Android simulator. Additional benchmarks have expanded coverage across different digital domains, such as Android-Lab (Xu et al., 2025b), SPA-Bench (Chen et al., 2025), B-MoCA (Lee et al., 2025), and GUI Odyssey (Lu et al., 2025). On the modeling side, CogAgent (Hong et al., 2024), AppAgent (Zhang et al., 2025), Mobile-Agent (Wang et al., 2024), UI-TARS (Qin et al., 2025), and AutoDroid (Wen et al., 2024) explore architectures ranging from fine-tuned VLMs to RL-trained agents (Bai et al., 2024; 2025) for mobile agents. Ferret-UI (You et al., 2024) develops a multimodal model for understanding mobile UI on both Android and iOS.

Dynamic iOS evaluations remain absent. It differs from Android in its UI framework (SwiftUI versus XML layouts), navigation patterns (edge-swipe instead of a hardware back button), and accessibility infrastructure (XCUITest versus UIAutomator). No mobile benchmark on any platform seeds applications with a user identity or evaluates reasoning over extensive personal data distributed across apps.

3 iOSWorld

Figure 2: Jordan Avery's digital life: 26 iOS apps across 10 domains sharing one identity. App names (**bold**) and analogues (*italics*) in callout columns; colored dots denote domains. Edge thickness $\propto$ shared data points; Mail is the primary hub. See Table 4.

3.1 Environment

We model the iOSWorld environment as a partially observable Markov decision process (POMDP) (Kaelbling et al., 1998): $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \Omega, T)$, where $\mathcal{S}$ is the set of simulator states, $\mathcal{A}$ is the action space, Ω is the observation space, and $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ defines deterministic

Action	Parameters	Mode
`tap_xy`	$x, y \in [0, 1000]$	Both
`type`	text string	Both
`swipe`	direction, optional origin	Both
`home`	—	Both
`wait`	duration (seconds)	Both
`stop`	answer string	Both
`tap`	accessibility identifier	XML only, adapter-dependent
`launch_app`	bundle identifier	XML only, adapter-dependent
`terminate_app`	bundle identifier	XML only, adapter-dependent
`open_url`	URL string	XML only, adapter-dependent

Table 1: Action space. The top block (6 actions) is available in both modalities. The bottom block requires the accessibility tree and is exposed when supported by the provider adapter (Table 8). The `tap` action targets elements by identifier, enabling pixel-perfect interaction without coordinate estimation.

transitions. At each step t, the agent receives a partial observation $o_t \in \Omega$ of state s_t and produces an action $a_t \in \mathcal{A}$, which transitions the simulator to s_{t+1}.

Observation space. We evaluate agents under two observation modalities. Unlike Android, where tools like UIAutomator expose accessibility data through open APIs, iOS is closed-source. The richest structured UI data available to third-party tools comes through Apple's XCUITest framework, which requires a Mac running Xcode. Additionally, a deployed agent without privileged information would have access only to what is visible on the screen. We evaluate under both settings to understand how much of the performance gap is due to visual grounding, reasoning, and privileged settings.

In the **vision-only** setting, the agent receives a screenshot at each step. Raw simulator captures are 1206×2622; we resize to 706×1536. This 1536-pixel cap on the longest edge stays within Anthropic's 1568-pixel API limit for Claude Computer Use, and we apply it uniformly across all providers for a fair comparison. The agent must visually identify UI elements, estimate their coordinates, and infer the application state from pixels alone. We do not evaluate in text-only mode since all frontier computer-use models require image input.

In the **vision + XML** setting, the agent additionally receives a cleaned accessibility tree in XML extracted via XCUITest. For each interactive element, the tree reports the element type (e.g., `Button`, `TextField`, `Cell`), display name, label, current value, center coordinates in a normalized 0–1000 space, and an accessibility identifier when available. The tree is filtered to interactive and visible elements, capped at 200 elements and 15 levels of depth.

Action space. The available actions differ between modalities, reflecting the information each setting provides and the provider adapter used for execution (Table 1).

In vision-only mode, agents are limited to six actions and must estimate all tap coordinates from screenshots. In vision+XML mode, adapters may expose additional actions. The most consequential are `tap`, which targets elements by their accessibility identifier and eliminates coordinate estimation error entirely, and `launch_app`, which opens applications through their bundle identifier rather than requiring the agent to navigate the home screen visually. Provider-native support varies; Table 8 gives the exact mapping.

Translating computer use agents to iOS. Frontier computer-use models each define their own action space for desktop environments. We adapt them to iOS through a unified translation layer: click becomes `tap_xy`, scroll becomes `swipe` with inverted direction, and coordinate systems are normalized to 0–1000 space. All models also receive iOS-specific system prompts emphasizing touchscreen-only interaction. For the open-source Qwen3.5 baseline we follow the official Qwen3-VL mobile-agent cookbook, exposing the cookbook's `mobile_use` tool (`click`, `long_press`, `swipe`, `type`, `system_button`, `wait`, `terminate`) on a 999×999 grid that we rescale to our 0–1000 schema. The full action mapping (Claude CU, OpenAI CUA, Gemini CU, Qwen mobile_use) is in Table 8.

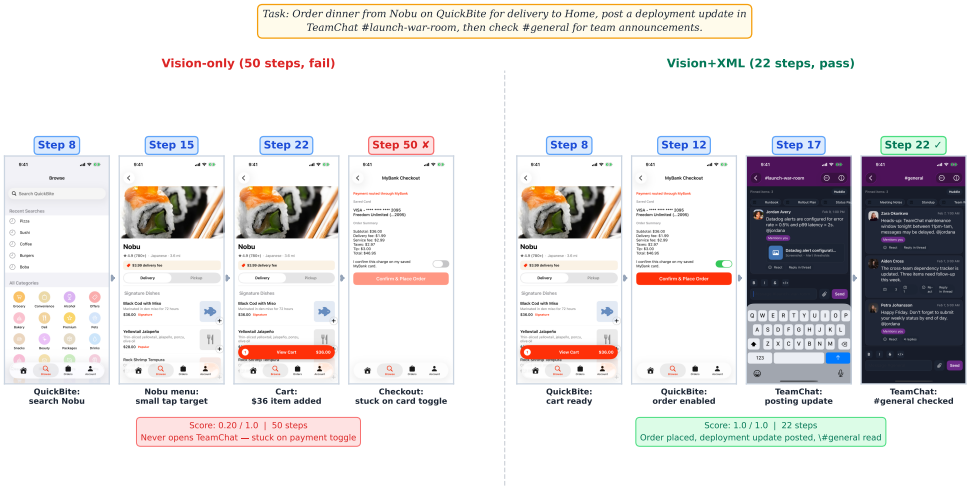

Figure 3: Same multi-app QuickBite → TeamChat task, two modalities, same model (Opus 4.6). **Vision-only** finds Nobu, adds an item, and reaches checkout, but spends the remaining budget trying to toggle payment confirmation and never opens TeamChat (50 steps, score 0.20). **Vision+XML** places the order, posts a deployment update in #launch-war-room, and checks #general announcements in 22 steps (score 1.0).

Infrastructure. The agent loop runs on macOS using Appium with the XCUITest driver, controlling an Xcode-managed iPhone simulator. Each task starts from a deterministic home state with all 26 apps pre-installed and seeded. The loop captures observations, sends them to the model, executes the returned actions, and repeats until the agent issues `stop` or reaches the step limit (Fig. 3 shows two same-task runs under both modalities). Each cloned simulator instance uses 2–4 GB of RAM. Accounting for OS overhead (∼8 GB), a 36 GB Mac Studio (M4 Max) safely supports 12 parallel workers and a 24 GB MacBook Pro (M4) supports 4. Workers are managed via `xcrun simctl clone`, with each clone receiving dedicated Appium and WebDriverAgent ports.

Evaluation. Each task is scored with an LLM-as-a-Judge (Zheng et al., 2023) framework with GPT-5.4 Mini that reviews the agent's full trajectory. Given per-step screenshots, actions, and the final answer, the judge produces a binary pass/fail judgment. Human validation on 128 Opus 4.6 trajectories confirms substantial agreement (κ=0.77 at task level, 89% accuracy; §4.3). We also explored a per-step variant that evaluates each screenshot independently, but it proved more lenient without improving discrimination; we use the trajectory-level judge throughout and report binary pass rate as our primary metric. Details on per-step evaluation and rubric-based scoring are in Appendix C.

3.2 App Ecosystem and User Identity

All 26 applications share a single user identity: **Jordan Avery**, a San Francisco-based professional living at 410 Brannan Street who works at Northstar Studio and trains for a half marathon (Fig. 2). Jordan's contacts, Maya Patel, Leo Chen, Kai Santos, appear as QuickChat correspondents, SplitPay payees, Mail senders, LockedIn connections, and TeamChat colleagues. A Chipotle order in QuickBite produces a charge in MyBank and a receipt in Mail; an upcoming SFO→JFK flight in SkyTrip aligns with a StayFinder booking and a Notes reminder. These cross-references make multi-app and memory tasks require evidence from more than one application.

Apps were developed or adapted in SwiftUI using Claude Code as a coding assistant, then manually verified by human developers for correct navigation, data rendering, and seed data consistency. The applications implement tab-based navigation, searchable lists, detail views, and editing flows that follow their real-world counterparts. Two apps build

Category	Task ID	Task Instruction	Apps	Rubrics
Single-App	ot-001	Search for restaurants in San Francisco with "Outdoor Seating" on DineSpot and make a reservation at Harborline Seafood for 2 tonight at 7 PM.	DineSpot	9
	msg-003	Search my QuickChat for "Brooklyn Half" and find which conversation mentioned it. Reply to that thread confirming I'm registered.	QuickChat	9
Multi-App	multi-009	Check my most recent Chipotle order on QuickBite. Then check my MyBank credit card for the corresponding charge. Find the receipt email in Mail and note any price differences in Notes.	4 apps	10
	multi-011	Check my StayFinder trip for Catalina Island (Apr 18–21). Look up the weather for those dates. Check my TasteRank "Want to Try" list for nearby restaurants and compile everything in Notes.	4 apps	10
Memory	mem-002	Look at my CityRide app and figure out my most common route based on my saved locations. Then request a ride along that route.	CityRide	4
	mem-005	Review my TrailBlaze activities to figure out my regular running schedule and favorite routes. Check the Weather for conditions during my typical run time and message my running group.	3 apps	6

Table 2: Example tasks from each category.

on open-source foundations: Notes is based on snowNotes[3] and Cinephile draws from MovieSwiftUI[4]. User data is encoded in Swift seed fixtures and JSON snapshots loaded at build time. The 26 apps span finance, messaging, travel, food, shopping, productivity, entertainment, fitness, utilities, and professional networking. Full details are in Table 4.

3.3 Task Design

iOSWorld includes 133 tasks in three categories of increasing difficulty (Table 2). **Single-app tasks** (27) test basic navigation and interaction within one app, such as logging a meal in CalTrack or finding an upcoming flight in SkyTrip. **Multi-app tasks** (60) span two to eight applications and require transferring information between them. For example, one task asks the agent to check a Chipotle order on QuickBite, find the corresponding charge in MyBank, locate the receipt email in Mail, and note any price differences in Notes. **Memory and personalization tasks** (46) require discovering latent patterns that are never stated. The agent is asked questions like "What is my most common commute route?" or "Find my most frequently ordered restaurant and place a reorder." Answering correctly requires exploring multiple apps, identifying patterns in the data, and synthesizing evidence into a coherent answer.

Task creation. We generated tasks using Claude Code (Anthropic, 2026a) with full access to each app's source code and seed data. The coding agent examined seeded JSON files, view controllers, and navigation flows, then produced tasks grounded in the actual app state. Each task is written in first-person voice and accompanied by rubric criteria that decompose the objective into verifiable steps (Appendix C). Human annotators reviewed and refined every task.

Quality assurance. We found that grounding tasks in real seed data required careful verification. Every task was manually executed end-to-end on the iOS simulator by human annotators and feasible for completion. Forty-four of the 175 candidate tasks required corrections: flight routes that did not exist in the seeded data, food items with mismatched names, and rubric criteria that referenced app states the agent could not reach. All 26 apps were independently tested to verify that UI elements, seed data, and navigation flows worked as intended.

The initial pool contained 175 tasks. We trimmed the single-app set to broad app coverage with minimal duplication and kept all multi-app and memory tasks, leaving a final set of 133 tasks. Memory tasks involve 4.4 apps per task on average, since answering them requires exploring several data sources. QuickChat appears in 44 of 133 tasks and Notes in 41, with CloudDocs at 35 and Mail at 29, making them the most frequently referenced apps.

[3] https://github.com/probablyhades/snowNotes
[4] https://github.com/Dimillian/MovieSwiftUI

Model	+XML	Single (27)		Multi (60)		Memory (46)		Overall (133)	
		Pass	Steps	Pass	Steps	Pass	Steps	Pass	Steps
Opus 4.6	✗	70.4%	23.1	20.0%	45.5	8.7%	49.4	26.3%	42.3
	✓	81.5%	16.4	**36.7**%	38.2	**54.3**%	39.3	**51.9**%	34.1
Sonnet 4.6	✗	77.8%	26.3	21.7%	46.9	10.9%	49.8	29.3%	43.7
	✓	**92.6**%	15.4	35.0%	42.0	34.8%	44.8	46.6%	37.5
GPT-5.4	✗	63.0%	32.5	11.7%	48.5	6.5%	48.3	20.3%	45.2
	✓	81.5%	12.1	26.7%	37.1	32.6%	37.2	39.8%	32.1
GPT-5.4 Mini	✗	70.4%	24.0	18.3%	46.3	10.9%	48.9	26.3%	42.7
	✓	66.7%	14.9	1.7%	43.8	4.3%	41.5	15.8%	37.2
Gemini 3 Flash	✗	70.4%	11.0	13.3%	23.9	21.7%	23.5	27.8%	21.2
	✓	70.4%	11.5	18.3%	38.0	17.4%	33.3	28.6%	31.0
Qwen3.5 35B-A3B	✗	40.7%	23.0	6.7%	36.2	4.3%	33.4	12.8%	32.6
	✓	48.1%	31.4	0.0%	43.9	2.2%	37.9	10.5%	39.3

Table 3: Pass rates (%) and average steps by task category. Rows with ✗ are vision-only. ✓ denotes vision+XML. Rubric-based scoring in Appendix C.

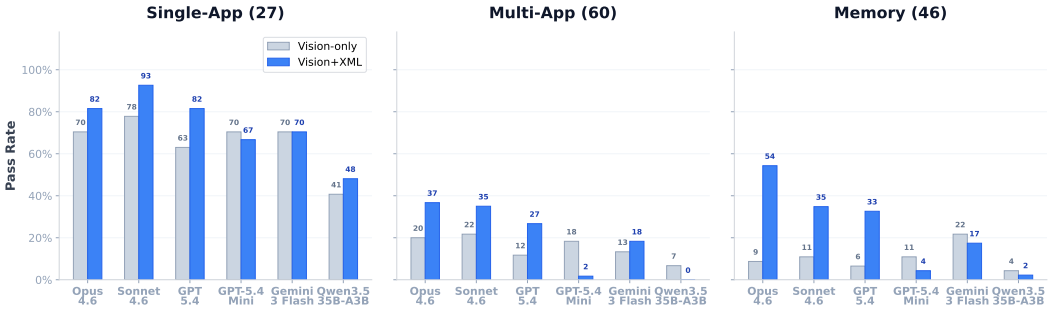

Figure 4: Pass rates by task category and observation modality across all six models. Vision+XML (blue) outperforms vision-only (gray) for the stronger frontier models. GPT-5.4 Mini and the open-source Qwen3.5 baseline do not show the same benefit from the additional modality.

4 Experiments

4.1 Setup

We evaluate five frontier computer-use models: Claude Opus 4.6 and Claude Sonnet 4.6 (Anthropic, 2026b), GPT-5.4 and GPT-5.4 Mini (OpenAI, 2026), and Gemini 3 Flash (Google, 2026). Each provider offers a dedicated computer-use API with native screenshot understanding and action generation. To compare against open-source phone-agent capability, we additionally include Qwen3.5 35B-A3B (Qwen Team, 2026), an open-weights MoE model with 35B total / 3B active parameters, served via vLLM and prompted with the official Qwen3-VL mobile-agent cookbook (`mobile_use` tool on a 999×999 coordinate grid). We test each model under both observation modalities, yielding twelve configurations. All runs use a maximum of 50 interaction steps per task with screenshots at 1536-pixel maximum dimension. Within each modality, all models receive equivalent system prompts adapted to their action vocabularies. We use GPT-5.4 Mini as the trajectory judge; human agreement analysis on 128 Opus 4.6 trajectories confirms substantial human–judge agreement (κ=0.77 task-level; Appendix I). Tasks were generated by Claude Code but reviewed and corrected by human annotators before evaluation.

4.2 Results

Privileged vision+XML access helps the stronger frontier models (see also Figs. 7 and 3): Opus rises from 26% to 52% overall (+25.6%), Sonnet from 29% to 47% (+17.3%), and GPT-5.4 from 20% to 40% (+19.5%). Fig. 3 shows the gap on a multi-app QuickBite → TeamChat

task: vision-only Opus reaches checkout but gets stuck on a small payment-confirmation control and never opens TeamChat, while vision+XML Opus places the order, posts the deployment update, and checks #general announcements in 22 steps. With vision+XML, Sonnet reaches 93% on single-app tasks and Opus leads on memory at 54% and multi-app at 37%, though multi-app tasks remain the hardest category. Fig. 5 traces a successful memory trajectory: Opus pulls balances from MyBank, checks SplitPay pending requests, then synthesizes a budget projection in CloudDocs — five apps, 29 steps, no app named in the prompt. In vision-only mode, frontier models cluster between 20% and 29%, with Sonnet (29%) and Opus (26%) leading via the strongest single-app numbers (78% and 70% respectively); Gemini at 28% is the most step-efficient (averaging only 21 steps per task vs. 42–45 for the Anthropic and OpenAI models). Opus leads overall at 52% with vision+XML, followed by Sonnet at 47%. GPT-5.4 Mini and Qwen3.5 do not show the same gain from the extra accessibility-tree context, suggesting a capacity limit rather than a problem with the modality itself (see §4.3).

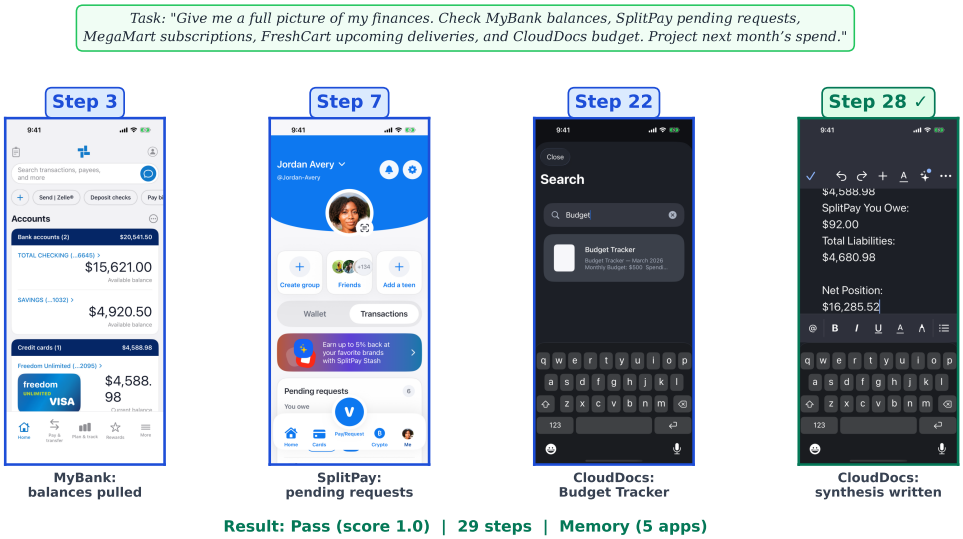

Figure 5: Successful memory trajectory (Opus 4.6, vision+XML, 29 steps, score 1.0): in response to "Give me a full picture of my finances," the agent pulls balances from MyBank, checks pending requests in SplitPay, opens the Budget Tracker in CloudDocs, and writes a synthesis spanning five apps — a latent-pattern task with no app named in the prompt.

4.3 Analysis

Why does XML help so much? The vision-to-XML gap is large due to iOS-specific factors: dense interfaces with small touch targets make coordinate estimation error-prone, app launching requires navigating home screens (which `launch_app` bypasses), the accessibility tree exposes off-screen elements, and iOS lacks a hardware back button. Because the XML setting changes both observation and available actions, this should be read as a privileged-access condition rather than an isolated test of accessibility-tree text (Fig. 3). Across the 26 Opus tasks where vision-only fails (score <0.5) and vision+XML passes outright, $\sim$70% feature a home-screen / app-switching failure eliminated by `launch_app`. The gain is also not uniform across categories: memory sees the largest absolute improvement (Opus: 9% $\rightarrow$ 54%) because the tree lets agents read element labels and values directly; multi-app benefits too (Opus: 20% $\rightarrow$ 37%; Sonnet: 22% $\rightarrow$ 35%) once the agent can launch apps and target elements precisely.

Smaller models struggle with the extra context. Two smaller-capacity configurations break the gain pattern. For GPT-5.4 Mini, vision+XML lands at 16% versus 26% vision-only: the additional $\sim$3,100 tokens per step appear to push it past its effective context limit, and 22 of the 35 tasks Mini solves with vision-only become failures under XML (Fig. 9). Qwen3.5 35B-A3B shows a related but more severe pattern: XML drops it from 13% to 11% overall and from 7% to 0% on multi-app, with $\sim$42% of its 119 XML failures dominated by action

loops (Fig. 10). The accessibility tree helps when the model can use it and becomes noise when it cannot.

Failure taxonomy. We classify all 422 vision+XML failures across the five frontier models into four mutually exclusive modes: *timed out* (48%), *gave up* (26%), *premature stops* (22%), and *looping out* (5%). Timing out dominates multi-app (54%) and memory (47%); premature stops dominate single-app (50%). GPT-5.4 Mini gives up on 46% of its failures, Gemini loops on 15%, and Qwen3.5 has a different profile entirely (∼42% looping, ∼30% gave-ups across 119 XML failures). Fig. 6 shows a representative timed-out failure; full breakdowns are in Appendix D.

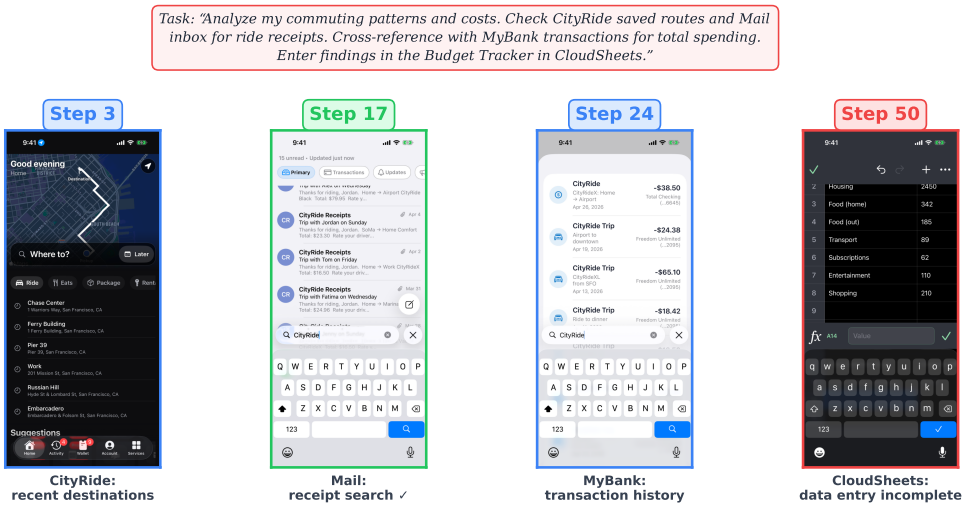

Figure 6: Representative timed-out failure (Opus 4.6, vision+XML, 50 steps, score 0.45): the agent explores CityRide (step 3), finds Mail receipts (step 17), and reaches MyBank transactions (step 24), but exhausts the 50-step budget before completing data entry in CloudSheets. Timed-out failures account for 48% of all frontier-model failures.

Scaling with interaction steps. Cumulative pass rate vs. step budget (Fig. 8, appendix) shows single-app saturating by step 20, multi-app scaling through step 40, and memory varying widely (Opus climbs from 17% at step 30 to 54% at step 50). GPT-5.4 Mini plateaus at 16% by step 25 and Qwen3.5 reaches only 11% by step 50.

Judge validation. The trajectory judge agrees with human annotators on 128 Opus 4.6 trajectories at κ=0.77 on binary task success (89% accuracy, F1=0.86) and κ=0.69 on individual rubric criteria (Pearson r=0.85). Per-annotator breakdowns and the per-step judge comparison are in Appendix I.

5 Conclusion

We introduced iOSWorld, an interactive native iOS benchmark with a persistent user identity spanning 26 purpose-built applications, designed to measure how personally intelligent today's phone agents are. Frontier models solve up to 93% of single-app tasks with privileged vision+XML access, but the best overall configuration still reaches only 37% on multi-app and 54% on memory tasks; the open-source Qwen3.5 35B-A3B baseline remains far behind (11% overall, 0% multi-app, ∼42% loop-failures). 48% of frontier failures are timeouts. Closing the gap to personally intelligent phone agents will require progress on loop recovery, retrieval-augmented memory, and user-aware planning.

Ethics Statement

Synthetic data. All data in iOSWorld is entirely synthetic. The Jordan Avery persona is fictional, and no real user data was collected, processed, or used at any stage. Benchmark runs use deterministic seeded data and do not depend on real user accounts, real services, or external databases. We chose this design specifically to enable research on personalization and memory tasks without the privacy risks inherent in real user data.

Malicious Agents. Phone agents capable of operating autonomously on a user's device carry significant dual-use risks. An agent with access to personal messaging, banking, ride-hailing, and email apps could be misused for surveillance, unauthorized transactions, social engineering, or data exfiltration. Even well-intentioned agents can cause harm through errors, such as sending a message to the wrong contact, making an unintended purchase, or leaking personal information across apps. The personalization and memory tasks in iOSWorld are particularly sensitive because they require agents to reason about personal data, which is the most damaging if mishandled. We encourage researchers to develop agents with explicit user consent mechanisms and action confirmation for irreversible operations.

iOS access and reproducibility. iOSWorld requires macOS with Xcode to run the iOS Simulator, which limits reproducibility to researchers with access to Apple hardware. We release all source code, seed data, and evaluation scripts. The closed-source nature of iOS means the vision+XML modality relies on XCUITest, a developer tool unavailable to a deployed consumer agent; vision-only numbers reflect deployed capability, while vision+XML represents an upper bound with privileged access.

Accessibility and intended use. Capable phone agents could improve accessibility for users with visual, motor, or cognitive impairments who find complex multi-step workflows difficult. iOSWorld is a research benchmark for measuring progress in a controlled simulator; results should not be interpreted as indicating readiness for deployment on real devices with real user data.

Annotator welfare. Task creation and verification were performed by research assistants compensated at standard institutional rates. The work involved interacting with simulated apps containing fictional data and did not expose annotators to harmful content.

References

Anthropic. Claude code. https://docs.anthropic.com/en/docs/claude-code/overview, 2026a.

Anthropic. Claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, 2026b.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *NeurIPS*, 2024.

Hao Bai, Yifei Zhou, Li Erran Li, Sergey Levine, and Aviral Kumar. Digi-q: Learning vlm q-value functions for training device-control agents. *ICLR*, 2025.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale. *ICML*, 2025.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, 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. *ICLR*, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *NeurIPS*, 2023.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks? *ICML*, 2024.

Google. Gemini 3 flash. https://blog.google/products/gemini/gemini-3-flash/, 2026.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *ACL*, 2024.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents. *CVPR*, 2024.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *ACL*, 2024.

Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. B-moca: Benchmarking mobile device control agents across diverse configurations. *CoLLAs*, 2025.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *ICLR*, 2018.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. Guiodyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *ICCV*, 2025.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: A benchmark for general ai assistants. *ICLR*, 2024.

OpenAI. Gpt-5.4. https://developers.openai.com/api/docs/models/gpt-5.4, 2026.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Qwen Team. Qwen3.5-35b-a3b. https://huggingface.co/Qwen/Qwen3.5-35B-A3B, 2026.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. *NeurIPS*, 2023.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. *ICLR*, 2025.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. *ICML*, 2017.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *ICLR*, 2024.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. *MobiCom*, 2024.

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

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking llm agents on consequential real world tasks. *NeurIPS*, 2025a.

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. *ACL*, 2025b.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *COLM*, 2025.

Pei Yang, Hai Ci, and Mike Zheng Shou. macosworld: A multilingual interactive benchmark for gui agents. *NeurIPS*, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *NeurIPS*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *ICLR*, 2025.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms. *ECCV*, 2024.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *CHI*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. *NeurIPS*, 2023.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *ICLR*, 2024.

A LLM Disclosure

We used large language models in several stages of this work. All drafting and structural decisions were made by human authors. Claude Code was used to polish prose, check grammar, and verify consistency with human-in-the-loop review. Figures and plots were generated programmatically via Claude Code from human-provided sketches, with the human author directing layout and content at every iteration. We used a multimodal LLM coding agent to perform high-level quantitative analysis, such as aggregating scores, computing pass rates, and to flag qualitative patterns in agent trajectories (e.g., identifying

failure modes from screenshots). All flagged results were reviewed, verified, and synthesized into written analysis by human authors. The 26 iOS applications were built in SwiftUI using Claude Code as a coding assistant with human developers verifying correctness, and tasks and rubrics were generated by Claude Code then reviewed, refined, and manually executed end-to-end by human annotators. Finally, we use GPT-5.4 Mini as an LLM-as-a-judge evaluator, validated against human annotators (κ=0.77).

B Application Details and Dataset Statistics

Table 4 lists all 26 applications. QuickChat (44 task references), Notes (41), CloudDocs (35), and Mail (29) are the most frequently involved apps.

App	Analogue	Key Seed Data and Features
MyBank	Chase	Checking $15,621; Savings $4,920; Credit $4,589/$8,000; 597 ledger transactions
SplitPay	Venmo	17 core users, 24 suggestions, 80 transactions, $650/mo rent from Arnav, 6 pending
QuickChat	WhatsApp	54 contacts, 40 conversations (26 DMs + 14 groups)
TeamChat	Slack	12 members, 11 channels (#eng-mobile, #launch-war-room), 8 DMs
Mail	Mail	88 inbox, 11 sent, 2 drafts; senders span all other apps
SkyTrip	Delta	Gold Medallion, 127K miles, 5 upcoming trips (SFO→JFK, SEA, ORD, HNL, LHR)
CityRide	Uber	Home/Office saved, 79 trips, 20 drivers, 5 ride types
StayFinder	Airbnb	Upcoming: Catalina Island, Barcelona sail. Past: Tahoe, Big Sur, Lisbon
QuickBite	DoorDash	82 restaurants, 5 saved addresses, 30+ past orders (Chipotle, Sweetgreen)
FreshCart	Instacart	13 stores, 112 products, 36 orders, scheduled/active/delivered
MegaMart	Amazon	Prime member, 10 depts, 17 cart items, 51 saved items, 42 orders
TasteRank	Beli	60+ restaurants, 8 saved lists (Date Night, Seoul Guide, etc.)
DineSpot	OpenTable	74 restaurants, 7 cities, outdoor/tasting/vegetarian filters
CalTrack	MyFitnessPal	77+ foods, 2,450 cal goal, 170g protein target, weight 182 lbs
CloudDocs	Google Docs	Mobile Benchmark Plan, Research Outline, Agent Systems Review
CloudDrive	Google Drive	Product Strategy, Shared Assets, Archive folders
CloudSheets	Google Sheets	Budget Tracker, Experiment Results (multi-sheet)
CloudSlides	Google Slides	Conference Practice Deck, Agent Systems Talk
Notes	Apple Notes	4 folders, 13 notes, pinned: Shopping List, Wifi Passwords
Cinephile	Letterboxd	Wishlists, Seenlist, custom lists, fan clubs
TicketBox	SeatGeek	18 venues, 66 events, Taylor Swift Eras Tour, Hamilton
TrailBlaze	Strava	4 athletes, 3 routes, 6 activities, PRs (5K: 19:42), clubs
ScoreZone	ESPN	Favorites: Lakers, Chiefs, Yankees; 40 seeded teams, scores
Weather	Weather	SF, NYC, Dallas; hourly and 10-day forecasts
Clock	Clock	Alarms, world clocks (Tokyo, London), stopwatch, timer
LockedIn	LinkedIn	Connections, job postings, professional profile

Table 4: The 26 iOS applications in iOSWorld with real-world analogues and seed data.

Per-app difficulty. Table 5 shows pass rate for Opus 4.6 (vision+XML) across all 26 apps. Cinephile is hardest (12%, 8 references) and CloudDrive (14%, 7 references), while CalTrack is easiest (65%, 17 references). Mail (59%, 29 references) and MyBank (59%, 22 references) are also among the strongest. QuickChat remains challenging (20%, 44 references) due to precise thread navigation across many tasks.

C Rubric-Based Evaluation Details

Each task is accompanied by a rubric, a list of independently verifiable criteria decomposing the objective into steps. The benchmark contains 1,123 rubric items across 133 tasks, ranging from 4 to 13 per task (mean 8.4). Multi-app tasks are the most rubric-dense at 9.4

App	Pass	Total	Rate
CalTrack	11	17	65%
Mail	17	29	59%
MyBank	13	22	59%
ScoreZone	5	9	56%
TeamChat	12	22	55%
QuickBite	9	17	53%
MegaMart	9	17	53%
StayFinder	5	10	50%
CloudDocs	17	35	49%
CityRide	8	17	47%
TrailBlaze	7	15	47%
TicketBox	7	16	44%
Notes	17	41	41%
LockedIn	4	10	40%
SplitPay	9	23	39%
Clock	6	16	38%
CloudSlides	3	8	38%
SkyTrip	4	11	36%
FreshCart	5	15	33%
TasteRank	4	12	33%
Weather	6	21	29%
CloudSheets	5	18	28%
QuickChat	9	44	20%
DineSpot	3	17	18%
CloudDrive	1	7	14%
Cinephile	1	8	12%

Table 5: Per-app pass rate for Opus 4.6 (vision+XML) across all 26 apps.

items on average, reflecting the number of intermediate steps needed to coordinate across applications.

Rubric scores reveal hidden progress. Binary pass rates understate agent capability. Under vision+XML, the average rubric score (fraction of criteria satisfied) ranges from 32% (Qwen3.5) to 81% (Opus), meaning frontier agents satisfy a majority of criteria even on tasks they ultimately fail. The rubric perfect rate (all criteria satisfied) tracks the binary pass rate within 1–2pp, confirming internal consistency between the holistic judge and per-criterion evaluation.

Per-step evaluation. We also evaluated a per-step variant in which the judge reviews each screenshot independently and we take the maximum across steps per criterion. This approach yields higher rubric scores (73–87% average) but proved more lenient than the trajectory judge when validated against human annotators (κ=0.51–0.61 vs. 0.77 for the trajectory judge). Its mean rubric score is 0.83 versus 0.70 for humans, producing more than twice as many false-positive criteria (188 vs. 79). Since iOSWorld tasks are relatively straightforward and compact (mean 8.4 criteria, max 50 steps), the trajectory judge provides sufficient discrimination without the added complexity of per-step evaluation. We use the trajectory-level judge throughout the main text.

D Failure Analysis Details

Methodology. We scan the action trace for $\geq$3 consecutive identical actions (action type + discretized parameters: tap coordinates rounded to integers, swipe direction, typed text). If a loop is detected and the run hit the 50-step limit: *looped out*. Hit the limit without a loop: *timed out*. Stopped before the limit with rubric ≥ 0.67: *premature stop*. Stopped early with rubric < 0.67: *gave up*.

Failure Mode	Category (%)			
	Single	Multi	Mem	All
Looped out	0	4	7	5
Timed out	19	54	47	48
Premature stop	50	21	16	22
Gave up	31	21	30	26

Table 6: Failure mode distribution under vision+XML (422 failures across the five frontier models × 133 tasks). The open-source Qwen3.5 baseline (119 XML failures) is summarized separately in the main-text Failure taxonomy paragraph because its failure distribution differs from the frontier-model aggregate. Categories are mutually exclusive.

Examples. *Timed out*: Sonnet hits 50 steps trying to reply in QuickChat (msg-003, score 0.45). *Premature stop*: GPT-5.4 reports fare and ETA correctly but stops at the final "Request" button after 8 steps without confirming the booking (uber-001, score 0.80). *Gave up*: Opus abandons a game-day planning task after 44 steps with only partial progress across ScoreZone, QuickChat, and SplitPay (mem-020, score 0.65).

Per-category (five frontier models). *Single-app*: 29/135 failures, 50% premature stops, mean rubric 0.81. *Multi-app*: 229/300, 54% timed out, 21% premature stops; pass rate by app count: 40% (2 apps), 32% (3), 31% (4), 6% (5–6), 0% (7+). *Memory*: 164/230, 47% timed out, 30% gave up.

Open-source baseline (Qwen3.5 35B-A3B). 119/133 vision+XML failures (10.5% pass rate). Per category: *Single-app*: 14/27 failures, 64% looping (loop-dominated even on simple tasks). *Multi-app*: 60/60 (0% pass), 42% looping, 32% timed out. *Memory*: 45/46, 38% looping, 42% gave up. The loop-dominance pattern is consistent across categories and contrasts sharply with the frontier models, where loops account for only 5% of failures overall.

E Additional Results Figures

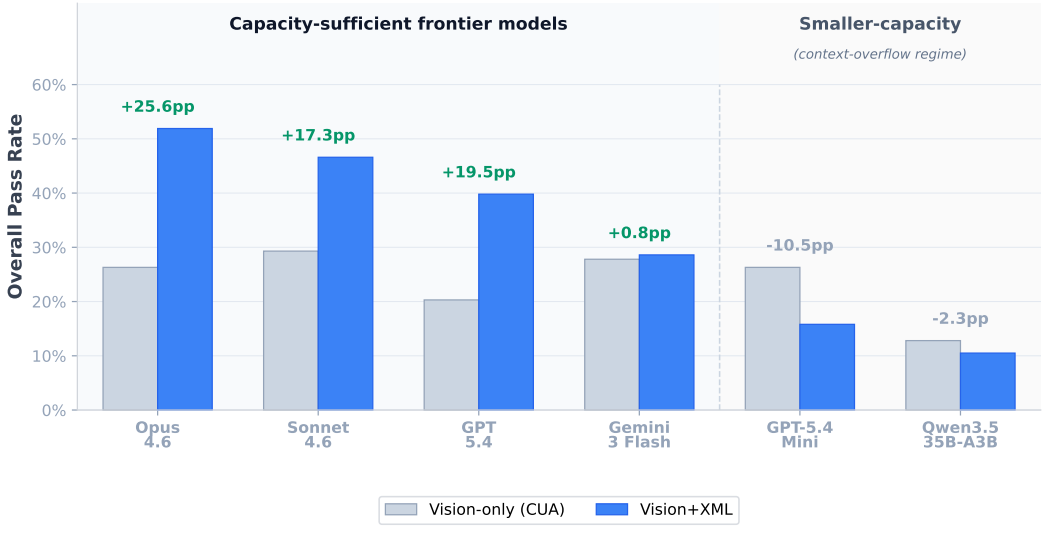

Figure 7: Vision-only vs. vision+XML accuracy per model, grouped by model capacity. Privileged vision+XML access improves the stronger frontier models (Opus +25.6pp, Sonnet +17.3pp, GPT-5.4 +19.5pp, Gemini +0.8pp). Smaller models (GPT-5.4 Mini, Qwen3.5 35B-A3B) do not benefit from the additional accessibility-tree input.

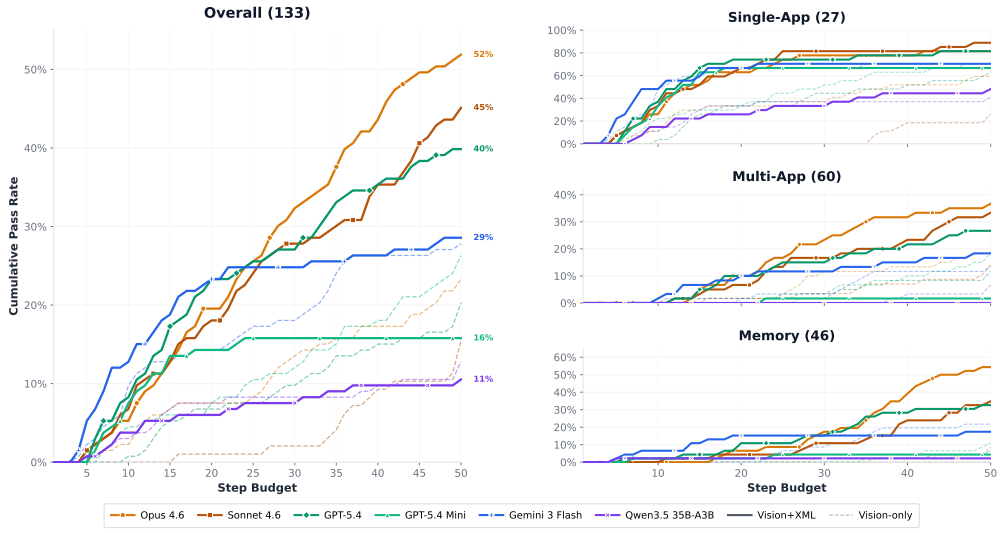

Figure 8: Cumulative pass rate vs. step budget. **Left:** overall. **Right:** by task category. Solid: vision+XML; dashed: vision-only. Single-app saturates by step 20; multi-app scales through step 40; memory shows varied scaling with Opus climbing steeply past step 30.

Figure 9: GPT-5.4 Mini on the same DineSpot reservation task under both modalities. **Left**: vision-only navigates cleanly to a confirmed booking in 24 steps (score 1.0). **Right**: vision+XML loops through filter menus for 30 steps and ultimately forgets the original goal (score 0.4, 37 steps). The accessibility tree overwhelms the smaller model's context capacity.

F App and Task Construction

App creation. Apps were created or adapted using Claude Code with a structured prompt specifying constraints (SwiftUI, deterministic seeded data, accessibility identifiers), workflows, data models, and seed quantities. Apps underwent iterative refinement and manual verification by human developers.

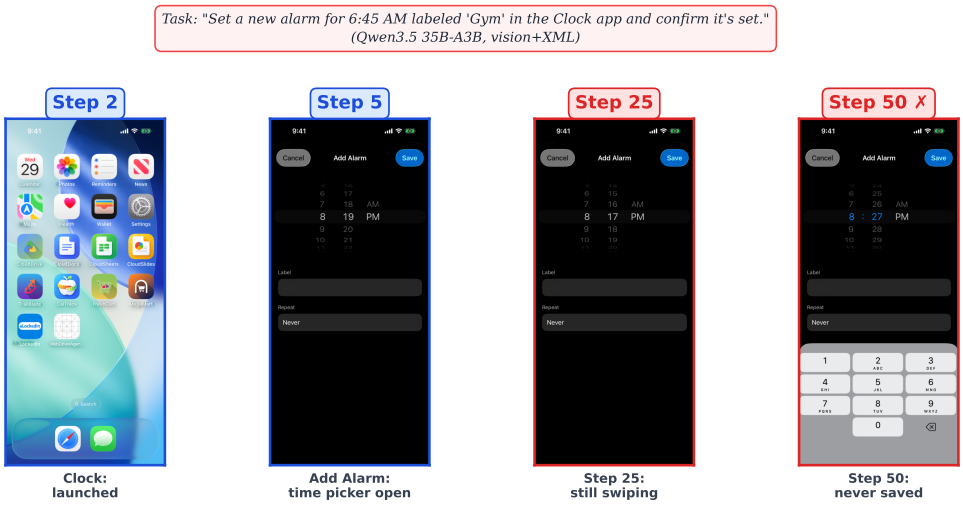

Figure 10: Qwen3.5 35B-A3B (vision+XML) on a simple "Set a 6:45 AM alarm labeled Gym" single-app task. The agent reaches the Add Alarm screen by step 5 but then issues the *same* swipe-down action on the time picker 38 consecutive times (steps 6–46), never adjusting to 6:45, never setting the label, and never tapping Save. The 50-step budget is exhausted on a task Opus and Sonnet both solve in 25 steps. This pattern accounts for ∼42% of Qwen3.5's 119 XML failures.

Action	Opus	Sonnet	GPT-5.4	Mini	Gemini	Qwen3.5
tap_xy	61%	61%	54%	37%	53%	65%
swipe	20%	23%	12%	35%	10%	13%
type	8%	8%	18%	21%	17%	4%
launch_app	–	–	–	–	18%	1%
wait	1%	1%	8%	3%	<1%	1%
home	7%	7%	7%	3%	<1%	16%

Table 7: Action type distribution per model under vision+XML. Qwen3.5 35B-A3B uses the cookbook `mobile_use` tool; `system_button(Home)` is mapped to home.

Task pipeline. Stage 1: Claude Code generated tasks grounded in app source code and seed data. Stage 2: A Python pipeline normalized app names, rewrote tasks in first-person voice, and generated rubric criteria. Stage 3: Human annotators executed every task on the simulator.

G Prompts

Agent system prompt. All models receive the following iOS-specific instructions. Action names are adapted per provider (e.g., `left_click` for Claude, `click` for OpenAI, `click_at` for Gemini; see Tab. 8). The version below is for Claude CU; the vision+XML variant appends the accessibility tree instructions at the end.

```
You are controlling an iOS Simulator (iPhone). This is a touch-screen
mobile phone with NO mouse cursor, NO physical keyboard shortcuts, and NO
right-click.
CURRENT STATE: You start on the iOS home screen. You must find and open
apps yourself.
HOW TO OPEN APPS:
- Tap an app icon on the home screen if it is visible.
- To search for an app: swipe DOWN from the MIDDLE of the home screen to
open Spotlight search, then type the app name and tap the result.
- Swipe left/right on the home screen to browse additional pages of apps.
```

```
TOUCH INTERACTIONS:
- Use 'left_click' for all touch/tap interactions (there is no mouse
cursor).
- To type text, click a text field first, then use the 'type' action to
enter text.
- To scroll content, use the 'scroll' action with delta_x/delta_y.

iOS-SPECIFIC BEHAVIOURS:
- HOME: Use key 'Home', or swipe up from the very bottom of the screen.
- APP SWITCHER: Swipe up from the bottom and pause mid-screen.
- BACK NAVIGATION: Look for a back button (top-left) or swipe from the
left edge.
- KEYBOARD DISMISS: Tap any area outside the text field.

COMPLETING THE TASK:
- When the task is complete, stop calling the computer tool and respond
with a text summary.
- IMPORTANT: If the task asks you to find, check, look up, or report ANY
information, you MUST include that exact information in your final text
response.

ACCESSIBILITY TREE (appended in vision+XML mode only):
On each turn you will also receive a text accessibility tree of the
current UI. The tree lists every element with its type, name/label, value,
accessibility IDs (shown as id="..."), and centre coordinates.

IMPORTANT: The coordinates in the tree are in the SAME coordinate space
as your action coordinates.  You can use tree coordinates DIRECTLY as
click/tap targets without any conversion or mapping.

How to use the two inputs together:
- The SCREENSHOT is ground truth for what is displayed on screen.
- The TREE provides precise element names and coordinates for targeting.
- Use tree coordinates DIRECTLY to click more precisely than visual
estimation.
- If the screenshot and tree disagree (e.g. an element appears in the tree
but not on screen), trust the screenshot; the element may be off-screen
or obscured.
- Elements marked [hidden] are in the DOM but not rendered on screen.
- If you need to find elements not currently visible, try scrolling.
```

Trajectory-level evaluation. The judge (GPT-5.4 Mini) receives the following prompt structure. Per-step screenshots are attached as images:

```
Goal: [task instruction]

You are evaluating whether an iOS agent successfully completed the
above goal. The agent executed N steps. Full trajectory with per-step
screenshots:
Step 1: [Screenshot 1 - before actions] Actions: [JSON]
Step 2: ...
[Screenshot N+1 - final state after all actions]

The agent's final answer was: "[answer]"

N screenshots are attached, one per step showing the screen state before
each action, plus one final screenshot showing the end state.

Evaluate each of the following rubric criteria individually:
1. [criterion] (weight: W)
2. ...

Respond with ONLY a JSON object (no code fences):
{"success": true/false, "reasoning": "overall assessment",
"rubric_results": [{"criterion": "...", "satisfied": true/false,
"reasoning": "..."}]}

success=true means the goal is fully and completely achieved. For each
rubric criterion, set satisfied=true only if there is clear evidence in
the trajectory and screenshots that the criterion was met. For tasks that
ask a question, evaluate whether the agent's final answer is correct.
```

Per-step evaluation. Each step is evaluated independently in a separate LLM call. The judge receives one screenshot, the agent's action, and its reasoning. For the final step, the agent's answer is also included:

```
Goal: [task instruction]
You are evaluating an iOS agent's progress at step K. The attached
screenshot shows the device screen after the agent's action.
Agent's action: [JSON]
Agent's reasoning: [text]
Agent's final answer: [text]   (last step only)
Which of the following rubric criteria are NOW satisfied based on the
screenshot, the agent's action, and its reasoning?
1. [criterion]
2. ...
Respond with ONLY a JSON object:
{"satisfied": [list of criterion numbers that are satisfied]}
Return an empty list if none are satisfied.  Only mark a criterion
satisfied if there is clear evidence from the screenshot AND the agent's
actions/reasoning.  Do not infer satisfaction from ambiguous or partial
evidence.
```

All steps are evaluated in parallel (up to 8 concurrent calls per task); we take the max across steps per criterion. Once a criterion is satisfied at any step, it remains satisfied.

CUA action translation. Each provider uses its own action vocabulary. We map all native actions to our unified iOS action schema (Table 8):

iOS Action	Claude CU	OpenAI CUA	Gemini CU	Qwen mobile_use
tap_xy	left_click	click	click_at	click
tap_xy $\times 2$	double_click	double_click	–	–
tap_xy $\times 3$	triple_click	triple_click	–	–
type	type	type	type_text_at	type
type (keys)	key	keypress	key_combination	system_button
swipe	scroll	scroll	scroll_at	swipe
swipe	–	–	scroll_document	–
swipe	left_click_drag	drag	drag_and_drop	–
home	key Home	keypress Home	go_home	system_button Home
wait	wait	wait	wait_5_seconds	wait
launch_app	–	–	open_app	–
hover	–	–	long_press_at	long_press
open_url	–	–	open_url	–
stop	(text)	(text)	(text)	terminate / answer

Table 8: Action translation from provider-native vocabularies to our iOS action schema. Scroll direction is inverted for all providers (scroll down = swipe up on touchscreen). Coordinates: Claude and OpenAI output pixel coordinates scaled to 0–1000; Gemini outputs 0–999 directly; Qwen mobile_use outputs 0–999 (cookbook contract). Actions marked "–" are not available for that provider. Qwen mobile_use follows the official Qwen3-VL mobile-agent cookbook tool schema.

H Example Trajectories

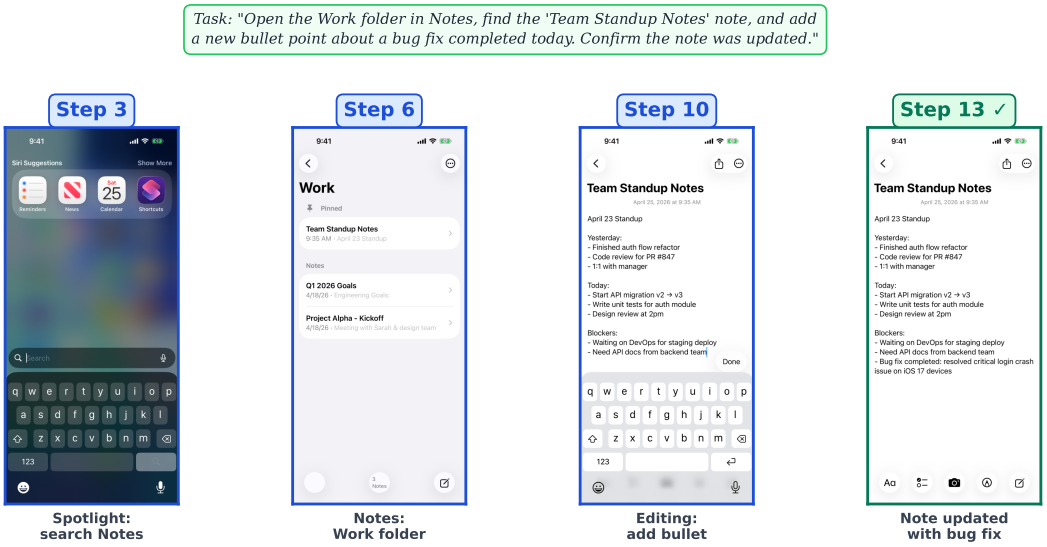

Figure 11: Successful single-app: Notes — Team Standup Notes, add bug-fix bullet (Sonnet 4.6 CUA, 13 steps, score 1.0).

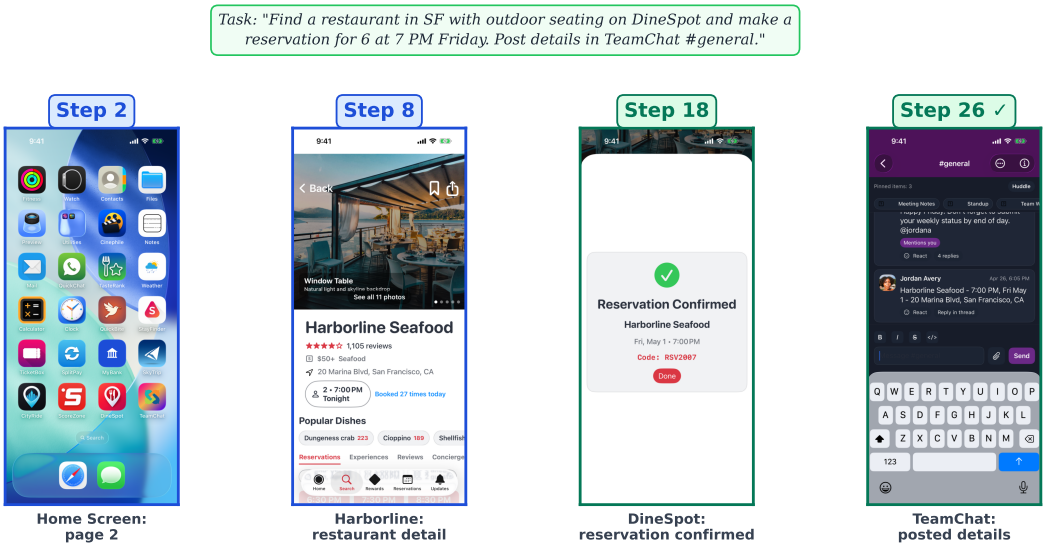

Figure 12: Successful multi-app: DineSpot reservation → TeamChat post (Opus 4.6, vision+XML, 26 steps).

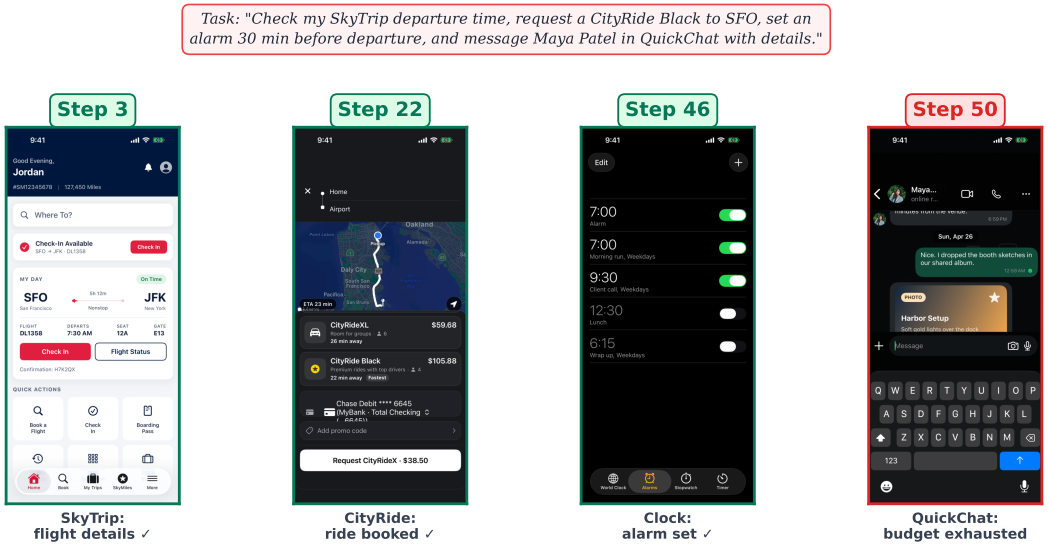

Figure 13: Failed multi-app: SkyTrip → CityRide → Clock → QuickChat, completed 3/4 subtasks but ran out of budget before messaging (50 steps, score 0.56).

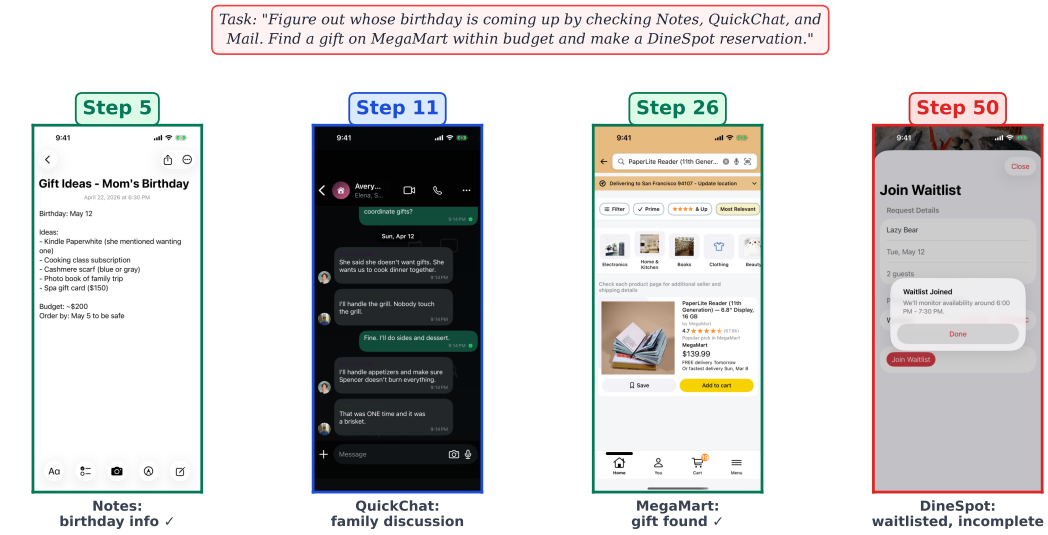

Figure 14: Failed memory: Notes → QuickChat → MegaMart → DineSpot, found birthday info and bought gift but ran out of budget for dinner reservation (50 steps, score 0.50).

I Human Agreement

To validate the automated evaluation pipeline, we collect human annotations on 128 trajectories from the Opus 4.6 vision+XML configuration, spanning all three task categories. Four annotators each reviewed a subset of trajectories and graded every rubric criterion as *pass* or *fail*, as well as providing an overall binary success judgment. When multiple annotations exist for the same task, we use the most recent one. We compare these human judgments against both automated judges, which are the trajectory-level judge, which sees the full action trace, and the per-step judge, which evaluates each screenshot independently and takes the max across steps.

Table 9 reports task-level and rubric-criterion agreement, plus the per-step judge comparison.

Comparison	Cohen's κ	F1	Acc.	ρ
Human vs. Trajectory Judge				
Binary task success	0.77	0.86	0.89	0.77
Rubric criteria (n=1,094)	0.69	0.90	0.86	0.69
Human vs. Per-Step Judge				
Binary task success	0.61	0.79	0.80	0.63
Rubric criteria (n=1,094)	0.51	0.87	0.81	0.56
Trajectory Judge vs. Per-Step Judge				
Binary task success	0.55	0.75	0.77	0.57
Rubric criteria (n=1,094)	0.55	0.88	0.83	0.59

Table 9: Extended agreement between human annotators and automatic judges on 128 trajectories (1,094 rubric criteria), including judge-vs-judge comparison. ρ: Spearman correlation.

The trajectory-level LLM judge achieves substantial agreement with human annotators across all metrics. At the task level, binary success judgments agree 89% of the time (κ=0.77, F1=0.86); at the rubric-criterion level, the judge correctly classifies 86% of individual criteria (κ=0.69, F1=0.90). The rubric-level κ is lower despite comparable accuracy because Cohen's κ is sensitive to marginal distributions. 67% of rubric criteria are satisfied, inflating expected chance agreement and mechanically suppressing κ. Accuracy and F1 are more directly interpretable here. The continuous rubric scores are highly correlated. Pearson r=0.85 and Spearman ρ=0.86 between human and trajectory-judge rubric fractions, with a mean absolute error of 0.10. The per-step parallel judge shows lower agreement (κ=0.51–0.61), primarily because it is more lenient: its mean rubric score is 0.83 versus 0.70 for humans, producing 188 false-positive criteria (compared to 79 for the trajectory judge). This aligns with the design difference, as per-step evaluation marks a criterion satisfied if *any* single screenshot shows evidence, which can overcount partial progress.

The 148 total disagreements between humans and the trajectory judge split into 79 false positives (LLM too lenient) and 69 false negatives (LLM too strict), indicating no strong systematic bias in either direction.

Per-annotator analysis. Table 10 breaks down agreement by annotator. Four annotators each graded 26–47 trajectories. Task-level κ ranges from 0.64 to 0.92, while rubric-level κ is more tightly clustered (0.67–0.72), suggesting that per-criterion judgments are more consistent across annotators than holistic task-level judgments. The annotator with the lowest task-level κ (0.64, 47 tasks) has the highest rubric-level κ (0.72), indicating that disagreements at the task level stem from borderline cases where most but not all criteria are satisfied, rather than from fundamentally different rubric interpretations. Human pass rates are consistent with the LLM judge across all annotators (36–50% human vs. 36–46% LLM), with no annotator showing a systematic leniency or strictness bias.

Annotator	Tasks	κ_{task}	Acc.	κ_{rubric}	Criteria
A	47	0.64	0.83	0.72	397
B	28	0.84	0.93	0.67	248
C	27	0.92	0.96	0.67	232
D	26	0.77	0.88	0.69	217

Table 10: Per-annotator agreement with the trajectory judge. κ_{task}: Cohen's kappa on binary task success. κ_{rubric}: Cohen's kappa on individual rubric criteria. All annotators show substantial agreement ($\kappa \geq 0.64$).