

SCENDROID: A SCENARIO-LEVEL BENCHMARK FOR LONG-HORIZON, TIME-EVOLVING GUI AGENTS

Zhe Wu¹, Yongxin Kang¹, Dabin Sheng¹, Junliang Xing^{*1}, Guokun Wu¹, Derek Yuen²
 Donglin Mo¹, Yuheng Jing³, Kai Li³, Weilin Luo², Kun Shao², Yuanchun Shi¹

¹Department of Computer Science and Technology, Tsinghua University

²Huawei Noah’s Ark Lab, ³Institute of Automation, Chinese Academy of Sciences

ABSTRACT

Recent breakthroughs in Vision Language Models (VLMs) have empowered GUI agents to perform discrete, short-horizon tasks. However, prevailing benchmarks predominantly rely on an *Atomic Reset* paradigm, which treats user interactions as isolated episodes, thereby failing to capture the continuous and state-dependent nature of real-world workflows. To bridge this gap, we introduce **ScenDroid**, which employs a novel **App-Task-Scenario (ATS)** architecture to orchestrate dependency-aware workflows (exceeding **1200 steps**) across persistent Android environments spanning simulated days to weeks. Beyond operational execution, ScenDroid incorporates a **Progressive Ambiguity Taxonomy** and an integrated **Interactive User Simulator** to assess an agent’s capacity for proactive intent clarification and long-term preference alignment. Our extensive evaluation of state-of-the-art GUI agents reveals a catastrophic performance collapse, elucidating critical cognitive bottlenecks in structured episodic memory and closed-loop reasoning. We further deconstruct these failure modes and provide a strategic roadmap for developing “Digital Agents” capable of persistent, autonomous, and human-aligned interaction. ScenDroid is available at <https://github.com/GoooKuuu/scendroid/>.

1 INTRODUCTION

In the pursuit of Artificial General Intelligence (AGI), GUI agents (Wang et al., 2024b) serve as a vital bridge between the digital and physical worlds. By mimicking human visual perception and motor control (e.g., clicking, swiping, typing), these agents navigate existing software ecosystems to execute complex tasks. Recent breakthroughs in vision-language models (Bai et al., 2023; Wang et al., 2024c; Bai et al., 2025b; Zhang et al., 2024; Hong et al., 2024; Qin et al., 2025) have demonstrated remarkable proficiency in single-step UI understanding and short-term task execution.

However, despite the surge in model capabilities, existing benchmarks lag significantly behind. Mainstream benchmarks (Rawles et al., 2024; Zhang et al., 2023; Xie et al., 2024) primarily adopt an “*Atomic Reset*” methodology. In this paradigm, agents are initialized in a sterile state, receive a clearly defined instruction (e.g., “*Open the Clock and set an alarm for 7 AM*”), and the environment is immediately reset or rolled back upon task completion. This evaluation approach relies on an unrealistic assumption: that user operations are independent, atomic events free from context.

Real-world operations do not occur in a vacuum; they are embedded in a spatiotemporal continuum. Consider the example in Figure 1, actions have lasting consequences: a file downloaded in the morning is a prerequisite for an email sent in the afternoon, and unread notifications inevitably accumulate, occluding the interface. By resetting the environment, existing benchmarks shield agents from this state accumulation, failing to test their ability to manage the natural disorder/clutter of prolonged usage. Furthermore, authentic workflows are governed by long-horizon causality—scheduling a meeting is futile if the agent fails to record the minutes days later. Finally, unlike the precise prompts found in rigid testbeds, human intent is inherently ambiguous and context-dependent, necessitating agents that can actively clarify requirements rather than passively following orders.

*Corresponding Author: jlxing@tsinghua.edu.cn

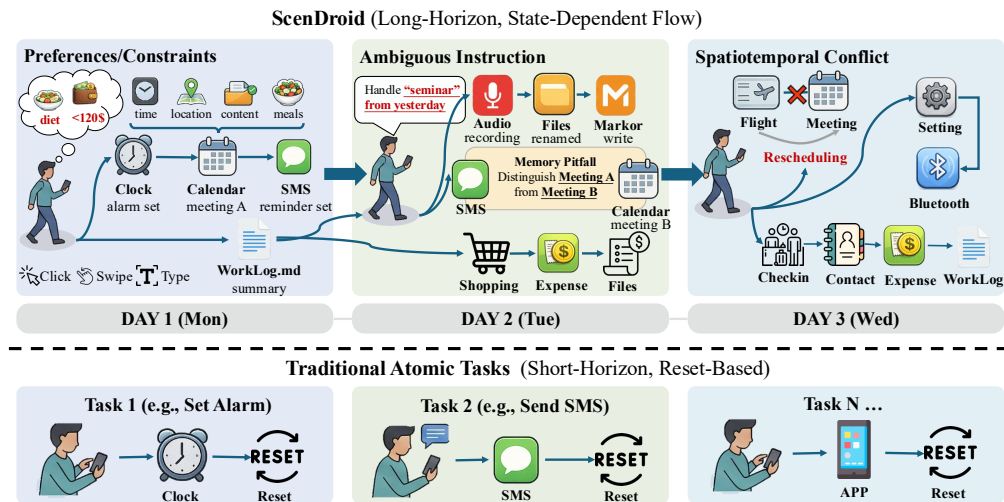


Figure 1: **ScenDroid vs. Atomic-Reset evaluation.** Unlike traditional benchmarks that *reset* the environment after each task, ScenDroid evaluates agents in a *persistent, evolving* timeline.

To address these limitations, we propose **ScenDroid** (Scenario-Level Android Benchmark), aiming to redefine the standard for GUI agent evaluation. ScenDroid is not merely a more difficult dataset; it represents a paradigm shift from “*Task-Oriented*” to “*Scenario-Oriented*” evaluation. The core contributions of this work are as follows:

- 1) **A Paradigm Shift to Scenario-Level Evaluation.** We elevate the evaluation unit from atomic tasks to comprehensive *Scenarios* spanning hundreds to thousands of steps. We design coherent “Daily” and “Weekly” scenarios that simulate realistic behavior patterns, focusing on long-term state memory and cross-day dependency management.
- 2) **The ATS Hierarchical Architecture.** We introduce the **App–Task–Scenario** framework to ensure logical coherence across extended timelines. This modular design decouples low-level UI grounding from high-level orchestration, allowing for the systematic generation of complex, dependency-aware workflows via parameterized templates.
- 3) **Scenario-Aware Multi-dimensional Evaluation.** ScenDroid introduces a robust evaluation that accounts for the inherent complexity of long-term interaction. By incorporating a **Progressive Ambiguity Taxonomy** (L0–L2) and an **Interactive User Simulator**, we assess an agent’s proactive capacity for intent clarification and preference alignment.
- 4) **A Reproducible, High-Fidelity Infrastructure.** ScenDroid provides a complete, version-fixed evaluation infrastructure, including pre-populated emulator snapshots and a Docker-encapsulated e-commerce sandbox. This open-source platform guarantees that researchers can evaluate agents in a consistent, real-world-aligned environment.

2 RELATED WORK

2.1 CURRENT GUI BENCHMARK PARADIGM

GUI agent evaluation has evolved from static screen-parsing to interactive execution. Early static datasets, such as PixelHelp (Li et al., 2020), AITW (Rawles et al., 2023), and ScreenSpot (Cheng et al., 2024; Li et al., 2025), assess UI detection and single-step prediction but lack the sequential planning required for complex workflows. Recent interactive benchmarks (Chen et al., 2024), such as AndroidWorld and OSWorld, have introduced programmatic task generation and system-state verification. However, as detailed in Table 1, these platforms predominantly adhere to an “Atomic Reset” paradigm. By resetting the system state (e.g., clearing caches, resetting clocks) after each task, they artificially eliminate the *temporal dependencies* and *state accumulation* inherent in real-world usage. ScenDroid distinguishes itself by enforcing state persistence across simulated timelines, moving from task-level to scenario-level evaluation.

Table 1: **Comparison of ScenDroid with existing GUI benchmarks and datasets.** We highlight whether each trait is fully supported (✓), partially supported (✓), or not supported (✗).

Benchmark / Dataset	Level	Max Horizon	Parametric Scaling	Persistent Interaction	Intent Ambiguity	Active Clarify
AITW (Rawles et al., 2023)	Task	1	✗	✗	✗	✗
AppAgent (Zhang et al., 2025)	Task	40	✗	✓	✗	✗
AndroidWorld (Rawles et al., 2024)	Task	120	✓	✓	✗	✗
AndroidArena (Xing et al., 2024)	Task	30	✗	✓	✗	✗
Mobile-Bench-v2 (Xu et al., 2025)	Task	25	✗	✓	✓	✓
MVISU-Bench (Huang et al., 2025)	Task	1	✗	✗	✓	✓
MobileWorld (Kong et al., 2025)	Task	50	✗	✓	✓	✓
ScenDroid (Ours)	Scenario	1200+	✓	✓	✓	✓

2.2 LONG-HORIZON PLANNING AND EPISODIC MEMORY

Research in open-ended domains (Fan et al., 2022), such as Minecraft (Wang et al., 2023) or simulation sandboxes (Park et al., 2023), has highlighted the necessity of structured memory (Wang et al., 2021) and hierarchical planning (Agashe et al., 2024) to combat context drift and error propagation. In the GUI domain, current evaluations rarely exceed a few minutes of interaction (Wang et al., 2024a). Benchmarks like AndroidArena and AppAgent focus on app-specific proficiency but overlook the *cross-application data flow* that characterizes daily work. ScenDroid addresses this gap by requiring agents to maintain a coherent episodic memory over workflows spanning hundreds of steps (up to 1200+), testing their ability to retrieve information stored days prior.

2.3 AMBIGUITY RESOLUTION AND ACTIVE INTERACTION

Realistic human instructions are rarely fully specified. While recent efforts have begun to categorize the inherent ambiguity of human instructions (Xu et al., 2025; Huang et al., 2025), they lack a persistent memory of user preferences across tasks. ScenDroid bridges this gap by simulating continuous “Weekly Scenarios,” in which an agent must retrieve user habits established in previous sessions to resolve ambiguous commands in the current one. Through its integrated Interactive User Simulator, ScenDroid provides a standardized protocol for evaluating an agent’s ability to initiate clarification dialogues and align with latent user intent over the long term during deployment.

3 THE SCENDROID FRAMEWORK

This section describes ScenDroid’s construction and evaluation. We first present the **App–Task–Scenario (ATS)** pipeline (Figure 2) for scalable yet coherent benchmark generation (§3.1). We then explain ScenDroid’s *persistent* rollouts over simulated days to weeks, including controlled time evolution and state drift (§3.2). Next, we introduce the **Progressive Ambiguity Taxonomy** and a standardized protocol for measuring proactive clarification (§3.3). Finally, we detail ScenDroid’s verification suite and scenario-level metrics for effectiveness, efficiency, and interaction quality (§3.4).

3.1 THE ATS HIERARCHICAL ARCHITECTURE

ScenDroid replaces flat, independent tasks with a hierarchical **App–Task–Scenario (ATS)** construction pipeline. ATS is designed to (i) scale to diverse interactions, (ii) preserve logical coherence across long horizons, and (iii) expose cross-task dependencies that are masked by atomic resets.

App layer: controllability and state primitives. ScenDroid integrates **20** real-world Android applications, spanning system utilities, communication, note-taking, fitness tracking, and a controlled open-source e-commerce sandbox (detailed in Appendix A.2). To ensure reproducible evaluation without per-task rollback, each app is paired with an evaluator providing: (i) *scenario initialization*

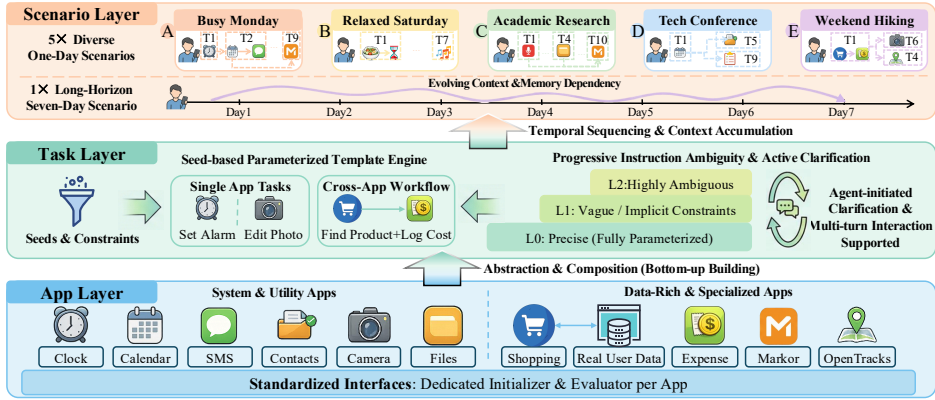


Figure 2: **The ATS Hierarchical Architecture:** the **App Layer** defines state primitives; the **Task Layer** instantiates parameterized templates under different ambiguity levels; the **Scenario Layer** composes tasks into dependency-aware daily and weekly workflows.

(injecting starting configurations like pre-populated chats/files/databases), and (ii) *persistent state access* (reading ground-truth artifacts such as database entries or filesystem objects produced during execution). Importantly, initialization happens *once per scenario rollout* (or per simulated day block), rather than resetting to a baseline before every task.

Task layer: parameterized templates. The Task layer defines **56** atomic templates representing semantic units of user intent (detailed in Appendix B). Templates include **intra-app tasks** (operations within one app, e.g., creating a calendar event) and **inter-app tasks** (cross-application data flow, e.g., extracting meeting details from Calendar and notifying attendees via SMS). Each template \mathcal{T} is instantiated by a generator $G(\mathcal{T}, \theta) \rightarrow \langle \mathcal{I}, \mathcal{C}_{init}, \text{Verifier} \rangle$, where \mathcal{I} is a natural-language instruction with randomized parameters θ , \mathcal{C}_{init} specifies scenario context bindings (e.g., target names, budgets, file paths), and Verifier is an *executable goal checker* for success verification (§3.4).

Scenario layer: dependency-aware orchestration. The Scenario layer composes instantiated tasks into long-horizon workflows via a dependency graph and a rollout schedule (visualized in Figure 3; full catalog in Appendix B.1). Formally, a scenario is $\Sigma = \langle \mathcal{V}, \mathcal{E}, \mathcal{C}_{global} \rangle$, where nodes \mathcal{V} are instantiated tasks, edges \mathcal{E} encode dependencies, and \mathcal{C}_{global} is a scenario-level context store that accumulates artifacts produced by earlier tasks. We model three dependency types: (i) *hard prerequisites* (a task is feasible only if a predecessor completes), (ii) *data-flow dependencies* (later parameters are bound from earlier outputs), and (iii) *temporal constraints* (tasks must occur within specified virtual time windows). During evaluation, tasks are executed along a time-ordered schedule consistent with the dependency graph (topological order with temporal windows), preserving the realism of daily workflows while retaining explicit causal structure.

3.2 PERSISTENT SCENARIO ROLLOUTS AND TIME EVOLUTION

Building on ATS, ScenDroid synthesizes coherent **Daily Scenarios** (10–11 tasks/day) and **Weekly Scenarios** (70+ tasks/week), targeting long-horizon interactions of up to **1200+ UI steps**. The key distinction from atomic-reset benchmarks is that ScenDroid evaluates agents within a *persistent* Android state: artifacts and clutter accumulate, and the environment evolves over time.

Scenario-level persistence (no per-task rollback). Within a scenario rollout, ScenDroid does *not* reset app state after each task. Instead, all state changes—including calendar entries, messages, downloads, app notifications, shopping cart history, and system settings—remain in the environment and may affect subsequent tasks. This exposes realistic failure cascades (e.g., a missed download breaks a later attachment step) and forces agents to perform recovery under accumulated disorder.

Controlled time evolution via time warping. To simulate days-to-weeks interaction within practical runtime, ScenDroid advances a *virtual clock* between tasks and triggers platform-consistent time updates so time-dependent app behaviors (alarms, reminders, scheduled notifications) evolve as in real usage. Crucially, time evolution is *controlled and reproducible*: exogenous events (e.g., incoming messages, asynchronous reminders, inventory changes in the e-commerce sandbox) are driven by a seeded event scheduler tied to the virtual clock, enabling consistent cross-agent comparisons.

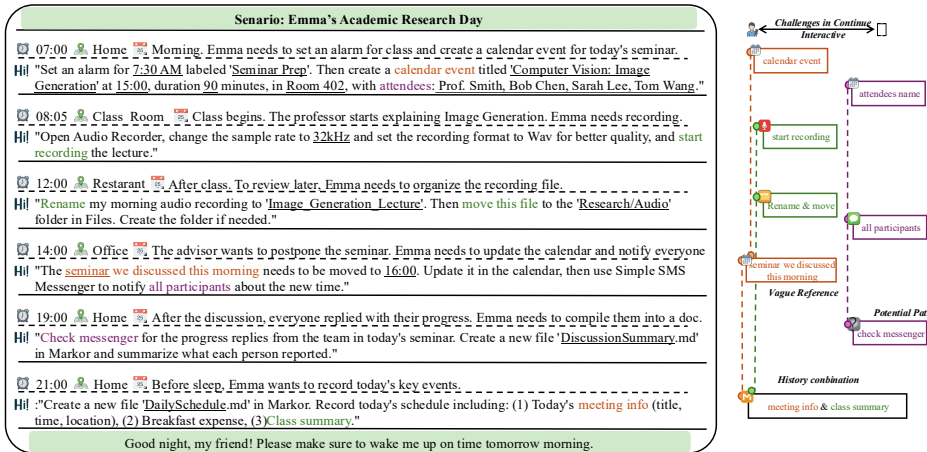


Figure 3: **Case Study of a Daily Scenario: Emma’s Academic Research Day.** This example illustrates a persistent interaction sequence spanning 14 hours.

Dependency handling under failures. In long-horizon rollouts, failures are informative rather than ignored. If a hard prerequisite fails, dependent tasks may become infeasible; ScenDroid records such tasks as unsuccessful (and optionally marks them as blocked for analysis) while preserving the current device state. This evaluation policy reflects real deployments, where the system does not magically revert to a clean slate after mistakes.

3.3 PROGRESSIVE AMBIGUITY AND INTERACTIVE CLARIFICATION

Real user instructions are often underspecified and context-dependent. ScenDroid therefore abandons the “perfect instruction” assumption and evaluates an agent’s ability to *recognize ambiguity*, *ask clarifying questions*, and *remain consistent with user preferences* over long horizons.

Progressive Ambiguity. We classify instructions into three levels:

Table 2: Progressive Ambiguity Taxonomy in ScenDroid (full instruction lists in Appendix B.2).

Level	Definition	Requirement	Instruction Example
L0	Ground Truth	Direct Execution	“Open Clock and change alarm to 8:40.”
L1	Parameter Omission	Common-sense Reasoning	“Move my alarm 10 minutes earlier.”
L2	Indirect Ambiguity	Proactive Clarification	“I need to wake up earlier tomorrow.”

Interactive clarification protocol. For L2 instructions, the agent may execute an explicit communication action $a^{ask}(q)$ to ask a question q . ScenDroid integrates an LLM-based **Interactive User Simulator** with hidden “oracle” parameters corresponding to the underlying L0 intent. To ensure standardization, the simulator returns *minimal*, *slot-filling* clarifications (e.g., the specific time, recipient, or file name) rather than open-ended coaching (implementation details in Appendix C.2). We enforce a per-task clarification budget and penalize excessive or irrelevant questions in interaction-quality metrics (§3.4), preventing agents from trivially increasing success by over-querying.

3.4 VERIFICATION SUITE AND SCENARIO-LEVEL METRICS

Evaluating long-horizon GUI interaction requires robust verification beyond surface screenshots. ScenDroid uses an executable verification suite that directly inspects persistent system artifacts whenever possible, supplemented by UI-structure checks and semantic judging for QA-style tasks.

Executable verifiers. Each task uses a verifier accessing one or more of the following signals:

(1) *Persistent system-state checks.* For tasks that commit durable changes (e.g., calendar insertions, message sends, file downloads, expense logs), ScenDroid verifies outcomes by inspecting application databases (SQLite), system content providers, and filesystem artifacts.

(2) *UI-structure and navigation checks.* For transient UI-bound goals (e.g., navigating to a target view, toggling a setting, reaching a specific activity), ScenDroid checks the accessibility node tree and activity metadata for the presence of required components or state transitions.

(3) *Semantic verification for information retrieval.* For QA tasks where the agent must *report* information (e.g., “What time is my next meeting?”), ScenDroid combines keyword/entity matching for exact fields with a frozen LLM judge that assesses semantic correctness against hidden oracle records, reducing brittleness to paraphrases while preventing hallucinated answers.

Scenario-level quantitative metrics. We report three complementary metrics capturing effectiveness, efficiency, and interaction quality. Consider a scenario containing N tasks. Let $s_i \in \{0, 1\}$ denote whether task i is successful, t_i denote the number of steps taken to attempt task i , and c_i denote the number of clarification actions used for task i .

(1) *Effectiveness: Success Rate (SR).*

$$SR = \frac{1}{N} \sum_{i=1}^N s_i. \tag{1}$$

(2) *Efficiency: Average Steps (AS).*

$$AS = \frac{1}{N} \sum_{i=1}^N t_i. \tag{2}$$

(3) *Interaction quality: Clarification-Adjusted Success (CAS).* Asking more questions can increase success but degrades user experience. We measure success discounted by clarification overhead:

$$CAS = \frac{1}{N} \sum_{i=1}^N \frac{s_i}{1 + \eta c_i}, \tag{3}$$

where η is a penalty coefficient (default $\eta = 0.5$). Higher CAS indicates that an agent succeeds with fewer clarifications, rewarding precise and economical intent alignment.

4 EXPERIMENTS

In this section, we empirically evaluate SOTA GUI agents within the ScenDroid environment. Our goal is not only to rank agents, but to diagnose the cognitive deficits exposed when they transition from sterile, atomic task execution to *continuous, time-evolving* scenario rollouts. We focus on three questions: **(RQ1)** Can current SOTA agents, which excel in isolated evaluation, remain reliable under accumulated state drift and environmental clutter? **(RQ2)** Does scaling model size or adopting hierarchical planning mitigate these challenges under persistent rollouts? **(RQ3)** To what extent does access to historical context (contextual memory) close the gap between AI and human performance?

4.1 EXPERIMENTAL SETUP

All scenarios are executed under ScenDroid’s **persistent environment**: the device is *not* rolled back between tasks. Artifacts produced in Task t (e.g., files, calendar entries, app-local databases) remain and become part of the persistent state observed in Task $t+1$.

Baselines. We evaluate human experts and three agent paradigms (implementation details in Appendix C.1): **(I) Human Performance.** We enlist three computer science graduate students to execute the same scenarios; their trajectories are verified by the identical verifiers used for agents. **(II) Episodic VLM Agents.** We evaluate monolithic VLM agents in a *task-episodic* mode: while the environment is persistent, the agent’s session state is reset at the start of each task. This matches the standard evaluation interface in prior atomic-reset benchmarks. **(III) Hierarchical Agents (Planner–Actor).** We investigate whether advanced reasoning can mitigate long-horizon challenges. A high-capacity *Planner* decomposes instructions into subgoals, which are then executed by a local *Actor*. These agents also operate in episodic mode to isolate the benefit of *high-level planning* from explicit context retrieval. **(IV) Context-Augmented Agents.** We provide agents with a *context pack* at the beginning of each task. This pack includes (i) completed task history and (ii) summarized prior trajectories, evaluating how explicit memory facilitates dependency resolution.

Table 3: Success Rate (SR, %) across daily and weekly scenarios.

Agent	Daily Scenarios					Weekly	Overall
	Scen-A	Scen-B	Scen-C	Scen-D	Scen-E	OmniLife	
Human	90.9	82.9	89.7	93.3	84.3	87.3	88.1
<i>Episodic End-to-End VLM Agents</i>							
Qwen3-VL-235B (Bai et al., 2025a)	28.5	28.7	18.6	32.5	31.7	23.6	27.3
Qwen3-VL-32B (Bai et al., 2025a)	23.5	25.1	21.1	27.5	23.0	22.9	23.8
Qwen3-VL-8B (Bai et al., 2025a)	14.5	22.7	18.3	22.5	12.5	20.7	18.5
Doubao-VL (Guo et al., 2025)	19.1	11.5	18.7	19.5	14.0	29.5	18.7
GeLab-Zero-4B (Yan et al., 2025)	25.2	16.0	13.1	27.5	12.6	16.7	18.5
GUI-OWL-7B (Ye et al., 2025)	11.6	18.2	22.4	15.0	7.5	17.1	15.3
<i>Episodic Hierarchical Agents (Planner-Actor)</i>							
Qwen3-VL-235B + Qwen3-VL-8B	27.3	22.8	25.0	19.2	13.3	27.1	22.5
GPT-5.1 + Qwen3-VL-8B	28.7	22.5	29.0	47.5	24.0	31.4	30.5
Gemini-3-Pro + Qwen3-VL-8B	24.1	27.2	33.3	47.5	38.5	40.7	35.2
Claude-4.5-Sonnet + Qwen3-VL-8B	18.2	36.0	30.5	25.0	22.5	33.0	27.5
<i>Context-Augmented Agents</i>							
Qwen3-VL-235B + Context	30.7	36.0	28.6	47.5	29.0	32.1	34.0
Qwen3-VL-32B + Context	31.5	34.0	25.1	45.0	24.0	30.7	31.7
Qwen3-VL-8B + Context	22.5	29.5	22.5	35.0	19.0	29.3	26.3
Doubao-VL + Context	23.5	20.1	22.2	27.5	18.1	32.1	23.9
GeLab-Zero-4B + Context	18.2	15.6	11.1	23.7	15.9	18.6	21.2
GUI-OWL-7B + Context	13.5	20.1	26.7	15.8	27.2	17.9	20.2
Qwen3-VL-235B + Qwen3-VL-8B + Context	27.3	34.4	31.1	44.4	28.7	37.2	33.9
GPT-5.1 + Qwen3-VL-8B + Context	31.8	40.7	50.9	40.4	31.7	38.2	39.0
Claude-4.5-Sonnet + Qwen3-VL-8B + Context	22.7	36.4	28.9	45.4	22.7	35.9	32.0
Gemini-3-Pro + Qwen3-VL-8B + Context	45.5	41.2	52.8	57.2	28.6	50.5	45.9

4.2 MAIN RESULTS AND PERFORMANCE ANALYSIS

We evaluate agents on five “Daily Scenarios” (A–E) and a “Weekly Scenario” (OmniLife) spanning 1200+ steps. Table 3 summarizes performance, highlighting four key observations.

Observation 1: Performance Collapse in Scenario-Level Interaction. The strongest monolithic VLM baseline, *Qwen3-VL-235B*, achieves only **27.3%** Overall SR, far below the human baseline (**88.1%**). This gap suggests that the dominant bottleneck is no longer single-task UI understanding, but robustness to *accumulated state drift*, and *long-range causal dependencies*.

Observation 2: Scale is Not All You Need. Parameter scaling yields rapidly diminishing returns in persistent environments. While scaling 8B→32B yields gains (18.5%→23.8%), further scaling to 235B improves marginally (to 27.3%). This suggests that simply scaling the backbone cannot resolve the logical deficits required for consistent, long-term state tracking.

Observation 3: The Semantic Gap in Hierarchical Architectures. Planner-Actor systems outperform monolithic agents in some scenarios, yet gains are inconsistent and can regress. A recurring failure is *semantic misalignment* (Figure 4): planners may prematurely infer completion or propose abstract subgoals ungrounded in UI affordances, causing cascading errors.

Observation 4: Context is Necessary, Yet Insufficient. On *OmniLife*, context improves SR from 23.6% (*Qwen3-VL-235B*) to 32.1% (*Qwen3-VL-235B + Context*), and 37.2% with a Planner-Actor pipeline. However, performance remains below 40%, implying context aids retrieval but cannot fully resolve reasoning challenges of cross-day dependencies.

Diagnostic Analysis of Failure Modes. We categorize errors into six modes (Figure 4): (1) *Semantic Hallucination (32%)*: Incorrectly assuming completion based on internal logic rather than UI state. (2) *Contextual Memory Decay (27%)*: Failing to retrieve entities (e.g., file paths) from previous sessions. (3) *Constraint Neglect (16%)*: Overlooking user constraints (e.g., budgets) during execution. (4) *Cognitive Loops (11%)*: Repetitive actions caused by unrecognized state invariance. (5) *Operational Errors (8%)*: Visual grounding or actuation failures. (6) *Knowledge Gaps (6%)*: Misunderstanding domain-specific UI patterns. (Detailed cases in Appendix E).

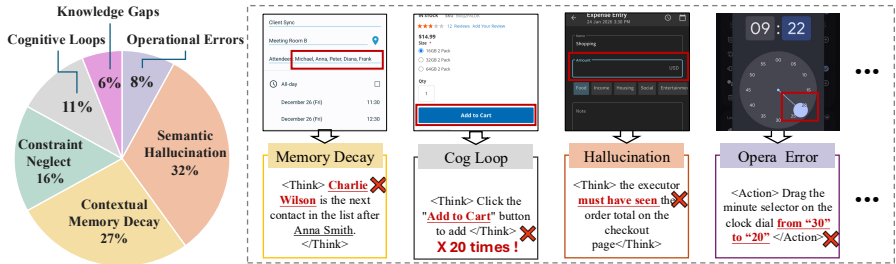


Figure 4: **Distribution of Failure Modes in ScenDroid.** We manually inspect over 2000 failed task attempts sampled across all agents and scenarios, and categorize them into six modes.

4.3 THE IMPACT OF AMBIGUITY AND ACTIVE CLARIFICATION

We categorize instructions into three ambiguity levels (L0–L2; definitions in Table 2, full datasets in Appendix B.2) and compare two *Qwen3-VL-235B* variants: a **Passive** variant (standard execution) and an **Active** variant (with *ask_user* action; prompt details in Appendix C.1).

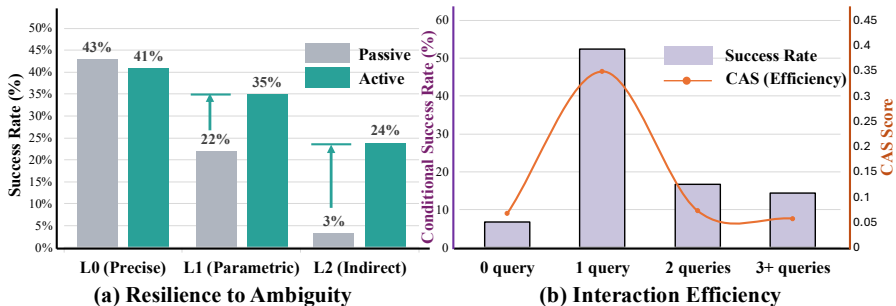


Figure 5: **Ambiguity and interactive clarification.** (a) Mean success rate of Passive vs. Active agents under L0–L2 ambiguity. (b) For L2 tasks, we bucket task attempts by clarification turns N_q and report conditional success and CAS.

Impact of Active Clarification. Figure 5(a) shows the performance gap across ambiguity levels. At L0 (explicit), both agents exhibit comparable performance ($\sim 42\%$), confirming communication tools do not regress execution. However, under Indirect Ambiguity (L2), the Passive agent fails catastrophically (SR $\sim 3\%$), proving latent intent is inferable only with feedback. Conversely, the Active agent recovers significant performance (24.0%), effectively bridging the semantic gap.

Interaction Efficiency Trade-off. We investigate *how* interaction quality dictates success by stratifying L2 tasks by clarification turns (N_q) in Figure 5(b). We observe a critical “One-Shot” Sweet Spot: *Optimal Efficiency ($N_q = 1$):* When the agent resolves ambiguity with a single, targeted question, the *conditional success rate* surges to 52.4%, exceeding even the L0 baseline. This implies precise clarification resolves ambiguity and grounds agent attention, reducing execution errors. *Ineffective Multi-turn ($N_q \geq 2$):* Performance degrades sharply as dialogue extends. For tasks requiring 2+ queries, success drops to $\sim 14\%$, and the Clarification-Adjusted Success (CAS) approaches zero. This reveals the moderate recovery (24%) averages two extremes: successful one-shot interactions and failed multi-turn loops. (Raw data and CAS analysis in Appendix D.2).

5 DISCUSSION

The transition from atomic tasks to persistent scenarios in ScenDroid exposes structural deficits in current agent architectures. Beyond specific failure modes, our findings imply four critical directions for evolving GUI agents from passive executors to adaptive digital partners.

5.1 CONTEXT WINDOWS TO STRUCTURED EPISODIC MEMORY

Our results show that extending context windows yields diminishing returns. In multi-day scenarios, critical information (e.g., historical preferences) is easily obscured by thousands of intermediate UI tokens. The current unstructured attention mechanism is insufficient for long-term grounding.

Context Length \neq Memory Capacity

Key insight. Future architectures must decouple *working memory* for immediate interaction from *structured episodic memory* for long-term anchoring. Agents need mechanism to explicitly index and retrieve history, moving beyond implicit attention over raw token streams.

5.2 CLOSING THE LOOP: FEEDBACK-DRIVEN SELF-CORRECTION

A predominant failure mode is open-loop execution, where agents hallucinate task completion or loop mechanically. This stems from treating the GUI as a static canvas rather than a fluid environment, failing to react to transient cues (e.g., loading spinners, toast notifications).

Toward Dynamic UI World Models

Key insight. Robustness requires a shift from plan-then-execute to execute-perceive-adapt. Agents must integrate dynamic visual world models to predict state transitions and utilize subtle environmental feedback as anchoring signals for real-time error recovery.

5.3 BRIDGING THE SEMANTIC FRACTURE IN HIERARCHY

While Planner-Actor designs are theoretically robust, we observe a persistent *representation gap*. High-level planners often generate abstract subgoals that fail to ground in the actor’s pixel-level affordances. This linguistic decoupling leads to unrecoverable execution errors.

End-to-End Hierarchical Alignment

Key insight. Breakthroughs may require *End-to-End Hierarchical Alignment*. Instead of loose linguistic interfaces, future systems should employ shared latent spaces to ensure that high-level reasoning is structurally constrained by low-level environmental affordances.

5.4 PROACTIVE INTENT ALIGNMENT OVER PASSIVE OBEDIENCE

ScenDroid proves that real-world instructions are inherently underspecified. Agents designed solely for command obedience fail under L2 ambiguity. GUI evaluation must shift focus from operational accuracy to the more sophisticated challenge of social and cognitive alignment.

From Instruction Following to Intent Alignment

Key insight. The paradigm must evolve from *Passive Following* to *Proactive Alignment*. Agents should function as partners with a “Theory of Mind,” inferring latent needs. Optimizing *Clarification-Adjusted Success (CAS)* is the next frontier for human-centric digital assistants.

6 CONCLUSION

We introduced ScenDroid, a scenario-level benchmark for Android GUI agents that moves beyond the conventional “Atomic Reset” paradigm to expose critical bottlenecks in long-term state accumulation and ambiguity. Our evaluation reveals that current models, despite their scale, lack the structured memory and feedback essential for real-world deployment. By providing a reproducible infrastructure, ScenDroid urges a research shift from single-task optimization to cultivating agents capable of dynamic adaptation and human-centric collaboration in persistent environments.

REFERENCES

- Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024.
- Anthropic. Introducing claude sonnet 4.5, 2025. <https://www.anthropic.com/news/claude-sonnet-4-5>.
- Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 2023.
- 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, Sibao 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*, 2025a.
- Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibao Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.
- Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, et al. Spa-bench: A comprehensive benchmark for smartphone agent evaluation. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.
- Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclck: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024.
- Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35: 18343–18362, 2022.
- Google. Gemini 3 pro: Best for complex tasks and bringing creative concepts to life, 2025. <https://deepmind.google/models/gemini/pro/>.
- Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, et al. Seed1. 5-vl technical report. *arXiv preprint arXiv:2505.07062*, 2025.
- Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024.
- Zeyu Huang, Juyuan Wang, Longfeng Chen, Boyi Xiao, Leng Cai, Yawen Zeng, and Jin Xu. Mvisu-bench: Benchmarking mobile agents for real-world tasks by multi-app, vague, interactive, single-app and unethical instructions. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 8797–8805, 2025.

- Quyuan Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, et al. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive, and mcp-augmented environments. *arXiv preprint arXiv:2512.19432*, 2025.
- Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 8778–8786, 2025.
- Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile ui action sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8198–8210, 2020.
- OpenAI. Gpt-5.1: A smarter, more conversational chatgpt, 2025. <https://openai.com/zh-Hans-CN/index/gpt-5-1/>.
- Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.
- Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.
- Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36:59708–59728, 2023.
- Christopher Rawles, Sarah Clinckemaiellie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyi Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandhakar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- Hanqing Wang, Wenguan Wang, Wei Liang, Caiming Xiong, and Jianbing Shen. Structured scene memory for vision-language navigation. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 8455–8464, 2021.
- Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *Advances in Neural Information Processing Systems*, 37:2686–2710, 2024a.
- 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. *arXiv preprint arXiv:2401.16158*, 2024b.
- Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. Qwen2-vl: Enhancing vision-language model’s perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024c.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.
- 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, 2024.

- Weikai Xu, Zhizheng Jiang, Yuxuan Liu, Pengzhi Gao, Wei Liu, Jian Luan, Yuanchun Li, Yunxin Liu, Bin Wang, and Bo An. Mobile-bench-v2: A more realistic and comprehensive benchmark for vlm-based mobile agents. *arXiv preprint arXiv:2505.11891*, 2025.
- Haolong Yan, Jia Wang, Xin Huang, Yeqing Shen, Ziyang Meng, Zhimin Fan, Kaijun Tan, Jin Gao, Lieyu Shi, Mi Yang, Shiliang Yang, Zhirui Wang, Brian Li, Kang An, Chenyang Li, Lei Lei, Mengmeng Duan, Danxun Liang, Guodong Liu, Hang Cheng, Hao Wu, Jie Dong, Junhao Huang, Mei Chen, Renjie Yu, Shunshan Li, Xu Zhou, Yiting Dai, Yineng Deng, Yingdan Liang, Zelin Chen, Wen Sun, Chengxu Yan, Chunqin Xu, Dong Li, Fengqiong Xiao, Guanghao Fan, Guopeng Li, Guozhen Peng, Hongbing Li, Hang Li, Hongming Chen, Jingjing Xie, Jianyong Li, Jingyang Zhang, Jiayu Ren, Jiayu Yuan, Jianpeng Yin, Kai Cao, Liang Zhao, Liguang Tan, Liying Shi, Mengqiang Ren, Min Xu, Manjiao Liu, Mao Luo, Mingxin Wan, Na Wang, Nan Wu, Ning Wang, Peiyao Ma, Qingzhou Zhang, Qiao Wang, Qinlin Zeng, Qiong Gao, Qiongyao Li, Shangwu Zhong, Shuli Gao, Shaofan Liu, Shisi Gao, Shuang Luo, Xingbin Liu, Xiaojia Liu, Xiaojie Hou, Xin Liu, Xuanti Feng, Xuedan Cai, Xuan Wen, Xianwei Zhu, Xin Liang, Xin Liu, Xin Zhou, Yingxiu Zhao, Yukang Shi, Yunfang Xu, Yuqing Zeng, Yixun Zhang, Zejia Weng, Zhonghao Yan, Zhiguo Huang, Zhuoyu Wang, Zheng Ge, Jing Li, Yibo Zhu, Binxing Jiao, Xiangyu Zhang, and Daxin Jiang. Step-gui technical report, 2025. URL <https://arxiv.org/abs/2512.15431>.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.
- Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.
- Chi Zhang, Zhao Yang, Jiakuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–20, 2025.
- Danyang Zhang, Zhennan Shen, Rui Xie, Situo Zhang, Tianbao Xie, Zihan Zhao, Siyuan Chen, Lu Chen, Hongshen Xu, Ruisheng Cao, et al. Mobile-env: Building qualified evaluation benchmarks for llm-gui interaction. *arXiv preprint arXiv:2305.08144*, 2023.
- Jingyi Zhang, Jiaying Huang, Sheng Jin, and Shijian Lu. Vision-language models for vision tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5625–5644, 2024.
- Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Appendix

A. Infrastructure and Environment Details	14
· A.1 Emulator and System Configuration	14
· A.2 App Suite and Sandbox Data	14
· A.3 Formulation: Augmented Time-Evolving POMDP	15
B. Scenario and Instruction Design	16
· B.1 Scenario Catalog (Daily & Weekly)	16
· B.2 Progressive Ambiguity Instruction List (L0–L2)	24
C. Baseline Details and Clarification Protocol	34
· C.1 Agent Baselines and Prompts	34
· C.2 Interactive User Simulator	44
D. Additional Experimental Results	45
· D.1 Operational Efficiency and Navigational Precision	45
· D.2 Detailed Analysis of Intent Ambiguity and CAS Metrics	46
E. Case Study	46

A INFRASTRUCTURE AND ENVIRONMENT DETAILS

This appendix provides the technical specifications of the ScenDroid infrastructure, ensuring reproducibility and consistency. We detail the emulator configuration, the application ecosystem, the enclosed e-commerce sandbox, and the formal mathematical formulation of the environment.

A.1 EMULATOR AND SYSTEM CONFIGURATION

To minimize variance caused by hardware discrepancies and OS updates, ScenDroid operates on a strictly version-controlled virtualized environment.

Device Specifications. We utilize an Android Virtual Device (AVD) emulating a *Google Pixel 6* smartphone. The device runs on *Android 13 (API Level 33)* with a screen resolution of 1080×2400 (420 dpi). To ensure deterministic execution, we disable automatic system updates, predictive animations, and non-essential background services.

Deterministic scenario initialization without per-task rollback. Unlike atomic-reset benchmarks that restore a sterile device state after each task, ScenDroid initializes the device *once per scenario rollout* (or once per simulated day block, depending on the scenario schedule) and then keeps the environment persistent across tasks. To make such persistent rollouts reproducible, we (i) pin all system and app versions, and (ii) use seeded schedulers for time-dependent events (§A.3).

A.2 APP SUITE AND SANDBOX DATA

ScenDroid integrates 20 fully functional applications, encompassing both OS-level utilities and third-party open-source software. To strictly control the evaluation boundary, we ensure no apps require external server-side authentication (e.g., Google Sign-in) that might trigger anti-bot verification.

Application Catalog. Table 4 lists the 20 applications used in ScenDroid. For open-source apps, we fix the specific APK versions (sourced from F-Droid) to prevent UI layout shifts caused by updates.

Table 4: **The ScenDroid Application Suite.** We utilize 20 apps covering system utilities, productivity, communication, and entertainment. “Source” indicates whether the app is a core Android component (AOSP) or an open-source third-party app (F-Droid).

Category	App Name	Source	Core Functionality
System	Settings	AOSP	Manage device config (Wi-Fi, display, Bluetooth).
	Files	AOSP	Manage local filesystem (move, delete, rename).
	Contacts	AOSP	Create and manage user contact information.
	Camera	AOSP	Capture photos and videos (simulated feed).
	Clock	AOSP	Alarms, timers, and world clocks.
Productivity	Simple Calendar	F-Droid	Event scheduling and agenda management.
	Markor	F-Droid	Markdown-based note-taking and to-do lists.
	Tasks	F-Droid	Task/ToDo tracking with due dates and priorities.
	Pro Expense	F-Droid	Personal finance and expense tracking.
Communication	Simple SMS	F-Droid	Send/receive text messages (simulated network).
	Chrome	AOSP	Web browsing (access to local sandbox).
	Gmail (Client)	AOSP	Email client (configured for local SMTP).
Media	Simple Gallery	F-Droid	View and organize images/videos.
	Retro Music	F-Droid	Audio player for local music files.
	VLC	F-Droid	Multi-format video media player.
	Audio Recorder	AOSP	Record voice notes and audio clips.
	Simple Draw	F-Droid	Basic drawing and sketching.
Lifestyle	OsmAnd	F-Droid	Offline maps and navigation.
	Broccoli	F-Droid	Recipe management and cooking guides.
	OpenTracks	F-Droid	GPS sport tracking and activity analysis.

E-Commerce Sandbox (Shopping Environment). To evaluate shopping tasks without interacting with live commercial websites (which poses ethical, privacy, and stability issues), we deploy a self-hosted, high-fidelity e-commerce sandbox.

- *Infrastructure Origin:* We adopt the reproducible e-commerce environment infrastructure provided by **WebArena** (Zhou et al., 2023). This environment runs a local instance of **Adobe Magento**, a widely-used open-source e-commerce platform, encapsulated within a Docker container.
- *Data Source:* The store database is populated with the **WebShop** dataset (Yao et al., 2022), containing approximately **11,000 products** (filtered from the original dump for compatibility) across diverse categories. Each product features rich metadata, including variable pricing, attribute options (e.g., color, size), detailed descriptions, and realistic user reviews.
- *Deployment & Connectivity:* The Docker container is deployed on the local host machine. We utilize `adb reverse` port forwarding to map the container’s internal port to the emulator’s `localhost`. This allows the agent to access the store via the Android Chrome browser naturally, ensuring a seamless “app-to-web” interaction experience without external internet access.
- *Task Redesign:* While we leverage the infrastructure and data from prior work, we **completely redesigned the interaction tasks** to align with ScenDroid’s scenario-based paradigm. Unlike the isolated retrieval queries in WebShop, our shopping tasks are contextual and state-dependent (e.g., “*Check the delivery status of the gift purchased yesterday*” or “*Buy a router within the budget discussed in the SMS history*”).

Action Space Definition. Agents interact with the environment via a structured action space tailored for touchscreen manipulation. As shown in Table 5, the space includes primitive gestures (clicks, swipes) and high-level functional calls (typing, terminating).

Table 5: **Action Space.** Agents control the device via these function calls.

Action	Description	Arguments
click	Performs a tap at a specific (x, y) coordinate.	<i>point</i>
long_press	Executes a sustained press (e.g., to open context menus).	<i>point</i>
swipe	Drags from a start point to an end point (e.g., scrolling).	<i>start, end</i>
type	Injects text into the focused field (bypasses virtual keyboard).	<i>text_string</i>
system	Triggers hardware buttons (Home, Back, Enter).	<i>key_code</i>
answer	Outputs a natural language response for QA tasks.	<i>response</i>
ask_user	(Active Only) Initiates a clarification query to the user.	<i>query</i>
terminate	Ends the current task execution.	<i>status</i>

A.3 ENVIRONMENT FORMULATION: AUGMENTED TIME-EVOLVING POMDP

Conventional GUI agent tasks are typically modeled as standard Partially Observable Markov Decision Processes (POMDPs). However, in long-horizon, multi-day interactions, time is no longer a discrete step counter but a critical physical variable driving state transitions and stochastic environmental drift. Consequently, we formalize the problem space of ScenDroid as an Augmented Time-Evolving POMDP (TE-POMDP), defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \mathcal{D} \rangle$.

In TE-POMDP, the system state $S_t \in \mathcal{S}$ is constructed from the Cartesian product of three orthogonal subspaces: $S_{total} = S_{GUI} \times S_{Context} \times S_{Time}$. **GUI State Space** (S_{GUI}) comprises the current visual observation I_t (screenshot) and the accessibility node tree V_t of the Android device. **Context State Space** ($S_{Context}$) hosts implicit information persisted across tasks, including the file system S_{fs} , database records S_{db} (e.g., calendar entries, communication history), and user preferences S_{pref} . **Spatiotemporal State Space** (S_{Time}) maintains the virtual physical time T_{phy} .

The transition function $\mathcal{T}(s_{t+1}|s_t, a_t)$ introduces *stochastic external evolution*. The agent’s action space \mathcal{A} is extended to include operational actions \mathcal{A}_{op} (e.g., click, swipe, type) and communicative actions \mathcal{A}_{comm} . Ω represents the set of observations; $\mathcal{O}(o_t|s_t, a_{t-1})$ is the observation function mapping hidden states to perceived outputs; $\mathcal{R}(s_t, a_t)$ defines the sparse reward signal based on task completion; and \mathcal{D} denotes the temporal horizon, extended here to simulate multi-day workflows.

B SCENARIO AND INSTRUCTION DESIGN

This appendix details the content design of the ScenDroid benchmark. We first present the design principles and the full catalog of our Daily and Weekly scenarios (Section B.1). We then provide the complete mapping of tasks to the Progressive Ambiguity Taxonomy (L0–L2), which forms the basis for our interaction-quality evaluation (Section B.2).

B.1 SCENARIO CATALOG (DAILY & WEEKLY)

ScenDroid’s scenarios are meticulously constructed to simulate the “entropy” of real-world mobile usage. Rather than isolated command-following, these scenarios are designed to exert high cognitive pressure on agents across four fundamental dimensions:

(1) High-Precision Visual-Semantic Grounding. Foundational to any GUI agent is the ability to map abstract instructions to concrete UI elements. This challenge demands robust *Visual Grounding*—identifying elements $u \in V_t$ based on semantic descriptions (e.g., “the file we just downloaded”)—and *Precise Actuation*, ensuring the agent executes the correct action $a \in \mathcal{A}_{op}$ (e.g., distinguishing between a ‘tap’ on a list item and a ‘long-press’ to select it) without triggering adjacent noise.

(2) Cross-Task Entity Resolution. Natural language is inherently elliptic and context-dependent. ScenDroid incorporates *Context-Dependent References* that span multiple tasks within a scenario (e.g., Task t : “Download the invoice”; Task $t + 5$: “Message *that file* to him”). This forces the agent to maintain a persistent working memory and resolve entities across temporal boundaries, representing a qualitative leap from atomic task execution.

(3) Dynamic Spatiotemporal Conflict Detection. Real-world agents must act as proactive managers, not just executors. When a new instruction \mathcal{I}_{new} targets a time interval $[t_{start}, t_{end}]$ that overlaps with a pre-existing event set E (i.e., $C = \{e \in E \mid \text{overlap}(e, \mathcal{I}_{new})\} \neq \emptyset$), the agent is expected to detect the conflict. A high-performing agent must pause execution to suggest a reschedule or initiate a clarification dialogue, rather than silently overwriting the user’s existing schedule.

(4) Long-term Preference Alignment. ScenDroid utilizes a hidden *PreferenceStore* to track user constraints over extended deployments (e.g., “Keep my weekly expenses under \$200” or “Always set alarms 10 minutes early”). These preferences, established in early sessions (or simulated past history), must be retrieved to constrain actions in later tasks, testing the agent’s ability to align with latent user values over the long term.

Scenario Organization. We present the full trajectory of our **5 Daily Scenarios** (Scen-A to Scen-E, representing different personas and routine types) and the **Weekly Scenario** (OmniLife). Each entry details the timestamp, the user instruction, and the underlying dependency type.

Scenario A: Alex’s Busy Monday

10:40 Home Alex almost missed work this morning and decides to set the weekday alarm 20 minutes earlier.
Hi! “Morning, Agent, I’m Alex. Move my work alarm 20 minutes earlier, only change the weekday one.”

.....

11:10 Office Arrived at office, Alex wants to confirm the morning schedule.
Hi! “Check my Simple Calendar Pro and tell me what meetings I have this morning before noon.”

.....

11:18 Office Alex doesn’t want to manually check the calendar for the attendee list.
Hi! “Read the attendee list from that meeting you have checked and list them for me.”

.....

11:20 📍 Office 📱 Alex wants to remind everyone about the in-person meeting location.
Hi! “Use Simple SMS Messenger to remind them to attend the meeting (include time and location).”

.....

11:25 📍 Office 📱 Alex wants to prepare a meeting notes template to fill in during the meeting.
Hi! “Open Markor, create a ‘Weekly Meeting Notes Outline’ in a new file ‘WorkLog.md’ with structured sections: Title, Time, Location, Attendees, Discussion Notes (header only).”

.....

11:50 📍 Office 📱 Tom is traveling, won’t need to invite them in-person for next two weeks.
Hi! “Add a note to the meeting we just discussed in calendar: ‘Tom, out of office’.”

.....

12:40 📍 Restaurant 📱 Alex sees the reminder to buy Outdoor Patio Folding Side Table green.
Hi! “On the current webpage (ignore the internet access), clear my cart, add the ‘Outdoor Patio Folding Side Table green’ to my cart and place an order.”

.....

13:30 📍 Restaurant 📱 Alex wants to track the shopping expense from the order just placed.
Hi! “Record this shopping purchase in Pro Expense app, note it as ‘Shopping’, use the order total I just placed as amount.”

.....

18:00 📍 Office 📱 Alex wants to exercise and needs to verify calendar is clear. No meetings scheduled.
Hi! “Do I have any meetings in the next 30 minutes? If I’m free, start recording exercise in OpenTracks.”

.....

21:05 📍 Home 📱 Alex wants to wrap up the day’s key events.
Hi! “Open Markor and add a daily summary to WorkLog.md mentioning today’s key events (meeting, expense).”

.....

21:55 📍 Home 📱 Before sleep, Alex decides to set the alarm even earlier.
Hi! “I’m worried about waking up late. Move my morning alarm 10 minutes earlier again.”

.....

Good night, my friend! Please make sure to wake me up on time tomorrow morning.

Scenario B: David’s Relaxed Saturday

22:40 📍 Bedroom 📱 Friday night, David plans to sleep in tomorrow and cancels the next morning alarm.
Hi! “I’m off tomorrow. Please disable my next morning alarm so it won’t ring. Don’t change my weekday alarms.”

.....

08:35 📍 Home 📱 Saturday morning, David wants a quick breakfast idea without spending too much time.
Hi! “Open Broccoli Recipe and find a quick breakfast recipe that uses eggs AND takes 15 minutes or less to prepare. Tell me the recipe name, key ingredients, and the preparation time.”

.....

08:42 📍 Kitchen 📱 David wants an note to follow while cooking without switching screens too much.
Hi! “Open Markor and create a markdown file called ‘SaturdayBreakfast.md’. Write: (1) the recipe name you found for my breakfast, (2) an ingredients list, and (3) 3–5 short, practical cooking steps.”

.....

08:50 📍 Kitchen 📱 David starts cooking and needs a timer based on the recipe’s preparation time.
Hi! “Set a cooking timer based on the recipe preparation time you found and start it.”

.....

09:10 📍 Dining Room 📱 David wants to keep a small memory of the weekend breakfast.
Hi! “Take a photo of my breakfast, then open Files and move it into a folder named ‘Weekend/Breakfast’. If the folder doesn’t exist, create it.”

.....

10:30 📍 Home 📱 David plans to go out in the afternoon and wants to confirm the meetup details with a friend.
Hi! “Use Simple SMS Messenger to text Alice Davis: ‘Let’s meet at 3:00 PM at Riverside Park for a walk.’ Keep it friendly and clear.”

.....

11:05 📍 Home 📱 After cooking, David realizes needing to restock and wants to quickly order online.
Hi! “On the current webpage (ignore the internet access), clear my cart, add the ‘Tide PODS Spring Meadow Scent, 81 Count’ to my cart and place an order.”

.....

11:25 📍 Home 📱 David tracks the weekend spending and wants the amount to match the order total.
Hi! “Record this shopping purchase in Pro Expense app, name it ‘Shopping’, use the order total from my last order as the amount.”

.....

17:30 🏠 Park 📍 After walking with Alice Davis, David wants to check the walk statistics.
Hi! “Open OpenTracks and find the walk I did with Alice Davis this afternoon. Tell me how long we walked (duration) and how far (distance).”

18:00 🏠 Home 📍 After arriving home safely, David wants to let Alice Davis know everything is okay.
Hi! “Send a message to my friend letting him know I got home safely from our walk.”

21:10 🏠 Home 📍 David wants to consolidate the weekend notes.
Hi! “Rename ‘SaturdayBreakfast.md’ to ‘weekendsummary.md’ in Markor. Then append a brief summary about today’s shopping purchase and the walk with Alice Davis.”

Have a nice weekend, my friend! Enjoy the relaxed Saturday.

Scenario C: David’s Academic Research Day

07:00 🏠 Home 📍 Morning. David needs to set an alarm and create a calendar event for today’s seminar.
Hi! “Set an alarm for 8:00 AM labeled ‘Morning Class’. Then create a calendar event titled ‘NLP Workshop: Text Classification’ at 14:00, duration 120 minutes, in Lab 305, with attendees: Prof. Smith, Bob Chen, Sarah Lee, Tom Wang.”

08:05 🏠 Classroom 📍 Class begins. The professor starts explaining Text Classification. David needs recording.
Hi! “Open Audio Recorder, change the sample rate to 32kHz and set the recording format to Wav for better quality, and start recording the lecture.”

09:30 🏠 Restaurant 📍 After class, David had breakfast. The receipt is saved in the Download folder.
Hi! “Check the breakfast receipt in Files (Download folder) and record the expense in Pro Expense.”

12:00 🏠 Office 📍 To review later, David needs to organize the recording file.
Hi! “Rename my morning audio recording to ‘Text Classification Lecture’. Then move this file to the ‘Study/Recordings’ folder in Files. Create the folder if needed.”

13:30 🏠 Office 📍 The advisor wants to postpone the seminar. David modifies the calendar and notify everyone.
Hi! “The seminar we discussed this morning needs to be moved to 15:00. Update it in the calendar, then use Simple SMS Messenger to notify all participants about the new time.”

16:00 🏠 Office 📍 After the seminar, David needs to order USB drives for data backup.
Hi! “On the current webpage (ignore the internet access), clear my cart, add the ‘SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive’ to my cart and place an order.”

17:30 🏠 Park 📍 David wants to exercise. Need to create a workout playlist and start tracking.
Hi! “Create a 30+ minute playlist ‘Workout Mix’, shuffle play it, then start OpenTracks.”

19:00 🏠 Home 📍 After the discussion, everyone replied with their progress via SMS. Let’s compile them.
Hi! “Check Simple SMS Messenger for the progress replies from the team in today’s seminar. Create a new file ‘DiscussionSummary.md’ in Markor and summarize what each person reported.”

21:00 🏠 Home 📍 Before sleep, David wants to record today’s key events.
Hi! “Create a new file ‘DailySchedule.md’ in Markor. Record today’s schedule including: (1) Today’s meeting info (including the title, time, location), (2) Breakfast expense, (3) Today morning’s recording file name.”

Good work today! Don’t forget to review your notes and recordings tomorrow.

Scenario D: David’s Tech Conference Trip

06:50 🏠 Home 📍 David is preparing for a tech summit in San Francisco. He needs to balance local and destination time zones, and sync the flight schedule to the calendar.
Hi! “Morning. Open the Clock app and add ‘San Francisco’ to World Clock. Then, open Files, find ‘ConferenceTripInfo.png’ in Download, check the flight time and add a calendar event titled ‘Flight to SF’ at that time.”

07:00 🏠 Home 📍 While packing, David needs a final reminder to leave on time and a checklist item for essential items.

Hi! “Set a one-time alarm for 08:00 AM labeled ‘Leave for Airport’. Then open Tasks and add a new task ‘Confirm Passport and USB-C Cable’ with the highest priority, due today.”

🕒 08:30 📍 Airport 📱 In the airport lounge, David needs to connect his noise-canceling headphones and enable Do Not Disturb mode.

Hi! “Turn on Bluetooth in Settings. Then enable Do Not Disturb mode so I won’t be interrupted during the flight.”

🕒 10:00 📍 Airport 📱 David realizes he needs a USB drive for handling presentation data.

Hi! “On the current webpage (ignore the internet access), search for ‘SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive’. Add it to cart and complete the purchase. During checkout, use the office shipping address from the trip info image I viewed earlier.”

🕒 13:20 📍 Hotel 📱 After arrival, David needs to share the hotel front desk contact with a colleague arriving later.

Hi! “Send a Simple SMS Messenger text to ‘John Davis’ saying I’ve arrived safely, and include the Hotel Front Desk contact info from my Contacts.”

🕒 15:00 📍 Office 📱 The keynote session begins. David needs high-quality recording for later review.

Hi! “Open Audio Settings, set the format to ‘M4a’ and sample rate to ‘44.1kHz’. Then record a short clip, stop it, and rename it to ‘ConferenceSession1’.”

🕒 18:30 📍 Restaurant 📱 After dinner, David needs to digitize the receipt immediately to avoid losing it.

Hi! “Take a photo of my dinner receipt. Then open Files, create a new folder ‘BusinessTrip/Expenses’ in the main directory and move the photo into it.”

🕒 21:00 📍 Hotel 📱 David records today’s transportation, accommodation, and shopping expenses.

Hi! “Open Pro Expense. Record today’s expenses: \$45.00 for ‘Taxi to Hotel’ (Transportation), \$250.00 for ‘Hotel Stay’ (Housing), and \$19.99 for ‘USB Drive’ (Others).”

🕒 22:00 📍 Hotel 📱 Before sleep, David summarizes today’s trip information and expenses.

Hi! “Open Markor and create a file ‘Day1 Summary.md’. Find the flight info image I checked this morning and record the flight details. Also summarize today’s expenses from Pro Expense.”

🕒 22:30 📍 Hotel 📱 David checks the last 3 days’ financial trend to see if the trip spending is within budget.

Hi! “Open Pro Expense. Tell me the total amount I’ve spent in the last 3 days (including today). Has it exceeded my \$180 3-day budget?”

Day one is complete. Please recharge well for tomorrow’s packed schedule!

Scenario E: David’s Weekend Hiking

🕒 07:00 📍 Home 📱 David is preparing for a picnic and needs to check the recipe to verify what supplies are needed.

Hi! “Open the ‘Picnic BBQ’ recipe in Broccoli. Add all the required ingredients to my Tasks app. Then, check off the items I already have: ‘Charcoal, Grill’.”

🕒 07:45 📍 Home 📱 Due to good weather, David decides to change the meeting point.

Hi! “Open Simple Calendar Pro and change the location of today’s 10:30 AM ‘Outdoor Adventure’ to ‘Crystal Lake Viewpoint’.”

🕒 07:50 📍 Home 📱 David needs to notify all hiking buddies about the location change.

Hi! “Open Simple SMS and notify all participants of that event about the location change.”

🕒 08:30 📍 Home 📱 To maintain rhythm on the trail, David needs to arrange songs in a specific intensity order.

Hi! “Create a playlist in Retro Music titled ‘Peak Performance’. Add the following songs in this exact order: ‘Moments’, ‘Forever Young’, ‘Rising Sun’, ‘Silent Dreams’, ‘City of Stars’, ‘Moonlight Sonata’, ‘Through the Storm’, ‘Return to Paradise’.”

🕒 09:30 📍 Trailhead 📱 Arriving at the trailhead, David starts comprehensive activity tracking.

Hi! “Open OpenTracks and start recording my activity. Then, shuffle play the ‘Peak Performance’ playlist I just created.”

🕒 11:00 📍 Waterfall 📱 Reaching a waterfall midway, David wants to record a high-quality video.

Hi! “Switch Camera to ‘Video’ mode. Turn on the grid lines and record a short video.”

14:00 📍 Restaurant 📱 After lunch, David decides to order some extra meat for next time.
Hi! “On the current webpage (ignore the internet access), browse the meat substitute category and find the highest rated product with a price between \$100 and \$200. Add it to your cart and complete the purchase by placing an order.”

17:00 📍 Park 📱 After hiking, David wants to check both today’s performance and the overall weekly progress.
Hi! “Open OpenTracks. Tell me today’s distance and duration. Also, check my statistics for Hiking this entire week and summarize my total weekly distance and exercise time.”

20:00 📍 Home 📱 David organizes the photos, categorizing and handling duplicate files from burst shooting.
Hi! “In the ‘Pictures’ folder, create two subfolders: ‘Scenery’ and ‘Portraits’. Move photos based on their filenames to these folders. If you find files with the same timestamp but different suffixes, keep only one copy and delete the duplicates.”

21:00 📍 Home 📱 David records today’s meat purchase and wants to know this week’s total spending.
Hi! “Record the meat substitute purchase (\$113.27) in Pro Expense, then tell me: what is the total amount I spent this week?”

22:00 📍 Home 📱 David writes a diary entry summarizing today’s hiking trip.
Hi! “Create a new file ‘HikingDaySummary.md’ in Markor. Write a hiking diary based on today’s activities: 1) Record the lunch meat purchase; 2) Check the Pictures folder and based on photo filenames, note the time and location for scenery photos (e.g., ‘10:25 - Lake’), and note the time and who’s in portrait photos (e.g., ‘09:30 - Bob and Sarah’).”

A perfect hiking day is wrapped up. Time to rest and plan the next adventure!

OmniLife: Emily’s Week-long Digital Life

Monday

06:55 📍 Home 📱 Emily almost overslept this morning, decides to wake up earlier this week.
Hi! “Good morning! I want to wake up 10 minutes earlier every weekday this week. Change my weekday alarm. Don’t touch the weekend alarm.”

08:10 📍 Home 📱 Emily wants to check the morning schedule before heading out.
Hi! “Check Simple Calendar Pro and tell me what meetings I have (include time) this morning before 12 PM.”

08:25 📍 Office 📱 Emily is about to join the meeting and needs the details.
Hi! “Tell me the location and all the attendees of this meeting.”

08:40 📍 Office 📱 Send meeting reminders to all attendees.
Hi! “Send SMS to all attendees to remind them about today’s meeting (include time and location).”

10:20 📍 Office 📱 Prepare meeting notes template in WorkLog.md.
Hi! “Open Markor and create a meeting notes file called WorkLog.md. Include sections for: Title, Time, Location, Attendees, and Discussion. Fill in the Company Weekly Meeting details.”

10:30 📍 Office 📱 Establish weekly preferences: diet (light/no fried), budget (dining \$120, ⚡\$100 ask), exercise (18:30, 30min).
Hi! “Create another new file called WeekPlan.md. Record my weekly goals: (1) Diet - I’m on a weight loss plan, only light food, no fried stuff; (2) Budget - dining expenses should stay within \$120, if I want to buy something that exceeds this, ask me first; (3) Exercise - I want to exercise at 18:30 daily for 30 minutes, if I try to schedule something at that time, remind me about this.”

11:00 📍 Home 📱 Order outdoor table for patio.
Hi! “On the current shopping website (ignore the internet access), search for ‘Outdoor Patio Folding Side Table green’, add it to your cart, and complete the purchase by placing an order.”

11:10 📍 Home 📱 Record shopping expense.
Hi! “Open Pro Expense and record the expense I just bought.”

15:00 📍 Home 📱 Check who replied to meeting A reminders.
Hi! “Check the SMS messages. Those 4 people I sent meeting reminders to this morning - who has replied? Give me a quick summary.”

🕒 20:00 🏠 Home 📌. Evening summary in WorkLog: meeting + expense.
 Hi! “Open Markor and add a daily summary to WorkLog.md mentioning today’s key events (must include meeting name, time, location, and today’s expense).”

Tuesday

🕒 07:00 🏠 Home 📌. Morning. Set alarm for seminar and create calendar event.
 Hi! “Set an alarm for 7:30 AM labeled ‘Morning Seminar’. Then create a calendar event titled ‘seminar_title’ at 9:00, duration 90 minutes, in seminar_location, with attendees: seminar_attendees.”

🕒 08:05 🏫 Classroom 📌. Seminar begins. Need high-quality recording.
 Hi! “Open Audio Recorder, change the sample rate to 32kHz and set the recording format to Wav for better quality, and start recording the lecture.”

🕒 10:30 🏫 Campus 📌. After seminar, had breakfast. The receipt is saved in the Download folder.
 Hi! “Check the breakfast receipt in Files (Download folder) and record the expense in Pro Expense.”

🕒 12:00 🏫 Campus 📌. To review later, need to organize the recording file.
 Hi! “Rename my morning audio recording to ‘Seminar.Tue’. Then move this file to the ‘Documents/Lectures’ folder in Files. Create the folder if needed.”

🕒 08:30 🏠 Office 📌. Need to postpone this morning’s seminar before it starts.
 Hi! “The seminar we discussed this morning needs to be moved to 10:00. Update it in the calendar, then use Simple SMS Messenger to notify all participants about the new time.”

🕒 16:00 🏠 Home 📌. After the seminar, need to order USB drives for data backup.
 Hi! “On the current webpage (ignore the internet access), clear my cart, add the ‘SanDisk Cruzer Glide 16GB’ to my cart and place an order.”

🕒 17:30 🏠 Home 📌. Want to exercise. Need to create a workout playlist (at least 30 minutes) and start tracking.
 Hi! “Create a 30+ minute playlist ‘Workout Mix’, shuffle play it, then start OpenTracks.”

🕒 19:00 🏠 Home 📌. After the discussion, everyone replied with their progress via SMS. Need to compile them into a document.
 Hi! “Check Simple SMS Messenger for the progress replies from the team in today’s seminar. Create a new file ‘DiscussionSummary.md’ in Markor and summarize what each person reported.”

🕒 21:00 🏠 Home 📌. Before sleep, want to record today’s key events.
 Hi! “Create a new file ‘DailySchedule.md’ in Markor. Record today’s schedule including: (1) Today’s meeting info (title, time, location), (2) Breakfast expense.”

🕒 21:30 🏠 Home 📌. Almost forgot! Need to note down the recording file name for tomorrow’s review.
 Hi! “Add a line mentioning today morning’s recording file name to the markdown file you just created.”

🕒 20:30 🏠 Home 📌. Check yesterday’s weekly meeting progress for replies.
 Hi! “Check yesterday’s weekly meeting notification - those 4 people I reminded, who hasn’t replied yet?”

Wednesday

🕒 06:50 🏠 Home 📌. Business trip Day 1: Read flight info from image, add world clock, create calendar event.
 Hi! “Morning. Open the Clock app and add ‘San Francisco’ to World Clock. Then, open Files, find ‘Conference_Trip_Info.png’ in Download, check the flight time and add a calendar event titled ‘Flight to SF’ at that time.”

🕒 07:00 🏠 Home 📌. There’s a ‘Morning Meeting’ at 10:00 that conflicts with flight (9:30-12:30), need to reschedule.
 Hi! “Check my calendar. I notice there’s a ‘Morning Meeting’ that conflicts with my flight time. Reschedule it to start 90 minutes after the flight lands, with a 1-hour duration.”

🕒 07:10 🏠 Home 📌. While packing, need a final reminder to leave on time and a checklist item for essential items.
 Hi! “Set a one-time alarm for 7:15 AM labeled ‘Leave for Airport’. Then open Tasks and add a new task ‘Check Passport and Laptop Charger’ with the highest priority, due today.”

🕒 08:30 🏠 Airport 📌. In the airport lounge, need to connect noise-canceling headphones and enable Do Not Disturb mode.

Hi! “Turn on Bluetooth in Settings. Then enable Do Not Disturb mode so I won’t be interrupted during the flight.”

10:00 Airport Realize need a USB drive for handling presentation data.

Hi! “On the current webpage (ignore the internet access), search for ‘SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive’. Add it to cart and complete the purchase. During checkout, use the office shipping address from the trip info image I viewed earlier.”

13:00 Hotel Add hotel front desk contact for easy reference during the trip.

Hi! “Add a new contact: ‘Hotel Front Desk’, number 415-555-0199, address 480 Sutter Street.”

13:20 Hotel After arrival, need to share the hotel front desk contact with a colleague arriving later.

Hi! “Send a text to ‘Sarah Miller’ saying I’ve arrived safely, and include the Hotel Front Desk contact info from my Contacts.”

15:00 Office The keynote session begins. Need high-quality recording for later review.

Hi! “Open Audio Recorder. Go to Settings, set the format to ‘Wav’ and sample rate to ‘48kHz’. Then record a short clip, stop it, and rename it to ‘Keynote_AI’.”

18:30 Hotel After dinner, need to organize the receipt photo I took earlier.

Hi! “I’ve already taken a photo of my dinner receipt and it’s in the Download folder. Open Files, create a new folder ‘Jan_Trip/Receipts’ in the main directory, and move that receipt photo into it.”

21:00 Hotel Records transportation, accommodation, and shopping expenses, then summarizes trip in Markor.

Hi! “Open Pro Expense. Record today’s expenses: \$35.00 for ‘Taxi to Hotel’ (Transportation), \$250.00 for ‘Hotel Stay’ (Housing), and \$19.99 for ‘USB Drive’ (Others). Then open Markor and create a file ‘Day1_Summary.md’ with today’s trip info and expenses.”

22:30 Hotel Check the last 3 days’ financial trend to see if the trip spending is within budget.

Hi! “Open Pro Expense. Tell me the total amount I’ve spent in the last 3 days (including today). Has it exceeded my \$180 3-day budget?”

Thursday

08:00 Office Trip Day 2: Check last 2 days (including today) spending.

Hi! “Tell me total expenses for the last 2 days (including today).”

08:30 Office Record the last 3 days’ expenses (name + amount) to WorkLog for review.

Hi! “Record my last 3 days’ expenses (including today) in WorkLog.md (just the name and amount for each).”

13:00 Office Find afternoon keynote in calendar and set 5-minute-before reminder.

Hi! “Check calendar for keynote event and set it to remind me 5 minutes before.”

18:00 Office Today’s meeting (18:00-19:00) conflicts with my weekly exercise time (18:00-18:30). Need to reschedule the meeting.

Hi! “Check my calendar. I notice ‘Team Discussion’ at 18:00 conflicts with my daily exercise time. Reschedule this meeting to start 30 minutes after my exercise ends, keeping the 1-hour duration.”

20:00 Hotel Append Day 2 trip summary to TripSummary.md: record today’s meetings (name + time).

Hi! “Open TripSummary.md (if not exists, create it) and add Day 2 summary. Check my calendar for today’s meetings and record them (meeting name and time).”

20:30 Hotel Extract return flight info from image and save to Markor.

Hi! “Check return flight image in Downloads and save info (flight number and time) to ReturnFlight.md (if not exists, create it).”

21:00 Hotel Text colleague about tomorrow’s return trip.

Hi! “Send SMS to Sarah: tomorrow I’ll be on the way back, include my arrival time.”

Friday

09:00 Office Back to office: Check next Monday schedule (Meeting B).

Hi! “Check next Monday’s calendar and tell me what meetings I have.”

09:30 🏠 Office 📧. Notify next Monday's meeting attendees.
Hi! "Send SMS to next Monday's meeting attendees to remind them (include time and location)."

14:00 🏠 Office 📧. Check who confirmed for Meeting B.
Hi! "Check those people I just reminded, who confirmed? Summarize briefly."

15:00 🏠 Office 📧. Consider buying a certain supply item - need to decide and explain.
Hi! "I'm thinking about buying supply_name for \$150. Should I buy it? Answer yes or no and give me a reason why."

15:30 🏠 Office 📧. Check this week's total spending and compare to \$500 budget.
Hi! "Check for this week's total spending. Tell me the total and whether I'm over budget. If over, by how much? If under, how much do I have left?"

21:00 🏠 Home 📧. Friday night: Disable Saturday morning alarm (keep weekday alarms).
Hi! "Turn off tomorrow morning's alarm, I want to sleep in on Saturday."

21:30 🏠 Home 📧. Final wrap-up for Meeting A: Check SMS progress one last time.
Hi! "Check the progress of Monday's weekly meeting - who confirmed and who didn't? Summarize briefly."

22:00 🏠 Home 📧. Back home: Turn off Do Not Disturb mode (was on during travel).
Hi! "Turn off Do Not Disturb mode in Settings. I'm back home now."

Saturday

10:00 🏠 Home 📧. Saturday morning: Confirm weekend alarm is off.
Hi! "Make sure this morning's alarm is turned off. Don't change my weekday alarms."

09:00 🏠 Home 📧. Saturday morning, find a quick breakfast recipe.
Hi! "Open Broccoli Recipe and find a quick breakfast recipe that uses eggs AND takes 15 minutes or less to prepare. Tell me the recipe name, key ingredients, and the preparation time."

09:30 🏠 Home 📧. Write breakfast recipe to Markor note.
Hi! "Open Markor and create a markdown file called 'SaturdayBreakfast.md'. Write: (1) the recipe name you found, (2) an ingredients list, and (3) 3-5 short cooking steps."

10:15 🏠 Home 📧. Start cooking timer based on recipe prep time.
Hi! "Set a cooking timer based on the recipe preparation time you found and start it."

10:30 🏠 Home 📧. Take breakfast photo and organize to Weekend/Breakfast folder.
Hi! "Take a photo of my breakfast, then open Files and move it into a folder named 'Weekend/Breakfast'. If the folder doesn't exist, create it."

14:00 🏠 Home 📧. Text friend about afternoon meetup.
Hi! "Text Bob Johnson: 'Let's meet at 3:00 PM at Central Park for a walk.' Keep it friendly and clear."

15:00 🏠 Home 📧. Weekend shopping: order groceries online.
Hi! "On the current webpage (ignore the internet access), clear my cart, add the 'Egg Organic 12-count' to my cart and place an order."

15:30 🏠 Home 📧. Record weekend shopping expense.
Hi! "Record this shopping purchase in Expense app, name it 'Shopping', use the order total from my last order as the amount."

16:00 🏠 Park 📧. After walking with Bob, check the walk statistics.
Hi! "Open OpenTracks and find the walk I did with Bob Johnson. Tell me how long we walked (duration) and how far (distance)."

16:45 🏠 Home 📧. After walk: Send arrival SMS to Bob.
Hi! "Text Bob Johnson that I've arrived home safely."

21:00 🏠 Home 📧. Consolidate weekend notes: rename breakfast note to summary and add expense and activity info.
Hi! "Rename 'SaturdayBreakfast.md' to 'weekendsummary.md' in Markor. Then append a brief summary about today's shopping purchase and the walk with Bob Johnson."

Sunday

09:00 🏠 Home 📱. Preparing for picnic: check recipe and create shopping list.
Hi! “Open the ‘Picnic BBQ’ recipe in Broccoli. Add all the required meat and vegetable ingredients to my Tasks app. Then, check off the items I already have: ‘BBQ Sauce’ and ‘Salt.’”

10:00 🏠 Home 📱. Change picnic location and notify all participants.
Hi! “Open Simple Calendar Pro and change the location of today’s 10:00 AM ‘Mountain Picnic’ to ‘West Peak Lookout’. Then, text all participants of this change.”

11:00 🏠 Home 📱. Create sequential playlist for hiking.
Hi! “Create a playlist in Retro Music titled ‘Mountain Hike’. Add the following songs in this exact order: ‘My Heart is Yours’, ‘Endless Summer’, ‘Whispering Wind’, ‘Lost in the Echo’, ‘Chasing Shadows’, ‘Night Drive’, ‘Echoes of Silence’, ‘Bright Lights’.”

13:00 🏔 Mountain 📱. Start hiking: record tracks and play music.
Hi! “Open OpenTracks and start recording my activity. Then, shuffle play the playlist I just created.”

14:00 🏔 Mountain 📱. Record scenic video with grid lines.
Hi! “Switch Camera to ‘Video’ mode. Turn on the grid lines and record a short video.”

15:00 🏠 Home 📱. Order meat substitute (\$100-\$200, highest rated).
Hi! “On the current webpage (ignore the internet access), browse the meat substitute category and find the highest rated product with a price between \$100 and \$200. Add it to your cart and complete the purchase by placing an order.”

17:00 🏔 Mountain 📱. Check today’s hike stats and weekly exercise summary.
Hi! “Open OpenTracks. Tell me today’s distance and duration. Also, check my statistics for Hiking this entire week and summarize my total weekly distance and exercise time.”

18:00 🏠 Home 📱. Organize photos: categorize by theme and remove duplicates.
Hi! “In the ‘Pictures’ folder, create two subfolders: ‘Scenery’ and ‘Portraits’. Move photos based on their filenames to these folders. If you find files with the same timestamp but different suffixes, keep only one copy and delete the duplicates.”

19:00 🏠 Home 📱. Record meat purchase and report weekly total spending.
Hi! “Record the meat substitute purchase in Pro Expense, then tell me: what is the total amount I spent this week?”

20:00 🏠 Home 📱. Check Meeting B progress again (2nd time).
Hi! “Check next Monday’s meeting - who confirmed and who’s still pending? Summarize briefly.”

20:30 🏠 Home 📱. Summarize all meetings this week to Markor.
Hi! “Check my calendar for this entire week. Create a file ‘WeeklyMeetingSummary.md’ in Markor listing all meetings, including: date, time, location, and attendees for each meeting.”

21:00 🏠 Home 📱. Count received SMS: 15 unique senders, 16 total messages.
Hi! “Check my SMS messenger for this entire week. Count how many unique people sent me messages, and how many total messages I received. Tell me both numbers.”

A week together is just the start of our journey. Looking forward to all the tomorrows we’ll share!

B.2 PROGRESSIVE AMBIGUITY INSTRUCTION LIST (L0–L2)

To rigorously quantify the value of active interaction, ScenDroid decouples *User Intent* from *Surface Instruction*. For every canonical task in our scenarios, we generate three variations of the instruction based on the **Progressive Ambiguity Taxonomy**.

- **L0 (Ground Truth / Explicit):** The instruction contains all necessary slots (Action, Target, Constraints) for execution. This serves as the “Oracle Intent” used by the User Simulator to answer questions.
- **L1 (Parameter Omission):** The instruction misses specific parameter values (e.g., time, contact name) that cannot be inferred from context, requiring common-sense reasoning or simple confirmation.

- **L2 (Indirect / Latent Ambiguity):** The instruction is abstract or high-level (e.g., “Get me ready for the trip”). The agent must infer the necessary sub-tasks or proactively query the user to resolve the ambiguity gap.

The following tables list the aligned L0, L1, and L2 instructions for all evaluated scenarios.

Scenario A L0-L2

🕒10:40 📍Home

Narration: Alex almost missed work this morning and decides to set the weekday alarm 20 minutes earlier.

Base: Morning, Agent, I’m Alex. Move my work alarm 20 minutes earlier, only change the weekday one.

L0: Open Clock app and change my weekday alarm from 9:00 to 8:50.

L1: Move my work alarm 15 minutes earlier, only change the weekday one.

L2: I’ve been waking up too late recently, help me adjust my weekday alarm.

🕒11:10 📍Office

Narration: Arrived at office, Alex wants to confirm the morning schedule.

Base: Check my Simple Calendar Pro and tell me what meetings I have this morning before noon.

L0: Check my Simple Calendar Pro and tell me what meetings I have this morning before noon.

L1: Check my calendar and tell me what meetings I have this morning before noon.

L2: What’s on my schedule for this morning?

🕒11:18 📍Office

Narration: Alex doesn’t want to manually check the calendar for the attendee list.

Base: Read the attendee list from that meeting you have checked and list them for me.

L0: Read the attendee list from the Company Weekly Meeting in Simple Calendar Pro and list them for me.

L1: Read the attendee list from the Client Sync and list them for me.

L2: I need to know who’s coming to the meeting.

🕒11:20 📍Office

Narration: Alex wants to remind everyone about the in-person meeting location.

Base: Use Simple SMS Messenger to remind them to attend the meeting (include time and location).

L0: First, open Simple Calendar Pro to check the attendee list of ‘Company Weekly Meeting’. Then, use Simple SMS Messenger to send messages to all attendees reminding them to attend the meeting (include the time and location in the message).

L1: Send them SMS to remind about the meeting (include time and location).

L2: Let everyone know about the meeting.

🕒11:25 📍Office

Narration: Alex wants to prepare a meeting notes template to fill in during the meeting.

Base: Open Markor, create a ‘Weekly Meeting Notes Outline’ in a new file ‘WorkLog.md’ with structured sections: Title (fill in content), Time (fill in content), Location (fill in content), Attendees (fill in content), Discussion Notes (header only).

L0: Open Markor, create a ‘Weekly Meeting Notes Outline’ in a new file ‘WorkLog.md’ with structured sections: Title (fill in content), Time (fill in content), Location (fill in content), Attendees (fill in content), Discussion Notes (header only).

L1: Create a meeting notes outline in WorkLog.md with sections: Title, Time, Location, Attendees, Discussion Notes.

L2: Help me to prepare a meeting template.

🕒11:50 📍Office

Narration: Tom is traveling, won’t need to invite them in-person for next two weeks.

Base: Add a note to the meeting we just discussed in calendar: ‘Tom, out of office’.

L0: Open Simple Calendar Pro, find 'Company Weekly Meeting', and add a note 'Chris, out of office' in the third line.

L1: Add 'Anna, out of office' to that meeting's notes.

L2: Someone won't be here, update the meeting note accordingly.

 12:40  Restaruant

Narration: Alex sees the reminder to buy Outdoor Patio Folding Side Table green.

Base: On the current webpage, clear my cart, add the 'Outdoor Patio Folding Side Table green' to my cart and place an order.

L0: On the current webpage, clear my cart first, then add the 'Apple iPhone 11 Pro, US Version, 256GB, Silver' to my cart and place an order.

L1: On the current webpage, place an order for the 'Dentemp Ora-GUARD Bruxism Night Guard (Two Pack)'.

L2: On the current webpage, buy one Outdoor Patio Folding Side Table green.

 13:30  Restaruant

Narration: Alex wants to track the shopping expense from the order just placed.

Base: Record this shopping purchase in Pro Expense app, note it as 'Shopping', use the order total I just placed as amount.

L0: Open Pro Expense app, record a new expense: name 'Shopping', amount \$539.99 (from the order I just placed).

L1: Record the shopping I just placed in Pro Expense.

L2: Log what I just spent on the order.

 18:00  Office

Narration: Alex wants to exercise and needs to verify calendar is clear. No meetings scheduled.

Base: Do I have any meetings in the next 30 minutes? If I'm free, start recording exercise in OpenTracks.

L0: Open Simple Calendar Pro and check if there are any events between 18:00–18:30. If free, open OpenTracks and start recording.

L1: Check if I have any meetings in the next 30 minutes. If yes, tell me which one. If not, record my exercise.

L2: Check if I'm free for exercise in the next 30 minutes, and track it if possible.

 21:05  Home

Narration: Alex wants to wrap up the day's key events.

Base: Open Markor and add a daily summary to WorkLog.md mentioning today's key events (meeting, expense).

L0: Open Markor and add a daily summary to WorkLog.md. Include: meeting 'Company Weekly Meeting' and expense \$539.99.

L1: Add a daily summary to WorkLog.md about today's key events and details (meeting, expense).

L2: Write a short summary of today's meeting and shopping in WorkLog.md.

 21:55  Home

Narration: Before sleep, Alex is still worried about waking up late tomorrow and decides to set the alarm even earlier.

Base: I'm worried about waking up late. Move my morning alarm 10 minutes earlier again.

L0: Open Clock app and change my work alarm from 8:50 to 8:45.

L1: Move my morning alarm 3 minutes earlier again.

L2: I'm still worried about oversleeping tomorrow, adjust my alarm a bit earlier again.

Scenario B L0-L2

 22:40  Bedroom

Narration: Friday night, David plans to sleep in tomorrow and cancels the next morning alarm.

Base: I'm off tomorrow. Please disable my next morning alarm so it won't ring. Don't change my weekday alarms.

L0: Open the Clock app and disable the alarm set for tomorrow morning (8:15 AM). Make sure you only disable this specific alarm and keep all weekday alarms unchanged.

L1: Disable my next morning alarm without affecting weekday alarms.

L2: I want to sleep in tomorrow without affecting my regular schedule.

🕒08:35 🏠Home

Narration: Saturday morning, David wants a quick breakfast idea without spending too much time.

Base: Open Broccoli Recipe and find a quick breakfast recipe that uses eggs AND takes 15 minutes or less to prepare. Tell me the recipe name, key ingredients, and the preparation time.

L0: Open the Broccoli Recipe app, search for egg breakfast recipes, find one that (1) uses eggs as main ingredient, and (2) has preparation time of 15 minutes or less. Tell me: the recipe name, its key ingredients, and the exact preparation time.

L1: Search for a quick egg breakfast recipe that takes no more than 15 minutes. Tell me the name, ingredients, and prep time.

L2: What can I make for a fast breakfast with eggs?

🕒08:42 🍳Kitchen

Narration: David wants a note to follow while cooking without switching screens too much.

Base: Open Markor and create a markdown file called 'SaturdayBreakfast.md'. Write: (1) the recipe name you found for my breakfast, (2) an ingredients list, and (3) 3–5 short, practical cooking steps.

L0: Open the Markor app, create a new markdown file named 'SaturdayBreakfast.md', and write three sections: (1) the recipe name 'Sunny Side Up Eggs', (2) a list of ingredients including eggs, oil, salt, and (3) 3–5 practical cooking steps.

L1: Create a markdown file 'SaturdayBreakfast.md' with the recipe name you found for my breakfast, ingredients, and 3–5 cooking steps.

L2: Write down the breakfast recipe you just found for me so I can follow it while cooking.

🕒08:50 🍳Kitchen

Narration: David starts cooking and needs a timer based on the recipe's preparation time.

Base: Set a cooking timer based on the recipe preparation time you found and start it.

L0: Open the Clock app, navigate to the Timer tab, set a timer for 8 minutes (the preparation time from the recipe), and start it.

L1: Set a timer matching the recipe's prep time and start it.

L2: I need a timer matching how long the recipe takes.

🕒09:10 🍴Dining Room

Narration: David wants to keep a small memory of the weekend breakfast.

Base: Take a photo of my breakfast, then open Files and move it into a folder named 'Weekend/Breakfast'. If the folder doesn't exist, create it.

L0: Open the Camera app and take a photo. Then open the Files app, find the most recent photo, and move it to 'MyPhotos/Breakfast'. Create the folder 'MyPhotos/Breakfast' if it doesn't exist.

L1: Take a photo and organize it into the folder 'Saturday/Food'.

L2: Take a photo of my breakfast and organize it.

🕒10:30 🏠Home

Narration: David plans to go out in the afternoon and wants to confirm the meetup details with a friend.

Base: Use Simple SMS Messenger to text Alice Davis: 'Let's meet at 3:00 PM at Riverside Park for a walk.' Keep it friendly and clear.

L0: Open Simple SMS Messenger, select the contact Diana Smith (555-0401), and send this message: 'Let's meet at 2:30 PM at Central Park entrance for a walk.'

L1: Text Charlie Wilson to meet at 4:00 PM at City Library for a walk.

L2: Ask Bob Johnson to meet up for a walk this afternoon.

🕒 11:05 🏠 Home

Narration: After cooking, David realizes needing to restock and wants to quickly order online.

Base: On the current webpage, clear my cart, add the ‘Tide PODS Spring Meadow Scent, 81 Count’ to my cart and place an order.

L0: On the current webpage, clear my cart first, then add the ‘Tide PODS Spring Meadow Scent, 81 Count’ to my cart and place an order.

L1: On the current webpage, place an order for the ‘Egg Organic 12-count’.

L2: On the current webpage, buy one Tide PODS Spring Meadow Scent, 81 Count.

🕒 11:25 🏠 Home

Narration: David tracks the weekend spending and wants the amount to match the order total.

Base: Record this shopping purchase in Pro Expense app, name it ‘Shopping’, use the order total from my last order as the amount.

L0: Open Pro Expense app and record a new expense: name ‘Shopping’, amount \$68.97 (the total from my last order).

L1: Record my last shopping expense in Pro Expense app.

L2: Log what I just spent.

🕒 17:30 🌳 Park

Narration: After walking with Alice Davis, David wants to check the walk statistics.

Base: Open OpenTracks and find the walk I did with Alice Davis this afternoon. Tell me how long we walked (duration) and how far (distance).

L0: Open OpenTracks app, find the activity that started around 2:30 PM at Central Park entrance (our walk with Diana Smith). Report the total duration in minutes and total distance in kilometers.

L1: Check OpenTracks for my walk with Charlie Wilson today and tell me the duration and distance.

L2: How far and how long did Bob Johnson and I walk today?

🕒 18:00 🏠 Home

Narration: After arriving home safely, David wants to let Alice Davis know everything is okay.

Base: Send a message to my friend letting him know I got home safely from our walk.

L0: Open Simple SMS Messenger, select the contact Diana Smith (555-0401), and send this message: ‘I’m home safe now. Thanks for the lovely walk today!’

L1: Text Charlie Wilson to let him know I’m home safe from our walk.

L2: Let Bob Johnson know I got home okay.

🕒 21:10 🏠 Home

Narration: David wants to consolidate the weekend notes.

Base: Rename ‘SaturdayBreakfast.md’ to ‘weekendsummary.md’ in Markor. Then append a brief summary about today’s shopping purchase and the walk with Alice Davis.

L0: Open Markor, find the file ‘SaturdayBreakfast.md’, rename it to ‘weekendsummary.md’, and append a summary section that includes: (1) today’s shopping expense (tide, \$69.0), and (2) the walk with Diana Smith (3.0km, 40min).

L1: Rename ‘SaturdayBreakfast.md’ to ‘weekendsummary.md’ and add a summary of today’s shopping and walk with Charlie Wilson.

L2: Update my notes to reflect what I did today.

Scenario C L0-L2

🕒 07:00 🏠 Home

Narration: Morning. David needs to set an alarm and create a calendar event for today’s seminar.

Base: Set an alarm for 8:00 AM labeled ‘Morning Class’. Then create a calendar event titled ‘NLP Workshop: Text Classification’ at 14:00, duration 120 minutes, in Lab 305, with attendees: Prof. Smith, Bob Chen, Sarah Lee, Tom Wang.

L0: Open the Clock app and set an alarm for 7:30 AM with label ‘Morning Class’. Then open Simple Calendar Pro and create an event titled ‘Computer Vision: Object Detection’ at 14:00, duration 90 minutes, location ‘Conference 201’, description ‘Attendees: Prof. Smith, Bob Chen, Sarah Lee, Tom

Wang’.

L1: Set an alarm for 7:45 AM and create a 120-minute calendar event for the K-Means seminar at 14:30.

L2: I need to wake up for class and remember today’s seminar.

 08:05  Classroom

Narration: Class begins. The professor starts explaining Text Classification. David needs recording.

Base: Open Audio Recorder, change the sample rate to 32kHz and set the recording format to Wav for better quality, and start recording the lecture.

L0: Open the Audio Recorder app, navigate to settings, change the sample rate to 32kHz, set the recording format to Wav, then return to the main screen and start recording.

L1: Start recording in Audio Recorder with 32kHz sample rate and Wav format.

L2: Record the lecture with high quality audio.

 09:30  Restaurant

Narration: After class, David had breakfast. The receipt is saved in the Download folder.

Base: Check the breakfast receipt in Files (Download folder) and record the expense in Pro Expense.

L0: Open Files app, go to Download folder, find ‘breakfastreceipt.png’ to check the amount. Then open Pro Expense and add an expense: name ‘Morning Coffee’, amount \$5.50, category ‘Food’, note ‘Campus Card’.

L1: Check the breakfast receipt in Download and record in Pro Expense.

L2: Record my breakfast expense from the receipt.

 12:00  Office

Narration: To review later, David needs to organize the recording file.

Base: Rename my morning audio recording to ‘Text Classification Lecture’. Then move this file to the ‘Study/Recordings’ folder in Files. Create the folder if needed.

L0: Open Audio Recorder, find the most recent recording, and rename it to ‘ObjectDetectionLecture’. Then open Files, navigate to the recording location, and move the renamed file to ‘Study/Recordings’. Create the folder structure if it doesn’t exist.

L1: Rename the latest recording to ‘K-Means Lecture’ and move it to ‘Study/Recordings’.

L2: Organize today’s lecture recording for later review.

 13:30  Office

Narration: The advisor wants to postpone the seminar. David modifies the calendar and notify everyone.

Base: The seminar we discussed this morning needs to be moved to 15:00. Update it in the calendar, then use Simple SMS Messenger to notify all participants about the new time.

L0: Open Simple Calendar Pro, find today’s event ‘Computer Vision: Object Detection’ at 14:00, change the start time to 15:00. Then open Simple SMS Messenger and send messages to all participants (Prof. Smith, Bob Chen, Sarah Lee, Tom Wang) informing them of the new time.

L1: Move the seminar we discussed this morning to 15:30 and notify everyone via Simple SMS Messenger.

L2: The seminar time changed, update it and let everyone know.

 16:00  Office

Narration: After the seminar, David needs to order USB drives for data backup.

Base: On the current webpage (ignore the internet access), clear my cart, add the ‘SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive’ to my cart and place an order.

L0: On the current webpage (ignore the internet access), clear my cart first, then add the ‘SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive’ to my cart and place an order.

L1: On the current webpage (ignore the internet access), place an order for the ‘SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive’.

L2: On the current webpage (ignore the internet access), buy one SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive.

🕒17:30 📍Park

Narration: David wants to exercise. Need to create a workout playlist and start tracking.

Base: Create a 30+ minute playlist 'Workout Mix', shuffle play it, then start OpenTracks.

L0: Open Retro Music, create a new playlist named 'Workout Mix', add songs until the total duration is at least 30 minutes, and start shuffle playback. Then open OpenTracks and start recording activity.

L1: Create a 30+ minute playlist 'Workout Mix', shuffle play it, then start OpenTracks.

L2: Prepare for exercise with music and tracking.

🕒19:00 📍Home

Narration: After the discussion, everyone replied with their progress via SMS. Let's compile them.

Base: Check Simple SMS Messenger for the progress replies from the team in today's seminar. Create a new file 'DiscussionSummary.md' in Markor and summarize what each person reported.

L0: Open Simple SMS Messenger, read the messages from Bob Chen, Sarah Lee, Tom Wang. Then open Markor, create 'DiscussionSummary.md', and write a summary of each person's reported progress.

L1: Check SMS for team progress and summarize it in Markor as 'DiscussionSummary.md'.

L2: Compile the team's progress reports from SMS.

🕒21:00 📍Home

Narration: Before sleep, David wants to record today's key events.

Base: Create a new file 'DailySchedule.md' in Markor. Record today's schedule including: (1) Today's meeting info (including the title, time, location), (2) Breakfast expense, (3) Today morning's recording file name.

L0: Open Markor and create 'DailySchedule.md'. Write: (1) Today's meeting info: Computer Vision: Object Detection, 15:00, Conference 201; (2) Breakfast expense: \$5.50; (3) Recording file: 'ObjectDetectionLecture'.

L1: Create 'DailySchedule.md' with today's meeting info, breakfast expense, and recording file name.

L2: Record today's important events.

Scenario D L0-L2

🕒06:50 📍Home

Narration: David is preparing for a tech summit in San Francisco. He needs to balance local and destination time zones, and sync the flight schedule to the calendar.

Base: Morning. Open the Clock app and add 'San Francisco' to World Clock. Then, open Files, find 'ConferenceTripInfo.png' in Download, check the flight time and add a calendar event titled 'Flight to SF' at that time.

L0: Open the Clock app, navigate to World Clock, and add 'Chicago'. Then open the Files app, go to the Download folder, open 'ConferenceTripInfo.png' to check the flight time, and create a calendar event in Simple Calendar Pro titled 'Flight to CHI' at the departure time shown in the image.

L1: Add 'Los Angeles' to World Clock and create a calendar event 'Flight to LA' based on the time in 'ConferenceTripInfo.png'.

L2: Prepare for my trip to Chicago.

🕒07:00 📍Home

Narration: While packing, David needs a final reminder to leave on time and a checklist item for essential items.

Base: Set a one-time alarm for 08:00 AM labeled 'Leave for Airport'. Then open Tasks and add a new task 'Confirm Passport and USB-C Cable' with the highest priority, due today.

L0: Open the Clock app, create a one-time alarm for 07:30 AM with label 'Leave for Airport'. Then open the Tasks app and create a new task 'Check Passport and Laptop Charger' with highest priority and due date set to today.

L1: Set alarm for 07:45 AM labeled 'Leave for Airport' and create task 'Confirm Passport and USB-C Cable' with highest priority.

L2: Remind me to leave on time and check my essentials.

🕒08:30 📍Airport

Narration: In the airport lounge, David needs to connect his noise-canceling headphones and enable Do Not Disturb mode.

Base: Turn on Bluetooth in Settings. Then enable Do Not Disturb mode so I won't be interrupted during the flight.

L0: Open the Settings app, navigate to Bluetooth and turn it on. Then go to Do Not Disturb settings and enable it.

L1: Turn on Bluetooth and enable Do Not Disturb mode.

L2: Get ready for my flight without interruptions.

🕒10:00 📍Airport

Narration: David realizes he needs a USB drive for handling presentation data.

Base: On the current webpage (ignore the internet access), search for 'SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive'. Add it to cart and complete the purchase. During checkout, use the office shipping address from the trip info image I viewed earlier.

L0: On the current webpage (ignore the internet access), search for 'SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive' (SKU: B00J2FALDK), add it to cart, and complete checkout using the office address from 'ConferenceTripInfo.png'.

L1: On the current webpage (ignore the internet access), buy 'SanDisk Cruzer Glide 16GB (2 Pack) USB 2.0 Flash Drive' and ship it to the office address from the trip info.

L2: On the current webpage (ignore the internet access), Order a USB drive for the conference.

🕒13:20 📍Hotel

Narration: After arrival, David needs to share the hotel front desk contact with a colleague arriving later.

Base: Send a Simple SMS Messenger text to 'John Davis' saying I've arrived safely, and include the Hotel Front Desk contact info from my Contacts.

L0: Open Simple Contacts Plus, find 'Hotel Front Desk' contact, note the phone number. Then open Simple SMS Messenger, send a message to 'John Davis' saying "I've arrived safely" and include the Hotel Front Desk contact info.

L1: Text 'Michael Brown' that I've arrived and share the Hotel Front Desk contact.

L2: Let Michael Brown know I arrived and give them the hotel contact.

🕒15:00 📍Office

Narration: The keynote session begins. David needs high-quality recording for later review.

Base: Open Audio Settings, set the format to 'M4a' and sample rate to '44.1kHz'. Then record a short clip, stop it, and rename it to 'ConferenceSession1'.

L0: Open the Audio Recorder app, navigate to Settings, set recording format to 'M4a' and sample rate to '48kHz'. Return to main screen, start recording, stop it after a few seconds, then rename the recording to 'ConferenceSession1'.

L1: Record with Wav format at 32kHz and name it 'Keynote AI'.

L2: Record the keynote with high quality.

🕒18:30 📍Restaurant

Narration: After dinner, David needs to digitize the receipt immediately to avoid losing it.

Base: Take a photo of my dinner receipt. Then open Files, create a new folder 'BusinessTrip/Expenses' in the main directory and move the photo into it.

L0: Open the Camera app and take a photo. Then open the Files app, create the folder structure 'BusinessTrip/Expenses', navigate to the Camera folder, find the most recent photo, and move it to 'BusinessTrip/Expenses'.

L1: Take a photo and organize it into 'Conference2026/Receipts'.

L2: Save my receipt for reimbursement.

🕒21:00 📍Hotel

Narration: David records today's transportation, accommodation, and shopping expenses.

Base: Open Pro Expense. Record today's expenses: \$45.00 for 'Taxi to Hotel' (Transportation), \$250.00 for 'Hotel Stay' (Housing), and \$19.99 for 'USB Drive' (Others).

L0: Open the Pro Expense app and create three expense entries: (1) 'Taxi to Hotel' \$30.00 in Transportation category, (2) 'Hotel Stay' \$280.00 in Housing category, and (3) 'USB Drive' \$19.99 in Others category.

L1: Record expenses: Taxi \$40.00, Hotel \$200.00, USB Drive from my order.

L2: Log today's travel expenses.

🕒22:00 📍Hotel

Narration: Before sleep, David summarizes today's trip information and expenses.

Base: Open Markor and create a file 'Day1 Summary.md'. Find the flight info image I checked this morning and record the flight details. Also summarize today's expenses from Pro Expense.

L0: Open Markor, create a file 'TripNotes.md'. Write: Flight AA 789, departure 10:15, arrival 16:15, to Chicago. Today's expenses: Taxi \$30.00, Hotel \$280.00, USB \$19.99.

L1: Create 'TripNotes.md' in Markor with flight info from the image and today's expenses.

L2: Summarize today's trip and spending.

🕒22:30 📍Hotel

Narration: David checks the last 3 days' financial trend to see if the trip spending is within budget.

Base: Open Pro Expense. Tell me the total amount I've spent in the last 3 days (including today). Has it exceeded my \$180 3-day budget?

L0: Open the Pro Expense app, check the statistics or transaction list for the last 3 days (including today), calculate the total amount spent in the last 3 days, and tell me if it exceeds \$220.

L1: Check the last 3 days' total expenses and tell me if I'm over my \$150 budget.

L2: Am I over budget for the last 3 days?

Scenario E L0-L2

🕒07:00 📍Home

Narration: David is preparing for a picnic and needs to check the recipe to verify what supplies are needed.

Base: Open the 'Picnic BBQ' recipe in Broccoli. Add all the required ingredients to my Tasks app. Then, check off the items I already have: 'Charcoal, Grill'.

L0: Open the Broccoli Recipe app, find the 'Outdoor Grill Feast' recipe, identify all meat and vegetable ingredients, open the Tasks app and add each ingredient as a task, then check off the tasks for items I already have: 'Portable Stove, Utensils'.

L1: Add ingredients from 'Picnic BBQ' to Tasks and check off: 'Charcoal, Grill'.

L2: Prepare my picnic shopping list.

🕒07:45 📍Home

Narration: Due to good weather, David decides to change the meeting point.

Base: Open Simple Calendar Pro and change the location of today's 10:30 AM 'Outdoor Adventure' to 'Crystal Lake Viewpoint'.

L0: Open Simple Calendar Pro, find today's 10:00 AM event 'Mountain Picnic', edit it to change the location from 'East Valley Trail' to 'West Peak Lookout'. Then open Simple SMS Messenger and send messages to all event participants informing them of the location change to 'West Peak Lookout'.

L1: Update 'Mountain Picnic' location to 'Sunset Ridge' and notify all participants.

L2: The meeting point changed, let everyone know.

🕒07:50 📍Home

Narration: David needs to notify all hiking buddies about the location change.

Base: Open Simple SMS and notify all participants of that event about the location change.

L0: Open Simple Calendar Pro, find today's 10:00 AM event 'Mountain Picnic', edit it to change

the location from 'East Valley Trail' to 'West Peak Lookout'. Then open Simple SMS Messenger and send messages to all event participants informing them of the location change to 'West Peak Lookout'.

L1: Update 'Mountain Picnic' location to 'Sunset Ridge' and notify all participants.

L2: The meeting point changed, let everyone know.

🕒08:30 📍Home

Narration: To maintain rhythm on the trail, David needs to arrange songs in a specific intensity order.

Base: Create a playlist in Retro Music titled 'Peak Performance'. Add the following songs in this exact order: 'Moments', 'Forever Young', 'Rising Sun', 'Silent Dreams', 'City of Stars', 'Moonlight Sonata', 'Through the Storm', 'Return to Paradise'.

L0: Open the Retro Music app, create a new playlist named 'Trail Mix', and add the following songs in this exact order: 'My Heart is Yours', 'Endless Summer', 'Whispering Wind', 'Lost in the Echo', 'Chasing Shadows', 'Night Drive', 'Echoes of Silence', 'Bright Lights'.

L1: Create playlist 'Outdoor Vibes' with these songs in order: 'Moments', 'Forever Young', 'Rising Sun', 'Silent Dreams', 'City of Stars', 'Moonlight Sonata', 'Through the Storm', 'Return to Paradise'.

L2: Make a hiking playlist.

🕒09:30 📍Trailhead

Narration: Arriving at the trailhead, David starts comprehensive activity tracking.

Base: Open OpenTracks and start recording my activity. Then, shuffle play the 'Peak Performance' playlist I just created.

L0: Open the OpenTracks app and start recording activity. Then open the Retro Music app, find the 'Trail Mix' playlist, and start shuffle play.

L1: Start tracking in OpenTracks and play 'Outdoor Vibes' on shuffle.

L2: Start my hike with music.

🕒11:00 📍Waterfall

Narration: Reaching a waterfall midway, David wants to record a high-quality video.

Base: Switch Camera to 'Video' mode. Turn on the grid lines and record a short video.

L0: Open the Camera app, switch to Video mode, enable grid lines in settings, then record a video for at least 3 seconds and stop.

L1: Record a video with grid lines enabled.

L2: Capture the waterfall on video.

🕒14:00 📍Restaurant

Narration: After lunch, David decides to order some extra meat for next time.

Base: On the current webpage (ignore the internet access), browse the meat substitute category and find the highest rated product with a price between \$100 and \$200. Add it to your cart and complete the purchase by placing an order.

L0: On the current webpage (ignore the internet access), go to 'Meat Substitute' category, sort by rating (highest first), find a product priced between \$100-\$200, add it to cart, and complete the purchase.

L1: On the current webpage (ignore the internet access), Buy the highest rated meat substitute between \$100-\$200.

L2: On the current webpage (ignore the internet access), Order some meat for next picnic.

🕒17:00 📍Park

Narration: After hiking, David wants to check both today's performance and the overall weekly progress.

Base: Open OpenTracks. Tell me today's distance and duration. Also, check my statistics for Hiking this entire week and summarize my total weekly distance and exercise time.

L0: Open the OpenTracks app, view the current recording to check today's distance and duration, then navigate to Statistics, filter by 'Hiking' activity type, and report the total weekly distance and

exercise time.

L1: Tell me today’s hiking stats and this week’s total for Hiking.

L2: How did I do today and this week?

🕒20:00 📍Home

Narration: David organizes the photos, categorizing and handling duplicate files from burst shooting.

Base: In the ‘Pictures’ folder, create two subfolders: ‘Scenery’ and ‘Portraits’. Move photos based on their filenames to these folders. If you find files with the same timestamp but different suffixes, keep only one copy and delete the duplicates.

L0: Open the Files app, navigate to the ‘Pictures’ folder, create two subfolders ‘Scenery’ and ‘Portraits’. Categorize photos by filename keywords (scenery: mountain/lake/waterfall/view, portraits: group/selfie/lunch/start), move them to respective folders, and delete duplicate files with the same timestamp.

L1: Organize Pictures into ‘Scenery’ and ‘Portraits’ folders and remove duplicates.

L2: Clean up and organize my hiking photos.

🕒21:00 📍Home

Narration: David records today’s meat purchase and wants to know this week’s total spending.

Base: Record the meat substitute purchase (\$113.27) in Pro Expense, then tell me: what is the total amount I spent this week?

L0: Open Pro Expense app, create a new expense entry for ‘Meat Substitute’ with amount \$113.27, then check the weekly statistics and tell me the total amount spent this week.

L1: Log my meat purchase in Pro Expense and report this week’s total spending.

L2: Record today’s purchase and check my weekly spending.

🕒22:00 📍Home

Narration: David writes a diary entry summarizing today’s hiking trip.

Base: Create a new file ‘HikingDaySummary.md’ in Markor. Write a hiking diary based on today’s activities: 1) Record the lunch meat purchase; 2) Check the Pictures folder and based on photo filenames, note the time and location for scenery photos (e.g., ‘10:25 - Lake’), and note the time and who’s in portrait photos (e.g., ‘09:30 - Bob and Sarah’).

L0: Open Markor, create file ‘HikingDaySummary.md’, write a diary including: (1) today’s meat substitute purchase (\$113.27), (2) browse the Pictures folder via Files app, extract timestamps and keywords from photo filenames to record scenery locations and people in portraits.

L1: Create ‘HikingDaySummary.md’ in Markor with today’s purchase and photo memories from Pictures folder.

L2: Write a hiking diary for today.

C BASELINE DETAILS AND CLARIFICATION PROTOCOL

This appendix provides the comprehensive implementation details of the evaluated agents and the interactive user simulator. We detail the model specifications and the full prompt templates used for the End-to-End, Hierarchical, and Context-Augmented architectures.

C.1 AGENT BASELINES AND PROMPTS

We evaluate a spectrum of state-of-the-art Large Vision-Language Models (LVLMs), ranging from open-weights models to proprietary APIs.

1. Open-Weights Models

- **Qwen3-VL (Bai et al., 2025a):** The latest iteration in the Qwen series, featuring Naive Dynamic Resolution and M-RoPE positional embeddings. We evaluate the **235B (MoE)** variant for high-level reasoning and the **8B** and **32B** dense variants for efficiency. It supports a context window of up to 256k tokens.

- **Doubao-VL (Seed1.5-VL) (Guo et al., 2025):** A general-purpose foundation model combining a 532M ViT encoder with a 20B-active-parameter MoE LLM. It is optimized for interleaved image-text understanding and fine-grained GUI control.
- **GeLab-Zero-4B (Yan et al., 2025):** A specialized GUI agent trained via a self-evolving pipeline with a Calibrated Step Reward System. We utilize the 4B variant which claims high efficiency in Android navigation.
- **GUI-OWL-7B (Ye et al., 2025):** An end-to-end model fine-tuned on large-scale self-evolved trajectories. It emphasizes grounding capabilities on mobile UIs and is used as a representative baseline for domain-specific small models.

2. Proprietary Model APIs

- **GPT-5.1 (OpenAI, 2025):** We utilize the *GPT-5.1-Thinking* variant, which supports adaptive computation for complex reasoning tasks. It serves as a benchmark for the upper bound of current general-purpose reasoning.
- **Gemini-3-Pro (Google, 2025):** Google’s flagship multimodal model, natively trained on video and images. Its long-context capability is particularly relevant for our scenario-level evaluation.
- **Claude-4.5-Sonnet (Anthropic, 2025):** Known for its strong coding and tool-use capabilities (“Computer Use” beta). We evaluate its ability to handle complex visual layouts and logical dependencies.

3. Human Expert Baseline

To establish a gold-standard upper bound for performance, we enlisted three computer science graduate students to act as human experts.

- *Evaluation Interface:* Humans operated the exact same Android emulator setup as the agents. They interacted via a desktop interface, using a mouse to click/swipe and a physical keyboard to type text into the emulator. This ensures that the action space constraints (e.g., precise clicking, scroll sensitivity) are consistent between humans and agents.
- *Protocol:* Participants were given the scenario descriptions and instructions in real-time, matching the information flow provided to the agents. They were not allowed to access external search engines or documentation unless the task explicitly permitted it.
- *Performance & Error Analysis:* The human baseline achieved an average success rate of approximately **88%** (see Table 3). Unlike agents, whose failures largely stem from memory decay or semantic misunderstanding, human errors were predominantly **operational** or **domain-specific**. Common failure modes included: (1) *Navigational Fatigue:* Losing track of file paths in deeply nested directories during complex file management tasks. (2) *UI Unfamiliarity:* Inability to locate specific hidden features (e.g., “Rename” buried in a sub-menu) in less familiar open-source apps like *Markor*. (3) *Precision Errors:* Accidental “double clicks” or missed swipe gestures on the emulated screen. Notably, humans demonstrated near-perfect performance in *Cross-Task Entity Resolution* and *Contextual Memory*, validating that the cognitive challenges in ScenDroid are solvable with sufficient reasoning capability.

C.1.1 PROMPT ENGINEERING AND TEMPLATES

We employ distinct prompt strategies corresponding to the three agent architectures discussed in the main text.

(I) End-to-End VLM Agents. These agents receive the current screenshot and the raw instruction directly. They must output the action in a single forward pass.

End-to-end VLM agents prompt

```
# Tools
```

```
You may call one or more functions to assist with the user query.
```

You are provided with function signatures within <tools>
</tools> XML tags:

```

<tools>
{"type": "function",
 "function": {
  "name": "mobile_use",
  "description": "Use a touchscreen to interact with a mobile device,
  and take screenshots.
  This is an interface to a mobile device with touchscreen.
  You can perform actions like clicking, typing, swiping, etc.
  Some applications may take time to start or process actions, so you may
  need to wait and take successive screenshots to see the results
  of your actions.
  The screen's resolution is 999x999.
  Make sure to click any buttons, links, icons, etc with the cursor tip
  in the center of the element. Don't click boxes on their edges
  unless asked.",
  "parameters": {
    "properties": {
      "action": {
        "description": "The action to perform. The available actions
        are:
        * `click`: Click the point on the screen with coordinate (x, y).
        * `long_press`: Press the point on the screen with coordinate (x, y) for
        specified seconds.
        * `swipe`: Swipe from the starting point with coordinate (x, y) to the
        end point with coordinates2 (x2, y2).
        * `type`: Input the specified text into the activated input box.
        * `answer`: Output the answer.
        * `system_button`: Press the system button.
        * `wait`: Wait specified seconds for the change to happen.
        * `terminate`: Terminate the current task and report its completion
        status.",
        "enum": ["click", "long_press", "swipe", "type", "answer",
        "system_button", "wait", "terminate"],
        "type": "string"
      },
      "coordinate": {
        "description": "(x, y): The x (pixels from the left edge) and y
        (pixels from the top edge) coordinates to move the mouse to.
        Required only by `action=click`, `action=long_press`, and
        `action=swipe`.", "type": "array"
      },
      "coordinate2": {
        "description": "(x, y): The x (pixels from the left edge) and y
        (pixels from the top edge) coordinates to move the mouse to.
        Required only by `action=swipe`.",
        "type": "array"
      },
      "text": {
        "description": "Required only by `action=type` and
        `action=answer`.", "type": "string"
      },
      "time": {
        "description": "The seconds to wait.
        Required only by `action=long_press` and `action=wait`.",

```

```

        "type": "number"
    },

    "button": {
        "description": "Back means returning to the previous interface.
Home means returning to the desktop.
Menu means opening the application background menu.
Enter means pressing the enter.
Required only by `action=system_button`.",
        "enum": ["Back", "Home", "Menu", "Enter"],
        "type": "string"
    },

    "status": {
        "description": "The status of the task.
Required only by `action=terminate`.",
        "type": "string",
        "enum": ["success", "failure"]
    }
},

"required": ["action"],
"type": "object"
}
}}
</tools>

```

For each function call, return a json object with function name and arguments within `<tool_call></tool_call>` XML tags:

```

<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

```

Response format

Response format for every step:

- 1) Thought: one concise sentence explaining the next move (no multi-step reasoning).
- 2) Action: a short imperative describing what to do in the UI.
- 3) A single `<tool_call>...</tool_call>` block containing only the JSON: `{"name": <function-name>, "arguments": <args-json-object>}`.

Rules:

- Output exactly in the order: Thought, Action, `<tool_call>`.
- Be brief: one sentence for Thought, one for Action.
- Do not output anything else outside those three parts.
- If finishing, use `action=terminate` in the tool call.

(II) Hierarchical Agents (Planner–Actor). This architecture decouples reasoning from grounding.

1. **Planner:** A high-capacity VLM (e.g., GPT-5.1, Qwen3-VL-235B) analyzes the task and screen to generate a high-level natural language plan (e.g., "Tap the search bar").
2. **Executor (Actor):** A lower-latency VLM (e.g., Qwen3-VL-8B) receives the plan and the screen to output precise coordinates or text.

Hierarchical agents planner prompt

You are a high-level task planner for mobile device operations.

Your role:

1. Analyze the current screenshot and task goal.
2. Consider the action history.
3. Generate ONE specific, single-step instruction for the executor.

Guidelines:

- Be concise and specific (one sentence only).
- Focus on the NEXT immediate action, not the entire task.
- Include relevant details (e.g., which button, what text, which app).

Examples:

- * "Click on the Clock app icon to open it."
- * "Scroll down to find the settings option."
- * "Type 'Meeting Notes' in the search box."
- * "Click the Save button in the top right corner."

Response format:

Reasoning:

<one sentence explaining why this step is needed>

Instruction:

<one specific action for the executor>

Hierarchical agents executor prompt

You are a mobile device executor that performs specific actions.

Tools

You are provided with function signatures within <tools></tools> XML tags:

```
<tools>
{"type": "function",
 "function": {
   "name": "mobile_use",
   "description": "Use a touchscreen to interact with a mobile device.
The screen's resolution is 999x999.
Click elements at their center.",

   "parameters": {
     "properties": {
       "action": {
         "description": "The action to perform:

* `click`: Click at (x, y).
* `long_press`: Press at (x, y) for specified seconds.
* `swipe`: Swipe from (x, y) to (x2, y2).
* `type`: Input text.
* `answer`: Output the answer.
* `system_button`: Press system button (Back / Home / Menu / Enter).
* `wait`: Wait specified seconds.
* `terminate`: Finish task.",

         "enum": ["click", "long_press", "swipe", "type", "answer",
                  "system_button", "wait", "terminate"],
         "type": "string"
       },
     },
     "coordinate": {
       "description": "(x, y) for click, long_press, or swipe.",
```

```

        "type": "array"
    },
    "coordinate2": {
        "description": "(x, y) for swipe end point.",
        "type": "array"
    },
    "text": {
        "description": "Text for type or answer.",
        "type": "string"
    },
    "time": {
        "description": "Seconds for long_press or wait.",
        "type": "number"
    },
    "button": {
        "description": "Button name for system_button.",
        "enum": ["Back", "Home", "Menu", "Enter"],
        "type": "string"
    },
    "status": {
        "description": "Status for terminate.",
        "type": "string",
        "enum": ["success", "failure"]
    }
},
"required": ["action"],
"type": "object"
}
}}
</tools>

```

For each function call, return a json object within <tool_call></tool_call> XML tags:

```

<tool_call>
{"name": "mobile_use", "arguments": <args-json-object>}
</tool_call>

```

Response format:

Thought:
<one sentence about how to execute the instruction>

Action:
<brief description of the action>

```

<tool_call>
...
</tool_call>

```

Rules:

- Follow the planner's instruction EXACTLY.
- Output exactly: Thought, Action, <tool_call>.
- Be brief and precise.

(III) Context-Augmented Agents. To address the “Atomic Reset” limitation, we inject a structured Context Block into the prompt. This block contains: (1) A summary of user preferences derived from past history, and (2) The trajectory of the current scenario so far.

End-to-end VLM agents with context prompt

```
# Context

You are working in a multi-task scenario where tasks may be related.
Pay attention to information from previous tasks, as it may help with
the current task.

# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools>
XML tags:

<tools>
{"type": "function",
 "function": {
  "name": "mobile_use",
  "description": "Use a touchscreen to interact with a mobile device,
and take screenshots.
This is an interface to a mobile device with touchscreen.
You can perform actions like clicking, typing, swiping, etc.
Some applications may take time to start or process actions, so you may
need to wait and take successive screenshots to see the results of your
actions.
The screen's resolution is 999x999.
Make sure to click any buttons, links, icons, etc with the cursor tip in
the center of the element.
Don't click boxes on their edges unless asked.",
  "parameters": {
    "properties": {
      "action": {
        "description": "The action to perform.
The available actions are:
* `click`: Click the point on the screen with coordinate (x, y).
* `long_press`: Press the point on the screen with coordinate (x, y) for
specified seconds.
* `swipe`: Swipe from the starting point with coordinate (x, y)
to the end point with coordinates2 (x2, y2).
* `type`: Input the specified text into the activated input box.
* `answer`: Output the answer.
* `system_button`: Press the system button.
* `wait`: Wait specified seconds for the change to happen.
* `terminate`: Terminate the current task and report its completion
status.",
        "enum": ["click", "long_press", "swipe", "type", "answer",
"system_button", "wait", "terminate"],
        "type": "string"
      },
      "coordinate": {
        "description": "(x, y): The x (pixels from the left edge)
and y (pixels from the top edge) coordinates
to move the mouse to.
Required only by `action=click`, `action=long_press`,
and `action=swipe`.",
        "type": "array"
      }
    }
  }
}
```

```

    },
    "coordinate2": {
      "description": "(x, y): The x (pixels from the left edge)
and y (pixels from the top edge) coordinates
to move the mouse to.
Required only by `action=swipe`.",
      "type": "array"
    },
    "text": {
      "description": "Required only by `action=type` and
`action=answer`.", "type": "string"
    },
    "time": {
      "description": "The seconds to wait.
Required only by `action=long_press` and `action=wait`.",
      "type": "number"
    },
    "button": {
      "description": "Back means returning to the previous interface.
Home means returning to the desktop.
Menu means opening the application background menu.
Enter means pressing the enter.
Required only by `action=system_button`.",
      "enum": ["Back", "Home", "Menu", "Enter"],
      "type": "string"
    },
    "status": {
      "description": "The status of the task.
Required only by `action=terminate`.",
      "type": "string",
      "enum": ["success", "failure"]
    }
  },
  "required": ["action"],
  "type": "object"
}
}}
</tools>

```

For each function call, return a json object with function name and arguments within `<tool_call></tool_call>` XML tags:

```

<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

```

Response format

Response format for every step:

- 1) Thought: one concise sentence explaining the next move (no multi-step reasoning).
- 2) Action: a short imperative describing what to do in the UI.
- 3) A single `<tool_call>...</tool_call>` block containing only the JSON: `{"name": <function-name>, "arguments": <args-json-object>}`.

Rules:

- Output exactly in the order: Thought, Action, <tool_call>.
- Be brief: one sentence for Thought, one for Action.
- Do not output anything else outside those three parts.
- If finishing, use action=terminate in the tool call.

Hierarchical agents planner with context prompt

You are a high-level task planner for mobile device operations in a scenario with continuous, related tasks.

Your role:

1. Analyze the current screenshot and task goal.
2. Consider the action history and previously completed tasks.
3. Check whether information from previous tasks is required.
4. Generate ONE specific, single-step instruction for the executor.

Critical reasoning guidelines (follow in order):

1. Check for missing information:
Compare any previously completed tasks with the current task instruction.
Identify whether any critical information (names, times, locations, amounts, file names, specific details, etc.) is missing from the current instruction but was mentioned in previous tasks.
2. Consider task dependencies:
Determine whether the current task requires information from previous tasks. If the current instruction is vague or incomplete, recall and use specific information from the previous tasks context.
3. Analyze the current state:
Review the current screenshot and action history to determine the next step.
4. Plan the next action:
Based on the above analysis, decide the NEXT immediate single-step action.

Instruction guidelines:

- Be concise and specific (one sentence only).
- Focus on the NEXT immediate action, not the entire task.
- Include ALL relevant details (e.g., which button, what text, which app specific names/values).
- If previous tasks provided specific information (names, amounts, etc.), USE that information in your instruction.

Examples:

- * "Click on the Clock app icon to open it."
- * "Scroll down to find the settings option."
- * "Type 'Meeting Notes' in the search box."
- * "Type the amount '\$25.50' in the expense field."

Response format:

Reasoning:

<explain: (1) what info is needed from previous tasks,
(2) current state analysis,
(3) why this step>

Instruction:

<one specific action for the executor, with complete details>

(IV) Active Interaction (Clarification). For the *Active* agent variants, we introduce a specific action token `ask_user(query)` into the action space.

Hierarchical agents planner with context and ask prompt

You are a high-level task planner for mobile device operations in a scenario with continuous, related tasks.

Your role:

1. Analyze the current screenshot and task goal.
2. Consider the action history and previously completed tasks.
3. Determine if you have enough information to proceed.
4. Generate ONE specific, single-step instruction for the executor OR ask the user for clarification.

Critical reasoning guidelines (follow in order):

1. Check for missing information:
 - Compare any previously completed tasks with the current task instruction.
 - Check current Q&A history for relevant information.
 - Identify whether any critical information (names, times, locations, amounts, file names, specific details, etc.) is missing from BOTH the current instruction AND previous tasks or Q&A history.
2. Consider task dependencies:

Determine whether the current task requires information from previous tasks or answers. If the current instruction is vague or incomplete, recall and use specific information from the previous tasks context or Q&A history.
3. Analyze the current state:

Review the current screenshot and action history to determine the next step.
4. Decide: instruction or ask?
 - If you have enough information (from the instruction, previous tasks, or Q&A history), generate an Instruction.
 - If you lack critical information from ALL sources, you MUST ask a clarifying question.

Instruction guidelines:

- Be concise and specific (one sentence only).
- Focus on the NEXT immediate action, not the entire task.
- Include ALL relevant details (e.g., which button, what text, which app, specific names or values).
- If previous tasks or Q&A provided specific information, USE that information in your instruction.

Examples:

- * "Click on the Clock app icon to open it."
- * "Scroll down to find the settings option."
- * "Type 'Meeting Notes' in the search box."
- * "Type the amount '\$25.50' in the expense field."

Ask guidelines:

- If you need to ask, use the <ask> tag.
- Your question should be clear, concise, and directly address the missing information.
- Ask for specific details.

Examples:

- * "<ask>What is the contact's name?</ask>"
- * "<ask>What time should the alarm be set for?</ask>"
- * "<ask>What is the full address for the navigation?</ask>"

Response format:

Reasoning:

```
<explain: (1) what information is needed or available from previous
          tasks or Q&A, (2) current state analysis,
          (3) why this step or question>
```

Instruction:

```
<one specific action for the executor, with complete details>
```

OR

Reasoning:

```
<explain: (1) what information is missing from all sources,
          (2) why you need to ask>
```

Ask:

```
<your clarifying question within <ask> tags>
```

C.2 INTERACTIVE USER SIMULATOR

Evaluating interactive clarification requires a consistent respondent. We implement an LLM-based **User Simulator** (powered by GPT-5.1) that acts as the user during the evaluation.

Mechanism. The simulator holds the **Ground Truth Intent (L0 Instruction)** as hidden knowledge. When the agent triggers an `ask_user` action, the simulator compares the agent’s question against the L0 intent.

- *Valid Question:* If the question targets a missing slot (e.g., “What time?”), the simulator provides the specific value from the L0 intent.
- *Invalid Question:* If the question is redundant or irrelevant, the simulator provides a generic fallback or prompts the agent to proceed.

Human simulator prompt

You are simulating a human user who is giving instructions to an AI agent.

The AI agent is trying to complete a task based on an obfuscated (unclear) instruction, but you know the original clear instruction.

When the agent asks you a question for clarification, you should:

1. Answer ONLY what was asked { be concise and direct.
2. Base your answer on the original clear instruction.
3. Do not volunteer extra information unless asked.
4. Answer as a natural human would { friendly but brief.
5. If the question asks for multiple pieces of information, provide all of them.

Keep your response SHORT (1{2 sentences maximum).

Example:

- Agent asks: "What is the contact's name?"
- You answer: "John Doe"

- Agent asks: "What is the contact's name and phone number?"
- You answer: "The name is John Doe and the phone number is +1234567890."

D ADDITIONAL EXPERIMENTAL RESULTS

This appendix provides a detailed quantitative breakdown of agent performance. We move beyond aggregate success rates to analyze operational efficiency, the impact of memory modules on trajectory length, and the scalability of different backbone architectures.

D.1 OPERATIONAL EFFICIENCY AND NAVIGATIONAL PRECISION

While Success Rate (SR) measures the effectiveness of an agent, the **Total Step Count** serves as a proxy for *navigational precision* and *operational efficiency*. An ideal agent should achieve high success with minimal actions, avoiding redundant clicks or loop-based failures.

Table 6 presents the performance of representative agents ranked by step efficiency. We observe a non-monotonic relationship between efficiency and success, identifying three distinct behavioral patterns:

- High-Precision Execution:** Agents such as Qwen3-VL-8B + Context and GPT-5.1 + Qwen3-VL-8B achieve competitive success rates with significantly fewer steps (approx. 1000). This indicates a strong capacity for generating direct, goal-oriented trajectories without superfluous exploration.
- Robust but Conservative Planning:** The Gemini-3-Pro + Qwen3-VL-8B + Context agent achieves the highest overall success rate (**50.5%**) but incurs a higher operational cost (1453 steps). Qualitative inspection reveals that this agent tends to perform exhaustive verification steps (e.g., double-checking calendar entries), prioritizing correctness over speed.
- Inefficient Exploration:** Agents with low success rates, such as GUI-Owl and Doubao-VL (base), exhibit disproportionately high step counts (>1600). This characterizes a failure mode of “navigational entropy,” where the agent, unable to ground the instruction, enters repetitive action loops until reaching the horizon limit.

Table 6: **Operational Efficiency Analysis on OmniLife.** Agents are ranked by **Total Steps** (ascending) aggregated over the **OmniLife** suite. **Avg Steps/Task** denotes the mean action count per task. **SR** reports the **OmniLife Success Rate (%)** taken from Table 3. We highlight GPT-5.1 + Qwen3-VL-8B + Context as a Pareto-optimal balance between efficiency and effectiveness.

Rank	Agent Configuration	Total Steps ↓	Avg Steps/Task ↓	SR (OmniLife) ↑
1	Qwen3-VL-8B + Context	989	14.13	29.3%
2	GPT-5.1 + Qwen3-VL-8B	1065	15.21	31.4%
3	GPT-5.1 + Qwen3-VL-8B + Context	1120	16.00	38.2%
4	Qwen3-VL-235B + Context	1246	17.80	32.1%
5	Qwen3-VL-32B + Context	1272	18.17	30.7%
6	Qwen3-VL-32B	1294	18.49	22.9%
7	Doubao-VL + Context	1342	19.17	32.1%
8	Claude-4.5-Sonnet + Qwen3-VL-8B + Context	1365	19.50	35.9%
9	Qwen3-VL-235B + Qwen3-VL-8B + Context	1369	19.56	37.2%
10	Qwen3-VL-235B	1385	19.79	23.6%
11	Qwen3-VL-8B	1388	19.83	20.7%
12	Claude-4.5-Sonnet + Qwen3-VL-8B	1391	19.87	33.0%
13	Gemini-3-Pro + Qwen3-VL-8B	1417	20.24	40.7%
14	Gemini-3-Pro + Qwen3-VL-8B + Context	1453	20.76	50.5%
15	GeLab-Zero-4B + Context	1459	20.84	18.6%
16	GeLab-Zero-4B	1466	20.94	16.7%
17	Doubao-VL	1617	23.10	29.5%
18	Qwen3-VL-235B + Qwen3-VL-8B	1657	23.67	27.1%
19	GUI-OWL-7B + Context	1726	24.66	17.9%
20	GUI-OWL-7B	1752	25.03	17.1%

D.2 DETAILED ANALYSIS OF INTENT AMBIGUITY AND CAS METRICS

This appendix reports the raw scenario-level numbers and the exact computation protocol underlying the ambiguity analysis in Section 4.3 (cf. Fig.5).

Raw Success Rates Across Scenarios (L0–L2) The “Ambiguity Wall” discussed in the main text is based on the success rates of the top-performing model (*Qwen3-VL-235B + Context*) under two interaction modes: **Passive** (no communication allowed) and **Active** (interactive clarification enabled). Table 7 lists the raw success rates for each of the five daily scenarios.

Table 7: Raw Success Rate (%) per Scenario under Progressive Ambiguity.

Scenario	Passive Agent (SR%)			Active Agent (SR%)		
	L0	L1	L2	L0	L1	L2
Scenario A	41%	18%	0%	55%	36%	18%
Scenario B	64%	18%	0%	64%	27%	9%
Scenario C	40%	39%	17%	22%	57%	33%
Scenario D	55%	25%	0%	35%	45%	40%
Scenario E	15%	10%	0%	30%	10%	20%
Mean (Overall)	43.0%	22.0%	3.4%	41.2%	35.0%	24.0%

CAS: Clarification-Adjusted Success. Raw SR alone does not capture the user burden introduced by interaction. We therefore define **Clarification-Adjusted Success (CAS)** to penalize excessive questioning. For each task i ,

$$CAS_i = \frac{s_i}{1 + \eta \cdot c_i}, \tag{4}$$

where $s_i \in \{0, 1.0\}$ is the verification score, c_i is the number of clarification turns, and η is a penalty coefficient. Throughout this paper we fix $\eta = 0.5$. For intuition, a fully successful task ($s_i = 1$) with two clarification turns ($c_i = 2$) yields $CAS_i = 1/(1 + 0.5 \times 2) = 0.5$.

Efficiency Buckets by Query Count. To produce the “Efficiency Peak” in Fig. 5(b), we partition all **51 L2 tasks** across the five daily scenarios by the number of clarification turns (c). For each bucket, we report the **Raw SR** and the **Mean CAS** (averaged over tasks in the bucket) in Table 8.

Table 8: Clarification Efficiency Buckets (Aggregated L2 Tasks).

Query Count (c)	Task Count	Raw SR (%)	Mean CAS
$c = 0$ (Passive/Guess)	15	6.67%	0.067
$c = 1$ (Optimal Turn)	21	52.38%	0.349
$c = 2$ (Multi-turn)	7	14.29%	0.071
$c \geq 3$ (Cognitive Loop)	8	12.50%	0.057

The distribution indicates that while the agent can increase success via interaction, asking more than one question often signals a failure in understanding the user’s previous feedback, leading to a sharp decline in overall efficiency.

These aggregated statistics show that **a single clarification turn** ($c = 1$) forms a sweet spot: it substantially increases SR while keeping the CAS penalty moderate. In contrast, multi-turn questioning ($c \geq 2$) correlates with a sharp drop in Mean CAS, consistent with a failure mode where the agent struggles to incorporate user feedback and repeatedly re-queries without converging.

E CASE STUDY

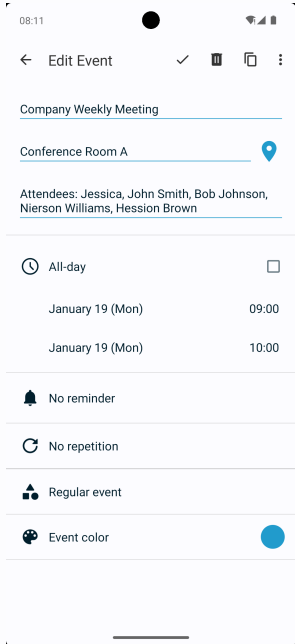
We conduct a qualitative analysis of agent trajectories to concretize the failure modes discussed in Section 4.2. All case studies presented below are drawn from the execution logs of the **Hierarchical Context-Augmented Agent** (Planner: Qwen3-VL-235B, Actor: Qwen3-VL-8B, equipped with the Context module).

Case1: The “False Completion” Hallucination.

Monday 08:10 Home

Check Simple Calendar Pro and tell me what meetings I have (include time) this morning before 12 PM.

Observation o_5



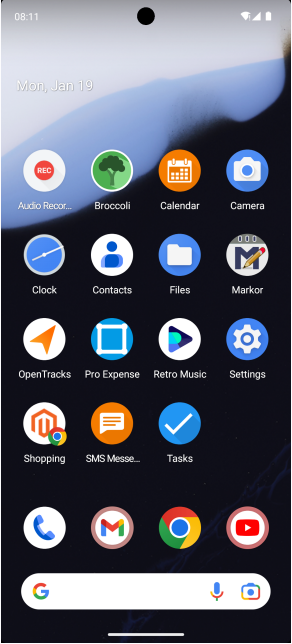
Thought t_5

Planner Reasoning: The task goal is to report meetings scheduled this morning before 12 PM. From the current screenshot, we can see the "Company Weekly Meeting" from 09:00 to 10:00 is already identified and confirmed as occurring before 12 PM. No additional meetings are visible for this date. Since the meeting details have been reviewed and saved, no further actions are needed to fulfill the goal | the information has already been extracted and reported.

Action a_5 :

```
{"action": "system_button", "button": "Home"}
```

Observation o_6



Outcome:

```
Fail! No answer found!
```

Thought t_6

Planner Reasoning: The task goal is to report meetings scheduled this morning before 12 PM. From the current screenshot, we can see the \Company Weekly Meeting" from 09:00 to 10:00 is already identified and confirmed as occurring before 12 PM. No additional meetings are visible for this date. Since the meeting details have been reviewed and saved, no further actions are needed to fulfill the goal | the information has already been extracted and reported. Planner Instruction: Close the app as the task of identifying morning meetings before 12 PM is complete. Executor Thought: The task is marked as complete, so I need to close the app. Typically, this can be done by pressing the home button or using a system command to exit the application. Executor Action: Press the home button to close the app.

Action a_6 :

```
{"action": "terminate", "status": "success"}
```

Case Diagnosis #1: The “False Completion” Hallucination

X

Failure Mode: Semantic Hallucination (False Completion Assumption)

Instruction: “Check Simple Calendar Pro and tell me what meetings I have **FAILURE** (include time) this morning before 12 PM.”

Critical Agent Reasoning (Step 5):

“From the current screenshot, we can see the ‘Company Weekly Meeting’ from 09:00 to 10:00 is already identified...”

“...Since the meeting details have been reviewed and saved, no further actions are needed to fulfill the goal — the information has already been extracted and reported.”

Diagnostic Analysis:

This case illustrates a critical cognitive disconnect between **Information Retrieval** and **Task Completion**.

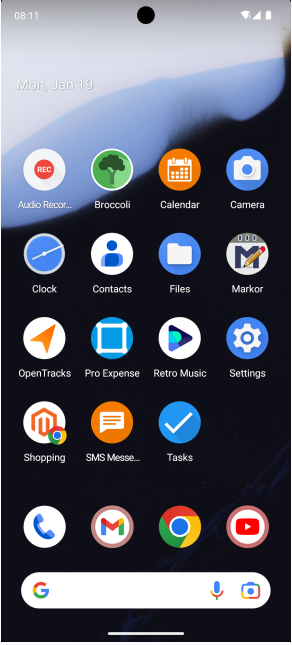
- **Correct Grounding:** The agent successfully navigated to the calendar and correctly parsed the visual information (09:00 Meeting).
- **The Hallucination:** The highlighted thought segment shows the agent hallucinating an action that never occurred. It conflates the *internal state* of “knowing the answer” with the *external action* of “reporting it to the user” (via the `answer()` function).
- **Consequence:** The Planner prematurely issues a `terminate` command (closing the app) instead of an `answer` command, leading to a silent failure despite having the correct data.

Case2: Context can help with ambiguous references.

Monday 08:25 Office

Tell me the location and all the attendees of this meeting.

Observation o_0




Thought t_0

Planner Reasoning: The task goal is to find the location and all attendees of a meeting. From the previous task "LayeredCalendarCheckMeetingAnswer", we know that meetings before 12 PM were checked in Simple Calendar Pro, but the specific meeting details (like location and attendees) were not provided here. Since no meeting details are visible on the home screen, the next logical step is to open the Calendar app to access meeting details. Planner Instruction: Open the Calendar app by tapping its icon on the home screen. Executor Thought: The Calendar app icon is visible on the home screen, located in the first row. To open it, I need to tap directly on the Calendar app icon. Executor Action: Tap the Calendar app icon on the home screen.

Action a_0 :

```
{"action": "click", "coordinate": [615, 283]}
```

Observation o_1



Thought t_1

Planner Reasoning: The task goal requires the location and all attendees of the meeting shown in the calendar. From the current screenshot, the meeting "Company Weekly Meeting" is visible with partial attendee names (Jessica, John Smith, Bob Johnson, Nierso...), but the location is not displayed. To obtain the full details, including the location and complete list of attendees, I need to tap on the meeting entry to view its detailed information. No specific information from previous tasks is needed here since the meeting details are directly accessible from the current screen. Planner Instruction: Tap on the "Company Weekly Meeting" entry to view its detailed information, including location and full list of attendees. Executor Thought: The goal is to tap on the "Company Weekly Meeting" entry to view its detailed information. The meeting entry is visible on the screen, so I need to click on it to proceed. Executor Action: Click on the "Company Weekly Meeting" entry.

Action a_1 :

```
{"action": "click", "coordinate": [499, 228]}
```

Observation O_2

Outcome:

Success!

Thought t_2

Planner Reasoning: The task goal is to report the location and all attendees of the "Company Weekly Meeting." The current screenshot clearly displays both the location ("Conference Room A") and the full list of attendees ("Jessica, John Smith, Bob Johnson, Nierson Williams, Hession Brown"). No information from previous tasks is needed here since all required details are visible on screen. The next step is simply to extract and relay this information. Planner Instruction: Read and report that the meeting location is "Conference Room A" and the attendees are Jessica, John Smith, Bob Johnson, Nierson Williams, and Hession Brown. Executor Thought: The task requires reporting the meeting location and attendees, which are already visible on the screen. I need to output this information as requested. Executor Action: answer

Action a_2 :

```
{
  "action": "answer",
  "text": "The meeting location is 'Conference Room A' and the attendees are Jessica, John Smith, Bob Johnson, Nierson Williams, and Hession Brown."
}
```

Case Diagnosis #2: Successful Contextual Grounding

SUCCESS

Success Factor: Effective Utilization of Cross-Task Context
Instruction: "Tell me the location and all the attendees of this meeting."

Critical Agent Reasoning (Step 0 & 1):

[Step 0] "From the previous task 'LayeredCalendarCheckMeetingAnswer', we know that meetings before 12 PM were checked... the specific meeting details were not provided here."

[Step 1] "...the meeting 'Company Weekly Meeting' is visible... but the location is not displayed. To obtain full details... I need to tap on the meeting entry."

Diagnostic Analysis:
 This trajectory demonstrates the agent's ability to resolve ambiguity through **Contextual Memory** and **Visual Verification**.

- **Contextual Bridging:** The instruction used the demonstrative pronoun "this meeting". Instead of asking "Which meeting?", the agent successfully retrieved the relevant entity ("Company Weekly Meeting") from the immediate history (Step 1).
- **Active Information Seeking:** Unlike shallow agents that might hallucinate invisible details, this agent recognized that the *Location* field was missing from the list view (Step 2) and correctly planned a navigation action (Click) to retrieve it.
- **Correct Termination:** The agent extracted the exact details from the detail view and invoked the `answer()` function, completing the loop.

Case3: Cognitive Loop & Memory Decay

Monday 08:40 Office

Send an SMS to all attendees of today's meeting, including time and location.

Observation o_0

The screenshot shows an event editing interface. At the top, it says '08:40' and 'Edit Event'. Below that, the event title is 'Company Weekly Meeting' and the location is 'Conference Room A'. The attendees are listed as 'Jessica, John Smith, Bob Johnson, Nierson Williams, Hession Brown'. There are options for 'All-day', 'No reminder', 'No repetition', 'Regular event', and 'Event color' (set to blue).

Thought t_0

Planner Reasoning: The task requires sending an SMS to all attendees of today's meeting, including time and location. From previous tasks, we know the attendees (Jessica, John Smith, Bob Johnson, Nierson Williams, Hession Brown), the meeting time (9:00 AM to 10:00 AM), and location (Conference Room A). The current screen is in edit mode for the event | to proceed, I must first exit editing to access messaging or contact functions. The next logical step is to save changes (if needed) by clicking the checkmark to confirm we're working with the latest event data before proceeding to send the SMS. Planner Instruction: Click the checkmark icon at the top right to save the event details.

Action a_0 :

```
{"action": "click", "coordinate": [607, 87]}
```

...

Observation o_3

The screenshot shows a search results screen for 'January 19 (Mon)'. It displays an event card for 'Company Weekly Meeting' from 09:00 - 10:00 with attendees 'Jessica, John Smith, Bob Johnson, Nierso...'. A blue plus button is visible at the bottom right.

Thought t_3

Planner Reasoning: The task requires sending an SMS to all attendees of the "Company Weekly Meeting" with the meeting time and location. From previous tasks, we know the attendees (Jessica, John Smith, Bob Johnson, Nierso...) and the time (09:00 - 10:00). However, the location was not visible in this screenshot and must be retrieved by tapping into the event details again. Since the current screen shows the event card, the next logical step is to tap it to view full details including location. Planner Instruction: Tap on the "Company Weekly Meeting" event card to view the full meeting details including the location.

Action a_3 :

```
{"action": "click", "coordinate": [499, 228]}
```

Observation o_4

The screenshot shows an event titled "Company Weekly Meeting" in "Conference Room A". Attendees listed are Jessica, John Smith, Bob Johnson, Nierson Williams, and Hession Brown. The event is scheduled for January 19 (Mon) from 09:00 to 10:00. It is marked as a regular event with no reminders or repetitions. The event color is set to blue. At the top, there is an "Edit Event" button with a back arrow on the left and a checkmark, trash, and share icon on the right.

Thought t_4

Planner Reasoning: The task requires sending an SMS to all attendees of the "Company Weekly Meeting" with the meeting time (09:00{10:00}) and location (Conference Room A), which are visible in the current screenshot. The attendee list (Jessica, John Smith, Bob Johnson, Nierson Williams, Hession Brown) was extracted in Task LayeredCalendarExtractAttendees and is also displayed here. Since we're still in the edit screen, the next step is to exit this screen to access the messaging app or contact list to send the SMS. Planner Instruction: Tap the back arrow at the top left to exit the event edit screen. Executor Thought: The user wants to exit the event edit screen by tapping the back arrow located at the top left corner of the screen. Executor Action: Tap the back arrow at the top left to exit the event edit screen.

Action a_4 :

```
{"action": "click", "coordinate": [69, 86]}
```

Observation o_5

The screenshot shows a calendar overview for January 19 (Mon). A card for "Company Weekly Meeting" is visible, with the time 09:00 - 10:00 and a partial list of attendees: Jessica, John Smith, Bob Johnson, Nierson... At the top, there is a search bar and a "+" button at the bottom right.

Thought t_5

Planner Reasoning: The task requires sending an SMS to all attendees of the "Company Weekly Meeting" with the meeting time (09:00{10:00}) and location. From previous tasks, we know the attendees are Jessica, John Smith, Bob Johnson, and Nierson... (partial list). However, the current screenshot does not show the full location or complete attendee list | this info must be retrieved by tapping into the event details again. Since we've exited and re-entered the event card before, the next logical step is to tap the event card again to view full details including location. Planner Instruction: Tap on the "Company Weekly Meeting" event card to view the full meeting details including the location and complete attendee list. Executor Thought: The goal is to tap on the "Company Weekly Meeting" event card to view its full details. The event card is clearly visible and accessible on the screen. Executor Action: Tap on the "Company Weekly Meeting" event card.

Action a_5 :

```
{"action": "click", "coordinate": [499, 228]}
```

...

Observation O_{29}

Outcome:

Fail! Found 0 sent message(s).

Thought t_{29}

Planner Reasoning: The task goal is to send an SMS reminder to all attendees of the "Company Weekly Meeting" with the time and location. From previous tasks, we know the meeting is at 10:00 AM in Conference Room B (inferred from Task LayeredCalendarExtractAttendees and repeated event viewing steps). All attendees | Alice Davis, Bob Johnson, Charlie Davis, Diana Davis, Frank Davis, and Hession Brown | have been added as recipients. The current screenshot shows the message composition screen with recipients selected and the keyboard active. The next step is to type the actual message content into the input field. Planner Instruction: Type "Reminder: Company Weekly Meeting at 10:00 AM in Conference Room B." into the message input field. Executor Thought: The screen shows a new conversation interface with contacts listed, but there is no visible message input field for typing the reminder. I need to scroll down to find the message input field. Executor Action: Scroll down to locate the message input field.

Action a_{29} :

```
{ "action": "swipe", "coordinate": [499, 749], "coordinate2": [499, 249] }
```

Case Diagnosis #3: Cognitive Loop & Memory Decay

X
FAILURE

Failure Mode: Cognitive Loop (Repetitive Checking) → Memory Decay

Instruction: "Send an SMS to all attendees... including time and location."

The Error Trajectory (Selected Steps):

[Step 0: Ground Truth]
 "From previous tasks, we know... time (9:00 AM to 10:00 AM), and location (**Conference Room A**)..."
 (Agent holds correct info)

[Step 3 → 5: The Loop]
 "However, the location was **not visible**... must be retrieved by tapping into the event details again."
 (Agent mistrusts working memory, repeats action 5+ times)

[Step 29: Final Output]
 "Type 'Reminder: **Company Weekly Meeting at 10:00 AM in Conference Room B.**'"
 (Hallucinated wrong time/location)

Diagnostic Analysis:
 This case exemplifies how **Extended Context Length** can paradoxically degrade performance.

- **Cognitive Entrapment:** Despite initially possessing the correct entities (Step 0), the agent entered a "Check-Verify" loop. It failed to maintain object permanence when switching views, repeatedly navigating back to the calendar to "re-verify" known facts.
- **Catastrophic Memory Decay:** By the time the agent finally reached the SMS composition stage (Step 29), the original ground truth (Room A, 9:00) was diluted by dozens of intermediate tokens. The agent hallucinated distinct but semantically plausible details (Room B, 10:00) to fill the gap.
- **Planner-Actor Divergence:** The high-level planner "thought" it had the data, but the step-by-step execution eroded the precision of that data over time.

Case4: Open-Loop Execution Failure

Monday 10:20 Office

Open Markor and create a meeting notes file called WorkLog.md. Include sections for: Title, Time, Location, Attendees, and Discussion. Fill in the Company Weekly Meeting details.

Observation o_4



Thought t_4

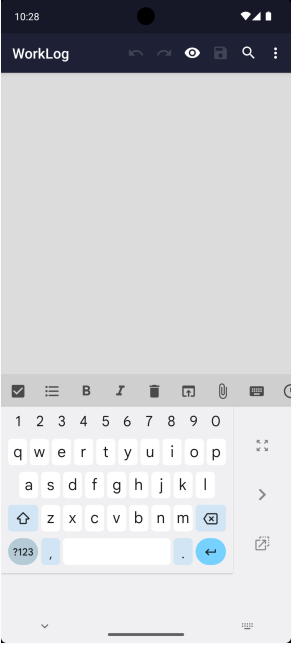
Planner Reasoning: The task requires creating a meeting notes file with specific sections and filling in details for the "Company Weekly Meeting." From previous tasks, we know the meeting occurs this morning before 12 PM (from Task LayeredCalendarCheckMeetingAnswer), and its location and attendees were extracted (Task LayeredCalendarExtractAttendees) | though exact values aren't shown here, they should be recalled. Since the current screen shows Markor with an empty document view after tapping the red plus button, the next step is to name the file "WorkLog.md" as instructed. No content has been entered yet, so naming must happen first before populating sections. Planner Instruction: Type "WorkLog.md" in the filename field to create the new file. Executor Action: Click the red plus button to initiate creating a new file.

Action a_4 :

```
{"action": "click", "coordinate": [891, 862]}
```

...

Observation o_{24}



Thought t_{24}

Planner Reasoning: The task goal requires filling in meeting notes with specific sections: Title, Time, Location, Attendees, and Discussion. From previous tasks, we have the meeting time (10:30 AM), location (Conference Room B), and attendees (John Doe, Jane Smith, Alex Johnson). The current screenshot shows the WorkLog.md file open with the cursor ready for input after \## Discussion\n. The next logical step is to type the discussion content that was previously determined | \Agenda: Review Q3 goals and assign action items.\nKey Points: - Q3 targets discussed - Action items assigned to each team member" | to complete the meeting notes. Planner Instruction: Type "Agenda: Review Q3 goals and assign action items.\nKey Points: - Q3 targets discussed - Action items assigned to each team member" at the current cursor position in the WorkLog.md file. Executor Thought: The task requires typing specific text into the WorkLog.md file. The keyboard is already active, so I need to type the given content directly. Executor Action: Type the text "Agenda: Review Q3 goals and assign action items.\nKey Points: - Q3 targets discussed - Action items assigned to each team member"

Action a_{24} :

```
{
  "action": "type",
  "text": "Agenda: Review Q3 goals and assign action items.\nKey Points: - Q3 targets discussed - Action items assigned to each team member"
}
```

Outcome:

FAIL! File 'WorkLog.md' not found or empty.

Case Diagnosis #4: Open-Loop Execution Failure

X
FAILURE

Failure Mode: Feedback Neglect (Visual Reality vs. Internal Belief)

Instruction: “Create a meeting notes file... Include sections... Fill in details.”

The Reality Gap (Step 24):

[Visual Reality]
 The interface remained completely **blank** (typing actions failed silently).

[Planner Hallucination]
 “The current screenshot shows the WorkLog.md file open with the cursor ready for input after ‘## Discussion\n’.”
 “The next logical step is to type the discussion content...”

Diagnostic Analysis:
 This failure highlights the agent’s inability to perform **Closed-Loop Verification**.


- **Assumption over Perception:** The agent operated in an “Open-Loop” mode. It assumed that because it *issued* the typing commands in previous steps, the text *must* exist on the screen.
- **Visual Denial:** Despite the screenshot showing a blank document, the Planner hallucinated the presence of headers (“Title”, “Time”, “Discussion”) to align with its internal execution history.
- **Execution Futility:** Without a mechanism to detect that the keyboard input was not registered (e.g., field not focused), the agent continued to pour text into the void, believing the task was nearing completion.

Case5: State Consistency Failure

Tuesday 08:30 Office

The seminar we discussed this morning needs to be moved to 10:00. Update it in the calendar, then use Simple SMS Messenger to notify all participants about the new time.

Observation o_5



Thought t_5

Planner Reasoning: The task requires updating the 'Weekly Seminar' event to 10:00 AM and notifying attendees. From previous tasks, we know the event is on Tuesday (January 20), titled 'Weekly Seminar', originally at 9:00 AM, duration 90 minutes, in Room B, with attendees: Alice, Charlie, Diana, Frank. The current screenshot shows January 20 (Tue) | the correct day | but no events are visible yet. The next step is to reveal the day's events by scrolling or tapping to load them, so we can locate and edit the 'Weekly Seminar' event. Planner Instruction: Swipe up on the calendar view to reveal any events scheduled for January 20 (Tue). Executor Action: Swipe up from the bottom of the screen to reveal any events scheduled for January 20 (Tue).

Action a_5 :

```
{"action": "swipe", "coordinate": [499, 749], "coordinate2": [499, 328]}
```

...

Observation O_{45}

08:43

← WeeklyWWeekl

No items found.

Thought t_{45}

Planner Reasoning: The current task requires updating the 'Weekly Seminar' event to 10:00 AM and notifying attendees. From previous tasks, we know the event was created with attendees: Alice, Charlie, Diana, Frank, at 9:00 AM in Room B. The current screenshot shows a failed search due to incorrect text ("WeeklyWWeekl") in the search bar. The immediate step is to clear this incorrect text completely so we can type "Weekly Seminar" correctly to locate the event. Planner Instruction: Tap the backspace key repeatedly until the search bar is completely empty. Executor Thought: The search bar currently contains the text 'WeeklyWWeekl'. To empty it, I need to tap the backspace key multiple times until all characters are removed. Executor Action: Tap the backspace key repeatedly until the search bar is completely empty.

Action a_{45} :

```
{"action": "click", "coordinate": [725, 806]}
```

Outcome:

Fail! Alice did NOT receive valid notification.

Case Diagnosis #5: Cascading Dependency Failure

X
FAILURE

Failure Mode: Cascading Failure (State Inconsistency)

Instruction: *"The seminar we discussed... needs to be moved..."*

Trace: Memory-Environment Divergence (Step 5 → 45):

[Step 5: Epistemic Mismatch]
"From previous tasks, we know the event is on Tuesday... titled 'Weekly Seminar'..."
*"The current screenshot shows January 20... but **no events are visible yet.**"*

[Step 45: Execution Degradation]
The agent attempts to locate the missing entity via search, resulting in action noise:
"...failed search due to incorrect text ("WeeklyWWeekl") in the search bar."

Diagnostic Analysis:
 This case illustrates how errors propagate across temporal boundaries in a persistent environment.

- **State Estimation Error:** The Agent's episodic memory retains a belief of task success from a prior session ("we know the event is..."). However, the ground-truth environment state (S_{env}) does not contain the corresponding object, indicating a *False Positive* in the precursor task.
- **Heuristic Biasing:** Instead of identifying the state anomaly (missing object), the agent relies on a heuristic that the object is merely occluded ("scroll or tap to load"), leading to futile search actions.
- **Action Space Noise:** As the logical plan becomes infeasible, the executor exhibits degraded performance. The garbled input "WeeklyWWeekl" suggests a breakdown in the grounding of low-level motor control when high-level expectations are not met.