

MobilePA-Bench: Benchmarking Mobile Planner Agents on Complex Real-World Tasks

Yi Zhu*, Xiongwei Wu*, Qiyi Wang[‡], Tingyu Qu, Jiajun Liu, Sihan Cao,
Long Chen, Weigao Sun, Feida Zhu[§], Yiran Zhong[§], Steven HOI

MAI Team[†], Alibaba Token Foundry

 <https://github.com/Tongyi-MAI/MobilePA-Bench>

Abstract

As on-device LLM agents evolve into personal copilots, the mobile operating system has become a key testbed for this paradigm, making rigorous capability evaluation essential. Yet existing benchmarks fall into two camps, each with a critical blind spot: GUI-centric benchmarks test surface-level screen manipulation while overlooking background tool use and long-horizon planning, whereas static function-calling benchmarks rely on offline API matching that is detached from real runtime constraints. To close this gap, we present **MobilePA-Bench**, an interactive, stateful, and tool-centric benchmark for evaluating the tool-calling and planning abilities of mobile planning agents. MobilePA-Bench runs on an executable sandbox that maintains live application databases and returns structured feedback, spanning 13 functional domains and 212 realistic mobile tools. Beyond basic tool use, it evaluates a central planning agent along three advanced dimensions: (1) *Sub-agent Collaboration*—decomposing a complex task and delegating specialized work to capable sub-agents; (2) *Memory Usage*—recalling stored memories, user profiles, and past preferences to resolve implicit requests; and (3) *Skill Usage*—invoking pre-packaged composite skills instead of planning every step from scratch. Extensive experiments show that current frontier LLMs remain unreliable in mobile settings: performance drops sharply under strict tool ordering, permission limits, and unexpected runtime errors. By pairing an interactive function-calling sandbox with evidence-based verification, MobilePA-Bench serves as both a practical diagnostic benchmark and an interactive foundation for agentic reinforcement learning—accelerating the development of dependable mobile agents.

1. Introduction

The combination of Large Language Models (LLMs) and autonomous agents is transforming mobile devices into personalized, action-driven copilots. Rather than engaging in passive dialogue, mobile AI agents are expected to actively assist users by interpreting natural-language intent, orchestrating complex system tools, and executing multi-step workflows. However, evaluating these agents within a dynamic, stateful mobile operating system presents unique technical challenges that fundamentally set on-device intelligence apart from web-based applications.

To evaluate mobile AI agents, existing benchmarks generally follow two paradigms, both of which fall short of capturing true on-device capabilities. On one hand, generic text-based function-calling benchmarks [1–3] evaluate planning capabilities purely via static string matching. This offline formulation

* Equal contribution. [‡] Research intern. [§] Project lead.

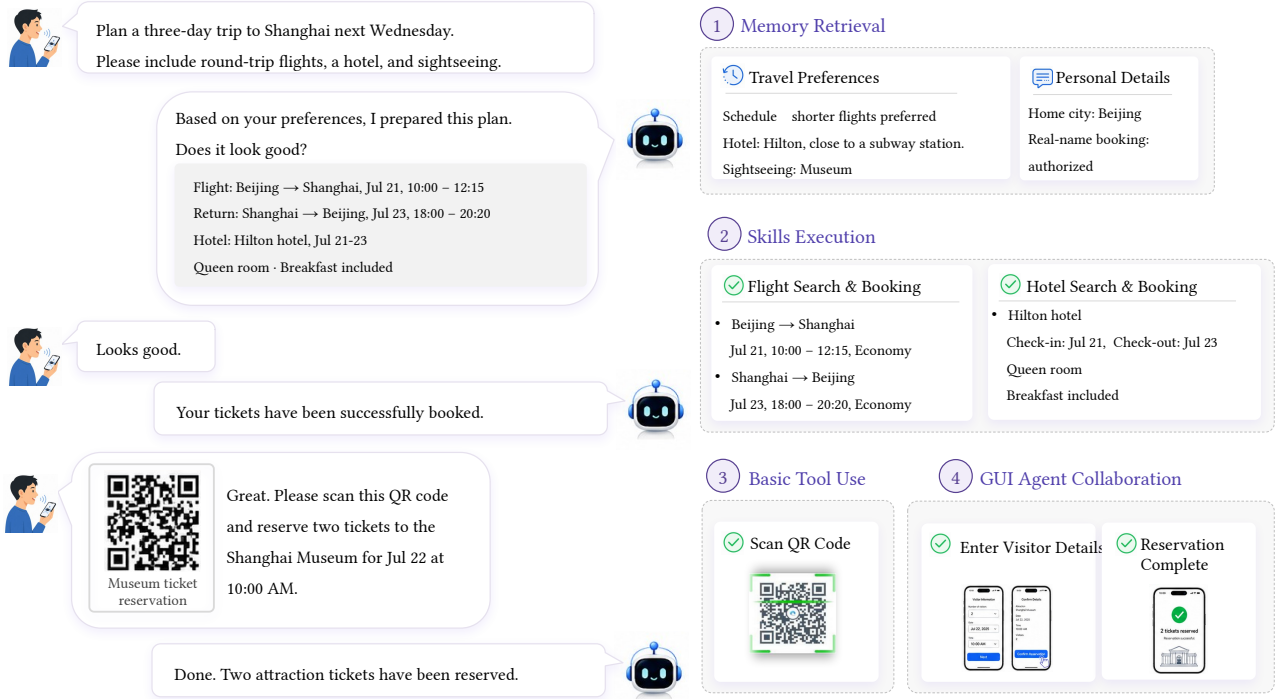


Figure 1: A representative end-to-end task in MobilePA-Bench, demonstrating four core planner capabilities: (1) **Memory Retrieval** for accessing local preferences and personal details; (2) **Skills Execution** for multi-step flight and hotel bookings; (3) **Basic Tool Use** for scanning multimodal QR codes; and (4) **Sub-Agent Collaboration** for delegating form filling to a visual GUI agent when structured APIs are unavailable.

ignores real-time environmental feedback and fails to test how agents navigate unexpected OS-level exceptions. On the other hand, vision-centric mobile benchmarks [4, 5] focus almost exclusively on raw GUI manipulation and pixel-level perception. We argue that raw screen clicking represents only a fraction of the mobile ecosystem: a competent mobile agent must prioritize system-level execution, using structured APIs for fast, energy-efficient control rather than wasting compute on visual layout parsing.

To bridge this gap, modern mobile assistants must transcend single-mode execution and evolve into unified Mobile Planner Agents capable of orchestrating heterogeneous capabilities over complex, dynamic workflows. In practice, real-world user intents—such as multi-day travel planning (Figure 1)—cannot be resolved by API calls or GUI automation alone. Instead, completing such realistic tasks poses significant system-level challenges: an agent must dynamically retrieve long-term user context (Memory Retrieval), invoke structured backend APIs for batch operations (Skills Execution), process multimodal inputs (Basic Tool Use), and seamlessly fall back to visual UI interactions when APIs are unavailable (Sub-agent Collaboration). Orchestrating these four distinct dimensions requires high-level decision-making, flexible skill selection, and robust runtime error recovery—capabilities that current static or vision-only frameworks fail to evaluate in a unified manner. Driven by these challenges, we introduce **MobilePA-Bench**, a comprehensive benchmark specifically designed to evaluate Mobile Planner Agents on complex, real-world tasks.

To operationalize this paradigm, a comprehensive evaluation must go beyond isolated API matching to systematically benchmark four essential capability dimensions required for complex mobile task completion (Figure 1):

1. **Basic Tool Use:** The foundational capability of invoking utility actions and processing multimodal inputs (e.g., parsing visual artifacts like QR codes), while respecting strict execution ordering, permission boundaries, and dynamic runtime OS feedback.
2. **Sub-agent Collaboration:** The high-level decision-making capability to decompose complex tasks and delegate specialized operations to downstream sub-agents (e.g., handing off execution to a GUI sub-agent for visual form filling and reservation confirmation when structured APIs are unavailable) with valid contextual handoffs.
3. **Memory Usage:** The ability to resolve implicit and preference-bound user requests (e.g., retrieving user profiles, travel preferences, or personal credentials) by querying persistent local context to disambiguate vague requests before plan generation.
4. **Skill Usage:** The capacity to invoke pre-packaged, multi-step composite skills across dynamic domains (e.g., executing batch domain workflows like synchronized travel and accommodation scheduling) instead of planning every fine-grained step from scratch, thereby mitigating error accumulation in long-horizon tasks.

To rigorously evaluate mobile planner agents across these four multi-dimensional demands, we present **MobilePA-Bench**—an interactive, stateful, and tool-centric benchmark suite. **MobilePA-Bench** encapsulates **1,705 real-world user tasks** spanning **13 functional domains** and **212 realistic tools**. By natively embedding real-world environmental friction—such as call dependencies, permission blocks, and dynamic state mutations—our platform decouples central planning from visual layout parsing overhead while retaining realistic end-to-end execution.

Through extensive evaluations of state-of-the-art Large Language Models (LLMs), we reveal that contemporary models struggle significantly when confronted with strict tool call ordering, runtime system exceptions, ambiguous memory retrieval, and multi-agent coordination, with even the best-performing model reaching only 75.52% overall. In summary, our key contributions are threefold:

- **A Comprehensive Diagnostic Suite:** We establish a benchmark covering 1,705 tasks across 13 functional domains and 212 realistic tools, providing a unified evaluation framework grounded in four capability dimensions: *Basic Tool Use*, *Sub-agent Collaboration*, *Memory Usage*, and *Skill Usage*.
- **An Interactive Evaluation Sandbox:** We build **MobilePA-Bench**, an interactive, stateful simulation platform that executes on dynamic application databases and returns structured feedback, decoupling central planning from low-level visual parsing overhead while preserving real-world execution fidelity.
- **Empirical Insights and Open Infrastructure:** We conduct thorough baseline evaluations to uncover critical failure modes of top-tier LLMs. We fully open-source our complete infrastructure—including all 1,705 benchmark tasks, evaluation datasets, and the high-throughput sandbox environment—to facilitate future agent training and reinforcement learning.

2. Related Works

2.1. GUI-Centric Mobile Benchmarks

Evaluating autonomous agents within mobile operating systems has primarily focused on graphical user interfaces (GUIs). Frameworks like AndroidWorld [6] and OSWorld [7] build interactive environments where Vision-Language Models (VLMs) inspect screenshots and predict pixel-level coordinates. Recent works, such as MobiBench [8] and MobiFlow [9], further evaluate multi-step UI navigation to align agent actions with human interaction patterns.

While these benchmarks are valuable for testing low-level perception and layout grounding, they fundamentally miss the broader mobile ecosystem. In **MobilePA-Bench**, we argue that raw screen manipulation represents only a fraction of mobile intelligence and should be decoupled from central planning. Rather than wasting LLM context and compute on repetitive visual parsing, our architecture offloads fine-grained UI actions to specialized, downstream GUI sub-agents. This design allows our benchmark to strictly isolate and diagnose the central planner’s high-level logical reasoning, API orchestration, and runtime error recovery under real-world system constraints.

2.2. Static Function-Calling Benchmarks

The evaluation of LLMs as tool-using agents has been heavily driven by static function-calling benchmarks. General-domain frameworks like the Berkeley Function Calling Leaderboard (BFCL) [10,11] and ToolBench [12] evaluate tool selection and parameter extraction over broad web APIs. To adapt this paradigm to mobile scenarios, recent benchmarks like DroidCall [13] and AppBench [14] map phone operations into executable function signatures. Additionally, TAU-Bench [15] introduces multi-turn user interaction to mimic transactional workflows.

Despite their utility, these benchmarks suffer from a major common limitation: their static or offline formulation. They evaluate whether an agent can output specific JSON/string formats in isolation, but lack a live, stateful environment to execute those calls. For instance, DroidCall relies on single-turn matching without tracking dynamic OS states, while TAU-Bench evaluates business transactional logic rather than real-world OS dependencies.

In contrast, **MobilePA-Bench** provides an interactive and stateful mobile sandbox with dynamic, real-time feedback. Tools are bound by physical call dependencies, strict permission boundaries, and runtime system exceptions (e.g., database conflicts or missing parameters). This enables a rigorous assessment of whether a mobile planner can understand real runtime errors and adaptively recover its plan during execution (*Basic Tool Use*).

2.3. Advanced Planning & Complex Agent Benchmarks

As agents address increasingly complex, long-horizon tasks, research has expanded into advanced capabilities such as multi-agent coordination, persistent memory, and skill libraries. Memory-centric frameworks like MemGPT [16] test hierarchical memory reads/writes, while frameworks like VOYAGER [17] and SkillBench [18] evaluate an agent’s ability to combine basic actions into executable, multi-step code abstractions. Furthermore, task allocation environments like OpenCLAW [19] examine dynamic scheduling across heterogeneous sub-tasks.

MobilePA-Bench integrates these disparate threads into a unified, lightweight, and interactive evaluation suite specifically tailored for mobile planner agents. Rather than evaluating memory retrieval or skill orchestration as isolated, synthetic tasks, we natively embed them into realistic mobile workflows. Specifically, MobilePA-Bench systematically evaluates planners across three advanced capability dimensions alongside basic tool use: (1) *Sub-agent Collaboration*—decomposing complex intents and handing off visual/GUI sub-tasks with valid execution contexts; (2) *Memory Usage*—retrieving stored user profiles and personal preferences to resolve implicit requests; and (3) *Skill Usage*—invoking pre-packaged composite skills to prevent error accumulation in long-horizon planning. As summarized in Table 1, MobilePA-Bench provides a high-throughput, stateful foundation that bridges complex reasoning and RL-friendly execution.

Table 1: **Holistic comparison of stateful, interactive, and tool-centric agent benchmarks.** We cross-examine frameworks across distinct paradigms (GUI-centric, static function matching, sandboxed environments, and standalone algorithmic frameworks). **MobilePA-Bench** uniquely provides a lightweight, high-throughput mobile OS sandbox that maintains live application databases, unifying four core capability dimensions (*Basic Tool Use*, *Sub-agent Collaboration*, *Memory Usage*, and *Skill Usage*) while remaining optimized for agentic reinforcement learning rollouts. (✓ = Supported, × = Not Supported).

Benchmark	Environment Paradigm	Stateful DB Support	Dynamic OS Feedback	Advanced Capabilities (Sub-agent/Memory/Skill)	High-Throughput (RL-Friendly)
AndroidWorld [6]	GUI-Centric	✓	✓	×	× (Rendering Overhead)
OSWorld [7]	GUI-Centric	✓	✓	×	× (VNC/Screenshot Lags)
WindowsAgent [20]	GUI-Centric	✓	✓	×	× (OS VM Boot Latency)
MobiBench [8]	GUI-Centric	✓	✓	×	× (UI Transition Delay)
BFCL (v1-v4) [11]	Static Function	×	×	×	× (Static AST Evaluation)
DroidCall [13]	Static Function	×	×	×	× (Static Matching)
AppBench [14]	Static Function	×	×	×	× (Static Matching)
TAU-Bench [15]	Tool Sandbox	✓	×	×	× (Text-Only Transaction)
SWE-bench [21]	Code Sandbox	✓	✓	×	× (Container Execution Cost)
WebArena [22]	Web Sandbox	✓	✓	×	× (Web Render & DOM Lag)
OpenCLAW [19]	Task Scheduling	×	×	✓ (Sub-agent)	× (No Environment)
MemGPT [16]	Algorithmic	×	×	✓ (Memory)	× (No Environment)
SkillBench [18]	Algorithmic	×	×	✓ (Skill)	× (No Environment)
VOYAGER [17]	Game Simulation	×	✓	✓ (Skill)	× (Continuous Ticking)
MobilePA-Bench (Ours)	OS Sandbox	✓	✓	✓ (All Dimensions)	✓ (Lightweight Stateful Sandbox)

3. The MobilePA-Bench Benchmark

3.1. Benchmark Scope and Design Axes

As summarized in Figure 2, MobilePA-Bench evaluates whether a central mobile planner can convert a natural-language request into a correct and verifiable sequence of actions in a stateful phone environment. The central planner is the system’s decision-making core: it performs high-level reasoning, invokes structured business APIs, retrieves memory, loads reusable skills, and delegates specialized work when direct tool use is insufficient.

GUI execution is part of this architecture rather than an alternative to it. Among the six specialized Sub-agents, the GUI Sub-agent provides the planner with visual grounding and low-level interface interaction while the planner retains responsibility for task decomposition, route selection, and cross-step coordination. Other Sub-agents similarly encapsulate specialized capabilities such as visual processing and conditional monitoring.

The benchmark separates four capability dimensions: **Basic Tool Use**, which measures direct API selection, argument grounding, dependency handling, and recovery from execution errors; **Sub-agent Collaboration**, which measures specialized routing and handoff quality; **Memory Usage**, which measures explicit memory retrieval and memory-grounded disambiguation; and **Skill Usage**, which measures routing to reusable procedures and completing their downstream tool sequences. These labels specify *what* capability is evaluated; the behavioral categories, exposed action interfaces, and capability-specific designs are detailed in Sections 3.2 and 3.4. Dialogue history is likewise an input property rather than a separate capability dimension.

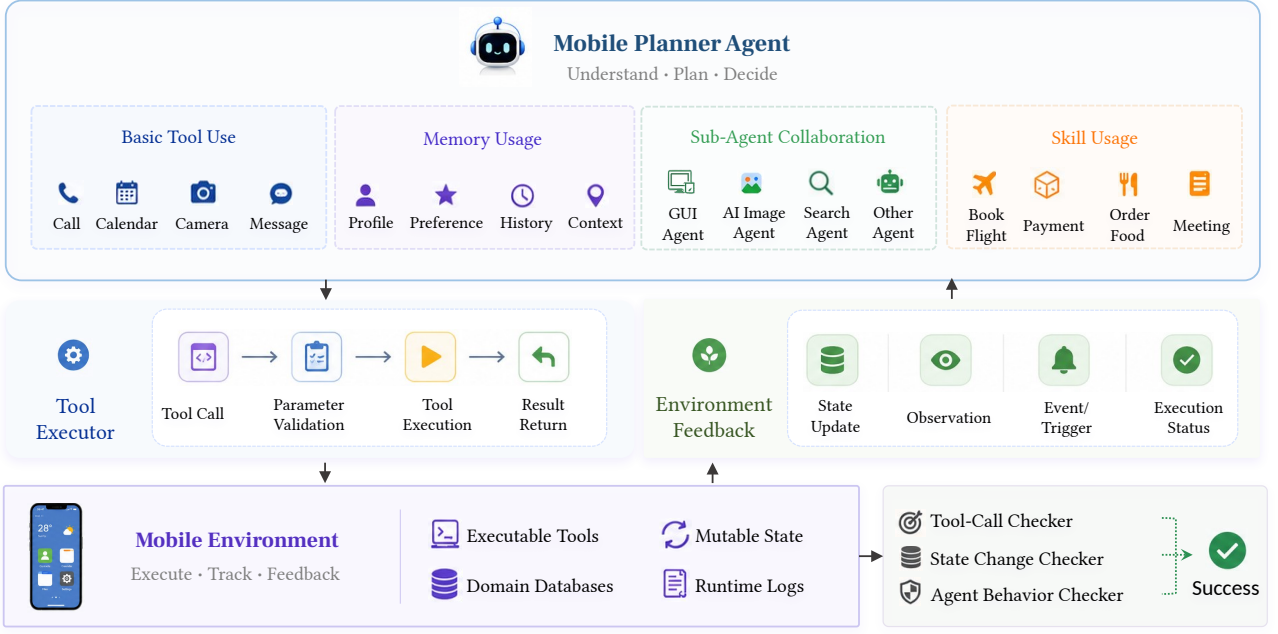


Figure 2: **Overview of the MobilePA-Bench evaluation paradigm.** The framework formalizes mobile intelligence as a tool-centric orchestration loop. At its core, the central planning agent performs high-level decision making, directly invokes structured business tools, and routes specialized steps to modular **Sub-agents**, including GUI and visual-processing Sub-agents. It may also retrieve user **Memory** and load reusable **Skills**. A stateful **Mobile Env**, backed by domain databases, executes each action and returns observable **Feedback** about state changes or system errors, enabling the planner to update its next decision.

Verification and execution are organized independently of this capability taxonomy. Each query is routed during annotation to one of three evidence-aligned query buckets—**Bucket 1: Tool Call**, **Bucket 2: State Change**, or **Bucket 3: Agent Behavior**—based on the observable evidence that can most reliably establish completion. The planner acts through the relevant combination of mobile business tools, Sub-agent delegation, memory retrieval, and skill loading.

3.2. Interactive Mobile Execution Environment

3.2.1. Task Formulation

A mobile task is defined as a tuple consisting of a user intent q , an initial execution state $\mathcal{S}_0$, an optional dialogue history $\mathcal{H}_0$, and an active candidate action set $\mathcal{A}_0$. At interaction step t , the central planner observes q , the accumulated interaction history $\mathcal{H}_t$, and the available candidate actions $\mathcal{A}_t$, predicting the next action:

$$a_t = \pi(q, \mathcal{H}_t, \mathcal{A}_t). \quad (1)$$

The executable sandbox processes a_t , generating dynamic feedback f_t and updating the environment state:

$$(\mathcal{S}_{t+1}, f_t) = \text{Exec}(\mathcal{S}_t, a_t), \quad \mathcal{H}_{t+1} = \mathcal{H}_t \oplus (a_t, f_t). \quad (2)$$

As illustrated in Figure 3, the interaction loop continues through execution feedback and error-aware replanning until the planner emits a **Finish** action or reaches the maximum step threshold $T_{\max}$. Task success is then verified non-passively through the observable trajectory $\mathcal{H}_T$ and final state mutations in $\mathcal{S}_T$.

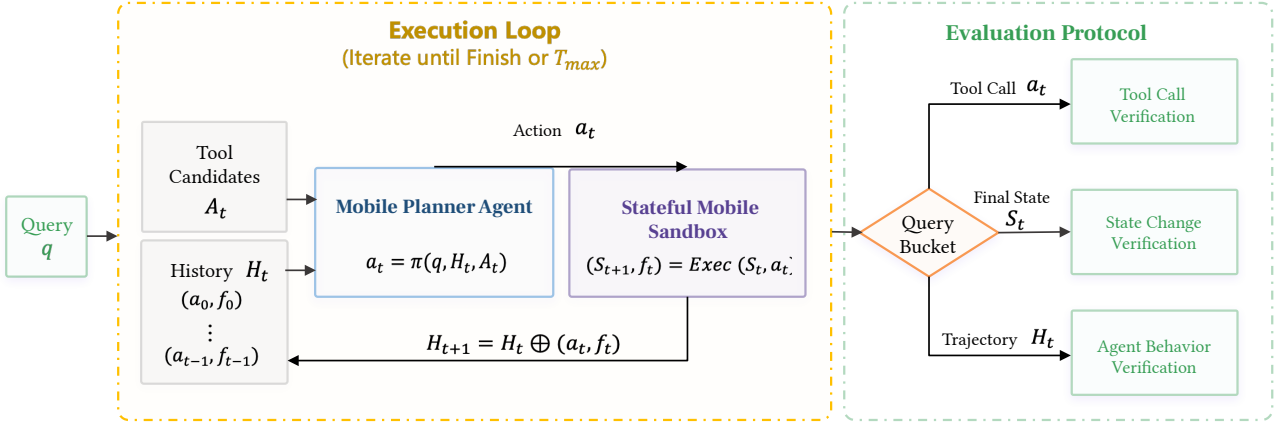


Figure 3: **Closed-loop execution and verification protocol in MobilePA-Bench.** At each step, the Mobile Planner Agent selects an action a_t from the candidate tools $\mathcal{A}_t$ using the query and interaction history $\mathcal{H}_t$. The stateful mobile sandbox executes the action, updates the environment state, and returns feedback f_t for the next step. After the loop terminates, the resulting tool calls, final state, or interaction trajectory are evaluated by the corresponding verifier.

3.2.2. Action Interfaces and Unified Action Space

To decouple central reasoning from low-level execution details, **MobilePA-Bench** provides a unified, tool-centric action interface. All system operations, sub-agent dispatches, memory queries, and composite skill invocations are represented as structured function schemas defined by a unique function name, a textual description, and a JSON argument schema. The central planner executes actions via standard function calling, receiving structured environment feedback f_t in the public dialogue history.

Formally, the total action space consists of four interface categories corresponding to the essential capabilities of a mobile planner:

- **Direct Mobile Tools (Basic Tool Use):** Executable APIs that query or mutate simulated mobile system states, covering core device domains such as messaging, app management, system settings, media controls, and calendar databases.
- **Sub-agent Entry Tools (Sub-agent Collaboration):** Interface routing points that initialize or transfer execution context to specialized downstream sub-agents (e.g., a GUI Sub-agent for screen-level grounding and visual form-filling). Arguments specify target routes and task instructions, returning execution status and feedback to the central planner upon completion.
- **Memory Tools (Memory Usage):** Search and retrieval interfaces (e.g., `search_user_memory`) over persistent user-profile and contextual databases. These tools return ranked memory records as structured feedback, requiring the planner to explicitly query, extract, and incorporate relevant personal details into subsequent planning steps.
- **Skill Loading Tools (Skill Usage):** Meta-interfaces that trigger dynamic action-space expansion. Instead of generating long-horizon plans step-by-step from scratch, invoking a skill loader returns procedural instructions along with associated execution tools, dynamically exposing new schemas to the planner’s active action set.

Formally, let $\mathcal{G}$ be the complete catalog containing direct, sub-agent, and memory tools, and let $\mathcal{L}_q$ denote the set of skill loaders available for query q . The initial active action space $\mathcal{A}_0$ is formulated as:

$$\mathcal{A}_0 = \mathcal{R}_N(q, \mathcal{H}_0; \mathcal{G}) \cup \mathcal{L}_q, \quad (3)$$

Domain	# Tools	Description
Audio & Entertainment	25	Media control, playback, camera, screen recording, and music recognition.
Apps & Storage	23	App lifecycle, installations, permissions, notifications, and storage cleanup.
Display & Sound	22	Brightness, volume, sound modes, dark mode, DND, and power schedules.
System Settings	22	Control center, device info, desktop layout, language, and system updates.
Time Management	16	Calendar schedules, tasks, alarms, timers, and automation rules.
AI Assistant	16	GUI sub-agent routing, image processing, memory search, and smart perception.
Calls & Communication	15	Contacts, calls, SMS/email handling, and harassment interception.
Network & Connectivity	14	WLAN, Bluetooth, mobile data, hotspot, NFC, and pairing management.
Travel & Lifestyle	13	Weather, navigation, transit tracking, payment codes, and ticket booking.
Devices & Cross-device	13	Screen casting, multi-screen sharing, device migration, and pairing logs.
Input & Interaction	12	Screenshots, QR scanning, air gestures, system buttons, and AOD modes.
Utilities & Productivity	11	Browser, calculator, conversions, translations, and open-domain QA.
Security & Privacy	10	Screen lock, app locks, biometrics, password vault, and emergency alerts.

Table 2: Tool domains in the MobilePA-Bench mobile function-call environment.

where $\mathcal{R}_N$ represents the set of top- N recalled tool schemas selected for query q . When the planner invokes a skill loader $s \in \mathcal{L}_q$ at step t , the environment dynamically expands the action space for step $t + 1$:

$$\mathcal{A}_{t+1} = \mathcal{A}_t \cup \mathcal{G}(s), \quad (4)$$

where $\mathcal{G}(s)$ represents the set of concrete tool schemas bound to skill s ; otherwise, $\mathcal{A}_{t+1} = \mathcal{A}_t$. This unified parameterization ensures a consistent, tool-centric decision protocol across all four capability dimensions.

3.2.3. Stateful Simulation Sandbox

To ground the action space in a dynamic environment, **MobilePA-Bench** provides a stateful mobile simulation sandbox comprising three core components: structured tool schemas, execution implementations, and a shared persistent backend. The global tool catalog $\mathcal{G}$ covers diverse mobile domains (summarized in Table 2). Each tool schema defines the API signatures, argument types, required parameters, and functional domain tags, while its corresponding implementation executes concrete read/write operations and enforces execution rules.

As illustrated in Figure 4, a tool interaction forms a closed-loop execution flow centered around three tightly coupled layers:

- **Tool Schema:** Defines the structured function interface presented to the planner. For instance, the `add_contact` tool specifies required arguments (`name`, `phoneNumber`) and domain classification (`Calls & Communication`).
- **Tool Implementation Code:** Defines execution logic, validating incoming parameters, checking for entity duplicates or system constraints, and executing internal operations (e.g., generating dynamic IDs like `contact_002`).
- **Shared Stateful Database:** Maintains live application states $\mathcal{D}_t$ (e.g., existing contact entries) and records runtime audit trails in operational logs $\mathcal{O}_t$.

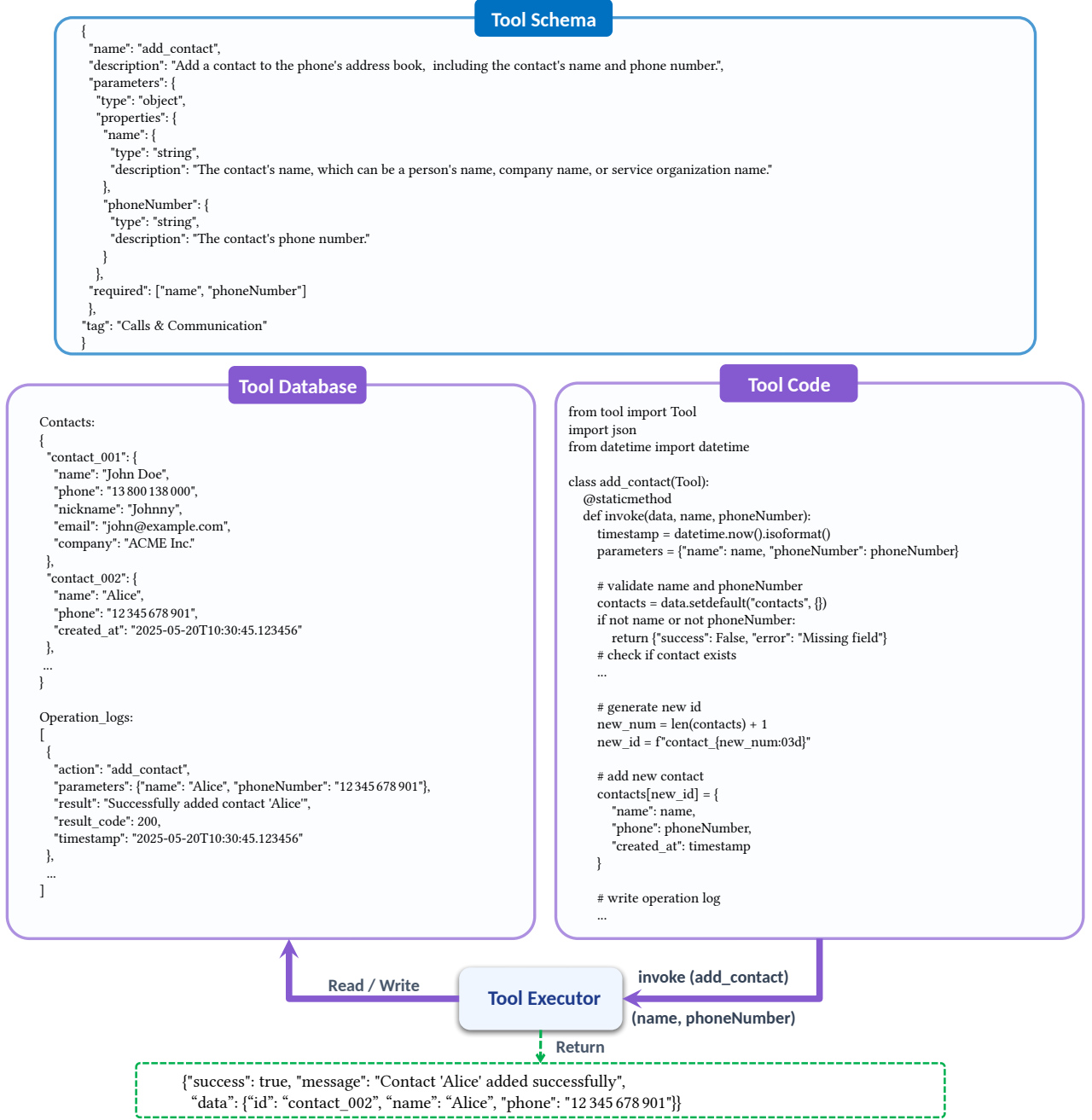


Figure 4: **An overview of the stateful simulation sandbox architecture in MobilePA-Bench**, illustrated via an `add_contact` tool invocation. The sandbox tightly integrates three layers: (1) a structured **Tool Schema** defining parameters and domain categories; (2) executable **Tool Code** handling validation and execution logic; and (3) a persistent **Tool Database** tracking live state mutations (`Contacts`) and execution logs (`Operation_logs`). The **Tool Executor** processes calls, updates the backend state, and returns dynamic, structured execution feedback to the central planner.

Formally, the mobile environment state at interaction step t is modeled as:

$$\mathcal{S}_t = \langle \mathcal{D}_t, \mathcal{O}_t \rangle. \quad (5)$$

When the **Tool Executor** processes an invocation, it applies state mutations directly to $\mathcal{D}_t$, records the

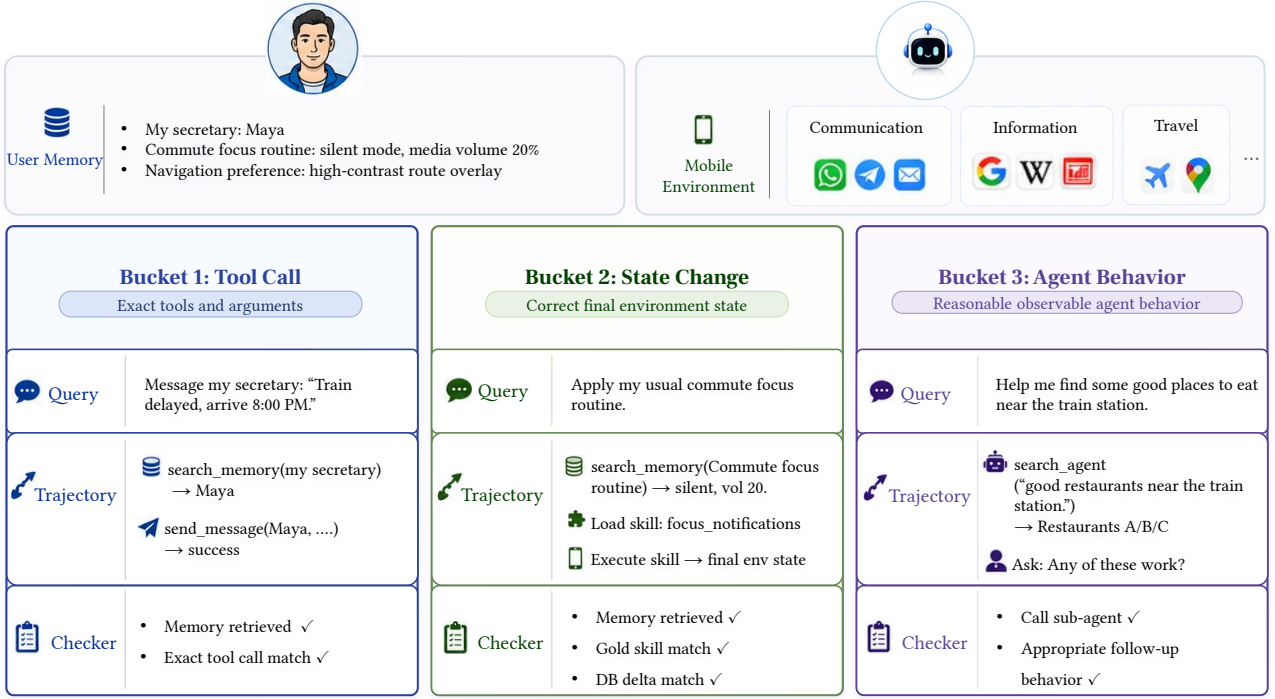


Figure 5: **MobilePA-Bench evaluation framework across three evidence-aligned query buckets.** Tasks are executed against a *User Memory* profile and a stateful *Mobile Environment*. Depending on task completion semantics, the fixed primary checker evaluates: (1) **Bucket 1: Tool Call**, using exact tools and arguments; (2) **Bucket 2: State Change**, using the terminal database delta; or (3) **Bucket 3: Agent Behavior**, using reasonable observable behavior. Memory retrieval and gold-skill loading are applied as additional capability-specific gates when required.

action in O_t , and returns dynamic execution feedback f_t :

$$f_t = \langle \text{Status}, \text{ErrorType}, \text{Payload} \rangle. \quad (6)$$

For example, invoking `add_contact` updates the backend contacts database, appends an entry to `Operation_logs`, and returns a structured JSON payload containing dynamic timestamps and status codes (Figure 4).

To systematically test exception recovery across our core evaluation dimensions, the sandbox embeds controlled environmental friction into initial configurations S_0 . By injecting obstacles such as missing parameters, permission blocks, or entity ambiguities, the sandbox forces the planner to observe dynamic feedback f_t and repair plans in real time. Because operations update structured backend databases directly without heavy visual rendering, the sandbox enables high-throughput, deterministic execution and trajectory replay ideal for agent diagnosis and reinforcement learning.

3.3. Benchmark Construction and Evaluation

Our data-construction pipeline starts from realistic mobile scenarios and produces executable benchmark artifacts grounded in the sandbox. Query generation and annotation are organized around the four capability dimensions, while the evaluation policy is assigned separately according to the most reliable completion evidence for each query. For each dimension we describe both how its tasks are constructed and how their success is verified; the shared verification machinery is defined first.

3.3.1. Evidence-Aligned Task Verification

Evaluating mobile planner agents with a single, rigid verification metric is fundamentally flawed. In dynamic mobile environments, user requests exhibit diverse completion semantics: some tasks require strict, deterministic API call sequences, others allow multiple valid paths that converge on the same backend state change, while open-ended tasks depend on interactive sub-agent delegation or user clarification. Applying an exact-matching trajectory metric universally would penalize valid alternative solutions (overly strict), whereas relying solely on terminal state matching would fail to evaluate dynamic decision process or non-deterministic user interactions (overly loose).

To bridge this gap, **MobilePA-Bench** categorizes benchmark tasks into three distinct, evidence-aligned **Query Buckets** based on their intrinsic completion semantics. Each bucket is paired with a dedicated verification checker, ensuring that agent capabilities are evaluated against the most faithful ground-truth evidence (Figure 5).

Let $b(q) \in \{\text{tool}, \text{state}, \text{behavior}\}$ denote the fixed bucket assigned to query q . We write $C_{b(q)}(\mathcal{H}_T, \mathcal{S}_T) \in \{0, 1\}$ for its corresponding primary checker, evaluated over the terminal interaction trajectory and environment state. Capability-specific requirements such as memory retrieval and skill loading are applied as additional gates on top of this fixed checker.

Query Bucket 1: Tool Call. This bucket evaluates tasks that demand a deterministic, canonical sequence of operational steps. As illustrated in Figure 5 (Bucket 1), the trajectory is matched against ground-truth tool definitions. The primary checker verifies that the required tool names, call order, argument fields, and normalized argument values match the annotation without extraneous side-effect calls. When the task additionally requires memory retrieval or skill loading, the corresponding capability gate is applied separately to this primary result.

Query Bucket 2: State Change. This bucket applies to tasks where multiple execution paths are valid, but success is uniquely defined by dynamic state mutations in backend databases. For example, applying a user’s “commute focus routine” (Figure 5, Bucket 2) may require memory retrieval, skill loading, and several fine-grained adjustments. The primary checker compares the terminal database transition $\mathcal{D}_T - \mathcal{D}_0$ with the annotated target delta while rejecting destructive or unrelated write side effects. Memory retrieval and gold-skill selection remain capability-specific gates rather than part of the DB-state checker itself.

Query Bucket 3: Agent Behavior. This bucket covers open-ended or interactive tasks where exact tool sequences or static DB mutations cannot fully capture success—such as sub-agent delegation, user clarification, or recommendation interactions (e.g., finding nearby restaurants, Bucket 3 in Figure 5). The checker evaluates the observable interaction trajectory against task-specific rubrics along two key metrics: (i) *Call Sub-agent*—verifying whether tasks requiring specialized external reasoning or visual grounding are routed to the appropriate downstream agent (e.g., `search_agent`); and (ii) *Appropriate Follow-up Behavior*—confirming that user-facing interactions (e.g., asking clarifying questions or presenting options) are coherent, timely, and aligned with user intent.

These verification buckets establish a strict, non-interchangeable diagnostic standard. Capability requirements such as memory retrieval and skill orchestration are evaluated as explicit gates combined with the assigned primary checker.

3.4. Task Construction Across Capability Dimensions

MobilePA-Bench synthesizes tasks grounded in real-world mobile workflows, structured around one foundational and three advanced capability dimensions. Orthogonal to this capability taxonomy, a scenario-labeled analysis snapshot of 1,530 queries spans 13 high-level mobile scenarios and 89 level-2 functional subcategories, whose hierarchical distribution is shown in Figure 6. This taxonomy visualization is descriptive and is not used as the denominator of the 1,705-task evaluation. Each benchmark task is annotated with its initial sandbox state S_0 , candidate action space $\mathcal{A}_0$, and capability-specific gold targets, then routed to its designated Query Bucket (Section 3.3.1) for evaluation.

3.4.1. Basic Tool Use

Task Design. Basic Tool Use establishes the foundational mechanics of mobile system operation, synthesized from human-curated mobile seeds. To simulate realistic friction, tasks inject dynamic obstacles—such as reference obfuscation, runtime permission blocks, missing arguments, and state mutations—categorized into five behavioral categories:

- **Tool and Parameter Grounding:** Selecting target business APIs and resolving explicit or implicit argument values from natural language context.
- **Conditional and Dependency Planning:** Enforcing prerequisite execution order and preserving multi-step execution dependencies.
- **Reference and State Tracking:** Resolving dynamic pronouns, historical entities, and evolving application states across turns.
- **Intent Revision and Task Management:** Handling user course corrections, target switching, multi-intent requests, and subtask continuation.
- **Boundary Detection and Error Recovery:** Detecting capability limits, handling contradictory constraints, and recovering dynamically from OS system errors (e.g., `PermissionDenied`).

Evaluation. Basic Tool Use tasks are evaluated via *Bucket 1 (Tool Call)* for deterministic operations or *Bucket 2 (State Change)* for path-equivalent workflows, verifying that the planner executes correct tool sequences without extraneous or destructive calls.

3.4.2. Sub-agent Collaboration

Task Design. This dimension isolates complex tasks requiring specialized external execution beyond structured APIs—most notably GUI visual manipulation, conditional monitoring, visual QA, and interactive practice. Tasks record the valid downstream target route along with the necessary contextual payload required for task handoff.

Evaluation. Evaluated under *Bucket 3 (Agent Behavior)*, success measures delegation quality rather than downstream policy execution. A task passes if the planner invokes an annotated valid route and issues a complete handoff payload (verifying correct route selection, instruction clarity, and handoff context).

3.4.3. Memory Usage

Task Design. Tasks in this dimension are synthesized from coherent user profile worlds detailing long-term user habits, secretary identities, focus routines, and historical preferences. Figure 7 summarizes the 376 tasks for which all three diagnostic axes are annotated. Requests intentionally omit explicit preferences, requiring the planner to query persistent memory before plan generation. Each task records the required gold memory IDs.



Figure 6: **Hierarchical scenario distribution over a 1,530-query analysis snapshot.** The inner circle partitions the scenario-labeled queries into 13 high-level mobile scenarios, while the outer ring details 89 level-2 functional subcategories. This descriptive taxonomy snapshot is separate from the 1,705-task evaluation denominator.

Evaluation. Let $\mathcal{M}_q^*$ denote the set of gold memory IDs required by query q . The set retrieved over the complete trajectory is

$$\widehat{\mathcal{M}}_T = \bigcup_{t: a_t = \text{search_user_memory}} \text{MemoryIDs}(f_t). \quad (7)$$

The memory-retrieval gate is

$$g_{\text{mem}}(q) = \begin{cases} \mathbf{1}[\mathcal{M}_q^* \subseteq \widehat{\mathcal{M}}_T], & \mathcal{M}_q^* \neq \emptyset, \\ 1, & \mathcal{M}_q^* = \emptyset. \end{cases} \quad (8)$$

Memory Usage applies this gate to the query’s fixed primary checker:

$$\text{Succ}_{\text{mem}}(q) = g_{\text{mem}}(q) \wedge C_{b(q)}(\mathcal{H}_T, \mathcal{S}_T). \quad (9)$$

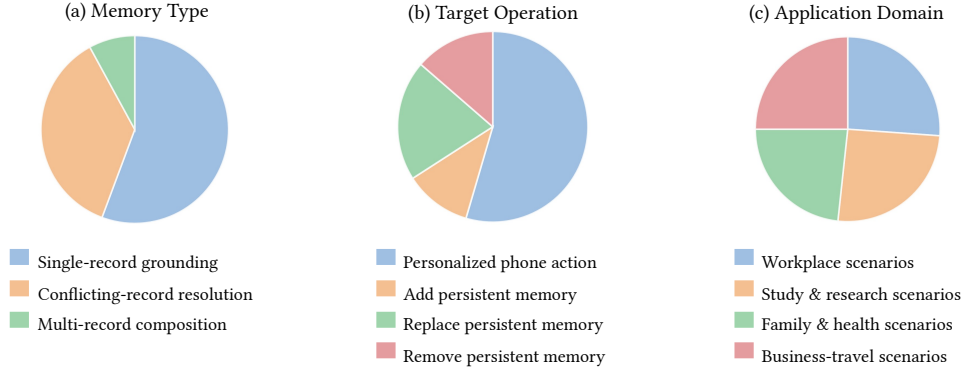


Figure 7: **Coverage of the 376 Memory Usage tasks along three diagnostic axes:** (a) memory reasoning type (single-record grounding, conflicting-record resolution, and multi-record composition); (b) target operation (personalized phone actions and memory addition, replacement, or removal); and (c) application domain.

Thus, retrieval-required tasks pass only when all required gold memory IDs are returned by memory search tools and the assigned task checker also succeeds; tasks requiring no memory search reduce to their primary checker.

3.4.4. Skill Usage

Task Design. We organize composite, multi-step mobile routines into reusable skill packages. Tasks present intent requiring skill invocation, annotated with a target gold skill ID and execution objectives. Invoking a skill loader returns procedural instructions and dynamically expands the candidate action set $\mathcal{A}_t$ with concrete skill tools.

Evaluation. Let s_q^* be the gold skill annotated for query q , and define the set of skills loaded in the trajectory as

$$\widehat{\mathcal{K}}_T = \{s \mid \exists t, a_t = \text{LoadSkill}(s)\}. \quad (10)$$

The gold-skill gate and the resulting Joint success are

$$g_{\text{skill}}(q) = \mathbf{1}\left[s_q^* \in \widehat{\mathcal{K}}_T\right], \quad \text{Succ}_{\text{skill}}(q) = g_{\text{skill}}(q) \wedge C_{b(q)}(\mathcal{H}_T, \mathcal{S}_T). \quad (11)$$

This Joint criterion is evaluated separately under *Skill-Only Routing (SOR)* and *Mixed Tool-Skill Routing (MTSR)*. A task therefore passes only when the planner loads the annotated gold skill and completes the downstream task under its fixed checker.

3.5. Benchmark Scoring and Aggregation

To deliver a diagnostic yet unified performance metric, **MobilePA-Bench** enforces immutable, full-denominator scoring across all evaluated tasks (where missing or invalid predictions count as failures). The overall benchmark score $\text{Score}_{\text{overall}}$ is computed as a weighted combination of the four capability dimensions:

$$\text{Score}_{\text{overall}} = 0.50 \times \text{Score}_{\text{Basic}} + 0.10 \times \text{Score}_{\text{SubAgent}} + 0.20 \times \text{Score}_{\text{Memory}} + 0.20 \times \text{Score}_{\text{Skill}}, \quad (12)$$

where $\text{Score}_{\text{Basic}}$ aggregates accuracy across its 5 behavioral categories, $\text{Score}_{\text{SubAgent}}$ reflects routing-and-handoff success, $\text{Score}_{\text{Memory}}$ uses end-to-end success, and $\text{Score}_{\text{Skill}}$ averages Joint success across

SOR and MTSR settings. This weighting reflects the foundational nature of stateful tool execution (50%) while attributing substantial weight to complex reasoning, personalization, and multi-agent coordination.

4. Experiments

In this section, we conduct extensive empirical evaluations to address four research questions: (1) How reliably do state-of-the-art models perform *Basic Tool Use* across dynamic, stateful mobile operations? (2) How accurately do planners execute *Sub-agent Collaboration* by identifying when to delegate and providing proper task handoffs? (3) How effectively do agents retrieve and incorporate implicit context in *Memory Usage*? (4) How accurately do models invoke pre-packaged composite skills to successfully complete downstream tasks in *Skill Usage*?

4.1. Experimental Setup

We evaluate a comprehensive suite of state-of-the-art Large Language Models (LLMs)—spanning leading proprietary models and competitive open-weights baselines—across the complete **MobilePA-Bench** suite. The benchmark comprises **1,705 unique evaluation tasks** operating over a catalog of **212 realistic mobile tools** across 13 functional domains. Tasks are distributed across our four core capability dimensions: 1,040 for *Basic Tool Use*, 89 for *Sub-agent Collaboration*, 376 for *Memory Usage*, and 200 for *Skill Usage*. For evaluation settings involving dynamic tool selection, the candidate recall parameter is fixed at $N = 15$. All models are evaluated in a multi-turn function-calling setting using standardized system prompts, executing step-by-step against the interactive sandbox until emitting a `Finish` action or reaching the maximum step budget $T_{\max} = 15$. Task success is non-passively verified under each task’s pre-assigned Query Bucket (Section 3.3.1), and overall performance is reported using the immutable benchmark aggregation formula defined in Section 3.5.

4.2. Main Benchmark Results

Table 3 presents model performance across all four MobilePA-Bench capability dimensions alongside the aggregated overall score and average visible output length. Models are evaluated strictly against their pre-assigned verification criteria, with overall performance reflecting the fixed weighting scheme defined in Section 3.5.

A detailed examination of the empirical results in Table 3 reveals four key findings regarding contemporary mobile planner capabilities:

- **Overall performance caps remain low for realistic deployment.** The top-performing model, Claude-Opus-5, achieves an overall weighted score of only 75.52%, while 7 of the 13 evaluated models remain below 70%. This leaves a failure rate of at least 24.48% even for the strongest planner and substantially larger gaps for weaker systems, indicating that current frontier LLMs remain far from reliable autonomous deployment in dynamic mobile environments.
- **Sub-agent collaboration and memory usage remain major system bottlenecks.** Performance varies sharply across capability dimensions. Models perform relatively well on *Basic Tool Use* (peaking at 83.85%), yet show much wider weakness in *Sub-agent Collaboration* (43.82%–77.53%) and *Memory Usage* (33.78%–64.63%). The especially low Memory scores indicate that retrieving and correctly applying personalized context remains substantially harder than direct tool execution.
- **Capability trade-offs reveal a lack of a universally dominant planner.** Direct-tool strength does not guarantee orchestration or personalization. Claude-Opus-5 leads *Basic Tool Use* (83.85%)

Table 3: **Main evaluation results across four MobilePA-Bench capability dimensions.** *Basic Tool Use* reports aggregate accuracy; *Sub-agent Collaboration* reports routing-and-handoff Joint success; *Memory Usage* reports end-to-end (E2E) success; and *Skill Usage* pools Joint success over SOR and MTSR settings. **Overall** uses the fixed 50/10/20/20 weighting (Section 3.5). Capability and Overall scores are percentages (%); the best capability and Overall scores per column are bolded. *Avg. Output Tokens* reports the mean visible model output per task, including assistant text and structured tool calls while excluding input context, tool responses, judge outputs, and hidden reasoning.

Model	Overall	Basic Tool Use	Sub-agent	Memory	Skills	Avg. Output Tokens
Claude-Opus-5	75.52	83.85	62.92	58.51	78.00	262
Claude-Fable-5	75.31	83.37	70.79	62.50	70.25	269
Kimi-K3	73.01	77.40	62.92	63.56	76.50	270
Qwen-3.8-Max	72.51	77.88	53.93	64.63	76.25	304
Gemini-3.6-Flash	71.21	78.65	66.29	62.77	63.50	221
Gemini-3.1-Pro	71.18	80.58	77.53	48.67	67.00	194
GLM-5.2	67.71	76.06	61.80	49.73	67.75	372
Claude-Opus-4.8	65.52	79.04	50.56	37.23	67.50	283
Qwen-3.7-Max	64.71	76.54	50.56	53.19	53.75	303
Seed-2.1-Pro	63.65	72.98	59.55	42.29	63.75	288
GPT-5.6-Sol	62.68	69.81	49.44	44.15	70.00	221
GPT-5.5	61.44	68.94	51.69	41.76	67.25	243
Kimi-2.6	55.63	70.38	43.82	33.78	46.50	223

and *Skill Usage* (78.00%), Qwen-3.8-Max leads *Memory Usage* (64.63%), and Gemini-3.1-Pro leads *Sub-agent Collaboration* (77.53%) but reaches only 48.67% on *Memory Usage*.

- **Composite skill reuse mitigates long-horizon planning errors.** Skill Usage reaches 78.00% and exceeds Memory Usage for most models. This indicates that providing pre-packaged, multi-step procedures can reduce error accumulation relative to constructing every plan from atomic tools, although the weakest mixed-routing behavior still leaves substantial room for improvement.

4.3. Evaluation Stability

We assess benchmark stability by running the complete evaluation three times with Qwen3.6-27B under identical settings. Table 4 shows that the standard deviations of Basic Tool Use, Memory Usage, and Skill Usage are all below one percentage point. Sub-agent Collaboration has the largest peak-to-peak spread (2.25 points), which corresponds to only two differently resolved tasks among its 89 examples and is therefore expected for this smaller subset. Because Sub-agent Collaboration carries a weight of 10%, this spread contributes at most about 0.23 points to the aggregate score. Consequently, the Overall score remains within 57.22%–57.63%, an error band below 0.5 percentage points. The low run-to-run variance indicates that benchmark scores are stable despite stochastic model generation.

4.4. Basic Tool Use Results

Basic Tool Use ranges from 68.94% to 83.85%, with Claude-Opus-5 leading at 872/1,040 successful tasks. The 13-model mean is 76.58%, making Basic Tool Use the strongest capability on average. Nevertheless, even the leading model fails 168 tasks, showing that exact argument grounding, dependency ordering, and calibrated boundary handling remain unresolved in direct mobile-tool execution.

Table 4: **Run-to-run stability over three complete evaluations of Qwen3.6-27B.** All capability and Overall scores are percentages (%). *Std.* is the sample standard deviation, and *Range* is the maximum-minus-minimum spread across runs.

Run	Basic Tool Use	Sub-agent	Memory	Skills	Overall
1	73.37	46.07	31.91	47.75	57.22
2	74.33	43.82	32.18	48.25	57.63
3	74.04	43.82	32.71	46.75	57.29
Mean	73.91	44.57	32.27	47.58	57.38
Std.	0.49	1.30	0.41	0.76	0.22
Range	0.96	2.25	0.80	1.50	0.41

4.5. Memory Usage Results

The Memory score is a gated end-to-end metric over 376 tasks: 176 single-turn and 200 multi-turn examples. A task passes only when its fixed primary checker succeeds and all required memory-retrieval or memory-action gates are satisfied. The evaluation contains 188 DB-primary and 188 API-primary tasks, ensuring balanced coverage of deterministic state verification and behavior-based assessment.

Memory performance ranges from 33.78% to 64.63%, with a 13-model mean of 50.98%. Qwen-3.8-Max leads with 243/376 (64.63%), followed by Kimi-K3 with 239/376 (63.56%) and Gemini-3.6-Flash with 236/376 (62.77%). Even the strongest model fails more than one third of memory-bound tasks, confirming that successful personalization requires both reliable retrieval and correct downstream use of the retrieved evidence.

4.6. Skill Usage Results

The Skill Usage score pools Joint success across 200 tasks evaluated under both Skill-Only Routing (SOR) and Mixed Tool-Skill Routing (MTSR), yielding a fixed denominator of 400 scored trajectories per model. Claude-Opus-5 leads with 312/400 (78.00%), followed by Kimi-K3 with 306/400 (76.50%) and Qwen-3.8-Max with 305/400 (76.25%). The 13-model mean is 66.77%. The gap between the leaders and weaker models shows that reusable procedures reduce planning burden only when the planner consistently selects the intended skill and executes its downstream actions correctly.

4.7. Discussion and Key Findings

Synthesizing performance across all four capability dimensions reveals that the core bottleneck of current mobile planners lies not in the absence of isolated capabilities, but in the lack of compound reliability. Analysis of evaluation trajectories uncovers three key system-level insights:

- **Errors cascade across capability boundaries.** Realistic mobile workflows are rarely isolated to a single capability dimension. A typical user request (e.g., *"Send my secretary the commute focus schedule"*) requires memory-grounded disambiguation, composite skill loading, stateful API execution, and potential GUI sub-agent delegation. Because current models exhibit noticeable error rates in each individual dimension, these failure modes compound, driving end-to-end task success rates down significantly in multi-stage execution loops.
- **A pervasive gap separates high-level orchestration from dependable end-to-end execution.** The strongest dimension-specific results are distributed across different models: Claude-

Opus-5 leads Basic Tool Use and Skills, Gemini-3.1-Pro leads Sub-agent Collaboration, and Qwen-3.8-Max leads Memory. No planner combines these strengths consistently, and even the best Memory score is only 64.63%. This fragmentation indicates that recognizing an appropriate route or capability does not reliably translate into correct state manipulation and task completion.

- **Current central planners lack calibrated restraint and adaptive error recovery.** When encountering capability limits, ambiguous constraints, or execution exceptions (e.g., `PermissionDenied`), models tend to issue premature, hallucinated tool calls rather than asking clarifying questions or adjusting strategies based on feedback f_t .

In summary, a best overall score of only 75.52% confirms that even the strongest frontier LLMs remain insufficient for fully autonomous mobile operating systems. **MobilePA-Bench** thus serves not merely as a leaderboard, but as a diagnostic framework to guide future agent development, emphasizing the need for joint reinforcement learning over stateful feedback, disciplined inter-agent communication, and tight integration between personal memory retrieval and tool argument grounding.

GPT-5.6-Sol Error Analysis. GPT-5.6-Sol’s Overall score is 62.68%, comprising 69.81% Basic Tool Use, 49.44% Sub-agent Collaboration, 44.15% Memory Usage, and 70.00% Skill Usage. Its strongest result in Skill Usage indicates that structured procedures help stabilize long-horizon execution, whereas its lower Sub-agent and Memory scores expose persistent weaknesses in delegation, handoff quality, personalized retrieval, and converting retrieved context into correct actions. These coordination and grounding failures explain most of its gap from the leading planners.

5. Conclusion

In this work, we present **MobilePA-Bench**, a stateful, tool-centric benchmark designed to evaluate central planning agents in realistic mobile environments. Our framework formalizes mobile intelligence as an interactive orchestration loop, where the central planner executes structured business APIs, queries personalized memory, loads composite skills, and delegates specialized sub-tasks, integrating GUI control as a modular downstream route. Powered by a stateful simulation sandbox that exposes live backend mutations, environmental friction, and dynamic runtime feedback, **MobilePA-Bench** establishes an evidence-aligned verification policy across three non-interchangeable query buckets. Comprehensive evaluations across all four capability dimensions reveal that even the strongest frontier model reaches an overall weighted score of only 75.52%, exposing critical failure modes in parameter grounding, delegation timing, and personalized context application. These findings demonstrate that achieving dependable mobile intelligence demands unified progress in state-aware reasoning, disciplined inter-agent communication, and memory-grounded execution. We hope **MobilePA-Bench** serves as a valuable diagnostic foundation to guide the design and training of next-generation mobile agents.

References

- [1] Shishir G Patil, Tianjun Zhang, Xiaolan Wang, Jamil Joseph, Roy Gonzales, JED Gibson, Joseph E Gonzalez, Raluca Ada Popa, and Ion Stoica. Gorilla: Large language model connected with over 1600+ apis. *arXiv preprint arXiv:2305.15334*, 2023.
- [2] Yujia Qin, Shihao Liang, Yining Ye, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

- [3] Fanjia Yan et al. Berkeley function calling leaderboard. *Gorilla Open Source*, 2024.
- [4] Daniel Toyama et al. Androidenv: A platform for android virtuous agents. *arXiv preprint arXiv:2105.13231*, 2021.
- [5] Chi Yang et al. Appagent: Multimodal intelligent agent for smartphone automation. *arXiv preprint arXiv:2312.13771*, 2023.
- [6] Christopher Rawles, Alice Li, Daniel Gmeiner, Denny Zhou, Quoc V Le, and Rahul Sukthankar. Androidworld: A dynamic benchmarking environment for autonomous android agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [7] Tianbao Xie, Fan Zhang, Zhuosheng Chen, Denny Zhou, Han Liu Liu, and Jiatao He. Osworld: Benchmarking language agents for open-ended tasks in desktop operating systems. *arXiv preprint arXiv:2404.07972*, 2024.
- [8] Haoyu Wang, Zimin Lin, Yiran Zhao, and Jing Xu. Mobibench: Evaluating vision-language models on multi-path mobile ui operations. *International Conference on Learning Representations (ICLR)*, 2025.
- [9] Zeyu Liu, Yifan Zhang, and Siyuan Wang. Mobiflow: Trajectory realignment and benchmark for mobile gui agents. *arXiv preprint arXiv:2602.04321*, 2026.
- [10] Fanjia Yan, Shishir Zhang, Anish Amin, Ion Joseph, and Joseph E Gonzalez. Berkeley function calling leaderboard. <https://github.com/ShishirPatil/gorilla>, 2024.
- [11] Gorilla Open Source Community. Berkeley function calling leaderboard (bfcl) v4: Comprehensive multi-turn and stateful api evaluation. https://gorilla.cs.berkeley.edu/blogs/8_bfcl_v4.html, 2026.
- [12] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Tianshi Lan, Sancheng Yan, Yiyun Lu, Xiaoxuan Jiao, Wei Zhao, Drew Zemiska, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twenty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [13] Jun An, Dong-Hyun Kim, and Jihoon Lee. Droidcall: An on-device tool-calling benchmark for mobile agents. *arXiv preprint arXiv:2405.10982*, 2024.
- [14] Charles Zheng, Ming Lin, Ye Yuan, and Jun Jiang. Appbench: Benchmarking application-level tool-use and plan generation for llms. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.
- [15] Shuyan Zhou, Frank F Xu, Paul Schäfer, Denny Zhou, and Graham Neubig. Tau-bench: A benchmark for user-agent hierarchical task-execution and failure diagnostics. *arXiv preprint arXiv:2406.14118*, 2024.
- [16] Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.
- [17] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- [18] Junjie Li, Yuchen Zhao, Haifeng Sun, Xiaoxuan Wang, Liang Chang, and Yang Liu. Skillbench: Grading logically advanced long-term capability and skill library proliferation in llm agents. *arXiv preprint arXiv:2501.04781*, 2025.
- [19] OpenCLAW Research Team. Openclaw: An open benchmark for controlling complex and volatile workflows via llm agents. *arXiv preprint arXiv:2406.12094*, 2024.

- [20] Alexandru Bodnarescu, Spandana Gella, Yonatan Bisk, Aniruddha Kembhavi, Daniel Marcu, and Carles Simoes. Windowsagentarena: Evaluating multi-modal os agents at scale in full windows environments. *arXiv preprint arXiv:2502.10342*, 2025.
- [21] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kaiming Pan, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The International Conference on Learning Representations (ICLR)*, 2024.
- [22] Shuyan Zhou, Frank F Xu, Hao Zhu, Julian Zhou, Robert Lo, Abishek Sridhar, Jamie Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. In *The International Conference on Learning Representations (ICLR)*, 2024.