https://doi.org/10.2352/EI.2026.38.7.IMAGE-269

A Survey of Mobile Agents: From Code Mobility to Large Multimodal Model-Driven Autonomy

Jianhang Chen; Google AI Innovation & Research; Mountain View, CA, USA

Abstract

This paper surveys mobile agents, tracing their evolution from the early paradigm of autonomous, migrating code to the contemporary era of sophisticated agents driven by Large Multimodal Models (LMMs). We begin by establishing a taxonomy that distinguishes historical network-centric agent architectures from modern LMM-native, user-centric systems designed for mobile environments [5]. We then analyze the operational workflows and architectural patterns that enable robust automation of complex tasks on Graphical User Interfaces (GUIs), with particular emphasis on the shift toward multi-agent frameworks, hierarchical control, and local-first execution [3, 15]. A critical review of representative state-of-the-art systems, including MobileAgent-v3.5, ClawMobile, Droidrun-appcard, and OpenClaw, is presented alongside an examination of benchmarks such as AndroidWorld that drive progress in the field [12, 9, 10, 11, 6]. Furthermore, we discuss the transition toward edge-native multimodal models [17, 18, 19, 13, 16, 20] and address novel vulnerabilities unique to LMM-powered GUI automation before concluding with future research directions toward generalized mobile autonomy [21].

1. Introduction

Historically associated with software that migrates code across networks [5], the term "mobile agent" is used in this survey to describe autonomous systems that operate mobile devices through Graphical User Interfaces (GUIs) and system APIs [6]. With modern devices possessing immense computational power, the primary bottleneck has shifted from network bandwidth to the cognitive load of navigating fragmented applications, necessitating the automation of the human-to-machine interface.

The emergence of Large Multimodal Models (LMMs) has substantially advanced mobile GUI agents by improving instruction following, screen understanding, and multi-step decision making. These models can combine natural-language goals with visual observations of application interfaces and action histories, enabling more flexible planning and execution than earlier rule-based approaches [3]. This progress has accelerated work on multi-agent coordination, hybrid local-cloud execution, dynamic evaluation benchmarks, and security analysis for visually grounded agents [6, 21].

This survey provides an analysis of the modern LMM-powered mobile GUI agent landscape. It deconstructs the foundational concepts, explores the operational workflows of leading frameworks, evaluates the micro-architectures of the neural models powering them, and examines the security vulnerabilities that must be resolved before widespread, trusted deployment [21].

2. Concepts and Taxonomy of Mobile Agents

To establish a rigorous analytical foundation, it is imperative to formally define the modern AI agent, distinguish it from traditional automated software, and trace the evolution of the cognitive architectures that enable sophisticated mobile autonomy.

2.1 Defining the Modern AI Agent

A modern AI agent transcends sequential programming by autonomously perceiving its environment, formulating dynamic plans, and acting to achieve user goals—exhibiting autonomy, reactivity, pro-activeness, and adaptability [1]. Autonomy enables independent execution without human intervention, reactivity allows swift responses to non-deterministic environmental changes like pop-ups, and pro-activeness elevates the agent to pursue goal-directed, long-horizon plans rather than simple stimulus-response behaviors. Finally, adaptability denotes the capacity for continuous self-refinement. The agent improves over time by learning from failed execution trajectories, environmental feedback, and persistent episodic memory retrieval.

2.2 Evolution of Cognitive Architectures

The cognitive models underpinning mobile agents have evolved from heuristic reflex systems to sophisticated deliberative frameworks powered by large multimodal models (LMMs). Early architectures evolved from Simple Reflex Agents to Belief-Desire-Intention (BDI) models [2]; however, BDI's symbolic reasoning struggled to scale to the rapidly changing screens of modern visual GUIs.

The reasoning capabilities inherent to LMMs have given rise to new patterns that address these limitations, acting as a neural realization of several BDI-like concepts. The ReAct (Reason-Act) paradigm is now foundational [3]. It prompts an LMM to generate both a reasoning trace (a "thought") and an executable action in an interleaved manner. By explicitly stating its reasoning before outputting commands, the agent can dynamically adjust its plan as it observes the consequences of its actions.

Building on ReAct, the Reflexion architecture endows agents with metacognitive self-reflection [4]. Following a failed attempt, the agent analyzes error feedback and its action trajectory, synthesizes natural language insights, and stores them in an episodic memory buffer. This memory is retrieved to improve future performance, enabling autonomous self-improvement without requiring underlying weight updates.

3. Architectures and Operational Workflows of Modern GUI Agents

Modern mobile agents capable of autonomously navigating heterogeneous graphical interfaces operate through a continuous,

cyclical process of perception, reasoning, and action. While classical approaches rely heavily on a monolithic neural network to orchestrate this loop, recent work has shown this to be inefficient.

To manage the complexity of the core agentic loop, contemporary systems frequently employ specialized multi-agent architectures that distribute cognitive labor across roles for planning, execution, verification, and memory updates [12, 15]. As illustrated in Figure 1, these tasks are divided across specialized nodes to ensure execution stability. A central *Manager* agent ($\mathscr{M}$) integrates the user's high-level instructions with external knowledge bases to formulate a sequence of discrete subgoals. These subgoals are dispatched to a *Worker* agent ($\mathscr{W}$), which contextualizes the task against shared episodic memory ($\mathscr{N}_t$) before executing physical interactions within the mobile GUI environment. Crucially, the execution phase is followed by a verification step: a *Reflector* agent ($\mathscr{R}$) analyzes the resulting environmental state transition to generate corrective feedback (ϕ_t), while a *Notetaker* agent ($\mathscr{C}$) extracts pertinent UI state data to update the shared memory for subsequent loop iterations.

3.1 The Core Agentic Loop and Modalities

The fundamental operational workflow executed by the Worker agent can be deconstructed into three distinct, iterative stages. The integrity of each stage depends on the success of the preceding one.

During perception, the agent comprehends the mobile environment by capturing high-resolution screenshots, which are typically supplemented with structured UI representations like the Android accessibility tree to mitigate the computational cost and hallucination risks associated with pure pixel processing [6].

To bridge the vision-text modality gap, intermediate representation techniques like Set-of-Mark (SoM) utilize computer vision or accessibility trees to overlay numerical text tags directly onto interactable visual elements [7]. This technique grounds the LMM's reasoning, allowing it to output a simple textual command (e.g., "Tap element 14") rather than attempting to guess the exact numerical (x, y) pixel coordinates of a button.

During the reasoning phase, the LMM serves as the cognitive engine by processing a context package that comprises the user's high-level instruction, the multimodal screen representation, and a trajectory history of recent actions to prevent looping behavior. The LMM's primary task is dynamic decomposition: breaking the macroscopic goal into a sequence of actionable sub-tasks, or determining the most appropriate next step [3].

Finally, in the action phase, the LMM generates an executable command based on its reasoning. This output is strictly formatted into a machine-readable schema, such as `click(element_id=15)` or `type(text="123 Main St", element_id=22)`. An external control module, acting as the agent's "hands," parses this command and translates it into a low-level device instruction. This is executed on the physical device or emulator, typically using the Android Debug Bridge (ADB) or native accessibility service APIs. The execution alters the state of the device application, and the agentic loop begins anew.

3.2 Smartphone-Native Architectures: ClawMobile and Execution-Aware Scheduling

A key realization in recent mobile agent research is that blindly relying on probabilistic LMM reasoning for every device interaction is highly inefficient, latent, and prone to failure. Standard mobile devices combine constrained execution contexts, fragmented control interfaces, and rapidly changing application states. The ClawMobile framework was developed explicitly to address these realities by treating mobile autonomy not merely as an AI challenge, but also as a runtime systems engineering problem [9].

ClawMobile introduces a hierarchical architecture where an Agent Orchestrator formulates macroscopic plans, while an Execution-Aware Scheduling strategy separates high-level reasoning from deterministic control pathways [9].

When a sub-task is formulated, the Agent Orchestrator does not immediately trigger a visual GUI interaction loop. Instead, the Execution-Aware Scheduler consults the agent's runtime memory to determine whether a deterministic control backend is available for the required action [9]. Deterministic backends—such as raw ADB shell commands or the Termux API—expose structured system interfaces with mathematically defined semantics for operations like toggling Wi-Fi, adjusting screen brightness, or retrieving battery status. If a matching API exists, the agent routes execution entirely through this deterministic pathway, bypassing the LMM's visual reasoning engine. This prioritizes predictable, bounded operations.

Only when direct, deterministic APIs are unavailable or insufficient does the runtime gracefully degrade by scheduling a semantic UI agent to perform probabilistic visual navigation [9]. This principled coordination between probabilistic planning and deterministic system interfaces drastically reduces token consumption, minimizes hallucination-induced execution errors, and improves stability and reproducibility on real-world devices.

3.3 Curiosity-Driven Knowledge Retrieval: The Droidrun-AppCard Framework

Even with advanced reasoning and hierarchical control, agents frequently encounter catastrophic failures when interacting with entirely unseen applications or highly complex interfaces due to incomplete knowledge. To address failures on unfamiliar UIs, the Droidrun-appcard framework introduces a curiosity-driven retrieval pattern that calculates a "curiosity score" based on execution uncertainty; when this score exceeds a threshold, the agent pauses to query app documentation, code repositories, and historical trajectories [10].

The retrieved knowledge is then organized into structured, modular representations termed "AppCards," which encode functional semantics, parameter conventions, interface structural mappings, and reliable interaction patterns for particular application states. Because AppCards are version-aware and modular, the enhanced agent selectively integrates only the most relevant fragments into its immediate reasoning process, compensating for knowledge blind spots, reducing aimless exploration [10], and yielding a finer-grained understanding of complex applications. When integrated with advanced reasoning backbones, the Droidrun-appcard system achieved a state-of-the-art success rate of 88.8% on the AndroidWorld benchmark, showing that targeted knowledge augmentation is as critical as raw model scale [10].

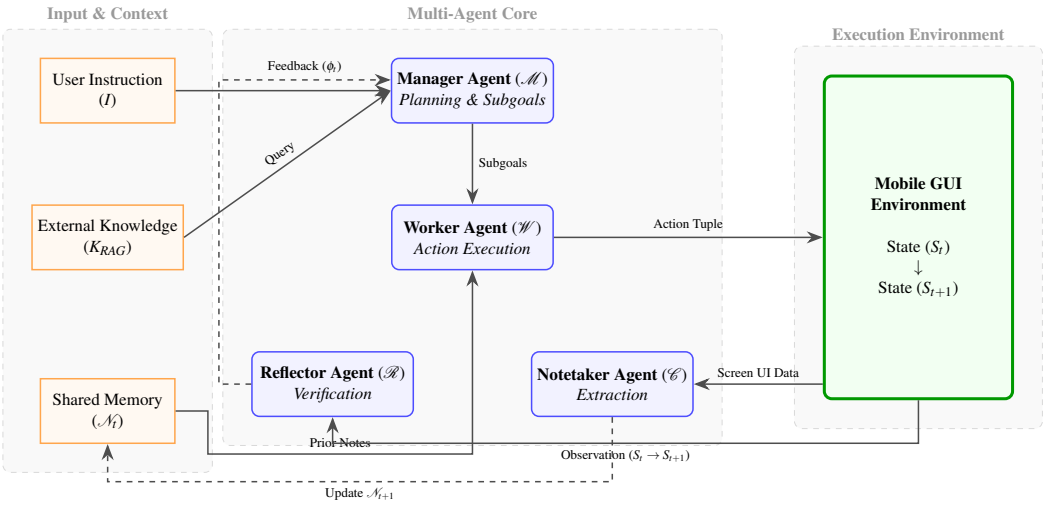


Figure 1. *Architectural overview of a modern multi-agent mobile GUI automation framework. The Manager agent ($\mathscr{M}$) decomposes the user instruction and external knowledge into executable subgoals. The Worker agent ($\mathscr{W}$) processes these subgoals alongside shared episodic memory ($\mathscr{N}_t$) to perform actions on the mobile environment. Subsequently, the Reflector ($\mathscr{R}$) evaluates the state transition ($S_t \rightarrow S_{t+1}$) to provide corrective feedback (ϕ_t), while the Notetaker ($\mathscr{C}$) extracts relevant UI data to update the shared memory state ($\mathscr{N}_{t+1}$) for future iterations.*

4. Local-First and Community-Extensible Frameworks: The OpenClaw Paradigm

While academic frameworks like ClawMobile push theoretical architectural boundaries, the practical deployment of mobile agents has also been advanced by open-source, local-first ecosystems. Foremost among these is OpenClaw (formerly known as Clawdbot and Moltbot), which gained large-scale community adoption in early 2026. Public materials reported that the project had surpassed 100,000 GitHub stars and attracted roughly 2 million visitors in a single week [11]. This momentum demonstrated the viability of running highly capable autonomous agents directly on consumer hardware.

4.1 The OpenClaw Architecture and Agentic Loop

Unlike traditional cloud-hosted chatbots, OpenClaw functions as a local gateway daemon that routes natural language instructions from messaging platforms like WhatsApp and Telegram through an LLM-powered agentic loop [11].

The OpenClaw agentic loop is a refined implementation of the ReAct paradigm, executing a precise sequence: intake, context assembly, model inference, tool execution, streaming replies, and persistence. During the context assembly phase, before any inference occurs, the runtime dynamically builds a dense context package. This package merges OpenClaw's base behavioral prompt, environment-level bootstrap context files, per-run instruction overrides, and, critically, a "skills prompt." The system rigorously enforces model-specific context window limits, actively maintaining a token "compaction reserve" buffer to ensure the LMM always has sufficient space to generate its reasoning and final replies during long-running sessions [3, 11].

When the model infers an action is required, it produces a structured tool call that OpenClaw's runtime intercepts, executes, and feeds back into the conversation context to continue the loop until the overarching goal is satisfied.

4.2 The OpenClaw Skills and File-Based Memory System

A major architectural challenge in agent design is managing the proliferation of tools without overwhelming the LMM's context window. OpenClaw addresses this through its Skills System. Skills are defined through local `SKILL.md` files, and the runtime can filter and selectively include only the relevant skill instructions in the model prompt for a given task [11].

Furthermore, OpenClaw maintains longitudinal continuity across stateless LLM APIs via a file-based memory architecture, storing all long-term knowledge as plain Markdown files within a local directory. This includes a `SOUL.md` file defining the agent's core personality and behavioral preferences, a `MEMORY.md` file for curated facts, and append-only daily logs [11].

To recall this data effectively, OpenClaw supports hybrid retrieval that combines semantic similarity with keyword relevance. Its memory configuration also supports options such as sqlite-vec-backed indexing, MMR diversity re-ranking, and temporal decay to improve the usefulness and freshness of retrieved context [11].

4.3 Android Native Deployment via Termux and Proot

An important extension of the OpenClaw paradigm is Android integration. In the officially documented architecture, Android operates as a companion node while the primary Gateway runs on another host [11]. In parallel, community-driven experiments have explored running gateway-like OpenClaw stacks directly on Android via Termux and proot-based environments, illustrating interest in fully self-hosted mobile deployments [11].

These experimental on-device deployments suggest that smartphones may eventually function as self-hosted AI agent servers without requiring a tethered PC. When paired with messaging clients such as Telegram or WhatsApp and local tool integrations, such setups point toward a highly localized deployment model that emphasizes low latency and data sovereignty [11].

5. State-of-the-Art Systems and Models

The rapid succession of architectural frameworks has been closely tied to the continuous evolution of the underlying vision-language models. The current state of the art is defined by systems that achieve unprecedented success rates on complex benchmarks, and by a decisive industry pivot toward efficient, edge-native LMMs [6, 17].

5.1 Leading Mobile Agents Frameworks

Recent empirical results highlight the pace of advancement in multimodal GUI automation. Powered by the GUI-Owl-1.5 model family (2B to 235B parameters), Mobile-Agent-v3.5 is a multi-platform GUI agent that achieves precise visual grounding via a Hybrid Data Flywheel training pipeline combining simulated and cloud-environment trajectories [12]. Furthermore, it uses a novel reinforcement learning framework, MRPO, engineered specifically to stabilize long-horizon policy optimization across heterogeneous device screens [12]. As a result, Mobile-Agent-v3.5 achieves a highly competitive 71.6% success rate on the AndroidWorld benchmark [12].

However, recent reported results vary depending on system design and evaluation setup. Droidrun reports 91.4% on AndroidWorld in a public technical blog post, emphasizing tighter execution feedback loops and stronger state observability during replanning [14]. In parallel, Minitap reports solving all 116 AndroidWorld tasks through multi-agent decomposition and error recovery, illustrating the potential of carefully engineered agent workflows [15].

5.2 The Transition to On-Device LMMs

A defining trend in mobile agent research is the aggressive push to transition inference from the cloud directly to the edge device. Relying on cloud-hosted, proprietary models introduces network latency, privacy risks regarding personal device data, and prohibitive API token costs for continuous background operation. The development of highly capable, compact LMMs is therefore a critical dependency for the widespread deployment of local-first frameworks. The landscape now spans both general-purpose compact multimodal models and increasingly specialized GUI-native families explicitly designed for local execution, privacy-sensitive automation, or cloud-edge collaboration. Six representative families illustrate this progression:

1. **The Gemma 3n Family:** Engineered with a mobile-first design for low-latency multimodal understanding on edge devices. Google describes Gemma 3n using effective-parameter configurations (E2B and E4B), together with MatFormer and Per-Layer Embeddings (PLE), to reduce active memory requirements while preserving capability. The family is positioned for operation within mobile-class memory budgets [17].
2. **Phi-4-mini-flash-reasoning:** Designed for scenarios where compute, memory, and latency are severely restricted, this is a 3.8 billion parameter model focused heavily on logic and analytical reasoning. Its defining characteristic is the abandonment of traditional, computationally expensive quadratic attention scaling in favor of a novel hybrid architecture called SambaY [18].
3. **Qwen-VL Series:** The Qwen2.5-VL and Qwen3-VL models have become important open-source backbones for multimodal reasoning and GUI grounding. The Qwen3-VL report explicitly supports grounding through normalized point and bounding-box prediction, making the family relevant for GUI-agent action generation and interface understanding [19].
4. **The GUI-Owl Model Family:** GUI-Owl and GUI-Owl-1.5 represent a line of native GUI-agent foundation models spanning small to very large checkpoints, with GUI-Owl-1.5 explicitly released in 2B, 4B, 8B, 32B, and 235B variants [13, 12]. Their distinguishing characteristic is that they are trained directly for GUI grounding, planning, action generation, and cross-platform control rather than being adapted only secondarily for interface manipulation. The smaller 2B–8B checkpoints are especially relevant for hybrid cloud-edge deployment and local inference, while larger checkpoints act as stronger teachers or remote coordinators.
5. **The Step-GUI Family:** Step-GUI introduces compact 4B and 8B GUI-specialized models trained through a calibrated self-evolving data pipeline that converts model-generated interaction traces into reliable supervision signals [16]. Beyond raw model quality, its broader significance for edge deployment lies in GUI-MCP, a hierarchical protocol that supports delegation to local specialist models while keeping sensitive data on-device. This makes Step-GUI particularly relevant for privacy-preserving mobile automation.
6. **UI-Venus-1.5:** UI-Venus-1.5 is a unified GUI-agent family comprising dense 2B and 8B variants as well as a larger 30B-A3B mixture-of-experts model [20]. Its architecture combines large-scale mid-training for GUI semantics, online reinforcement learning with full-trajectory rollouts, and model merging across grounding, web, and mobile capabilities. The 2B and 8B checkpoints make the family especially notable as a bridge between compact deployment targets and strong end-to-end GUI competence.

Taken together, these families show that edge-native mobile-agent progress is no longer driven solely by smaller generic VLMs. Instead, the field is increasingly shaped by compact, task-specialized GUI models that co-optimize grounding accuracy, long-horizon decision-making, and local execution constraints.

6. The Evaluation Benchmarks and Metrics

The objective measurement of mobile agent capabilities is an exceptionally complex endeavor. Early evaluation methodologies often relied on static datasets of pre-recorded screenshots or required costly, subjective human evaluation, both of which are inadequate for assessing true generalization and execution robustness. Real-world mobile applications are dynamic ecosystems characterized by changing content, varying layouts, non-deterministic network behaviors, and intrusive UI elements. A meaningful benchmark must therefore provide an interactive, reproducible environment that reflects this complexity [6].

6.1 Dynamic Instantiation and Durable Rewards

AndroidWorld has established itself as the premier benchmark for mobile agents. Operating on a live, highly instrumented Android emulator, it features 116 programmatic tasks spanning

Architectural Comparison of Leading Edge-Native and Edge-Oriented Model Families [17, 18, 19, 13, 12, 16, 20].

Model Family	Published Size	Key Architectural Innovations	Primary Edge Use Case
Gemma 3n (E4B)	Effective 4B (Raw 8B)	MatFormer; Per-Layer Embeddings (PLE) offload; MobileNet-V5-300M vision encoder (multi-resolution).	On-device multimodal perception under tight memory.
Phi-4-mini-flash	3.8B (Text-only)	SambaY decoder-hybrid; Gated Memory Units (GMU); Mamba (SSM) + sliding-window attention.	High-speed logical reasoning; planner module.
Qwen3-VL (Family)	Dense: 2B-32B MoE: 30B-235B	Interleaved-MRoPE; DeepStack; normalized point and bounding-box grounding for GUI-oriented actions.	General-purpose VLM backbone for GUI agents.
GUI-Owl-1.5 (Family)	2B–235B	Native GUI-agent training; self-evolving GUI trajectory production; cross-platform grounding, planning, and action generation.	Local GUI grounding and cloud-edge GUI control.
Step-GUI	4B / 8B	Calibrated Step Reward System; self-evolving trajectory supervision; GUI-MCP for local specialist delegation.	Privacy-preserving local GUI automation.
UI-Venus-1.5	Dense: 2B / 8B MoE: 30B-A3B	GUI-semantic mid-training; online RL with full-trajectory rollouts; model merging across grounding/web/mobile agents.	Unified end-to-end GUI control with smaller deployable checkpoints.

20 diverse, real-world applications [6]. Its rigorous design philosophy is centered on two fundamental innovations.

First, unlike legacy benchmarks with fixed evaluation sets, AndroidWorld utilizes Dynamic Task Instantiation via parameterized task templates. For every evaluation run, the framework dynamically generates a unique task instance by populating the template with randomized but contextually plausible data strings (e.g., assigning a distinct contact name to save, generating a unique SMS message to transmit, or selecting a specific geographical location to navigate toward). This mechanism ensures that agents are rigorously tested on their ability to perceive and generalize in real time, effectively eliminating the possibility of models artificially inflating scores by memorizing, hardcoded action sequences during training [6].

Second, the benchmark relies on Durable Reward Signals. Task success is not determined by comparing the agent's executed action sequence to a "golden" or "expert" trajectory. Instead, success is evaluated by programmatically inspecting the underlying state of the device's operating system after the agent declares completion. For instance, to verify that a contact was correctly saved, AndroidWorld's internal logic executes an SQL query directly against the device's secure SQLite contact database. To verify a file modification, it directly parses the Android file system. This methodology is robust precisely because it evaluates success regardless of the specific GUI path the agent took to achieve the goal, accommodating substantial variance in UI states and individual agent planning strategies [6].

6.2 Evaluating Multimodal Grounding and Factuality

As mobile agents increasingly eschew accessibility trees in favor of direct visual perception via LMMs, evaluating their spatial grounding and visual factuality becomes paramount. ScreenSpot evaluates GUI grounding by checking whether a predicted click coordinate falls within the ground-truth bounding box (and some settings also allow bounding-box predictions) [8]. Edge-optimized models perform especially well in this domain due to their specialized visual encoders.

Performance Metrics of State-of-the-Art Agent Systems on the AndroidWorld Benchmark [6, 15, 14, 10, 16, 12].

Agent System	SOTA Rate	Key Evaluated Capability
Minitap (mobile-use)	100%	Claimed solution across all tasks under official protocol.
Droidrun	91.4%	Evaluates dynamic replanning and tight execution feedback loops.
Droidrun + AppCard	88.8%	Evaluates curiosity-driven retrieval capabilities.
Step-GUI-8B	80.2% (Pass@3)	End-to-end AndroidWorld result reported under a Pass@3 evaluation protocol.
Mobile-Agent-v3.5	71.6%	Evaluates multi-platform, long-horizon reinforcement learning.

7. Security and Safety of Mobile Agents

The deployment of highly autonomous, LMM-powered agents with broad system-level permissions to perceive and manipulate a user's personal device introduces a severe, complex, and unmitigated set of security and safety challenges [21, 22].

7.1 Traditional Threat Models vs. Multimodal Attack Surfaces

Early academic research on mobile code agents focused on a bipartite threat model: host-on-agent attacks and agent-on-host attacks. Mitigations for these legacy threats relied heavily on strict software sandboxing, rigid access control policies, and cryptographic code obfuscation. Modern LMM-powered GUI agents operate under a radically different security paradigm. The device's visual GUI is no longer merely a display interface for human consumption; it is transformed into a direct, continuous input vector to the LMM's reasoning core. This blurring of the boundary between benign display content and potentially malicious instruction means that every pixel an agent observes is a potential vector for adversarial attack [21].

7.2 The AgentScan Threat Taxonomy

Recent systematic security analyses have culminated in the development of comprehensive evaluation frameworks such as AgentScan, which systematically probe mobile LLM agents across their entire end-to-end operational lifecycle [21].

7.2.1 The LLM Layer (Reasoning Compromise)

Attacks situated at the LLM layer directly target the neural model's language processing and decision-making capabilities, including Obfuscated Malicious Instructions and Glitch Tokens that systematically disrupt tokenization [21].

7.2.2 The GUI Layer (Perception Deception and Action Hijacking)

Because mobile agents rely heavily on visual and structural perception, manipulating the GUI through Prompt Injection via Display, Transparent Overlay, Pop-up Interference, and Viewtree Interference has proven to be an exceptionally reliable attack vector [21, 22].

7.2.3 The System Layer (Execution Exploitation)

These vulnerabilities exploit the agent's trust in the underlying operating system and its application routing mechanics, including Package Name Forgery and Log Leakage [21].

7.3 Defensive Paradigms and Mitigations

Mitigating these attack surfaces demands a defense-in-depth redesign, implementing rigorous cross-modal verification to match visual pixel data with accessibility trees, while leveraging execution-aware scheduling to inherently reduce GUI-layer risk [21, 9].

8. Conclusion and Future Directions

The trajectory of mobile agent research has pivoted from the historical challenge of migrating executing code across networks to the establishment of generalized, LMM-driven, user-centric autonomy natively integrated into complex smartphone ecosystems [5]. Recent progress shows that robust, high-performing systems combine multi-agent decomposition, verification, memory, and careful runtime systems design [15, 9, 11]. The aggressive hardware miniaturization of foundational neural models signals that the future of mobile autonomy is inherently edge-native [17, 18, 13, 16, 20]. Solving the critical vulnerabilities of transparent overlays and display-based prompt injections will require verifiable UI rendering pipelines, strict isolation protocols, and advanced multimodal models that can detect adversarial spatial manipulations natively [21].

References

[1] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2), 1995.

[2] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, 1995.

[3] Shunyu Yao et al. ReAct: Synergizing reasoning and acting in language models. arXiv:2210.03629, 2022.

[4] Noah Shinn et al. Reflexion: Language agents with verbal reinforcement learning. arXiv:2303.11366, 2023.

[5] Danny B. Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, 1999.

[6] Christopher Rawles et al. AndroidWorld: A dynamic benchmarking environment for autonomous agents. arXiv:2405.14573, 2024.

[7] Jianwei Yang et al. Set-of-Mark prompting unleashes extraordinary visual grounding in GPT-4V. arXiv:2310.11441, 2023.

[8] Kanzhi Cheng et al. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of ACL 2024 (Long Papers)*, 2024.

[9] Hongchao Du et al. ClawMobile: Rethinking smartphone-native agentic systems. arXiv:2602.22942, 2026.

[10] Sijia Li et al. Curiosity driven knowledge retrieval for mobile agents. arXiv:2601.19306, 2026.

[11] OpenClaw contributors. OpenClaw: Your own personal AI assistant. GitHub repository: openclaw/openclaw, 2026.

[12] Haiyang Xu et al. Mobile-Agent-v3.5: Multi-platform fundamental GUI agents. arXiv:2602.16855, 2026.

[13] Jiabo Ye et al. Mobile-Agent-v3: Fundamental agents for GUI automation. arXiv:2508.15144, 2025.

[14] DroidRun. Achieving 91.4% on Android World: A new approach to mobile UI automation. DroidRun Blog, October 3, 2025.

[15] Pierre-Louis Favreau et al. Do multi-agents dream of electric screens? Achieving perfect accuracy on AndroidWorld through task decomposition. arXiv:2602.07787, 2026.

[16] Haolong Yan et al. Step-GUI technical report. arXiv:2512.15431, 2025.

[17] Omar Sanseviero and Ian Ballantyne. Introducing Gemma 3n: The developer guide. Google Developers Blog, June 26, 2025.

[18] Liliang Ren et al. Decoder-hybrid-decoder architecture for efficient reasoning with long generation. arXiv:2507.06607, 2025.

[19] Shuai Bai et al. Qwen3-VL technical report. arXiv:2511.21631, 2025.

[20] Veuns-Team et al. UI-Venus-1.5 technical report. arXiv:2602.09082, 2026.

[21] Liangxuan Wu et al. From assistants to adversaries: Exploring the security risks of mobile LLM agents. arXiv:2505.12981, 2025.

[22] Earlence Fernandes et al. Android UI deception revisited: Attacks and defenses. In *Financial Cryptography and Data Security*, 2016.

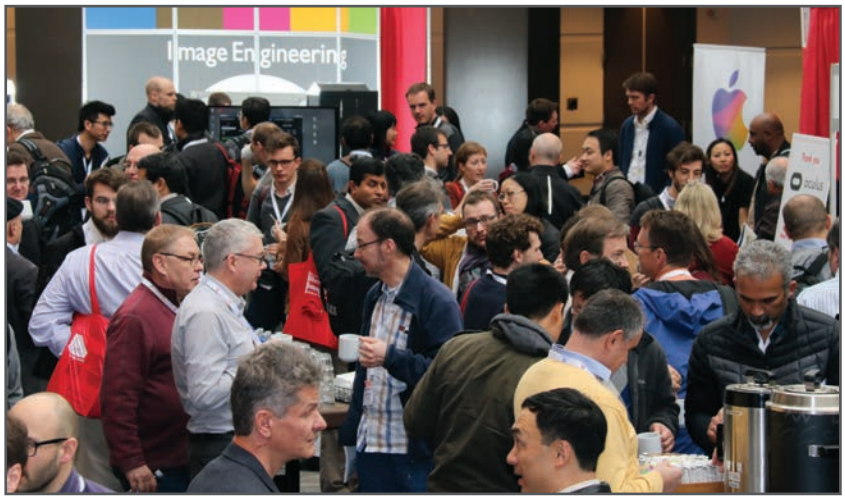
Author Biography

Jianhang Chen received his BS in Information and Computational Science from Beihang University, his MS in Robotics from National Taiwan University, and his PhD in Electrical and Computer Engineering from Purdue University. He has worked as a Senior Software Engineer on the AI Innovation & Research (AIR) team at Google in Mountain View, CA. His work focuses on the development of Gemini Nano and Gemma post-training, on-device intelligence, and hybrid computing.

JOIN US AT THE NEXT EI!

electronic IMAGING

Imaging across applications . . . Where industry and academia meet!



• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

ADVANCING IMAGING SCIENCE 75 1947 IS&T 2022 imaging.org