

# Demo: MobileClaw: A Notification-Driven, Human-Gated Mobile LLM Agent

Boyuan Yang  
University of Pittsburgh  
Pittsburgh, PA, USA  
by.yang@pitt.edu

Alexander Du<sup>†</sup>  
North Allegheny Senior High  
School  
Wexford, PA, USA  
alex.j.du2009@gmail.com

Ozzie Hassel<sup>†</sup>  
University of Pittsburgh  
Pittsburgh, PA, USA  
ouh4@pitt.edu

Jash Uganandan<sup>†</sup>  
University of Pittsburgh  
Pittsburgh, PA, USA  
jau63@pitt.edu

Wei Gao  
University of Pittsburgh  
Pittsburgh, PA, USA  
weigao@pitt.edu

## Abstract

Smartphone assistants built on large language models (LLMs) today either drive an application's screen display or wait to be asked through chatting; neither is integrated into the event stream produced by the mobile OS and user-installed applications. We demonstrate MobileClaw, an Android system component that attaches an external LLM to the OS notification channel. A background service captures every posted notification and asks the model for recommended further actions; an interactive front end carries out the user's requests through built-in skills, operated through OS API calls rather than screen automation. The model is treated as an untrusted component: nothing it decides takes effect unless the user confirms it. Notification text and everything the app reads from the device pass a sanitizer before they leave the phone, and privileged operations run in a separate process behind an allowlist it re-checks itself. Demo attendees may fire synthetic or real notifications, manipulate local resources on the device, approve or reject suggested actions, and watch the model's reasoning trace in real time.

## CCS Concepts

• **Human-centered computing** → Ubiquitous and mobile computing systems and tools; • **Computing methodologies** → *Intelligent agents*.

<sup>†</sup>These authors contributed equally to this work.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

MobiCom '26, Austin, TX, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2505-0/2026/10

<https://doi.org/10.1145/3795866.3848463>

## Keywords

Mobile Systems, Android Notifications, Language-Model Assistants

## ACM Reference Format:

Boyuan Yang, Alexander Du, Ozzie Hassel, Jash Uganandan, and Wei Gao. 2026. Demo: MobileClaw: A Notification-Driven, Human-Gated Mobile LLM Agent. In *The 32nd Annual International Conference on Mobile Computing and Networking (MobiCom '26)*, October 26–30, 2026, Austin, TX, USA. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3795866.3848463>

## 1 Introduction

LLM-based smartphone assistants are becoming a research interest due to their proximity to everyday life. Existing work takes two approaches: one drives an application's user interface from the screen [6, 7, 9]. Although it integrates with existing GUIs well, the operation could be slow and fragile under UI changes. The other exposes device functions as typed remote procedures, similar to Model Context Protocol (MCP) servers on the desktop [1]. AppFunctions on Android [4], as an example, demonstrates such a possibility; yet it requires every installed app to be adapted, which is not realistic in the near future. Instead, we find *notifications* on a mobile OS could act as a natural bridge between the user, the assistant and on-device applications, which is not well explored yet. This inspires us to demonstrate MobileClaw, a mobile assistant that connects LLM, OS, apps and the user together.

We present MobileClaw, a mobile LLM assistant with a strong focus on the OS notification channel. As an event-driven LLM agent, it listens to system notifications [3], reacts to the incoming notifications as well as the user's text inputs. Based on the context and system memory, the LLM suggests certain actions to the user for approval. MobileClaw also provides multiple tools to demonstrate its capability, which

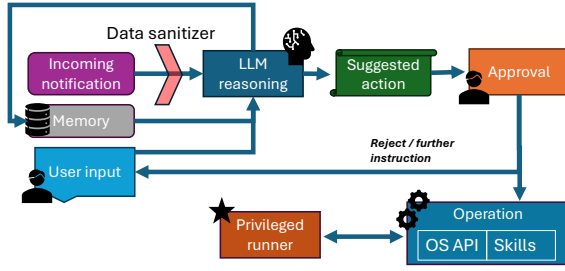


**Figure 1: Information and action flow within Mobile-Claw.**

are implemented as built-in Skills [2]. We design multiple showcases to demonstrate the capabilities of a notification-driven mobile LLM agent.

## 2 System Implementation

Figure 1 shows the basic design on the information and action flow within the system. Notifications and user inputs trigger LLM reasoning with local memory providing context. LLM-suggested actions, if approved by the user, get executed through Skills and OS integrations. To reduce attack surfaces, the system adopts safety measures including user approval gating, a data sanitizer on LLM inputs and an isolated privileged tool runner. The system is implemented as an Android App with 16484 LoC in Kotlin.

**Notification-triggered LLM Reasoning.** Our demo uses a cloud LLM, Gemini 2.5 Flash, for reasoning. The app talks to the model through the OpenAI-compatible chat interface, so the model can be replaced if needed, such as running a model on the phone itself or on a computer the user owns. We find that running the system on local models like Qwen3-8B [8] is possible albeit much slower with degraded tool call performance. The reasoning is triggered by notifications. When a new notification arrives, the listener service packages it into a request to the model, including the notification’s title, text, attached images, and up to ten other existing notifications. The last five decisions the model made, and the recent chat with the user are also provided. The model answers with a suggested action, which is one of *reply* (with the drafted reply text), *dismiss*, *summarize*, *escalate*, *snooze* (with a duration), *press a button* of the notification, or *do nothing*, together with a priority, a confidence between 0 and 1, and a one-sentence reason. The app checks this answer and only acts when the confidence is at least 0.5. Every decision is stored and shown to the user together with the model’s reason. Besides the notification itself, the model’s request carries local memory maintained by the model itself. Memory lives in an on-device database with three parts: past decisions, the chat history, and a set of facts about the user. The facts are

written and read through three functions that the model can call, remember, recall and forget, in the same way as any other tool.

**Tool Integration.** The actions available to the model are implemented in the app and grouped into *skills*. Our skill is a Kotlin module inside the app that declares a set of functions, callable as tools. Some skills are thin wrappers over OS APIs, including messaging, device usage retrieval, calendar access, media control, etc. Other skills are designed to demonstrate the system’s extended capabilities: file search runs over an on-device AppSearch index that the app builds in the background; and on Android 16 the app can call functions that other apps expose through AppFunctions [4]. At the time of writing there are eleven skills with 23 functions in total. To give a few examples: *reply\_notification*, *create\_event*, *music\_control*, *search\_files*, *kill\_process*, etc.

**Safety Approaches.** Since the model is given real control over the phone, the design places three checks between the model and the device. First, the system is user-gated. A function with an effect outside the app always asks for user approval in a dialog or notification. In the settings, the user can turn this prompt off for routine actions. Even then, a dangerous operation such as deleting or overwriting files still requires user confirmation. Every proposal and its outcome goes to a log that the user can read in the app. The dialog shows what will actually happen, such as the recipient and the text of a message, the path and the content of a file to be written, or the exact command line of a privileged command.

Second, notification content is filtered before it reaches the cloud model. A sanitizer replaces one-time codes, card and account numbers, PINs, phone numbers, e-mail addresses, license keys and credential strings with placeholders. It is applied to notification text, to anything the app reads from the device on the model’s behalf, such as files and search results, and again before the model’s output is written to the on-device database. The sanitizer works on text only with our current implementation, and the text typed by the user is provided to the model verbatim. A fact that the model wants to remember is sanitized before it is stored, with one exception: a fact is stored unchanged if the user explicitly typed it in the chat. A fact the model inferred on its own reaches the model only when the model asks for it with the recall tool.

Third, privileged access is kept in a separate process. Operations beyond a normal app’s permissions are carried out by a small helper process through Shizuku [5]. The app talks to the helper over Binder with an argument-vector interface to avoid command injection. The helper also checks every command against its own allowlist of programs before running it. On a custom ROM, the same helper could be installed as a privileged system component.

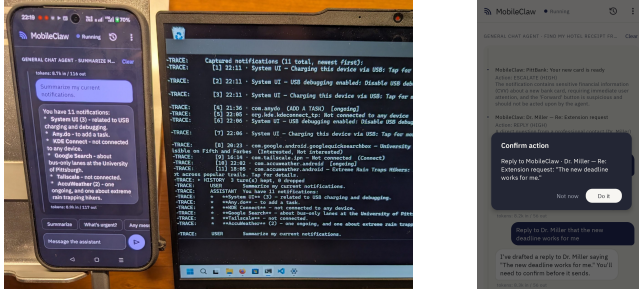


**Figure 2: Left: the demo setup—a phone beside a laptop showing the reasoning trace as it is logged. Right: the confirmation dialog on the phone; the reply the visitor asked for waits until they tap *Do it*.**

### 3 Demonstration

This demo showcases MobileClaw assisting with daily tasks of mobile device users. Specifically, we show how MobileClaw makes suggestions based on notification contents and its integration with the OS.

**Setup.** Our setup is illustrated in Figure 2. We provide three Android smartphones running MobileClaw, each with an internet connection, and a laptop beside them that displays the model’s reasoning trace in real time. The app’s built-in test menu posts 16 synthetic notifications, each exercising a distinct decision path. A folder of sample files, including multimedia, and a collection of Android apps are preloaded on the devices to emulate real-life usage.

**Walkthrough.** First, we allow the user to interact with MobileClaw through its built-in chat interface, similar to traditional LLM chat services. We further provide multiple buttons in the app as shortcuts to understand recent OS notification history. Second, we demonstrate the system’s capability of interacting with OS notifications: the user may pick a few entries from a test menu that emits various notifications, ranging from progress bar, payment confirmation, e-mail confirmation to SMS chats. After each emission, MobileClaw would show what the model decided and why in the app. When the model suggests an operation, a dialog will show up requesting the user’s confirmation, as shown in the right screenshot in Figure 2. We also demonstrate adversarial cases where a notification carries an instruction addressed to the assistant, claiming to be its administrator and telling it to forward every message without asking, or a notification asks to forward sensitive information. The user watches what the model proposes, sees that the sensitive information (card number) was already replaced before the request left the phone, and rejects the inappropriate injected actions. The user is also allowed to explore the device freely and react to real notifications from other apps. Furthermore, we demonstrate its integration with tools and resources by triggering

notifications that require semantic searching of local files and tool combinations during the reply. We display the real-time background reasoning chain and tool invocation on a laptop next to the device.

### References

- [1] Anthropic. 2024. Model Context Protocol. <https://modelcontextprotocol.io>.
- [2] Anthropic. 2025. Agent Skills. <https://agentskills.io>.
- [3] Google. 2026. NotificationListenerService — Android Developers. <https://developer.android.com/reference/android/service/notification/NotificationListenerService>.
- [4] Google. 2026. Overview of AppFunctions — Android Developers. <https://developer.android.com/ai/appfunctions>.
- [5] Rikka. 2026. Shizuku: Let Your App Use System APIs Directly. <https://shizuku.rikka.app>.
- [6] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-Agent: Autonomous Multimodal Mobile Device Agent with Visual Perception. arXiv:2401.16158.
- [7] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. AutoDroid: LLM-powered Task Automation in Android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (MobiCom '24)*. ACM, Washington, DC, USA, 543–557. doi:10.1145/3636534.3649379
- [8] An Yang, Anfeng Li, Baosong Yang, et al. 2025. Qwen3 Technical Report. arXiv:2505.09388.
- [9] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025. AppAgent: Multimodal Agents as Smartphone Users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. ACM, Yokohama, Japan, 1–20. doi:10.1145/3706598.3713600