Poster: Efficient and Accurate Mobile Task Automation through Learning from Code

Shihe Wang, Li Zhang, Mengwei Xu
Beijing University of Posts and Telecommunications
{shihewang,li.zhang,mwx}@bupt.edu.cn

ABSTRACT

With the emergence and continuous prosperity of large language models (LLMs), artificial intelligence (AI) agents have experienced rapid advancements. Most mobile AI agents merely imitate human operations, executing actions based on the human user interface (UI). The restricted input impairs the efficiency and accuracy of mobile tasks. We propose an unexplored approach: learning from the source code. Source code is the plain interaction for mobile applications, which can be used to enhance the UI understanding of mobile agents, improve action execution accuracy, and reduce the average action completion steps. The implementation of the agent prototype is preliminary evaluated on 5 open-source applications and 22 tasks, reducing the average number of task completion steps by 54%.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing**; • **Computing methodologies** → *Artificial intelligence.*

KEYWORDS

Task automation, large language model, code execution

ACM Reference Format:
Shihe Wang, Li Zhang, Mengwei Xu. 2024. Poster: Efficient and Accurate Mobile Task Automation through Learning from Code . In *ACM International Conference on Mobile Systems, Applications, and Services (Mobisys '24), June 3–7, 2024, Minato-ku, Tokyo, Japan.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3643832.3661397

1 INTRODUCTION

The emergence of large language models (LLMs) has revolutionized mobile agents with their superior performance of understanding mobile UI and behaving like human for daily tasks [3]. By instructing mobile agents a task description in natural language, LLM-powered mobile agents will predict and act a sequence of actions based on UI representations. As shown in Figure 1, to complete the task *"check my recycle bin in SMS Messages"*, an LLM-powered mobile agent may execute three sequential click actions to the "Recycle Bin" screen of the application.

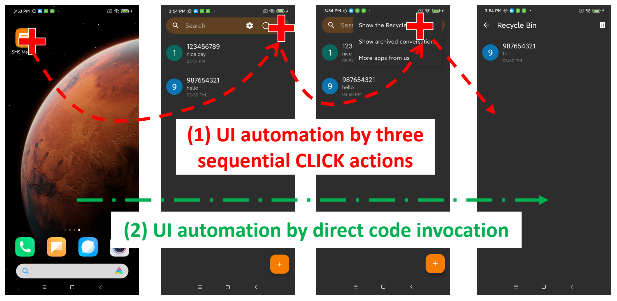
(1) UI automation by three sequential CLICK actions
(2) UI automation by direct code invocation
Search
123456789
987654321
Show the Recycle Bin
Show archived conversations
More apps from us
Recycle Bin

Figure 1: A task "check my recycle bin in SMS Messages" could be completed by (1) three sequential click *operations on the UI* or (2) direct screen navigation by *invoking codes to start an activity*, where the knowledge is learned from the source code.

However, mobile task automation by merely relying on UI operations may lead to inefficiency and inaccuracy. (1) A lengthy execution path is prone to error in individual steps. It significantly increases the likelihood of errors from mobile agents, leading to end-to-end task failure. (2) Mobile agents usually require to ingest a mobile UI representation (e.g., the screenshot or view hierarchy (VH) of the current screen) at each step for action prediction. It will result in high response latency and computation overhead. (3) Current mobile applications are highly complex: they usually offer a wide range of functionalities; some features may require multiple levels of navigation to find, which can sometimes be challenging even for humans. Without sufficient knowledge, LLMs are unable to make sound decisions during the task execution process.

To solve these problems, we propose an efficient yet unexplored approach, learning from the source code of mobile applications, to enhancing capabilities of mobile agents in task automation. Application source code can assist the mobile task automation process for several reasons: (1) The source code of an application not only contains functional implementation, but also descriptions and correlations of UIs (e.g., exposed by different activities). A previous study [4] indicates that LLMs can treat APIs as an extension of tools for invocation, thereby enhancing task completion capabilities. Learning from source code enables mobile agents to better understand and surf inside a complex application. (2) With enough knowledge learned from the code, mobile agents could generate and execute code to fast complete a task by skipping cumbersome UI operations. This can be done by invoking application APIs or directly starting a specific activity for screen navigation.

During the prototype implementation, we initially extracted key information from the source code. Given the complexity and diversity of Android application source code, we simplified its structure by extracting all activities, leveraging the few-shot capabilities of

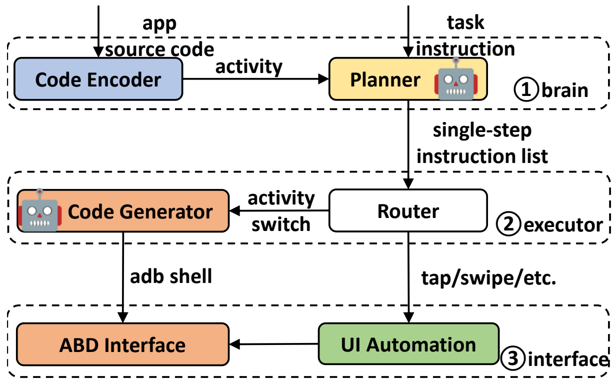

Figure 2: The system architecture.

LLMs and the tree structure of the source code. Subsequently, we employed task planning that breaks down natural language instructions into single-step actions, making them more comprehensible to LLMs. Finally, we implemented automatic adb shell invocation for activity switching.

This approach is currently limited to open-source applications. Developers are encouraged to share their code as applications grow more complex and super apps increasingly require in-app agents for task automation. Sharing source code can reduce the construction costs of in-app agents and improve their efficiency and accuracy during execution.

Contribution. (1) We observe the inefficiency of relying solely on UI operations of existing mobile agents for task automation. (2) We propose an approach that learns from source code to enhance UI understanding and automation capabilities of mobile agents. (3) We exemplify the approach through screen navigation, a fundamental component of most UI automation tasks, to demonstrate the potential of our approach. Our evaluation on 5 real-world mobile applications shows that our approach reduces the average number of task completion steps by 54%, from 2.31 to 1.04. Our straightforward design of mobile task automation through learning from code achieves task completion rate exceeding 59%.

2 DESIGN

Overall Workflow. We employed a three-layer architecture for mobile agent design. The core architecture is shown in Figure 2. From top to bottom, the three layers are the brain, executor, and interface. The brain layer is responsible for receiving application source code and task instructions. It generates an action plan by breaking down a natural language-based task description into concrete, individual actions to be performed. This action plan is then sent to the executor layer, where the router determines whether a task can be completed directly through code execution or requires UI operations such as tapping and swiping. For tasks executable by code, it generates the target code; otherwise, it initiates the native UI automation process. The interface layer is responsible for interacting with Android devices, including executing `adb shell` commands.

Brain. We adopt multiple representations of UI elements within a specific app. The source code could enhance the UI understanding of mobile agents which powered by LLMs. The app's resource code is initially processed by a code encoder. The AndroidManifest file serves as the code input, and the code encoder extracts activity and intent information using the DOM tree of XML. Each activity is incorporated into the system prompt and sent to the LLM as a planner. The planner receives human natural language instruction as a user prompt, combining the code and natural language into the LLM prompt. The planner will generate the initial action sequence and send it to the executor.

Executor. The executor executes the action according to the action plan and current UI representation. The current UI representation includes the current activity, a pixel-level screenshot, and a VH. The executor decides the current action, and sends either an activity route or a screen action (e.g., click or swipe) along two different paths. The activity route path is processed by a code generator that generates an adb shell command to jump to the target activity. The screen action path is processed by the screen action interface.

Interface. The adb interface sends the adb shell command to Android devices. The screen action interface can utilize existing agents [2] to execute actions on-screen.

Apps	Task Counts	Baseline Steps	Steps	TCR
SMS Message	5	12	5 (58%↓)	80%
Calendar	4	9	4 (55%↓)	50%
Gallery	5	14	5 (64%↓)	60%
Contacts	4	9	4 (55%↓)	50%
Notes	4	7	5 (28%↓)	50%
Total	22	51	23 (54%↓)	59%

Table 1: Evaluation results for 5 real-world open-source applications. 1) Average reduction rate of task completion steps using invoking code. 2) Average task completion rate (TCR).

3 PRELIMINARY EVALUATION

Table 1 shows the results of our preliminary experiments. The mobile agent prototype is evaluated on 5 real-world open-source applications [1] involving 22 tasks[1]. We used the number of actions for completing a specific task by humans as the baselines steps. The planner of the brain is based on OpenAI's gpt-3.5-turbo model. In summary, our approach skips lengthy steps through app API invocation, and reduces the average number of task completion steps by up to 54% and achieves an average task completion rate of 59%. In the future, we plan to apply PEFT to frequently updated application code bases and fully utilize edge hardware accelerators for efficient mobile task automation.

REFERENCES

[1] 2019. Simple Mobile Tools. https://github.com/SimpleMobileTools.

[2] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 2023. Cogagent: A visual language model for gui agents. *arXiv preprint arXiv:2312.08914* (2023).

[3] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. 2024. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459* (2024).

[4] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789* (2023).

[1]https://github.com/LlamaTouch/LlamaTouch/tree/main/dataset/SimpleTools