Recovering Internal Command Interfaces from Applications for Direct Agent Invocation

Prince Modi[1], Varad Kottawar[1], and Ayush Govind[1]

[1]University of California San Diego

Note: Prince Modi, Varad Kottawar, and Ayush Govind are graduate (MS) students.

1. Introduction

Computer-use agents (CUAs) increasingly rely on large language models (LLMs) to operate desktop applications by emulating human interaction: perceiving the screen, locating on-screen elements, and issuing sequences of clicks, drags, and keystrokes. This GUI-based approach forces the LLM to decompose even simple tasks into long, fine-grained action chains, inflating the number of LLM calls required per task and compounding the risk of cascading failures when any single step is misinterpreted or mislocated.

Recent work has proposed abstracting this GUI navigation burden away from the LLM. In particular, DMI [7] introduces a declarative interface layer built on top of the OS accessibility APIs (e.g., Windows UI Automation [5]), allowing an LLM to specify a desired control state rather than the low-level navigation sequence needed to reach it. While this substantially reduces interaction steps and improves task success rates, it remains fundamentally dependent on the completeness and correctness of the application's accessibility metadata, and still routes every action through the accessibility layer rather than the application's own internal logic.

This raises a natural question: if the goal is to let an LLM invoke application functionality directly, why stop at the accessibility layer? Many desktop functions are backed by internal function calls that GUI controls and accessibility elements trigger indirectly. We investigate the feasibility of recovering these internal command interfaces directly, using dynamic instrumentation (Frida [2]) to hook and invoke an application's internal functions, bypassing both the GUI and the accessibility tree entirely.

We hypothesize that recovering and directly invoking internal command interfaces offers a lower-overhead, more direct invocation path for CUAs than either GUI-based or accessibility-based approaches, particularly for applications where accessibility metadata is sparse or unavailable.

2. Approach

We use Frida to recover callable internal interfaces from an existing desktop application through dynamic instrumentation. We attach Frida to a running application (GIMP [3] in our case) and use its Stalker component to record the execution trace generated while a user manually performs a target operation (e.g., cropping or flipping an image in GIMP).

We identify the executed basic blocks corresponding to the user action. For each candidate block, we resolve the associated instruction addresses to symbols when symbol information is available, producing a set of candidate functions involved in the operation. We then instrument these functions to capture their invocation context, including the argument values observed during normal execution.

Using the recovered function addresses together with the observed calling arguments, we construct direct invocations of the same internal functions through Frida. These invocations are issued programmatically, without interacting with the application's GUI. In our current prototype, the tracing, symbol resolution, argument capture, and function invocation stages are implemented using Frida with Python and JavaScript. Generative AI tools are utilized to assist in writing the code for this Frida-based instrumentation prototype.

3. Proof of Concept

We validated our approach on the following operations in GIMP: image cropping, horizontal and vertical flipping, and rotate operations. For each, we used Frida's Stalker component to trace the basic blocks executed during a manual invocation of the corresponding menu action, then resolved the executed addresses to symbols where available. We figured it out empirically that the flip operation's entry point was reached via a tail call rather than a direct call instruction, and was therefore invisible to a Stalker configuration that only records stack-frame creation events. Enabling block-level

Method	Tokens	Time (s)
trycua/cua (baseline)	40k	897
trycua/cua + (our hook)	13k	329

Table 1: Comparison of tokens consumed and wall-clock time across baseline and hooked configurations (mean over 10 runs).

event tracing resolved this.

Having identified candidate functions, we next recovered their calling signatures empirically. For `gimp_image_crop`, we hooked the function and compared captured argument values against crops performed with known, operator-specified geometry (e.g., position $(10, 20)$, size 100×50), which confirmed the argument ordering.

Using the recovered function addresses and calling conventions, we constructed direct invocations of the operations via Frida's `NativeFunction` interface. The invocations produced the expected visible effect on the target image with no GUI interaction, and in all the cases, the operation's undo and redo behavior remained intact and correctly interleaved with subsequent manual edits performed through the GUI, which we attribute to the operations managing their own undo bracketing internally rather than relying on caller-side setup

4. Evaluation

For evaluation, we tested our approach by providing the hooks we generated to the CUAs as an additional tool, callable on top of the existing tools already available to the agent.

We measured tokens consumed and wall-clock time over 10 runs for a task similar to the following:

```
task = (
   "Launch GIMP, and wait for it to fully load. "
   "In GIMP, do exactly these steps in order. "
   "1. Select the Crop tool. "
   "2. On the canvas, drag from pixel (x1,y1) to pixel
        (x2,y2) to define the crop rectangle. "
   "3. Press Enter to commit the crop. "
   "4. Save the cropped image."
)
```

We evaluated this using `trycua/cua` [6], configured to use Claude Code.[1]

We first measured the two metrics above as a baseline, then provided the agent with our tool and measured the same two numbers over 10 runs. Table 1 summarizes these results.

[1]We also attempted this evaluation using a locally hosted Qwen2.5-VL-32B-Instruct [1] model served via vLLM [4] with tensor parallelism across two L40 GPUs; all GUI-only (baseline) runs failed under this setup, so we did not proceed to test the hooked configuration with this model.

5. Discussion

We were able to successfully provide a hook for a task using our approach; as observed, the task showed a faster end-to-end time, along with decreased token usage.

However, a key open question for this approach is how well it generalizes beyond the single application we evaluated. For example, GIMP manages its own undo stack internally within each function call we hooked, meaning that invoking its internal functions directly still preserves the application's expected state-management behavior without additional effort on our part. There may be applications where this is not true, in which case the functionality of this approach may break down beyond the scenario we studied.

6. Future Directions

Our preliminary result validates this approach for a few tasks on a single application; extending it to a general-purpose system requires automating the discovery of internal command interfaces, rather than manually identifying and hooking individual functions as we did here. We envision two complementary directions for this automation. Where an application's source is available, its code-base could be analyzed directly to identify candidate functions corresponding to user-facing functionality, rather than relying on manual inspection. Where source is unavailable, automated exploration of the compiled binary may offer a viable alternative, though the specific mechanism for this remains an open question we intend to explore. Realizing either direction would move this approach from a small set of hand-hooked examples toward a generalizable framework applicable across different applications.

7. Acknowledgments

Generative AI tools were used to assist in implementing the Frida-based instrumentation prototype and for language editing. The research idea, experimental design, evaluation, and conclusions were developed by the authors.

References

[1] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

[2] Frida. Frida: A world-class dynamic instrumentation toolkit. https://frida.re. Accessed: 2026-07-10.

[3] GIMP Development Team. GIMP: GNU Image Manipulation Program. https://www.gimp.org. Accessed: 2026-07-10.

[4] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.

[5] Microsoft. Windows UI Automation. https://learn.microsoft.com/en-us/windows/win32/winauto/entry-uiauto-win32. Accessed: 2026-07-10.

[6] trycua. Cua: Open-source infrastructure for computer-use agents. https://github.com/trycua/cua, 2025. Accessed: 2026-07-10.

[7] Yuan Wang, Mingyu Li, and Haibo Chen. From Imperative to Declarative: Towards LLM-friendly OS Interfaces for Boosted Computer-Use Agents. In *Proceedings of the 21st European Conference on Computer Systems*, EUROSYS '26, page 296–310, New York, NY, USA, 2026. Association for Computing Machinery.