HIDAgent: A Platform-Agnostic Hardware Bridge for Personal Computer Use Agents

Jeffrey P Bigham
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
jbigham@cs.cmu.edu

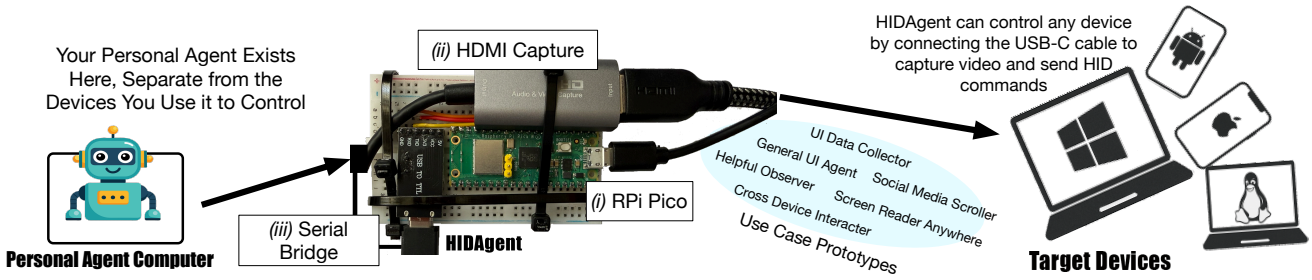


Figure 1: HIDAgent makes computer use agents "personal" by separating them from the devices they control. It is built from three hardware components (*(i)* a RP2040 microcontroller, *(ii)* an HDMI to USB converter, and *(iii)* a CH340 USB-to-Serial bridge) and a Python library we introduce to make prototyping new personal computer use agents easy with a set of functions designed to support screen interpretation and general HID (mouse/keyboard) emulation. We introduce six use case prototypes that demonstrate some of the interaction scenarios that are opened up by moving computer use agents to a separate device.

Abstract

Computer Use Agents powered by performant VLMs are increasingly capable of visually interpreting GUIs and executing actions like humans do. However, their method of controlling UIs remains tethered to system-specific APIs or VNC connections, limiting agents to specific platforms and open OS environments. This paper introduces HIDAgent, a platform-agnostic hardware bridge that allows agents to observe any device via HDMI capture and control it via emulated HID mouse and keyboard events—all over a single physical connection. Built for less than $30 using off-the-shelf components, HIDAgent enables agents to operate across mobile, desktop, and even legacy or "closed" systems without local software installation. We demonstrate the flexibility of this approach through six diverse use case prototypes, ranging from cross-device automation to accessibility. By decoupling the agent from the host OS, HIDAgent facilitates research into a future where personal agents are mobile and independent of the devices they control.

CCS Concepts

• **Computing methodologies** → **Intelligent agents**; • **Human-centered computing** → **User interface toolkits**.

This work is licensed under a Creative Commons Attribution 4.0 International License.
UIST '26, Detroit, MI, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2856-3/26/11
https://doi.org/10.1145/3830398.3830721

Keywords

computer use agents, GUI agents, HID emulation, toolkit

ACM Reference Format:
Jeffrey P Bigham. 2026. HIDAgent: A Platform-Agnostic Hardware Bridge for Personal Computer Use Agents. In *The 39th Annual ACM Symposium on User Interface Software and Technology (UIST '26), November 02–05, 2026, Detroit, MI, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3830398.3830721

1 Introduction

Computer Use Agents have become increasingly capable [5, 7, 11, 29], with multiple research prototypes and products promising to complete arbitrary computer tasks on a user's behalf (*e.g.*, OpenAI Operator [25], Anthropic Computer Use [2], Vercept Vy [35], Taxy AI [31]). While end-to-end task completion is imperfect [40], performance is impressive and continues to improve. UI Agents have also progressed toward using computers more like people do—by visually interpreting graphical user interfaces (GUIs) on screen and issuing appropriate actions to control them (*e.g.*, mouse clicks and keyboard entry) [36]. Interestingly, even as UI Agents have improved, the model for how people are assumed to interact with these agents has not changed all that much.

In this paper we introduce an alternative model that we call, "Personal Computer Use Agents", in which the computer use agent is under the control of the user and goes with the user from device to device. This differs from how Computer Use Agents operate today, since existing agents generally must be installed on a target computer (or connected to a remote computer using VNC) [28, 41], and cannot easily be moved from one device to another (*e.g.*, a personal iPhone to a Windows laptop to a friend's Android phone).

To enable this, we introduce HIDAgent, which works differently than existing computer use agent stacks because it is a separate hardware device that operates devices it is physically connected to.

Moving UI Agents to a separate hardware device allows research into underexplored interaction scenarios, such as,

- scenarios in which a user wants to or benefits from keeping their agent separate from the system being operated (*e.g.*, when users don't trust the target computer, or in "bring your own agent" interactions);
- scenarios in which the target system does not allow custom agent software to be installed or installing software on it might be a hassle (*e.g.*, UI agent user studies on participants' personal devices, platforms that would otherwise prevent agents other than its preferred agent from running); and,
- scenarios that benefit from the same agent being able to control multiple different kinds of computers (*e.g.*, share an agent and its memory between your Android phone and your Windows laptop) and/or for control to be easily moved from, (*e.g.*, an iPhone to your Linux desktop)

HIDAgent is an open-source external hardware/software toolkit that operates HID-compatible *target* computing system by directly receiving the pixels of the screen via HDMI capture and then sending appropriate keyboard and mouse commands directly as Human Interface Device (HID) inputs. HIDAgent is built using three off-the-shelf components costing less than $30 and a supporting Python library. HIDAgent's novel quality is that it allows UI Agents created using it to operate mobile phones (iOS, Android, etc), desktop computers (Mac, Windows, Linux) and even other kinds of devices (*e.g.*, virtual reality headsets) by plugging its USB-C cable into a different device. The HIDAgent code and hardware instructions are released as open source[1] for researchers to use to investigate new ways of interacting with UI agents.

In addition to the idea of a "personal computer use agent", a primary contribution of this paper is thus the HIDAgent toolkit, which allows a control computer to capture the pixels of the screen from target devices and issue HID commands to operate them. We further introduce a procedure to determine the correspondence between pixels on captured screenshots to HID space, which is necessary for a generalized solution. Toolkit contributions in HCI "are evaluated in a holistic fashion according to what they make possible and how they do so" [38]. Accordingly, to validate HIDAgent, we built six use case prototypes using it, each of which demonstrates different aspects of what is made possible by separating agent control into an external hardware device. As examples, our "Helpful Observer" prototype watches the screen until certain criteria is met that could make it useful for the agent to intervene – we explored this in a "bring your own agent" accessibility application. Our "Extensible UI Agent" prototype allows existing computer use agents to control any device, enabling researchers to, *e.g.*, conduct studies on the personal devices of participants or conduct studies beyond the restricted environments of current UI agent stacks [28, 41].

2 Related Work

UI Frameworks and Toolkits that control and computationally represent graphical user interfaces (GUIs) were created soon after the first GUIs [22]. Simulating human input device (HID) events (*e.g.*, mouse movements, keyboard events, etc) with software is well-supported on modern platforms, often via accessibility APIs, although usually requires special permissions on the host computer. Software such as the PyAutoGUI Python library [30] unifies support for keyboard and mouse simulation across desktop platforms (Mac, Linux, Windows). Some platforms, *e.g.*, iOS, do not allow third party use of their APIs. Instead of simulating input devices in software, HIDAgent operates as an external HID device, appearing to the target computer as a regular mouse and keyboard, which means no software needs to be installed. As a result, HIDAgent can control any device that can be operated with a keyboard and mouse (including mobile devices). In this way, it follows from hardware devices (KVM over IP) popular in IT management for remote computer control that transmit video and keyboard/mouse commands (*e.g.*, [27, 33]), although these require a human operator.

General computer use agents must understand the GUI. Platforms provide semantic representations of UI elements via APIs such as the DOM on the web and accessibility APIs on other platforms, which form the foundation of most automation approaches [14]. However, these APIs only work reliably when developers provide complete metadata, and more than 90% of iOS screens [44], Android screens [6], and web pages [37] contain one or (more often) many errors. As a result, *Understanding User Interfaces from Pixels* has become a common approach for reliable access to GUIs as computer vision has improved. Early work in this space built from predefined visual templates [12, 42]. Screen Recognition was one of the first comprehensive approaches for identifying all elements in a GUI screenshot using computer vision [44] (released in iOS 14 VoiceOver). A related problem is UI element grounding, which is the connection of a natural language description to a specific coordinate in a screenshot image (*e.g.,* "the hamburger menu" $->$ (800,100)), captured in popular benchmarks like ScreenSpot [8] and ScreenSpot-Pro [18]. Visual language models (VLMs) have shown continued improvement on UI grounding [3, 9, 20, 43], with top performing models like Qwen3-VL 30B-A3B nearly saturating the ScreenSpot benchmark (94.5) [3]. HIDAgent only has access to the pixels of the screen, but this has started to work as well as operating over unreliable semantic representations.

UI Automation and UI Agents have a long history in HCI, from early macro systems [17, 19] to web agents that leveraged the DOM for inspection and manipulation [1, 4, 17, 23]. Automatic understanding of GUIs from their pixels now makes a similarly reliable GUI representation available across platforms. Several benchmarks have been released for UI agent task completion [5, 7, 11, 29], which generally evaluate an agent's ability to successfully complete a given task based on a provided description, *e.g.*, "update the cells in the spreadsheet labeled 'UIST 2026 revision' with the status DONE". Although performance continues to improve, UI agents still fail to reliably complete benchmark tasks of even moderate complexity [29]. Recent cross-platform benchmark environments, such as Mobile-Agent-v3 [41] and UI-TARS [28], do so with either target virtual machines or specialized software that must be installed. As a result, HIDAgent may make development of future cross-platform benchmarks easier (especially on currently unsupported platforms, like iOS). More importantly, it may encourage benchmarks for new interaction scenarios supported by HIDAgent.

[1] https://github.com/CMUBigLab/hidagent/

3 HIDAgent

The HIDAgent toolkit (Figure 1) allows target devices to be observed and operated from a control computer. HIDAgent is composed of both hardware sufficient for capturing screenshots from devices with USB-C/HDMI video output and for performing HID input actions. The included Python library supports using this hardware to create a wide variety of different agents (Section 5). The HIDAgent toolkit hardware is made up of three off-the-shelf hardware components: *(i)* a Raspberry Pi Pico (RP2040) microcontroller, *(ii)* an HDMI to USB converter, and *(iii)* a CH340 USB-to-Serial bridge.

3.1 Simulating HID Events

Most computers cannot directly serve as HID output devices because their USB chips are limited to receiving HID input, but many low-cost microcontrollers have this capability. We used a Raspberry Pi Pico (RP2040)[2] to programmatically simulate HID events (~$4 USD). The RP2040 only includes one USB-C port, which is used as the HID output device connected to the target computing system. We therefore needed another way to connect the control computer to the RP2040. We used the built-in Universal Asynchronous Receiver/Transmitter (UART) functionality on the RP Pico to receive serial commands from the control computer. Many modern laptops also cannot do this directly, and so we used a low-cost USB-C to TTL UART serial bridge. For this, we chose the CH340[3] (~$6 USD), although there are other comparable options.

We wrote custom controller code for the RP2040 in CircuitPython[4], which provides a limited version of the Python language able to run on the RP Pico microcontroller. This code primarily *(i)* receives serial commands over UART sent from the control computer and replies back to indicate the commands were sent, *(ii)* sends the HID commands to the target device, and *(iii)* provides convenience functions (*e.g.*, run_app) and smooths subsequent commands for more reliable control. Commands are sent via JSON, *e.g.*:

```
send: {"type": "click", "x": 121, "y": 2145}
```

One challenge we ran into initially was that the target devices would get confused if the HID commands were sent too quickly, since humans do not interact so quickly, and so the RP2040 code includes appropriate delays built in, implemented via short sleep commands of, *e.g.*, 0.1 seconds. Because these delays are only introduced between commands, they do not introduce noticeable latency. Commands take 0.6 seconds on average to execute start to finish, which is sufficient for computer use agent control.

3.2 HIDAgent.py

The control computer uses core functions provided by HIDAgent.py to receive and process visual screenshots from the target device, which include (Appendix A): get a screenshot from the device, localize an image patch on the screen, recognize the GUI elements on the screen, and perform a diff between two screenshots, move the mouse to a specific location, click the mouse button at a specific location, perform a keypress (including combo keys), perform a drag from a first location to a second location, and run a specified application. To support debugging, HIDAgent.py provides an optional web-based logging interface on localhost, implemented with Python's http.server, which we found helpful to keep track of GUI states and commands sent, since they can go by quickly on the target device. Each entry in the timestamped visual log contains the captured screenshot, a graphical mark on the screenshot of any command performed on that screen, and a textual log of the command issued and other customizable metadata. HIDAgent also supports sending commands manually to the target device using this interface: clicks on the displayed screenshot are forwarded on to the target device, along with any captured keyboard events.

3.3 New Device Calibration

HID devices operate using relative input that differs based on operating system scaling and acceleration (easy for a human operator, but not supported by existing computer use agents). Thus, HIDAgent cannot natively target specific coordinates without calibration. To address this, we developed an automated procedure that maps HID increments to screen pixels. We first capture 1920x1080 HDMI screenshots and crop out black borders to isolate the active display. During calibration, the cursor moves independently along each axis while dense optical flow detects motion [13], which remains robust regardless of cursor shape or color (maintaining generality across cursor sizes/shapes). To ensure accuracy, we average three frames at each position to reduce camera sensor noise/variation and apply linear regression with iterative outlier rejection to the resulting data. This mapping is then verified by moving the cursor to five key locations and clustering the detections to filter out any measurements exceeding a 50px error threshold. To achieve reliable positioning from a known starting point, HIDAgent first resets the cursor to the $(0, 0)$ origin by over-indexing "up" and "left" movements, relying on the OS to clip the coordinates at the screen boundary. To bypass non-linear acceleration curves that would otherwise distort the pixel-to-HID mapping, large movements are decomposed into rapid, uniform 10px steps. This approach ensures predictable, linear travel across the screen with minimal latency (< 2.0 seconds even for long traversals).

3.4 Platform Considerations

HIDAgent is inherently cross-platform because it emulates the mouse and keyboard. Many common platforms work seamlessly, although users may need to set appropriate settings or permissions to allow it work. iOS required the most set up – we needed to enable both Assistive Touch and Allow Full Keyboard access in the accessibility settings to allow for mouse and keyboard control, although video capture via HDMI worked without setup. Android required us to "Allow Video Mirroring" via a dialog that appeared when we connected HIDAgent before it would work. Both Mac and Windows displayed keyboard configuration windows when connecting HIDAgent, but both could be dismissed or ignored. HIDAgent also currently requires a wired USB-C connection.

To characterize the latency of HIDAgent's pipeline components, we ran 100 trials using a Mac (M3) control computer and an iPhone 17 target device. Screen capture took 39 ms (SD 0.93), sending and receiving confirmation of a keypress took 57 ms (SD 2.3), a `click_mouse(100,100)` command completes in 504 ms (SD 1.6),

[2] https://www.raspberrypi.com/products/raspberry-pi-pico/
[3] https://cdn.sparkfun.com/datasheets/Dev/Arduino/Other/CH340DS1.PDF
[4] https://circuitpython.org

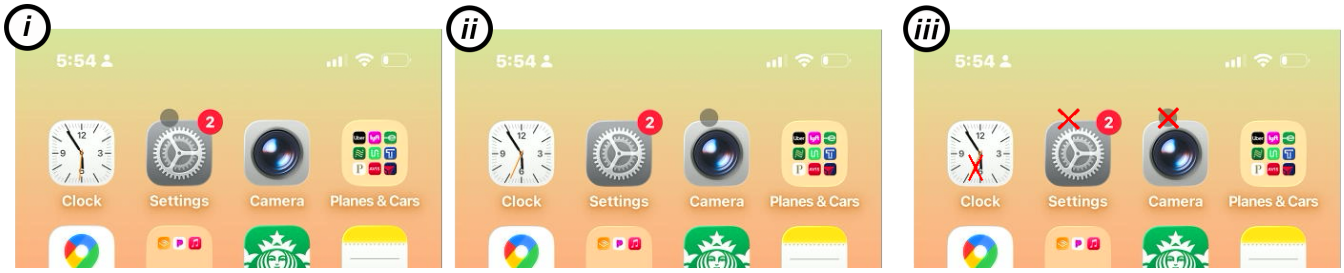


Figure 2: HID space to pixel space calibration is done by observing the cursor's movement in response to HID mouse move events. This example shows the faint cursor on an Apple iPhone screen before *(i)* and after *(ii)*. The pixel locations that were automatically detected to have changed are shown in the third image marked with red x's *(iii)*, which includes pixel changes detected in the clock icon because the seconds hand has moved. Our calibration procedure uses optical flow and multiple image capture for robustness to extraneous changes.

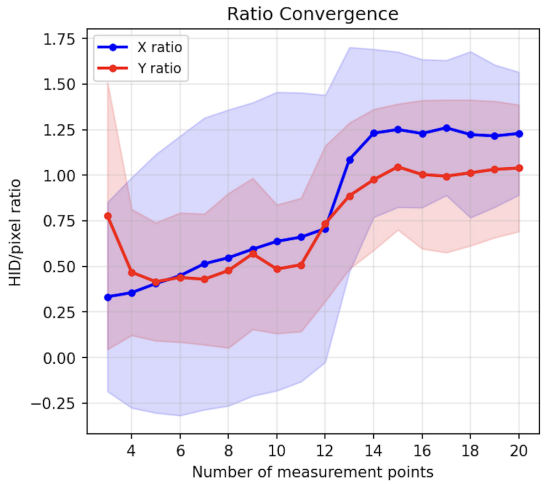


Figure 3: X/Y HID coordinates to pixel coordinates convergence averaged over 10 trials on an iPhone 17. We have found generally 15 points to be sufficient. Final error rates are around 1%, which we have found sufficient for most computer use agent applications, and could be improved with platform-specific training data.

and `gui_diff()` took 4.6 ms (SD 0.3). These hardware-layer latencies are small (and optimizable), especially relative to, *e.g.*, end-to-end agent performance, which is dominated by upstream cloud VLM inference (Claude 3.5 Sonnet: ~3.7 s, SD 323 ms).

HIDAgent is a dual-use technology. Nothing about it directly subverts existing security protections—an agent would still need to enter a password to access a locked system, and platforms such as macOS require explicit permission before a new HID device can be used. Modern operating systems further enforce exponential backoff that defeats naive brute-force password guessing. Nonetheless, an external hardware bridge like HIDAgent can facilitate VLM-driven local information scraping or cross-platform macro impersonation on devices to which a person has physical but not administrative access. We explicitly acknowledge these threat vectors and position HIDAgent as a tool intended for user-empowered accessibility and local research, not for circumventing the security or usage policies of systems a user does not control.

4 Toolkit Walkthrough

Every HIDAgent agent is an *observe-and-act* loop: `get_screenshot()` observes, a model interprets, `click_mouse()` or `type_text()` acts, and either `gui_diff()` or another model interpretation of the screenshot confirms that it resulted in a change. Appendix B gives a complete voice-control agent in roughly twenty lines, granting voice control to a device that does not natively support it with nothing installed on that device. HIDAgent makes capturing the current screen state, identifying the relevant UI element, and acting on it easy; the specific agent only needs to contribute the speech recognition results. The toolkit thus has a low *threshold* and a high *ceiling* [21]: substituting within the same loop (*e.g.*, `recognize_gui_elements()` or `patch_location()` for faster or fully local operation, or a different model) changes what it does without changing the structure. It offers *expressive leverage* [24] because the same script drives mobile, desktop, and "closed" systems unchanged: the abstraction is the pixel and the HID event, not any one platform's automation interface.

5 Use Case Prototypes

To validate the HIDAgent toolkit we created six distinct prototypes using it. Each prototype is described below and the source code for each is also released as open source. Our prototypes show how the capabilities of the HIDAgent toolkit allow for flexible exploration of personal computer use agents; our evaluation is limited to a descriptive analysis of our observations.

- **Extensible UI Agent**
  We implemented two generic UI Agents extensible by other researchers. The first connects HIDAgent to Claude in Computer Use mode via a straightforward adaptation of the Computer Use Tool[5] boilerplate, mapping its required actions (screenshot, click, type, key combinations, mouse move, wait) to HIDAgent functions. The second uses entirely on-device processing with Omniparser [20] and Gemma:27b [32]. We informally tested both on Android, iOS, Mac, and Windows. Both could complete some task steps, though Claude performed substantially better. Key observations: agents struggled to open applications on Android (no universal keyboard shortcut), performed poorly on high-resolution

[5]https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/computer-use-tool

desktops compared to mobile (a known limitation [18]), and made platform-inappropriate suggestions (*e.g.*, tried to Firefox on iOS) unless explicitly prompted with the target OS—suggesting models are rarely evaluated across platforms.

Because HIDAgent acts as an external input device without OS-level coordination, a user and the agent may attempt to control the same device concurrently. Such actions can produce cursor conflicts or actions taken against a stale UI state. To address this, we prototyped a lightweight human-takeover safeguard that leverages HIDAgent's low-latency execution layer. We call `gui_diff()` immediately before and after each VLM call, and if a meaningful structural change is detected (indicating human intervention), the agent drops its queued HID command, yields control, and enters a passive state. This does not fully solve human–agent collaboration, but it provides a concrete mechanism for human takeover and demonstrates the utility of HIDAgent's API.

- **Universal UI Data Collection**
  Few datasets exist for visual UI agent task completion, partly because data collection is difficult to set up—existing efforts like Rico [10] required crowd workers on remote Android desktops, and iOS datasets [39, 44] were not released publicly. Our Universal UI Data Collection prototype uses HIDAgent to create an automated collection pipeline: starting from any screen, it captures a screenshot with Omniparser parse results, randomly clicks to navigate, and uses gui_diff to detect new screens. Because HIDAgent works across platforms, this enables data collection on devices like iPhones that were previously inaccessible for this purpose.
- **Screen Reader Anywhere**
  Accessibility applications like screen readers require deep OS integration and careful per-application coding [14], and a person's preferred accessibility software often isn't available on devices they encounter, such as a public kiosk or a friend's phone. This has led to calls for intermediate "bots" that operate the original interface and expose a preferred accessible interface [34]. Our Screen Reader Anywhere prototype (Figure 4) continually captures screenshots from the target device, parses them with Omniparser to identify UI elements, and renders an accessible web page on the control computer. Actions on this page (clicks, typing) are forwarded back to the target device via HIDAgent, enabling non-visual access on devices lacking a screen reader or on which the user does not have permission to run a built-in screen reader.
- **Cross-Device Interaction**
  HCI researchers have long studied interactions that go across multiple computing devices we have access to at any given moment [15]. HIDAgent can be easily moved from one device to another, making possible novel cross-device agent interactions in which context is stored on the control computer even as the target computer changes to a new one. This was the simplest prototype to make because it simply involved using the UI Agent prototype previously described in a new way – as an example, with HIDAgent plugged into a laptop, we asked the agent to find how to enable high contrast mode on an iPhone and then enable it. The idea was that it could start with a web page, and find the information.

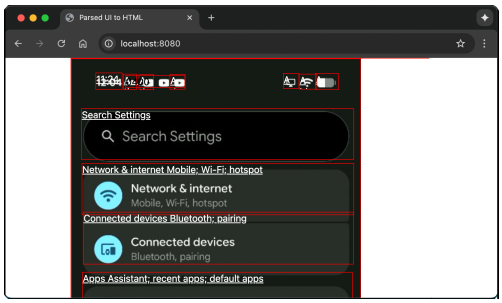


Figure 4: This screenshot shows how the Screen Reader Anywhere prototype brings a screenshot from in this case an Android target device into an accessible web page. Content from the pixels of the settings screen is made readable in the HTML and actions performed on the HTML (*e.g.*, buttons clicked) are transferred back to the original interface via HIDAgent, enabling a blind user to use their preferred screen reader to control any device HIDAgent can operate.

We then unplugged HIDAgent from the laptop and plugged it into an iPhone to complete the interaction.

- **The Helpful Observer**
  The Helpful Observer prototype allows users to ask for help related to their current screen by accepting input, taking a screenshot, and then offering potential advice. For instance, "how much is this flight in US dollars" when looking at possible trips on an airline app. The tips can be generated by either Claude or Gemma:27b running locally. Past studies have shown that agents can sometimes be better at figuring out how to navigate confusing or complex user interfaces [16]. The potential of this prototype in the context of HIDAgent is that you could bring your agent that has experience with you and your past interactions to a new computer just by plugging HIDAgent in, which could help unlock where and how future agents are allowed to be used. Our prototype required questions to be typed into the command line on the control computer, but connecting this to voice input could provide a new channel for improving interaction.
- **The Social Media Scroller**
  The Social Media Scroller prototype creates a recording of the content in a social media app in order to process or remix it in a different form. It finds the screen recording button on an iPhone, opens the Instagram app, chooses the main feed icon, and then for 2 minutes issues a scroll down command the length of half the screen. We separately built an iPhone application that parses the video into individuals posts and can then reorder and curate them (for instance, removing ads/suggested posts and reordering them), which may allow for new kinds of research into reordering visual social media feeds that has previously been restricted to the Web (*e.g.*, this study on rearranging feeds on X based on a measure of partisan animosity [26]). This prototype essentially allows a specific macro to be created and played back to support the narrow agent use case of social media recording, and may represent a broad class of similar prototypes.

6 Discussion & Future Work

We have introduced the idea of "Personal Computer Use Agents" that are separate from the devices that they control. The HIDAgent toolkit can observe and operate a wide variety of different devices, bringing new opportunities and challenges for Computer Use Agent research. In working with our prototypes, HIDAgent allowed us to find areas where state-of-the-art agent models perform poorly. For example, agents often didn't understand what operating system they were controlling, recommending, *e.g.*, running Firefox when operating an iPhone. This suggests that computer use agents are not often trained across multiple operating systems, which makes sense given that a single agent does not currently operate all operating systems. Computer Use Agents are also known to work best on lower resolution systems, and so when we plugged HIDAgent into our high-resolution work setup it didn't work well. HIDAgent may generally expose areas where existing computer use agents are insufficiently general.

We have considered a number of features that could extend HIDAgent's utility. For instance, since HIDAgent doesn't initially know much about a newly connected device, a "context crawler" could explore the system, try various ways of interacting with it, and peek into settings to understand the operating system and attached peripherals—emulating how a person might explore an unfamiliar device. The HIDAgent control computer we used was a Mac laptop, although a wide variety of other computing devices could be used. Even a minimal microcontroller could suffice if processing is done in the cloud (*e.g.*, via an LLM API). KVM over IP products suggest that a Raspberry Pi 4 may be powerful enough to capture screenshots and emulate HID events, allowing HIDAgent to run as a truly portable device, or even be integrated into a phone case. While technically feasible, we have not yet solved how to accomplish both screenshot capture and HID communication from an existing mobile phone.

Finally, platform makers have incentives to prevent agents on their platforms from inter-operating on other platforms, especially as more important interactions happen at the agent level. HIDAgent and the research it enables may help push back on that and lead to more work helping users maintain agency over their agents.

7 Conclusion

We have introduced the idea of "Personal Computer Use Agents", which are computer use agents that exist separate from the computers they control and which are controlled by and travel with the person using them. To facilitate research into personal agents, we introduced HIDAgent, a hardware/software toolkit that enables computer use agents to observe and operate HID-compatible devices. Through several prototypes, we demonstrated new interaction scenarios made possible with this toolkit that would not be possible with a purely software-based agent. We believe HIDAgent can be useful for HCI researchers in continuing to innovate and study how humans can work alongside computer use agents, and generally push back on the notion that computer use agents need to be silo'd onto specific platforms and use cases as they are today.

References

[1] James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. 2007. Plow: A collaborative task learning agent. In *AAAI*, Vol. 7. 1514–1519.

[2] Anthropic. 2024. Computer use (Beta). https://docs.anthropic.com/en/docs/build-with-claude/computer-use

[3] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. 2025. Qwen3-VL Technical Report. arXiv:2511.21631 [cs.CV] https://arxiv.org/abs/2511.21631

[4] Jeffrey P. Bigham, Tessa Lau, and Jeffrey Nichols. 2009. Trailblazer: enabling blind users to blaze trails through the web. In *Proceedings of the 14th International Conference on Intelligent User Interfaces* (Sanibel Island, Florida, USA) *(IUI '09)*. Association for Computing Machinery, New York, NY, USA, 177–186. doi:10.1145/1502650.1502677

[5] Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Guozhi Wang, Dingyu Zhang, Shuai Ren, and Hongsheng Li. 2025. AMEX: Android Multi-annotation Expo Dataset for Mobile GUI Agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 2138–2156. doi:10.18653/v1/2025.findings-acl.110

[6] Sen Chen, Chunyang Chen, Lingling Fan, Mingming Fan, Xian Zhan, and Yang Liu. 2022. Accessible or Not? An Empirical Investigation of Android App Accessibility. *IEEE Transactions on Software Engineering* 48, 10 (2022), 3954–3968. doi:10.1109/TSE.2021.3108162

[7] Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, Yuan Yao, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2025. GUICourse: From General Vision Language Models to Versatile GUI Agents. arXiv:2406.11317 [cs.AI] https://arxiv.org/abs/2406.11317

[8] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 9313–9332.

[9] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, YenSung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, Andrew Head, Rose Hendrix, Favyen Bastani, Eli VanderBilt, Nathan Lambert, Yvonne Chou, Arnavi Chheda, Jenna Sparks, Sam Skjonsberg, Michael Schmitz, Aaron Sarnat, Byron Bischoff, Pete Walsh, Chris Newell, Piper Wolters, Tanmay Gupta, Kuo-Hao Zeng, Jon Borchardt, Dirk Groeneveld, Crystal Nam, Sophie Lebrecht, Caitlin Wittlif, Carissa Schoenick, Oscar Michel, Ranjay Krishna, Luca Weihs, Noah A. Smith, Hannaneh Hajishirzi, Ross Girshick, Ali Farhadi, and Aniruddha Kembhavi. 2024. Molmo and PixMo: Open Weights and Open Data for State-of-the-Art Vision-Language Models. arXiv:2409.17146 [cs.CV] https://arxiv.org/abs/2409.17146

[10] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 845–854. doi:10.1145/3126594.3126651

[11] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2Web: Towards a Generalist Agent for the Web. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 28091–28114. https://proceedings.neurips.cc/paper_files/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets_and_Benchmarks.pdf

[12] Morgan Dixon and James Fogarty. 2010. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 1525–1534. doi:10.1145/1753326.1753554

[13] Gunnar Farnebäck. 2003. Two-Frame Motion Estimation Based on Polynomial Expansion. In *Image Analysis*, Josef Bigun and Tomas Gustavsson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 363–370.

[14] Chris Fleizach and Jeffrey P Bigham. 2024. System-class accessibility: The architectural support for making a whole system usable by people with disabilities. *Queue* 22, 5 (2024), 28–39.

[15] Peter Hamilton and Daniel J. Wigdor. 2014. Conductor: enabling and understanding cross-device interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 2773–2782. doi:10.1145/2556288.2557170

[16] Faria Huq, Zora Zhiruo Wang, Frank F. Xu, Tianyue Ou, Shuyan Zhou, Jeffrey P. Bigham, and Graham Neubig. 2025. CowPilot: A Framework for Autonomous and Human-Agent Collaborative Web Navigation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (System Demonstrations)*, Nouha Dziri, Sean (Xiang) Ren, and Shizhe Diao (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 163–172. doi:10.18653/v1/2025.naacl-demo.17

[17] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy) *(CHI '08)*. Association for Computing Machinery, New York, NY, USA, 1719–1728. doi:10.1145/1357054.1357323

[18] Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025. ScreenSpot-Pro: GUI Grounding for Professional High-Resolution Computer Use. In *Proceedings of the 33rd ACM International Conference on Multimedia* (Dublin, Ireland) *(MM '25)*. Association for Computing Machinery, New York, NY, USA, 8778–8786. doi:10.1145/3746027.3755688

[19] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6038–6049. doi:10.1145/3025453.3025483

[20] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203* (2024).

[21] Brad Myers, Scott E. Hudson, and Randy Pausch. 2000. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction (TOCHI)* 7, 1 (2000), 3–28. doi:10.1145/344949.344959

[22] Brad A. Myers. 2024. *Pick, Click, Flick! The Story of Interaction Techniques* (1 ed.). Vol. 57. Association for Computing Machinery, New York, NY, USA.

[23] Jeffrey Nichols, Zhigang Hua, and John Barton. 2008. Highlight: a system for creating and deploying mobile web applications. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (Monterey, CA, USA) *(UIST '08)*. Association for Computing Machinery, New York, NY, USA, 249–258. doi:10.1145/1449715.1449757

[24] Dan R. Olsen, Jr. 2007. Evaluating User Interface Systems Research. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. Association for Computing Machinery, 251–258. doi:10.1145/1294211.1294256

[25] OpenAI. 2025. Introducing Operator. https://openai.com/index/introducing-operator/. Accessed: 2025-04-09.

[26] Tiziano Piccardi, Martin Saveski, Chenyan Jia, Jeffrey Hancock, Jeanne L. Tsai, and Michael S. Bernstein. 2025. Reranking partisan animosity in algorithmic social media feeds alters affective polarization. *Science* 390, 6776 (2025), eadu5584. arXiv:https://www.science.org/doi/pdf/10.1126/science.adu5584 doi:10.1126/science.adu5584

[27] PiKVM. 2025. https://pikvm.org/

[28] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. 2025. UI-TARS: Pioneering Automated GUI Interaction with Native Agents. *arXiv preprint arXiv:2501.12326* (2025).

[29] Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. 2025. OS-Genesis: Automating GUI Agent Trajectory Construction via Reverse Task Synthesis. arXiv:2412.19723 [cs.AI] https://arxiv.org/abs/2412.19723

[30] Al Sweigart. 2025. Welcome to PyAutoGUI's documentation! https://pyautogui.readthedocs.io/. Accessed: 2025-11-15.

[31] TaxyAI. 2024. Taxy AI. https://taxy.ai/

[32] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. 2025. Gemma 3 Technical Report. arXiv:2503.19786 [cs.CL] https://arxiv.org/abs/2503.19786

[33] TinyPilot. 2025. https://tinypilotkvm.com

[34] Gregg Vanderheiden and Crystal Yvette Marte. 2024. Will AI allow us to dispense with all or most accessibility regulations?. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '24)*. Association for Computing Machinery, New York, NY, USA, Article 571, 9 pages. doi:10.1145/3613905.3644059

[35] Vercept. 2025. Vy, the AI-Powered Interface of the Future. https://vercept.com/. Accessed: 2025-04-08.

[36] Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, Bin Wang, Chuhan Wu, Yasheng Wang, Ruiming Tang, and Jianye Hao. 2025. GUI Agents with Foundation Models: A Comprehensive Survey. arXiv:2411.04890 [cs.AI] https://arxiv.org/abs/2411.04890

[37] WebAim. 2025. https://webaim.org/projects/million/. Accessed: 2025-11-09.

[38] Jacob O. Wobbrock and Julie A. Kientz. 2016. Research contributions in human-computer interaction. *Interactions* 23, 3 (April 2016), 38–44. doi:10.1145/2907069

[39] Jason Wu, Rebecca Krosnick, Eldon Schoop, Amanda Swearngin, Jeffrey P Bigham, and Jeffrey Nichols. 2023. Never-ending Learning of User Interfaces. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 113, 13 pages. doi:10.1145/3586183.3606824

[40] Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. 2025. TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks. arXiv:2412.14161 [cs.CL] https://arxiv.org/abs/2412.14161

[41] Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, Jitong Liao, Qi Zheng, Fei Huang, Jingren Zhou, and Ming Yan. 2025. Mobile-Agent-v3: Fundamental Agents for GUI Automation. arXiv:2508.15144 [cs.AI] https://arxiv.org/abs/2508.15144

[42] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: using GUI screenshots for search and automation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (Victoria, BC, Canada) *(UIST '09)*. Association for Computing Machinery, New York, NY, USA, 183–192. doi:10.1145/1622176.1622213

[43] Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In *European Conference on Computer Vision*. Springer, 240–255.

[44] Xiaoyi Zhang, Lilian de Greef, Amanda Swearngin, Samuel White, Kyle Murray, Lisa Yu, Qi Shan, Jeffrey Nichols, Jason Wu, Chris Fleizach, Aaron Everitt, and Jeffrey P Bigham. 2021. Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 275, 15 pages. doi:10.1145/3411764.3445186

A HIDAgent.py API

Below we list the primary functions in the HIDAgent API, which allow for retrieving the visual screen contents and issuing keyboard and mouse commands.

- **get_screenshot()** - identify the connected USB device that is most likely the HDMI capture card, capture one frame, and return it as a Python image object
- **move_mouse(x,y)** - move the mouse to location (x,y) on the target computer
- **click_mouse(x,y,<left|right>)** - move the mouse to location (x,y) on the target computer and click the left or right mouse button
- **type_text(<string>)** - send a string of characters to the target computer
- **keypress(list[keys])** - performs a single keypress (keydown + keyup) of the provided array of keyboard keys on the target computer, which can be provided either as characters or string denoting control keys (*e.g.*, ctrl, alt, cmd, space).
- **hid_calibration(optional:steps, optional:step_size)** - performs the calibration procedure described in 3.3. Optional parameters allow for control over the number of steps and the step_size.

HIDAgent.py also provides several convenience functions that we found especially useful in our development of use case prototypes that we believe are broadly useful (Section 5):

- **recognize_gui_elements(screenshot_img)** - takes a screenshot image of a GUI as input, and outputs a JSON list of recognize user interface elements, their location, their type, and any content associated with them (*e.g.*, a label). Currently, this is implemented as a lightweight wrapper around Omniparser [20].
- **llm_screenshot_query(screenshot_img, query, optional:<model>)** - takes a screenshot image of a GUI and a query to run on it, and then returns the model's response as a string. The default option currently first runs the screenshot through `recognize_gui_elements` and passes the results and image encoded in base64 to the Gemma:27b model [32], allowing all of the processing to run locally. We have also built support for OpenAI GPT5 and Anthropic Computer Use (Section 5).
- **run_application(application_name)** - exposes ability to run a provided application into one command in a way that works across multiple platforms, *i.e.*, the following three commands in sequence (cmd/windows key + spacebar, <application name>, enter)
- **gui_diff(screenshot1, screenshot2)** - takes in two screenshot images that are the same resolution and returns the location and size of each region that differs, or an empty list if none do.
- **patch_location(patch_img, screenshot)** - takes in an image of patch and returns its location on the provided screenshot or None if it is not detected. This provides an easy mechanism for specific scripting when you know exactly what you want the agent to click on, similar to Sikuli [42].

B A Voice Control Agent Example

The code below creates a complete voice-control agent, and has the four steps of the HIDAgent observe-and-act loop marked. The capture card and serial bridge are found on first use; calibration is run once per device and reloaded automatically thereafter (Section 3.3). Speech recognition and synthesis run on the control computer; the target contributes only HDMI out and USB in, so the same script runs unchanged on iOS, Android, Windows, and macOS.

```
import json
from hidagent.hidagent import (
    get_screenshot, click_mouse, type_text, gui_diff,
    llm_screenshot_query)

while True:
    said = listen()              # speech, on the
                                 # control computer
    before = get_screenshot()    # OBSERVE the target
    act = json.loads(            # INTERPRET the pixels
        llm_screenshot_query(before,
            f"For the request '{said}', return JSON "
            '{action, x, y, text} for the element '
            "to act on.", model="claude"))
    if act["action"] == "click": # ACT as a mouse...
        click_mouse(act["x"], act["y"])
    elif act["action"] == "type":# ...or as a keyboard
        type_text(act["text"])
    if not gui_diff(before,      # CONFIRM it landed
                    get_screenshot()):
        speak("Nothing changed. Try saying that "
              "another way.")
```