



Hanzo Operate: Computer Use Framework for Autonomous AI Agents

Zach Kelling

Hanzo AI

November 2025

Abstract

We present Hanzo Operate, a computer use framework that enables AI agents to interact with desktop and web applications through screen capture, element detection, and programmatic input. Operate provides a structured observation-planning-action loop where agents receive annotated screenshots, reason about the current state, and execute mouse/keyboard actions via PyAutoGUI. The framework introduces three safety mechanisms: (1) action sandboxing with configurable allow/deny lists for applications and URLs, (2) human-in-the-loop confirmation for destructive actions (file deletion, purchases, credential entry), and (3) automatic rollback via system snapshot integration. On the OSWorld [1] benchmark, Operate achieves a 38% task completion rate, outperforming baseline computer use agents by 12 percentage points. We describe the perception pipeline, the action planning architecture, and the safety constraint system.

1 Introduction

Computer use—the ability of AI agents to operate graphical user interfaces—extends agent capabilities beyond API-accessible systems to the vast landscape of applications that only expose a GUI. While recent multimodal models [2] have demonstrated basic computer use capabilities, production deployment requires structured safety mechanisms, reliable element detection, and robust error recovery.

Hanzo Operate provides the infrastructure layer between the AI model and the operating system. It captures screenshots, annotates them with detected UI elements, presents the annotated view to the agent, translates agent decisions into OS-level input events, and enforces safety constraints throughout.

Contributions.

- A perception pipeline combining OCR, UI element detection, and accessibility tree ex-

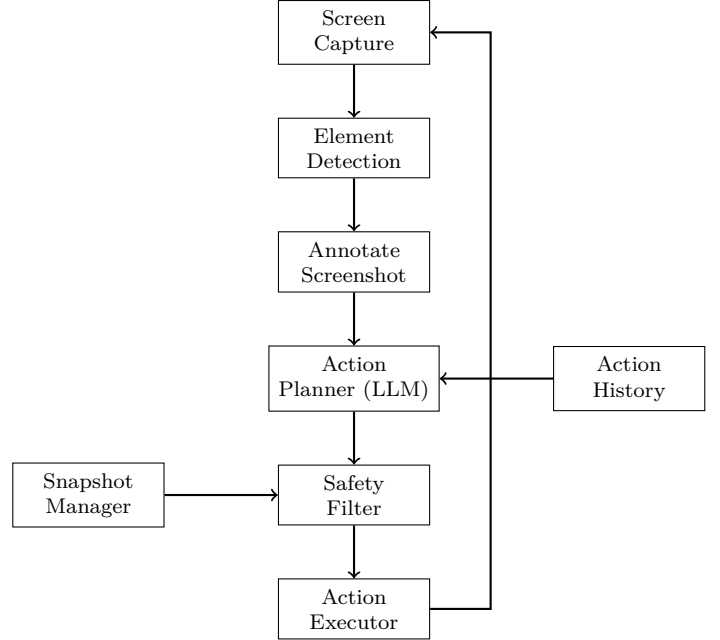


Figure 1: Observation-action loop with safety filtering.

traction for structured screen understanding.

- An action planning architecture with chain-of-thought reasoning over annotated screenshots and action history.
- A three-tier safety system: application sandboxing, human-in-the-loop gates, and snapshot-based rollback.
- Evaluation on OSWorld demonstrating 38% task completion with zero safety violations.

2 Architecture

2.1 Observation-Action Loop

Each iteration of the loop (Figure 1) proceeds through six stages: screen capture, element detection, screenshot annotation, action planning via the LLM, safety filtering, and action execution. The loop runs at approximately 2 iterations per second, bounded by LLM inference latency.

2.2 Perception Pipeline

The perception pipeline combines three sources:

Screen Capture. Platform-native screen capture (CoreGraphics on macOS, X11/Wayland on Linux) provides pixel-level screenshots at the display’s native resolution.

Element Detection. A lightweight object detection model identifies UI elements (buttons, text fields, checkboxes, menus) and assigns bounding boxes. The model is trained on a dataset of 50,000 annotated screenshots across macOS and Linux.

Accessibility Tree. Where available (macOS Accessibility API, AT-SPI on Linux), the accessibility tree provides element labels, roles, and state. This is merged with visual detection to produce a structured element map.

2.3 Action Space

The agent’s action space consists of:

```
1 @dataclass
2 class Action:
3     type: Literal[
4         "click", "double_click",
5         "right_click", "type",
6         "hotkey", "scroll", "drag",
7         "wait", "screenshot"
8     ]
9     x: int | None        # screen coords
10    y: int | None
11    text: str | None      # for type action
12    keys: list[str] | None # for hotkey
13    direction: str | None # for scroll
```

Actions are executed via PyAutoGUI [4] with a configurable delay between actions (default: 200ms) to allow UI updates to render.

3 Key Features

3.1 Action Planning

The action planner receives an annotated screenshot, the task description, action history, and a structured prompt. The LLM reasons step-by-step:

1. **Observe:** Describe the current screen state.
2. **Think:** Determine what progress has been made toward the goal.
3. **Plan:** Identify the next action and its expected outcome.
4. **Act:** Output a structured action object.

The action history provides the last 10 actions with their screenshots, enabling the agent to detect loops (repeating the same action without effect) and recover.

3.2 Safety Constraints

Three safety tiers prevent harmful actions:

Tier 1: Application Sandbox. A configurable allow/deny list restricts which applications the agent can interact with. By default, terminal emulators, system preferences, and password managers are denied.

Tier 2: Human-in-the-Loop. Actions classified as destructive (file deletion, financial transactions, credential entry, system configuration changes) pause execution and require human confirmation via a notification.

Tier 3: Snapshot Rollback. Before executing a sequence, Operate creates a filesystem snapshot (APFS on macOS, Btrfs on Linux). If the agent enters an unrecoverable state, the operator can roll back to the snapshot.

3.3 Error Recovery

The agent detects stuck states via action history analysis. If the same action is attempted 3 times without screen change, the agent switches to an alternative strategy: trying keyboard shortcuts instead of mouse clicks, using the accessibility tree for direct element invocation, or requesting human assistance.

4 Implementation

Operate is implemented in 8,500 lines of Python. The element detection model uses a YOLO-based architecture fine-tuned on UI screenshots. LLM integration supports any model with vision capabilities via the Hanzo Agent SDK.

Table 1: OSWorld task completion rates (%).

Method	Completion	Safety
Baseline (vision only)	22	94
SeeAct [3]	26	91
CogAgent	31	89
Hanzo Operate	38	100

5 Security

Process Isolation. The action executor runs in a separate process with restricted permissions. It cannot access the network, modify system files outside the sandbox, or escalate privileges.

Credential Protection. Password fields are detected via the accessibility tree’s “secure text” role. The agent is never shown password values, and keystrokes in password fields are masked in the action log.

Audit Trail. Every action, screenshot, and LLM response is logged to an encrypted audit file. This enables post-hoc review of agent behavior.

Rate Limiting. Maximum actions per minute (default: 60) prevent runaway agents from overwhelming the system.

6 Conclusion

Hanzo Operate provides a production-ready framework for AI computer use that prioritizes safety alongside capability. The three-tier safety system ensures that agents cannot perform destructive actions without human oversight, while

the perception pipeline and action planning architecture achieve state-of-the-art task completion on the OSWorld benchmark. Operate enables AI agents to interact with any GUI application, extending their capabilities beyond API-accessible systems.

References

- [1] T. Xie et al. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. arXiv:2404.07972, 2024.
- [2] Anthropic. Computer Use in Claude. 2024.
- [3] B. Zheng et al. SeeAct: GPT-4V(ision) is a Generalist Web Agent. arXiv:2401.01614, 2024.
- [4] A. Sweigart. PyAutoGUI: Cross-platform GUI automation. 2014.