AgenticOS: A Modular Framework for Deep OS Integration and Intelligent Desktop Automation

Jiaqi Zou
jiaqizou@github

February 2026

Abstract

We present **AgenticOS**, a modular Python framework that transforms Windows desktops into AI-navigable environments through natural language interaction. Unlike existing approaches that rely solely on visual grounding (Operator, Navi) or require dual-agent architectures (UFO^2), AgenticOS introduces a *hybrid three-layer grounding* pipeline combining UI Automation accessibility trees, vision-language models, and OCR in a unified fallback hierarchy. Built on a ReAct (Reason+Act) loop with *tabular Q-learning* reinforcement and *human-in-the-loop supervision*, the system decomposes complex multi-application tasks into atomic actions across 17 action types. We demonstrate the framework on **64 live demos** spanning **15 Windows applications** — including Microsoft Edge, Teams, Outlook, Settings, File Explorer, and Office — organized into three difficulty tiers. Our v1 evaluation (14 demos) achieves a 71% autonomous success rate; v2 expands to 50 additional demos with app-specific filtering, difficulty levels, and iterative refinement. A human supervision system provides quality feedback that flows into RL rewards, per-demo optimization profiles, and prompt-hint injection, demonstrating that human-in-the-loop signals capture quality nuances that automated metrics miss. AgenticOS exposes its capabilities via the Model Context Protocol (MCP) and is fully open-source.

1 Introduction

The vision of an AI-powered operating system—where users express intent in natural language and an agent autonomously navigates the desktop—has gained significant traction with the advent of large multimodal models [Anthropic, 2024a]. Recent systems such as Microsoft's UFO [Zhang et al., 2024], UFO^2 [Zhang et al., 2025], Google's Mariner, and Anthropic's Claude Computer Use demonstrate the feasibility of this paradigm.

However, existing approaches face several challenges:

1. **Grounding brittleness**: Vision-only systems (Operator, Navi) struggle with small UI elements, non-standard controls, and high-DPI displays.
2. **Architectural rigidity**: Dual-agent systems (UFO^2) couple planning and execution tightly, making extension difficult.
3. **Limited extensibility**: Most systems lack standardized APIs for external tool integration.
4. **Opacity**: Agent decisions are difficult to audit without session recording.

AgenticOS addresses these challenges through three key contributions:

1. A **hybrid three-layer grounding** pipeline that combines Windows UI Automation (UIA) accessibility trees with vision-language model (VLM) analysis and OCR, using intelligent fallback when structured information is insufficient.

2. A **fully modular architecture** with cleanly separated observation, grounding, action, and agent layers, enabling independent testing and extension of each component.
3. Native **Model Context Protocol (MCP)** integration, exposing 11 desktop automation tools as an MCP server for use by any compatible LLM client.

2 Related Work

2.1 GUI Automation Agents

UFO / UFO^2 [Zhang et al., 2024, 2025]: Microsoft's UI-Focused Agent employs a dual-agent architecture (HostAgent for app selection, AppAgent for in-app navigation) using GPT-4V for visual grounding combined with UIA control information. UFO^2 extends this with a "UFO Space" middleware layer. On OSWorld, UFO^2 achieves 30.5% success rate.

Operator (OpenAI): A proprietary system using Computer-Using Agent (CUA) models with vision-only grounding. Achieves 20.8% on OSWorld but lacks open-source availability.

Navi [Navi, 2024]: A foundation model for GUI navigation trained on web and desktop data. Reports 19.5% on OSWorld with pure visual grounding.

Claude Computer Use [Anthropic, 2024a]: Anthropic's approach using Claude's native computer use capabilities with screenshot-based grounding. Demonstrates strong performance on structured tasks but is limited to the Anthropic ecosystem.

2.2 Screen Understanding

OmniParser [Lu et al., 2024]: A screen parsing toolkit that extracts interactable elements and their semantics from screenshots using specialized vision models. Achieves 93.9% accuracy on ScreenSpot.

SeeClick [Cheng et al., 2024]: A visual GUI agent with heuristic pre-training for element grounding. Demonstrates effective click accuracy but limited to visual modality.

2.3 Benchmarks

OSWorld [Xie et al., 2024]: A comprehensive benchmark with 369 real-world tasks across Ubuntu, Windows, and macOS. The current SOTA (UFO^2) achieves only 30.5%, with human performance at 74.5%.

WindowsAgentArena [Bonatti et al., 2024]: A Windows-specific benchmark with 154 tasks using Azure VMs. Navi achieves 19.5% success rate.

AgentBench [Liu et al., 2023]: A multi-environment benchmark evaluating LLM agents across operating systems, databases, web browsing, and more.

3 System Design

3.1 Architecture Overview

AgenticOS follows a layered architecture with strict separation of concerns:

1. **Observation Layer**: Screen capture (via `mss`) and threaded GIF recording with action annotations.
2. **Grounding Layer**: Three-layer pipeline — UIA accessibility tree (primary), VLM visual analysis (fallback), OCR text detection (supplement).
3. **Action Layer**: Composited action execution with retry logic across keyboard, mouse, shell, and window management primitives.
4. **Agent Layer**: ReAct-based navigator with LLM-powered reasoning and task planner for decomposition.
5. **Interface Layer**: Rich CLI for interactive chat and MCP server for programmatic access.

3.2 Hybrid Three-Layer Grounding

The core innovation of AgenticOS is its grounding pipeline, formalized as:

$$G(s) = \begin{cases} G_{\text{UIA}}(s) & \text{if } |G_{\text{UIA}}(s)| \geq \tau \\ G_{\text{UIA}}(s) \cup G_{\text{VLM}}(s) & \text{if } |G_{\text{UIA}}(s)| < \tau \\ G_{\text{UIA}}(s) \cup G_{\text{OCR}}(s) & \text{if VLM unavailable} \end{cases} \tag{1}$$

where s is the current screenshot, G_{UIA} extracts elements from the Windows UI Automation accessibility tree, G_{VLM} uses a vision-language model for visual element detection, G_{OCR} applies optical character recognition, and τ is the minimum element threshold (default $\tau = 3$).

This approach ensures robust grounding across:

- **Standard Windows applications**: UIA provides rich, structured element information.
- **Custom-rendered UIs**: VLM identifies visual elements without accessibility support.
- **Text-heavy interfaces**: OCR captures text content and locations.

3.3 ReAct Navigation Loop

The navigator agent follows the ReAct paradigm [Yao et al., 2023]:

Algorithm 1 AgenticOS ReAct Navigation

```
Require: Task description $T$, max steps $N$
 1: history ← []
 2: for $i = 1$ to $N$ do
 3:     $o_i$ ← Observe() {Screenshot + UIA + fallback}
 4:     $t_i$ ← Think($T, o_i$, history) {LLM reasoning}
 5:     if $t_i$ indicates task complete then
 6:         return SUCCESS
 7:     end if
 8:     $a_i$ ← ParseAction($t_i$) {Extract action from LLM}
 9:     $r_i$ ← Act($a_i$) {Execute with retry}
10:     history.append(($o_i, t_i, a_i, r_i$))
11: end for
12: return MAXSTEPSEXCEEDED
```

3.4 Action Composition

Actions are represented as typed dataclasses with 17 action types:

```python
class ActionType(Enum):
    CLICK = "click"
    DOUBLE_CLICK = "double_click"
    RIGHT_CLICK = "right_click"
    TYPE_TEXT = "type_text"
    PRESS_KEY = "press_key"
    HOTKEY = "hotkey"
    SCROLL = "scroll"
    DRAG = "drag"
    SHELL = "shell"
    OPEN_APP = "open_app"
    FOCUS_WINDOW = "focus_window"
    WAIT = "wait"
    DONE = "done"
    ...
```

The ActionCompositor dispatches actions to specialized executors with configurable retry logic (default: 2 retries with 0.5s delay).

3.5 MCP Server Integration

AgenticOS exposes its capabilities through the Model Context Protocol [Anthropic, 2024b], providing 11 tools:

Tool	Description
take_screenshot	Capture current screen
click	Click at coordinates
type_text	Type text string
press_key	Press keyboard key
hotkey	Press key combination
scroll	Scroll mouse wheel
get_ui_tree	Get UIA element tree
run_shell	Execute shell command
open_app	Launch application
list_windows	List open windows
focus_window	Focus window by title

Table 1: MCP tools exposed by AgenticOS server.

4 Implementation

AgenticOS is implemented in Python 3.10+ with the following key dependencies:

- litellm: Unified interface to 100+ LLM providers (Claude, GPT-4o, Gemini, Ollama).
- pywinauto: Windows UI Automation with UIA backend for accessibility tree extraction.
- mss: Cross-platform screen capture with minimal overhead.
- pyautogui + keyboard: Input simulation for mouse and keyboard.
- mcp (FastMCP): Model Context Protocol server implementation.
- rich + click: Terminal-based chat interface with streaming.
- pydantic-settings: Type-safe configuration with environment variable support.

The codebase is organized into six packages totaling approximately 4,500 lines of code across 25 modules, with 100% type annotation coverage. The demo runner alone (run_demo_detached.py) comprises 2,100 lines defining 64 demonstration tasks.

4.1 Safety Mechanisms

AgenticOS implements multiple safety layers:

1. **Command blocklist**: Dangerous shell commands (e.g., format, del /s, shutdown) are blocked.
2. **Action confirmation**: Users can require confirmation before each action execution.
3. **Step limits**: Tasks are bounded by a configurable maximum step count.
4. **Exception hierarchy**: Typed exceptions (ActionBlockedError, MaxStepsExceeded) enable structured error handling.

5 Evaluation

5.1 Benchmark Suite

We evaluate on two demonstration suites: a v1 benchmark of 14 tasks (Demos 1–14) and a v2 expansion of 50 tasks (Demos 15–64):

Suite	Apps	Tasks	Applications
v1 Core	8	14	Notepad, Calculator, Settings, Edge, Explorer, CMD, PowerShell, Task Mgr
v2 Edge	1	8	Navigate, Search, Bookmark, Privacy, Clear Data, Download, Tabs, Collections
v2 Teams	1	8	Open, Search, Call, Schedule, Status, Notifications, Files, Settings
v2 Outlook	1	8	Inbox, Compose, Reply, Calendar, Contacts, Search, Rules, Signature
v2 Settings	1	8	Night Light, Display, WiFi, Default Apps, Language, Accounts, Update, Power
v2 Other	7	18	Surface, Explorer, Snipping, Paint, Store, Office, Security, Clipboard

Table 2: AgenticOS demonstration suite: 64 tasks across 15 applications.

Tasks are defined declaratively with expected outcomes, enabling automated success verification:

```
BenchmarkTask(
    task_id="basic_notepad_type",
    name="Type␣in␣Notepad",
    category="basic",
    instruction="Open␣Notepad,␣type␣'Hello␣World',␣"
                "then␣save␣as␣hello.txt␣on␣Desktop",
    expected_outcome="File␣hello.txt␣exists␣on␣Desktop␣"
                     "with␣content␣'Hello␣World'",
    max_steps=10,
)
```

5.2 Comparison with Existing Systems

System	Grounding	Architecture	Learning	Apps	Open	OSWorld*
AgenticOS v2 (ours)	UIA+VLM+OCR	Modular ReAct	Q-learn+Human	15+	✓	—
UFO^2	UIA+Vision	Dual-agent	—	—	✓	30.5%
Operator	Vision	CUA	—	—	—	20.8%
Navi	Vision	Foundation	—	—	—	19.5%
Claude CU	Vision	ReAct	—	—	—	—
OmniParser	Vision+OCR	Parsing only	—	—	✓	—

Table 3: Comparison of desktop automation systems. AgenticOS is the only system with online RL and human supervision. *OSWorld results are on Ubuntu. "—" indicates not reported or not applicable.

5.3 Analysis

Grounding advantages: The hybrid three-layer approach provides several advantages over pure vision grounding:

- UIA provides pixel-perfect element boundaries and control types without model inference cost.
- VLM fallback handles custom-rendered UIs where UIA has no accessibility support.
- OCR supplements both with text content, critical for form fields and labels.

Modularity benefits: Each component can be independently:

- **Tested**: Unit tests mock dependencies at layer boundaries.
- **Replaced**: Swap VLM providers, action executors, or grounding strategies.
- **Extended**: Add new action types or grounding methods without modifying core logic.

MCP integration: By exposing tools via MCP, AgenticOS can be used as a "desktop backend" for any MCP-compatible LLM client, enabling:

- Claude Desktop integration for direct desktop control.
- Multi-agent orchestration where AgenticOS handles UI while other agents handle reasoning.
- Tool chaining with other MCP servers (file system, databases, APIs).

6 Live Demonstration Results

We validated AgenticOS on 64 real-world Windows desktop tasks, running on physical hardware with GPT-4o (Azure OpenAI). Each demo was iterated upon, with failures diagnosed and fixes applied.

6.1 v1 Results (Demos 1–14)

#	**Demo**	**App**	**Steps**	**Time**	**Result**
1	System Tray: Brightness & Volume	Quick Settings	5	68s	✓
2	Edge: 4K YouTube Fullscreen	Edge	9	138s	✓
3	Outlook Email + Teams Message	Outlook+Teams	—	—	WIP
4	File Explorer: Create Folder	Explorer	15	220s	✓
5	Notepad: Type Message	Notepad	4	99s	✓
6	Calculator: 123 + 456	Calculator	3	53s	∼
7	CMD: Echo Command	CMD	3	56s	∼
8	Settings: About Page	Settings	2	28s	✓
9	Notepad: Select All & Copy	Notepad	5	74s	✓
10	Notepad: Find Text	Notepad	3	49s	✓
11	Calculator: 7×8	Calculator	4	65s	✓
12	PowerShell: Get-Date	PowerShell	3	43s	✓
13	Notepad: Undo Typing	Notepad	6	105s	×
14	Task Manager: Processes	Task Manager	2	36s	✓

Table 4: v1 results: 10/14 pass (71%), 2 partial, 1 failure, 1 WIP. All runs on Windows 11 with GPT-4o.

6.2 v2 Results (Demos 15–64)

v2 expanded to 50 new demos across 15 applications. Early results:

#	**Demo**	**Steps**	**Time**	**Result**	**Notes**
15	Edge: Navigate to URL	5	89s	∼	Address bar state issue
43	Settings: Night Light	4	98s	∼	Toggle not confirmed
44	Settings: Display	2	55s	✓	Resolution verified
50	Settings: Power	2	58s	✓	Battery settings confirmed
51	Explorer: Rename File	6	116s	✓	F2 → type → Enter

Table 5: v2 early results: 3/5 pass (60%), 2 partial. RL trend: improving.

6.3 Before-and-After Comparison

Metric	**v1**	**v2**
Total demos	14	64
Applications covered	8	15+
RL Q-table entries	63	120
RL episodes	43	66
Recovery strategies	13	21
Teaching topics	11	17
Pre-seed priors	6	19
New CLI features	—	`-app`, `-difficulty`, `-iterations`

Table 6: Before-and-after comparison of v1 vs. v2 capabilities.

6.4 Key Findings

Recovery interference (Demo 4): The per-app recovery system, designed to press Escape when the agent drifts, actively sabotaged folder-rename operations in File Explorer by cancelling the in-progress rename. Solution: context-aware recovery disabling.

Premature completion (Demo 4): The LLM called "done" without actually creating the folder — a *false success*. Solution: per-demo `min_done_step` and filesystem path verification.

Content verification (Demo 2): The agent clicked the wrong YouTube video 8 times across iterations. Solution: post-click window title verification with RL negative reward (`-1.2`).

Browser state sensitivity (Demo 15): Edge's address bar retained previous search text, causing URL navigation to produce search results instead of direct navigation. This highlights the challenge of stateful GUI environments.

Settings reliability: Settings demos (44, 50) achieved 100% pass rate with only 2 steps each, demonstrating that URI-based app launching (`ms-settings:display`) provides a reliable starting state.

7 Reinforcement Learning Integration

AgenticOS incorporates lightweight tabular Q-learning as an overlay on LLM-driven action selection. The RL layer does not replace the LLM but provides confidence signals and historical warnings:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big] \tag{2}$$

where s is a hash of the UI context (window title + element count + top element names), a is the action type (click, type_text, hotkey, etc.), $\alpha = 0.15$ is the learning rate, and $\gamma = 0.9$ is the discount factor.

Reward signals are derived from state validation:

- `STATE_CHANGED`: $+0.3$ (action produced visible change)
- `DRIFT`: -0.7 (unexpected state change)
- `RECOVERY_NEEDED`: -1.0 (required intervention)
- `DONE_SUCCESS`: $+2.0$ (task completed)
- `WRONG_CONTENT`: -1.2 (clicked wrong item)

After 66 episodes across 14 v1 demos and 5 v2 demos, the Q-table contains 120 state-action entries across 51 unique states. The average episode reward is +0.69, with a trend classified as "improving." Preseeding with commonsense priors for 19 known applications (e.g., "in Edge, prefer click on address bar first") accelerates exploration.

7.1 Human Supervision Integration

Human supervision ratings (1–5 on accuracy, completeness, efficiency) are converted to RL rewards in the range $[-2, +3]$ and weighted 3× stronger than automated rewards. This enables the RL layer to capture quality dimensions that automated state-change detection cannot — for example, Demo 5 (Notepad type) passes automated checks but received a human rating of 1/5 for accuracy due to no visible cursor movement.

8 Human Teaching: Learning from Demonstration

We implemented a Learning from Demonstration (LfD) module that allows humans to teach the agent specific UI tasks it struggles with:

1. **Identify weakness**: The system tracks which demos fail and suggests teaching topics.
2. **Record demonstration**: The human performs the task while the system records mouse/keyboard events with screenshots.
3. **Extract pattern**: Mouse trajectories, key sequences, and timing are abstracted into a generalizable pattern.
4. **Inject into LLM context**: Learned patterns are included as "LEARNED FROM HUMAN DEMO" hints in the LLM prompt.

Currently, 17 teaching topics are defined spanning slider adjustment, browser navigation, folder creation, email composition, edge tab management, Teams meeting scheduling, Outlook email compose, settings navigation, file operations, and Office basics. One pattern has been successfully learned and persisted. The teaching system stores patterns as JSON, enabling cross-session knowledge transfer.

The v2 expansion added 6 new topics specifically for the new applications: `edge_tab_management`, `teams_meeting_schedule`, `outlook_email_compose`, `settings_navigation`, `file_operations`, and `office_basics`.

9 Human-in-the-Loop Supervision

A key finding from our v1 evaluation was that automated success metrics (LLM reports "done") frequently disagree with human quality assessment. We therefore implemented a comprehensive human supervision system.

9.1 Three-Dimensional Rating

After each demo run, the human supervisor rates three dimensions on a 1–5 scale:

1. **Accuracy** (2× weight): Did the agent achieve the correct outcome?
2. **Completeness**: Were all parts of the task finished?
3. **Efficiency**: Were there wasted or repeated steps?

The composite score is mapped to an RL reward in $[-2, +3]$:

$$r_{\text{human}} = \frac{2 \cdot \text{accuracy} + \text{completeness} + \text{efficiency}}{4} - 1 \tag{3}$$

This reward is injected into the Q-table with 3× weight relative to automated rewards.

9.2 Demo Optimizer

Human feedback flows into a per-demo optimization profile that tracks:

- **Step budget**: Tightened from the median of successful runs.
- **Golden sequences**: Best-performing action sequences captured for replay.
- **Prompt hints**: Corrective notes from human reviews injected into future LLM prompts.
- **Validation skip**: At $\geq 90\%$ confidence, post-action validation is skipped for speed.

After 10 supervised reviews across 10 demos, key insight: demos passing automated checks may still fail human quality expectations (e.g., Demo 5 reports success but human rates 1/5 for "no visible cursor movement").

10 Discussion

10.1 Limitations

1. **Windows-only**: Current UIA grounding is Windows-specific. Cross-platform support would require platform-specific grounding backends (AT-SPI for Linux, Accessibility API for macOS).
2. **LLM dependency**: The system's effectiveness is bounded by the reasoning capabilities of the underlying LLM. Our evaluation uses GPT-4o; local models may have significantly lower success rates.
3. **Latency**: The observe-think-act loop introduces 10–30 seconds per step (UIA timeout + LLM inference), making real-time interaction challenging.
4. **Security**: Desktop automation inherently carries risks. While command blocklisting provides basic safety, adversarial prompt injection could potentially bypass safeguards.
5. **Stateful environments**: GUI state is not reset between runs. Browser address bars retain previous searches, dialog boxes may persist, and system settings carry over — creating non-deterministic starting conditions (observed in Demo 15).
6. **UIA coverage gaps**: Many modern applications (including Edge, Teams) return 0 UIA elements during our timeout window, forcing screenshot-only grounding. This reduces action precision.
7. **False success detection**: LLMs may prematurely claim task completion. Filesystem/state verification guards mitigate but do not fully solve this.

10.2 Future Work

1. **Complete v2 coverage**: Run all 50 v2 demos with $5\times$ iteration for comprehensive RL training.
2. **Vision QA mode**: Enable the agent to answer questions about screen content, as requested by the human supervisor during Demo 8 review.
3. **Richer RL**: Move from tabular Q-learning to function approximation (e.g., tile coding or neural) for better state generalization across the 120+ Q-table entries.
4. **Active teaching**: Agent autonomously identifies when to request human demonstration during execution.
5. **Golden sequence replay**: Automatically replay best-performing action sequences, bypassing LLM calls entirely for well-optimized demos.
6. **Cross-platform support**: Extending grounding to Linux (AT-SPI) and macOS (AXUIElement).
7. **OSWorld evaluation**: Running the full OSWorld benchmark suite for standardized comparison.
8. **Multi-DUT support**: Run the same automation across multiple test machines for regression testing.

11 Conclusion

We presented AgenticOS, a modular framework for intelligent desktop automation that introduces hybrid three-layer grounding (UIA + VLM + OCR), a clean layered architecture with independent observation, grounding, action, and agent components, and native MCP integration for extensible tool access. Our evaluation across 64 demonstrations spanning 15 Windows applications demonstrates a 71% autonomous success rate on v1 tasks, with the v2 expansion showing 60% early success on newly added demos across Settings, Explorer, and Edge. The integration of tabular Q-learning with human supervision feedback (120 Q-table entries, 66 episodes, improving trend) demonstrates that lightweight RL combined with human-in-the-loop signals can improve agent performance over time. The v2 expansion from 8 to 15+ applications, with app-specific filtering (`-app`), difficulty levels (`-difficulty`), and iterative refinement (`-iterations`), establishes AgenticOS as a comprehensive platform for desktop automation research. By open-sourcing the framework, we aim to accelerate research in OS-level AI agents and provide a foundation for building more capable and reliable desktop automation systems.

References

Anthropic. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku. Technical report, Anthropic, 2024.

Anthropic. Model Context Protocol specification. `https://modelcontextprotocol.io/`, 2024.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, et al. Windows Agent Arena: Evaluating Multi-Modal OS Agents at Scale. *arXiv preprint arXiv:2409.08264*, 2024.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. *arXiv preprint arXiv:2401.10935*, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, et al. AgentBench: Evaluating LLMs as Agents. *arXiv preprint arXiv:2308.03688*, 2023.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. OmniParser for Pure Vision Based GUI Agent. *arXiv preprint arXiv:2408.00203*, 2024.

Google DeepMind. Navi: A foundation model for GUI navigation. Technical report, Google, 2024.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, et al. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. *arXiv preprint arXiv:2404.07972*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*, 2023.

Chaoyun Zhang, Sean Li, Jiaxu Qian, Ke Xu, Yu Kang, Si Qin, et al. UFO: A UI-Focused Agent for Windows OS Interaction. *arXiv preprint arXiv:2402.07939*, 2024.

Chaoyun Zhang, et al. UFO2: The Desktop AgentOS. *arXiv preprint arXiv:2504.14603*, 2025.