Rethinking OS Interfaces for LLM Agents

Yuan Wang[★◇†] Mingyu Li[★◇] Haibo Chen[★◇‡]
[★]Key Laboratory of System Software (Chinese Academy of Sciences)
[◇]Institute of Software, Chinese Academy of Sciences
[‡]Shanghai Jiao Tong University
[†]University of Chinese Academy of Sciences

Abstract

Declarative Model Interface (DMI) is a novel OS interface tailored for LLM-based computer-use agents. Current agents use GUIs designed for humans, forcing LLMs to plan high-level goals and execute many low-level actions at the same time. Our key idea is to separate policy from mechanism: LLMs handle only high-level semantic planning (policy), while DMI takes care of low-level navigation and interaction (mechanism). We evaluate DMI on Microsoft Office (Word, PowerPoint, Excel). Compared to a leading GUI-based baseline agent, DMI boosts success by 67% and cuts steps by 43.5%, with 61% of successes achieved in just one LLM call.

1 Introduction

Computer-use agents (CUAs) powered by large language models (LLMs) automate complex workflows [2–12]. State-of-the-art CUAs use two main interfaces: application programming interfaces (APIs) and graphical user interfaces (GUIs). API-based methods give higher success rates and fewer steps [5, 13–15]. But most applications do **not** expose APIs. GUI-based methods are more **universal** [2, 3, 5, 6, 16]. They let multimodal LLMs see the screen and act (click, scroll, type). Yet they suffer from long action chains, increasing LLM calls and dropping success rates [13, 17–20].

Using a GUI application involves two distinct aspects. We call the first **policy**: deciding which GUI control to call, in what order, based on task semantics. We call the second **mechanism**: locating and operating the controls through navigation and interaction. In GUIs, the user (or agent) must first navigate spatial layouts to reveal controls, then perform precise interactions (click, drag, type, etc.) to activate them. This forces the LLM to handle both high-level planning and low-level execution in the same loop. The combined load overwhelms its reasoning, strains visual understanding, and creates long chains of actions with many round-trips.

Our key insight is that *a good portion of interaction logic with GUI is deterministic and can be executed independently of the LLM.* We propose Declarative Model Interface (DMI), which abstracts imperative GUI navigation and interaction into three declarative primitives: ***access***, ***state***, and ***observation***, where LLMs declare which control to access, what state to reach, or which result to observe for structured retrieval.

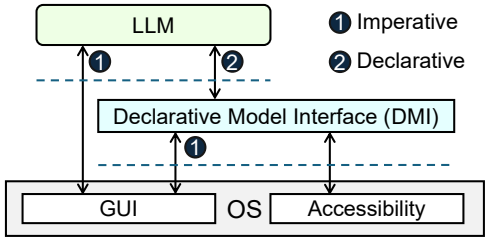

Figure 1. Overview of the DMI abstraction layer. DMI provides novel **OS interfaces** designed for LLM users.

This cleanly separates policy (semantic, non-deterministic decisions) from mechanism (low-level UI actions).

We validate DMI via case studies on Microsoft Word, Excel, and PowerPoint, covering text, spreadsheet, and graphics tasks. On OSWorld-W [4], DMI surpasses UFO2 [5] with a 29.6% absolute success gain (67% relative), 43.5% fewer steps, and 39% lower completion time. UFO2 failures are largely mechanism-driven (visual recognition, fine-grained interaction, navigation), whereas DMI re-centers failures to policy issues (80.9%).

Contributions. DMI is the first OS interface designed for LLM-based computer-use agents, aimed at reducing LLM load, shortening sequences, and improving reliability.

2 Motivation

2.1 Why GUI mismatches with LLMs?

M1: Procedural control access. GUI controls are hierarchically organized behind menus, tabs, and dialog boxes. This design deliberately narrows choices at each step, decomposing control localization into low-cognitive-load decisions [21]. The design assumes users (i) struggle with large decision spaces, (ii) have poor exact syntax recall, but (iii) excel at visual recognition [22]. These assumptions do not hold for LLMs. Given a global view, whether from trained knowledge or task-specific documentation in the prompt, an LLM can directly identify the appropriate control and generate structured invocations. However, an imperative GUI requires LLMs to first produce navigation sequences that make the target controls reachable. This increases the length of action chains and introduces systemic fragility, where any planning or execution error can cascade into complete task failure.

M2: Iterative interaction. Many GUI controls rely on iterative interactions (e.g., text selection involves repeated

This is a condensed version of "From Imperative to Declarative: Towards LLM-friendly OS Interfaces for Boosted Computer-Use Agents [1]," to appear in EuroSys 2026.

Table 1. Task examples of imperative GUI vs. declarative DMI.

Task	GUI	DMI
1	`click`("Design") → `click`("Format Background") → `click`("Solid fill") → `click`("Fill Color") → `click`("Blue") → `click`("Apply to All")	`visit`(["Blue", "Apply to All"])
2	iterative interaction (`drag` and `drop`)	`set_scrollbar_pos(80%)`

cursor movements). This design enforces high-frequency "observe–act" loops that transform challenging, high-precision outputs (exact coordinates) into manageable visual judgments ("is the current state acceptable?"). This assumes (i) perception is effortless and accurate, and (ii) frequent "observe–act" loops incur minimal cost. Neither assumption holds for LLMs: inference latency makes frequent loops expensive (humans adjust 3–5 times/s vs. 10–120+ s per round-trip for LLMs [23, 24]), and their visual perception is unreliable [25–27].

2.2 Insights

I1: Policy–mechanism decoupling. GUI design couples **policy** (function orchestration) with the **mechanism** (control navigation and interaction). Table 1 illustrates this coupling through two real-world examples. However, much of this mechanism is **deterministic** and can be resolved algorithmically without LLM involvement. Rather than forcing LLMs to plan fragile navigation sequences and complex interactions, we propose offloading deterministically solvable mechanisms to an abstraction layer, thus enabling LLMs to focus on nondeterministic, policy-level decisions that require semantic reasoning.

I2: Deterministic navigation. Given a target control, computing an access path and navigating to it becomes a deterministic problem. Once an application is released, its control transitions form a finite state machine [28], with control reachability and dependencies modeled as a directed graph where nodes represent controls. While controls may be accessible through multiple paths, removing back-edges and duplicate nodes with their successors yields a tree structure, producing a **unique** path identifiable from the control's identifier alone. The LLM needs only to declare the target control rather than specify the concrete navigation sequence, removing responsibility for path correctness from the model.

I3: Finite interaction operations. While controls exhibit diverse behaviors, Windows UI Automation (UIA) categorizes them into finite sets: 41 control types (e.g., Button, ListItem, Edit) and 34 control patterns (e.g., TextPattern, ScrollPattern, SelectionPattern) [29]. These universals enable us to abstract low-level, fine-grained interactions into state and result declarations such as `set_scrollbar_pos`(x_percent, y_percent) or `select_lines`(start_index, end_index). With these wrapped primitives, LLMs only need to specify the desired control state, eliminating requirements for precise visual-coordinate reasoning or high-frequency interaction.

2.3 Challenges

Challenge #1: Navigation path ambiguity. Application navigation relationships exist implicitly and are not exposed as explicit data structures. Even with UI topology, *path ambiguity* exists: cycles and merge nodes in the navigation graph prevent mapping controls to unique access paths.

Cycles: Loops (e.g., A → B → A) may yield infinite traversal sequences.

Merge nodes: Controls may be reachable via multiple paths (e.g., A → C and B → C).

Identical controls can enact different functions depending on path-related context. For instance, in Word's color picker, a color cell is reachable via "Font Color," "Outline Color," or "Underline Color" with different properties.

Challenge #2: Limited LLM context windows. Navigation topology should be converted to textual prompts for LLM comprehension. However, modern applications expose numerous controls (>5,000 in Microsoft Office), making full topology prohibitively expensive for LLMs.

Challenge #3: Inaccurate long-horizon planning. With DMI, the LLM can plan multiple commands in a single call, even when target controls are not currently visible. However, unexpected intermediate outcomes can invalidate subsequent steps and cascade into task errors. Additionally, real-world UI interaction is inherently unstable, for example, control name variations can break element identification.

3 Design and Implementation

3.1 Overview

DMI abstracts navigation as *access* declaration, and abstracts interaction as *state* and *observation* declarations.

- *Access declaration:* Given a control identifier, DMI deterministically navigates from any current state to that control and performs a primitive interaction (e.g., click).
- *State declaration:* Given a desired control end state (e.g., scrollbar position; selection state for a control or for text), DMI transitions the control from any current state to the target state, encapsulating compound interactions such as drag and keyboard–mouse coordination.
- *Observation declaration:* Given an information request (e.g., a control's text content), DMI returns structured data rather than relying on pixel-level recognition, and handles any compound interactions needed to reveal hidden content (e.g., expanding a table item).

3.2 Path-Unambiguous Navigation Topology

Navigation modeling: Following prior work [30, 31], we model UI navigation with an automated prototype, with modest human intervention. The result is a UI Navigation Graph (UNG). UNG is a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where

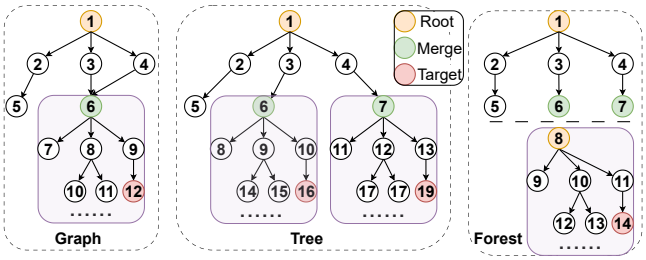

Figure 2. Navigation Topology. To access the bottom-right node along node 4, imperative GUI navigation relies on graph and requires the explicit path ⟨1, 4, 6, 9, 12⟩. With declaration, only ⟨19⟩ is required by tree, while the number of nodes explodes. For forest, the path required is ⟨7, 14⟩.

nodes are accessibility-exposed controls and edges denote click-induced reachability.

From directed graph to forest: To enable unambiguous control access by ID, we transform the single-source UNG into a **path-unambiguous** topology where each control has a unique access path. We first remove cycles by deleting back-edges, yielding a single-source DAG. We then resolve merge nodes (multiple in-edges) by converting the DAG into a forest with a main tree and shared subtrees. Naively cloning a merge node and its descendants per in-edge guarantees uniqueness but can cause exponential growth and exceed LLM context limits (e.g., Microsoft Office). Simply dropping in-edges is also invalid, since different paths to the same control may carry different semantics.

We propose an externalization algorithm that converts the single-source DAG into a forest while controlling topology size. Processing merge nodes bottom-up, it estimates the cloning cost of duplicating each node's descendant subgraph across incoming edges; if the cost exceeds a threshold, the subgraph is externalized as a shared subtree and reached via reference nodes, otherwise it is cloned. This ensures linear growth. The LLM outputs only the target control ID or with (typically one) shared-subtree `ref_id` (see Figure 2).

3.3 Context-Efficient Descriptions for Controls and Navigation

Compressed, hierarchical description: We represent the navigation hierarchy as compact structured text, retaining non-leaf navigation nodes so each control is described by its properties plus its hierarchical path.

Query on demand: To reduce context cost, DMI provides a pruned core topology by default (e.g., six levels). If the core is insufficient, the LLM issues `further_query` command to expand specific branches or fetch the full forest.

3.4 Access Declaration: The `visit` Interface

Interface specification: `visit` accepts an array of structured commands and executes them sequentially in a single call. Specifically, it supports four categories of commands:

- **Control access:** Navigate to the target control and perform a primitive interaction (i.e., click).
  Target in the main tree:
  ```
  {"id": "<target_id>"}
  ```
  Target in a shared subtree:
  ```
  {"id": "<target_id>",
  "entry_ref_id": ["<ref_id>", "..."]}
  ```
- **Access-and-input text:** Access an Edit type control and input text:
  ```
  {"id": "<target_id>", "text": "<text>"}
  ```
- **Shortcut keys (auxiliary):** Issue a keyboard shortcut (e.g., pressing ENTER to commit an edit):
  ```
  {"shortcut_key": "<key_combination>"}
  ```
- **FurtherQuery:** Request additional topology when the default core topology is insufficient.
  ```
  {"further_query": ["<node_id>", "..."]}
  ```

Balancing efficiency and accuracy: `visit` (i) bundles navigation and a primitive action (click/text) into an atomic control-access command and allows multiple commands per call. (ii) Excludes complex interactions: when composite steps are needed, DMI stops further visit commands, forcing observe-then-replan, and forbids mixing visit with other interfaces in the same turn. (iii) Supports essential keyboard shortcuts (e.g., ENTER) to be used sparingly.

Handling unstable UI Interaction: We employ fuzzy matching that combines control type, ancestor hierarchy, and name similarity when exact matching fails. We provide structured error feedback and use a failure retry mechanism.

Handling improper LLM instruction-following: Despite explicit instructions to output only target control identifiers, LLMs sometimes include navigational nodes. The key insight is that functional nodes are topology leaves, while navigational nodes are non-leaves. We filter out non-leaf nodes, allowing the executor to retain only commands targeting functional nodes and handle navigation independently.

3.5 State and Observation Declaration: Interaction-related Interfaces

Under UIA, a control describes its functionality through a finite set of control patterns (e.g., TextPattern, ScrollPattern, SelectionPattern). We leverage these control patterns and corresponding UIA interfaces to encapsulate control interactions to simplify complexity. In this way, we realize *State Declaration* and *Observation Declaration* (see § 3.1).

State and Observation declarations reduce dependency on precise visual perception and simplify complex interaction.

Table 2. `state` declaration and `observation` declaration interfaces. UIA defines 34 control patterns in total. These interfaces are extensible. For example, `set_texts` builds on `TextPattern`, `set_toggle_state` builds on `TogglePattern`, and `set_expanded/set_collapsed` builds on `ExpandCollapsePattern`.

Interface	Control Pattern	Description
`set_scrollbar_pos`	`Scroll`	Set scrollbar position to `x%`
`select_lines`	`Text`	Select one (or contiguous) line(s)
`select_paragraphs`	`Text`	Select one paragraph or a contiguous paragraph range
`select_controls`	`Select`	Single or multi-select controls
`get_texts`	`Text&Value`	Retrieve a control's text

For example, an Excel cell's full content may not be fully visible and would require several clicks to reveal. DMI ships with a set of pre-wrapped, high-value, and complex interactions, and can be customized to add additional UIA control patterns accordingly (see Table 2).

Supporting precise perception by default: Before each LLM call, `get_texts()` is invoked in passive mode on all `DataItem` controls, and a truncated, structured result is forwarded into the prompt. In cases where truncated results are insufficient, LLMs can call `get_texts()` in active mode to retrieve the full content.

Separating control access and complex interactions: To prevent mixing with `visit` interface within the same turn and preserve accuracy (see 3.4), the use of static IDs from the navigation topology is explicitly prohibited by interaction-related interfaces. Instead, those interfaces can only operate on controls specified by their label from the current screen's accessibility tree.

3.6 Implementation

DMI is implemented in over 18K lines of Python code. It builds on the pywinauto library [32] to drive the Windows UI Automation (UIA) framework. Full implementation is detailed in [1].

4 Evaluation

4.1 Setup

Case studies: We evaluate Microsoft Word, Excel, and PowerPoint as representative, feature-rich productivity applications. These applications collectively cover a range of scenarios, from text and tabular editing to graphics, and expose over 5,000 controls each.

Benchmark: We draw tasks from the widely used OSWorld benchmark [4]; specifically, the OSWorld-W (Windows) portion comprising 27 single-app scenarios for PowerPoint, Excel, and Word. We evaluate on the full set of single-app scenarios. We exclude the multi-app subset because it exercises the operating system's controls. Although modeling the OS controls is feasible, it would have required a significant amount of time for modeling.

Table 3. Results across interfaces and models.

Interface	Knowledge	Model	Reasoning	SR	Steps	Time(s)
GUI-only	/	GPT-5	Medium	44.4%	8.16	392
GUI-only	Nav.forest	GPT-5	Medium	42.0%	8.41	353
GUI+DMI	**Nav.forest**	**GPT-5**	**Medium**	**74.1%**	**4.61**	**239**
GUI-only	/	GPT-5	Minimal	23.5%	8.42	251
GUI+DMI	**Nav.forest**	**GPT-5**	**Minimal**	**40.7%**	**5.52**	**140**
GUI-only	/	5-mini	Medium	17.3%	7.14	171
GUI-only	Nav.forest	5-mini	Medium	23.5%	6.32	**150**
GUI+DMI	**Nav.forest**	**5-mini**	**Medium**	**43.2%**	**4.43**	167

Baseline: We adopt Microsoft's Windows CUA framework UFO-2 [5] as our baseline; it natively integrates UIA. For a fair comparison, we use the combination UFO2-base + action sequence (denoted UFO2-as). Here, UFO2-base is GUI-only. UFO-2 also offers Office-specific COM APIs, but those are not general and are therefore excluded. The action sequence mechanism reduces steps by allowing the LLM to emit multiple actions in a single turn, provided all referenced controls are currently visible in the app UI. The baseline labels accessible-tree controls and passes the labels in the prompt; to distinguish these labels from our numeric IDs in the navigation topology, labels are alphabetic (e.g., "A", "HF").

Our approach: On top of UFO2-as, we introduce our declarative DMI **(GUI+DMI)**. Prompts instruct the LLM to prefer DMI and fall back to GUI when necessary.

Methodology: Each task is capped at 30 steps and is run three times, then averaged. Office experiments use Microsoft 365 builds. The LLMs are OpenAI GPT-5 and GPT-5-mini. We compare GPT-5 at medium (default) vs minimal reasoning to emulate "reasoning" vs "non-reasoning" modes.

4.2 Offline Phase: UI Navigation Modeling Cost

Modeled graphs: We build the UNG and then transform it into a path-unambiguous forest. In the raw modeled graphs, Excel/PowerPoint/Word each exceeds 4K controls. To control context cost, we extract a core topology from the forest: Excel $\approx$ 2K, Word $\approx$ 1K, PowerPoint $\approx$ 1K controls. Word and Excel take a forest; PowerPoint's core topology is a single tree. No nested references exist among shared subtrees.

Cost: Automated modeling per application takes < 3 hours. Some manual setting is required. Average manual effort is around 1.5 person-days per application.

4.3 Online Phase: End-to-end Performance

Terminology: Reasoning denotes the configured reasoning effect; SR is the average success rate. Step counts averaged

LLM calls (i.e., round trips). Time is the average completion time. All metrics are calculated on successful cases only.

Settings: We compare three settings: (1) GPT-5, reasoning mode (core setting); (2) GPT-5, minimal reasoning; and (3) GPT-5-mini, reasoning mode.

Overall improvement: In the core setting, DMI yields substantial improvements over the baseline: raising success from 44.4% to 74.1% **(1.67×)**, cutting steps from 8.16 to 4.61 **(−43.5%)**, and reducing completion time by **39%.** Under minimal reasoning, success improves from 23.5% to 40.7%, steps drop from 8.42 to 5.52, and time declines from 251s to 140s. With GPT-5-mini, DMI yields an absolute +25.9% success gain (2.5× over GUI-only) and 38% fewer steps; average time is comparable (171s baseline vs 167s DMI) because we report only successful runs, and GUI-only primarily succeeds on shorter/easier cases. Table 3 shows the overall results.

One-shot task completion: In the core setting, with DMI, over 61% of successful trials are completed in 4 steps. UFO2's framework incurs a fixed 3-step overhead. For single-app tasks, DMI enables the LLM to complete the core user intent in a single LLM call.

4.4 Overhead

Token cost: DMI adds context tokens mainly from the navigation forest (>80%), with the rest from the usage prompt and truncated get_texts() payloads. Controls average ∼15 tokens (o200k_base); core topologies add ∼30K tokens (Excel) and ∼15K tokens (Word/PowerPoint). Despite this, DMI cuts interaction rounds, so total tokens per task are lower than the baseline in core scenarios.

Per-step latency: Per-call latency is higher for small models but overall time drops due to fewer steps; for large models, per-call latency is similar to the baseline.

4.5 Ablation Study

We ablate DMI to separate the effects of static topology and the declarative interface. Supplying only the navigation forest to UFO2-as (interface disabled) yields little benefit, while full DMI improves substantially, showing the declarative interface is the main performance driver (Table 3).

4.6 Failure Analysis

We analyze failures in the core setting and compare distributions between DMI and a GUI-only baseline.

GUI+DMI (policy-centric failures): The dominant causes are: ambiguous task descriptions (42.9%); misinterpretation of control semantics (28.6%); weak visual-semantic understanding (14.3%); misunderstanding of subtle task semantics (9.5%); and topology/modeling inaccuracies (4.8%). Tasks solved without DMI (GUI-only) **remain solvable** with DMI in our evaluation.

GUI-only baseline (mechanism-dominated failures): The baseline shows many mechanism failures: control localization and navigation errors (14/45), and composite-interaction errors (7/45). It also shows additional task-semantic errors and misperception of control functionality (6/45). A further 18 failures overlap with the policy-centric categories above. Because the LLM must split attention between policy and mechanism, more semantic mistakes appear. We observed similar errors in the ablation experiment, reinforcing that coupling policy with mechanism leads to avoidable mistakes.

Summary: The failure analysis pinpoints why DMI works: it cuts out mechanism-related errors (navigation and composite interaction) and re-centers policy, where LLMs are comparatively strong. Compared to GUI, DMI (DMI + GUI) aligns better with LLM capabilities, substantially reducing reliance on precise vision and high-frequency interactions.

5 Limitations

Unsupported GUI applications: Legacy applications from the Windows XP era and video-game applications and specialized graphics software often lack UIA support, falling outside our methodology's scope.

Completeness of DMI primitives: Tasks like freely drawing a shape (e.g., canvas) or position-precise manipulation (e.g., figure adjustment) are inherently dependent on fine-grained, **non-standard** position-related operations. Such tasks are difficult to express in a stateful manner and cannot be easily simplified using a declarative approach. In this case, the LLM can fall back to GUI for broader generality.

6 Related Work

GUI-based large action models: UI-TARS [6], OS-ATLAS [33], Aguvis [34], and UGround [7] train specialized large action models to navigate human-oriented GUI better.

Skill-based approaches: AppAgentX [35] encapsulates UI sequences as skills while relying on visual element matching. Similarly, AXIS [13] translates UI traces into application API calls, but its robustness is contingent upon the availability and stability of those APIs.

GUI ripping: GUITAR [30] abstracts GUIs into event-flow graphs and synthesizes test cases from the model (similar to [28, 30, 31, 36–38]).

Graph-based methods: AutoDroid-v1/v2 [28, 38] builds UI transition graphs and fine-tunes an on-device LM to plan action sequences beyond the current view. The method is still imperative and resolves missing controls by choosing the closest reachable path, which can be ambiguous—an issue addressed by DMI.

7 Concluding Remarks

OSes have long evolved interfaces to serve different users—from command-line interface (CLI) for experts to GUI for general users. LLMs are a fundamentally new class of OS users with distinct characteristics (large context-memory, reasoning) and constraints (weak visual capability, latency, token costs, imperfect instruction-following) that existing OS interfaces don't consider or address. As a result, LLMs underperform with GUI. DMI addresses this gap by separating policy from mechanism to build LLM-friendly OS interfaces.

References

[1] 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 ACM European Conference on Computer Systems (EuroSys)*, 2026.

[2] Claude Computer Use, 2024. URL https://docs.claude.com/en/docs/agents-and-tools/tool-use/computer-use-tool.

[3] OpenAI Operator, 2025. URL https://openai.com/index/introducing-operator/.

[4] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

[5] Chaoyun Zhang, He Huang, Chiming Ni, Jian Mu, Si Qin, Shilin He, Lu Wang, Fangkai Yang, Pu Zhao, Chao Du, Liqun Li, Yu Kang, Zhao Jiang, Suzhen Zheng, Rujia Wang, Jiaxu Qian, Minghua Ma, Jian-Guang Lou, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Ufo2: The desktop agentos, 2025. URL https://arxiv.org/abs/2504.14603.

[6] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL https://arxiv.org/abs/2501.12326.

[7] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=kxnoqaisCT.

[8] Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. Screenagent: A vision language model-driven computer control agent. In *Proceedings of the Thirty-ThirdInternational Joint Conference on Artificial Intelligence*, IJCAI-2024, page 6433–6441. International Joint Conferences on Artificial Intelligence Organization, August 2024. doi: 10.24963/ijcai.2024/711. URL http://dx.doi.org/10.24963/ijcai.2024/711.

[9] 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. Mobile-agent-v3: Fundamental agents for GUI automation. *CoRR*, abs/2508.15144, 2025. doi: 10.48550/ARXIV.2508.15144. URL https://doi.org/10.48550/arXiv.2508.15144.

[10] Reyna Abhyankar, Qi Qi, and Yiying Zhang. Osworld-human: Benchmarking the efficiency of computer-use agents. *CoRR*, abs/2506.16042, 2025. doi: 10.48550/ARXIV.2506.16042. URL https://doi.org/10.48550/arXiv.2506.16042.

[11] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 6864–6890. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.371. URL https://doi.org/10.18653/v1/2024.acl-long.371.

[12] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, pages 5295–5306. ACM, 2024. doi: 10.1145/3637528.3671620. URL https://doi.org/10.1145/3637528.3671620.

[13] Junting Lu, Zhiyang Zhang, Fangkai Yang, Jue Zhang, Lu Wang, Chao Du, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. AXIS: efficient human-agent-computer interaction with api-first llm-based agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 7711–7743. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.381/.

[14] Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. Computerrl: Scaling end-to-end online reinforcement learning for computer use agents. *CoRR*, abs/2508.14040, 2025. doi: 10.48550/ARXIV.2508.14040. URL https://doi.org/10.48550/arXiv.2508.14040.

[15] Zhuocheng Shen. Llm with tools: A survey, 2024. URL https://arxiv.org/abs/2409.18807.

[16] Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. UFO: A ui-focused agent for windows OS interaction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 597–622. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.26. URL https://doi.org/10.18653/v1/2025.naacl-long.26.

[17] Chaoyun Zhang, Shilin He, Liqun Li, Si Qin, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Api agents vs. gui agents: Divergence and convergence, 2025. URL https://arxiv.org/abs/2503.11069.

[18] Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, Ran Xu, Liyuan Pan, Caiming Xiong, and Junnan Li. Gta1: Gui test-time scaling agent, 2025. URL https://arxiv.org/abs/2507.05791.

[19] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Representation Learning*, volume 2024, pages 32808–32824, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/8b4add8b0aa8749d80a34ca5d941c355-Paper-Conference.pdf.

[20] Shuai Wang, Weiwen Liu, Jingxuan Chen, Weinan Gan, Xingshan Zeng, Shuai Yu, Xinlong Hao, Kun Shao, Yasheng Wang, and Ruiming Tang. GUI agents with foundation models: A comprehensive survey. *CoRR*, abs/2411.04890, 2024. doi: 10.48550/ARXIV.2411.04890. URL https://doi.org/10.48550/arXiv.2411.04890.

[21] Microsoft GUI Interface Design Principle, 2025. URL https://learn.microsoft.com/en-us/power-platform/well-architected/experience-optimization/interaction-design.

[22] Ben Shneiderman. Direct manipulation: A step beyond programming languages (abstract only). *SIGSOC Bull.*, 13(2–3):143, May 1981. ISSN 0163-5794. doi: 10.1145/1015579.810991. URL https://doi.org/10.1145/1015579.810991.

[23] Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. Llm-inference-bench: Inference benchmarking of large language models on ai accelerators. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1362–1379. IEEE, 2024.

[24] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. Serverlessllm: Low-latency serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, pages 135–153. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/fu.

[25] Zhiyuan Huang, Ziming Cheng, Junting Pan, Zhaohui Hou, and Mingjie Zhan. Spiritsight agent: Advanced GUI agent with one look. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pages 29490–29500. Computer Vision Foundation / IEEE, 2025. doi: 10.1109/CVPR52734.2025.02746. URL https://openaccess.thecvf.com/content/CVPR2025/html/Huang_SpiritSight_Agent_Advanced_GUI_Agent_with_One_Look_CVPR_2025_paper.html.

[26] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large language model-brained GUI agents: A survey. *Trans. Mach. Learn. Res.*, 2025, 2025.

[27] Liang Mi, Weijun Wang, Wenming Tu, Qingfeng He, Rui Kong, Xinyu Fang, Yazhu Dong, Yikang Zhang, Yuanchun Li, Meng Li, Haipeng Dai, Guihai Chen, and Yunxin Liu. Empower vision applications with lora lmm. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 261–277, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3717472. URL https://doi.org/10.1145/3689031.3717472.

[28] Hao Wen, Shizuo Tian, Borislav Pavlov, Wenjie Du, Yixuan Li, Ge Chang, Shanhui Zhao, Jiacheng Liu, Yunxin Liu, Ya-Qin Zhang, and Yuanchun Li. Autodroid-v2: Boosting slm-based gui agents via code generation, 2025. URL https://arxiv.org/abs/2412.18116.

[29] UIA Control Pattern, 2025. URL https://learn.microsoft.com/en-us/windows/win32/winauto/uiauto-controlpatternsoverview.

[30] Atif M. Memon, Ishan Banerjee, and Adithya Nagarajan. GUI ripping: Reverse engineering of graphical user interfaces for testing. In *10th Working Conference on Reverse Engineering, WCRE 2003, Victoria, Canada, November 13-16, 2003*, pages 260–269. IEEE Computer Society, 2003. doi: 10.1109/WCRE.2003.1287256. URL https://doi.org/10.1109/WCRE.2003.1287256.

[31] Inês Coimbra Morgado, Ana CR Paiva, and João Pascoal Faria. Dynamic reverse engineering of graphical user interfaces. *International Journal On Advances in Software*, 5(3):224–236, 2012.

[32] pywinauto, 2025. URL https://github.com/pywinauto/pywinauto.

[33] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. Os-atlas: A foundation action model for generalist gui agents, 2024. URL https://arxiv.org/abs/2410.23218.

[34] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous GUI interaction. *CoRR*, abs/2412.04454, 2024. doi: 10.48550/ARXIV.2412.04454. URL https://doi.org/10.48550/arXiv.2412.04454.

[35] Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, and Chi Zhang. Appagentx: Evolving GUI agents as proficient smartphone users. *CoRR*, abs/2503.02268, 2025. doi: 10.48550/ARXIV.2503.02268. URL https://doi.org/10.48550/arXiv.2503.02268.

[36] Android Monkey Tool, 2023. URL https://developer.android.com/studio/test/other-testing-tools/monkey?hl=en.

[37] Tanja EJ Vos, Peter M Kruse, Nelly Condori-Fernández, Sebastian Bauersfeld, and Joachim Wegener. Testar: Tool support for test automation at the user interface level. *International Journal of Information System Modeling and Design (IJISMD)*, 6(3):46–83, 2015.

[38] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, ACM MobiCom 2024, Washington D.C., DC, USA, November 18-22, 2024*, pages 543–557. ACM, 2024. doi: 10.1145/3636534.3649379. URL https://doi.org/10.1145/3636534.3649379.