# UFO2: The Desktop AgentOS

# UFO2: 桌面代理操作系统

Chaoyun Zhang*

张朝云 *

Microsoft

微软

He Huang

黄鹤

Microsoft

微软

Chiming Ni†

倪晨鸣 †

ZJU-UIUC Institute

浙大-伊利诺伊香槟分校研究院

Jian Mu†

穆坚 †

Nanjing University

南京大学

Si Qin

秦思

Microsoft

微软

Shilin He

何世霖

Microsoft

微软

Lu Wang

王璐

Microsoft

微软

Fangkai Yang

杨方凯

Microsoft

微软

Pu Zhao

赵璞

Microsoft

微软

Bo Qiao

乔博

Microsoft

微软

Chao Du

杜超

Microsoft

微软

Liqun Li

李利群

Microsoft

微软

Yu Kang

康宇

Microsoft

微软

Zhao Jiang

江钊

Microsoft

微软

Suzhen Zheng

郑素珍

Microsoft

微软

Rujia Wang

王如佳

Microsoft

微软

Jiaxu Qian†

钱家旭 †

Peking University

北京大学

Minghua Ma

马明华

Microsoft

微软

Jian-Guang Lou

娄建光

Microsoft

微软

Qingwei Lin

林青伟

Microsoft

微软

Saravan

萨拉文

Rajmohan

拉吉莫汉

Microsoft

微软

Dongmei Zhang

张东梅

Microsoft

微软

## Abstract

## 摘要

Recent Computer-Using Agents (CUAs), powered by multimodal large language models (LLMs), offer a promising direction for automating complex desktop workflows through natural language. However, most existing CUAs remain conceptual prototypes, hindered by shallow OS integration, fragile screenshot-based interaction, and disruptive execution.

近年来，基于多模态大语言模型 (LLMs) 驱动的计算机使用代理 (CUAs) 为通过自然语言自动化复杂桌面工作流提供了有前景的方向。然而，大多数现有 CUAs 仍停留在概念原型阶段，受限于浅层的操作系统集成、脆弱的基于截图的交互以及具有破坏性的执行。

We present UFO2, a multiagent AgentOS for Windows desktops that elevates CUAs into practical, system-level automation. UFO $^2$ features a centralized HosTAGENT for task decomposition and coordination, alongside a collection of application-specialized APPAGENTS equipped with native APIs, domain-specific knowledge, and a unified GUI-API action layer. This architecture enables robust task execution while preserving modularity and extensibility. A hybrid control detection pipeline fuses Windows UI Automation (UIA) with vision-based parsing to support diverse interface styles. Runtime efficiency is further enhanced through speculative multi-action planning, reducing per-step LLM overhead. Finally, a Picture-in-Picture (PiP) interface enables automation within an isolated virtual desktop, allowing agents and users to operate concurrently without interference.

我们提出 UFO2，一个面向 Windows 桌面的多智能体 AgentOS，将 CUA 提升为实用的系统级自动化。UFO $^2$ 以集中式 HosTAGENT 负责任务分解与协调，配合一组具备本地 API、领域知识和统一 GUI-API 操作层的应用专用 APPAGENT。该架构在保持模块化与可扩展性的同时，支撑稳健的任务执行。混合控件检测管道将 Windows UI Automation (UIA) 与基于视觉的解析融合，以支持多样的界面风格。通过推测性多动作规划降低每步 LLM 开销，从而提升运行时效率。最后，画中画 (PiP) 界面在隔离的虚拟桌面中启用自动化，使代理与用户能并发操作而互不干扰。

We evaluate UFO$^2$ across over 20 real-world Windows applications, demonstrating substantial improvements in robustness and execution accuracy over prior CUAs. Our results show that deep OS integration unlocks a scalable path toward reliable, user-aligned desktop automation.

我们在 20 多款真实 Windows 应用上评估了 UFO$^2$，显示其在鲁棒性与执行准确率上相较先前 CUA 有显著提升。我们的结果表明，深度 OS 集成为朝向可靠、与用户对齐的桌面自动化打开了可扩展的道路。

The source code of UFO$^2$ is publicly available at https: //github.com/microsoft/UFO/, with comprehensive documentation provided at https://microsoft.github.io/UFO/.

UFO$^2$ 的源代码已在 https: //github.com/microsoft/UFO/ 公开，可在 https://microsoft.github.io/UFO/ 查阅完整文档。

Keywords: Computer Using Agent, Large Language Model, Desktop Automation, Windows System

关键词: 计算机使用代理，大型语言模型，桌面自动化，Windows 系统

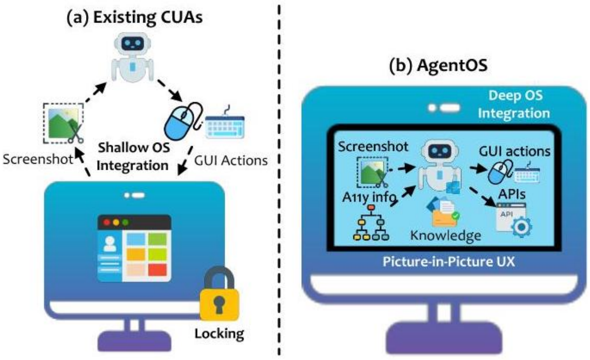


Figure 1. A comparison of (a) existing CUAs and (b) desktop AgentOS UFO2.

图 1. 比较 (a) 现有的 CUA 与 (b) 桌面 AgentOS UFO2。

## ACM Reference Format:

## ACM 引用格式:

Chaoyun Zhang et al.. 2025. UFO $^2$ : The Desktop AgentOS. In . ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Chaoyun Zhang 等. 2025. UFO $^2$ : The Desktop AgentOS. In . ACM, New York, NY, USA, 24 页. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 Introduction

## 1 引言

Automation of desktop applications has long been central to improving workforce productivity. Commercial Robotic Process Automation (RPA) platforms such as UiPath [1], Automation Anywhere [2], and Microsoft Power Automate [3] exemplify this trend, using predefined scripts to replicate repetitive user interactions through the graphical user interface (GUI) [4, 5]. However, these script-based approaches often prove fragile in dynamic, continuously evolving environments [6]. Minor interface changes can break the underlying automation scripts, requiring manual updates and extensive maintenance effort. As software ecosystems grow increasingly complex and heterogeneous, the brittleness of script-based automation severely limits scalability, adaptability, and practicality.

桌面应用的自动化长期以来是提高工作效率的核心。商业机器人流程自动化 (RPA) 平台如 UiPath [1]、Automation Anywhere [2] 和 Microsoft Power Automate [3] 就是这一趋势的典型，使用预定义脚本通过图形用户界面 (GUI) 重现重复的用户交互 [4, 5]。然而，这些基于脚本的方法在动态、持续演进的环境中常显脆弱 [6]。界面细微变化即可破坏自动化脚本，需人工更新并投入大量维护工作。随着软件生态系统日益复杂与异质，脚本式自动化的脆弱性严重限制了可扩展性、适应性与可行性。

Recently, Computer-Using Agents (CUAs) [7] have emerged as a promising alternative. These systems leverage advanced multimodal large language models (LLMs) [8, 9] to interpret diverse user instructions, perceive GUI interfaces, and generate adaptive actions (e.g., mouse clicks, keyboard inputs) without fixed scripting [7]. Early prototypes such as UFO [10], Anthropic Claude [11], and OpenAI Operator [12] demonstrate that such LLM-driven agents can robustly automate tasks too complex or ambiguous for conventional RPA pipelines. Yet, despite these advances, current CUA implementations remain largely conceptual: they mainly optimize visual grounding or language reasoning [12-16], but give little attention to system-level integration with desktop operating systems (OSs) and application internals (Figure 1 (a)).

近来，计算机使用代理 (CUAs) [7] 已成为有前景的替代方案。这些系统利用先进的多模态大型语言模型 (LLMs) [8, 9] 来理解多样的用户指令、感知 GUI 界面并生成自适应操作 (例如鼠标点击、键盘输入)，无需固定脚本 [7]。早期原型如 UFO [10]、Anthropic Claude [11] 和 OpenAI Operator [12] 表明，这类由 LLM 驱动的代理可以稳健地自动化那些对传统 RPA 管道而言过于复杂或模糊的任务。然而，尽管已有进展，目前的 CUA 实现仍多为概念性: 它们主要优化视觉定位或语言推理 [12-16]，而很少关注与桌面操作系统 (OS) 及应用内部的系统级集成 (见图 1(a))。

Reliance on raw GUI screenshots and simulated input events has several drawbacks. First, visual-only inputs can be noisy and redundant, increasing cognitive overhead for LLMs and reducing execution efficiency [17]. Second, existing CUAs rarely leverage the OS's native accessibility interfaces, application-level APIs, or detailed process states-a missed opportunity to significantly enhance decision accuracy, reduce latency, and enable more reliable execution. Finally, simulating mouse and keyboard events on the primary user desktop locks users out during automation, imposing a poor user experience (UX). These limitations must be resolved before CUAs can mature from intriguing prototypes into robust, scalable solutions for real-world desktop automation, and motivates our central research question:

*Corresponding author. Email: chaoyun.zhang@microsoft.com

* 通信作者。电子邮件: chaoyun.zhang@microsoft.com

†This work was done during their internship at Microsoft.

† 该工作完成于其在微软实习期间。

仅依赖原始 GUI 屏幕截图与模拟输入事件存在若干缺点。首先，仅视觉输入可能嘈杂且冗余，增加 LLM 的认知负担并降低执行效率 [17]。其次，现有 CUA 很少利用 OS 的本地辅助功能接口、应用级 API 或详细的进程状态——这是显著提升决策准确性、降低延迟并实现更可靠执行的良机。最后，在主用户桌面上模拟鼠标和键盘事件会在自动化期间将用户锁定在外，带来糟糕的用户体验 (UX)。这些限制必须被解决，CUA 才能从有趣的原型成长为面向真实世界的稳健、可扩展的桌面自动化解决方案，这也推动了我们的中心研究问题:

How can we build a robust, deeply integrated system for desktop automation that flexibly adapts to evolving interfaces, reliably orchestrates diverse applications, and minimizes disruption to user workflows?

我们如何构建一个稳健、深度集成的桌面自动化系统，能灵活适应不断变化的界面、可靠地编排多样应用，并将对用户工作流程的干扰降至最低?

In response, we present **UFO$^2$** , a new AgentOS for Windows that reimagines desktop automation as a first-class operating system abstraction. Unlike prior CUAs that treat automation as a layer atop screenshots and simulated input events, UFO$^2$ is architected as a deeply integrated, multi-agent execution environment-embedding OS capabilities, application-specific introspection, and domain-aware planning into the core automation loop, as illustrated in Figure 1(b).

为此，我们提出 **UFO$^2$** ，一个面向 Windows 的新 AgentOS，将桌面自动化重新设想为一种一等的操作系统抽象体。不同于将自动化视为建立在截图和模拟输入事件之上的层的先前 CUA，UFO$^2$ 被设计为深度集成的多智能体执行环境——在核心自动化循环中嵌入 OS 能力、应用特定的内省与领域感知规划，如图 1(b) 所示。

At its foundation, UFO$^2$ provides a modular, system-level substrate for natural language-driven automation. A centralized coordinator, the HosTAGENT, interprets user instructions, decomposes them into semantically meaningful subtasks, and dynamically dispatches execution to specialized APPAGENTS - expert modules tailored for specific Windows applications. Each APPAGENT is equipped with an extensible toolbox of application-specific APIs, a hybrid GUI-API action interface, and integrated knowledge about the application's capabilities and semantics. This architecture enables robust orchestration across multiple concurrent applications, supporting workflows that span Excel, Outlook, Edge, and beyond.

在其基础上, UFO$^2$ 提供了面向自然语言驱动自动化的模块化系统级基底。集中协调器 HosTAGENT 解读用户指令，将其分解为语义上有意义的子任务，并动态分派执行给专门的 APPAGENT ——为特定 Windows 应用定制的专家模块。每个 APPAGENT 配备可扩展的应用专用 API 工具箱、混合 GUI-API 操作接口，以及关于该应用能力与语义的集成知识。该架构实现了跨多并发应用的稳健编排，支持跨越 Excel、Outlook、Edge 等的工作流。

To enable reliable execution across the full spectrum of application UIs, UFO$^2$ introduces a hybrid control detection pipeline, combining Windows UI Automation (UIA) APIs with advanced visual grounding models [18]. This allows agents to introspect and act on both standard and custom UI components, bridging the gap between structured accessibility trees and pixel-level perception. Moreover, UFO$^2$ continuously incorporates external documentation, patch notes, and past execution traces into a unified vectorized memory layer, enabling each AppAGENT to incrementally refine its behavior without retraining.

> 为了在各类应用 UI 上实现可靠执行，UFO$^2$ 引入了混合控件检测流水线，结合了 Windows UI Automation (UIA) API 与先进的视觉定位模型 [18]。这使得智能体能够检测并作用于标准与自定义 UI 组件，弥合了结构化无障碍树与像素级感知之间的鸿沟。此外，UFO$^2$ 持续将外部文档、补丁说明和过去的执行痕迹整合进统一的向量化记忆层，使每个 AppAGENT 能在无需重训的情况下逐步优化其行为。

At the interaction layer, UFO$^2$ exposes a unified GUI-API execution model, where agents seamlessly combine traditional GUI actions (e.g., clicks, keystrokes) with native Windows or application-specific APIs. This hybrid approach improves execution efficiency, reduces brittleness to UI layout changes, and enables more expressive, higher-level operations. To further minimize the latency associated with LLM-based action planning, UFO$^2$ incorporates a speculative multi-action execution engine that proactively infers and validates action sequences using lightweight control-state checks at a single inference step-substantially reducing inference overhead without compromising correctness.

> 在交互层面，UFO$^2$ 暴露出统一的 GUI-API 执行模型，智能体可无缝结合传统 GUI 操作 (如点击、按键) 与原生 Windows 或应用特定的 API。这种混合方法提升了执行效率，降低了对 UI 布局变化的脆弱性，并支持更具表达力的高层操作。为进一步减少基于 LLM 的动作规划带来的延迟，UFO$^2$ 集成了一个推测性多动作执行引擎，通过在单次推理步使用轻量的控件状态检查主动推断并验证动作序列——在不牺牲正确性的前提下大幅降低推理开销。

Finally, to ensure a practical and non-intrusive user experience, UFO$^2$ introduces a novel Picture-in-Picture (PiP) interface: a secure, nested desktop environment where agents can execute independently of the user's main session. Built atop Windows' native remote desktop loopback infrastructure, PiP enables seamless, side-by-side user-agent multitasking without disruption on the user's main desktop, addressing one of the most persistent UX limitations of existing CUAs.

> 最后，为了确保实用且不干扰用户的体验，UFO$^2$ 引入了一种新颖的画中画 (PiP) 界面: 一个安全的嵌套桌面环境，使智能体可独立于用户主会话执行。基于 Windows 本地远程桌面回环基础设施构建，PiP 实现了无缝的并排用户-智能体多任务处理，不会干扰用户主桌面，解决了现有 CUA 中最持久的 UX 限制之一。

Together, these design principles position UFO$^2$ not just as a smarter agent, but as a new OS-level abstraction for automation-transforming desktop workflows into programmable, composable, and robust entities. In summary, this paper makes the following contributions:

> 综上，这些设计原则使 UFO$^2$ 不仅成为更智能的智能体，而且成为面向自动化的新的操作系统级抽象——将桌面工作流转变为可编程、可组合且稳健的实体。总之，本文做出以下贡献:

- Deep OS Integration: We design and implement UFO$^2$ , a multiagent AgentOS that deeply embeds within the Windows OS, orchestrating desktop applications through introspection, API access, and fine-grained execution control.

> - 深度操作系统集成: 我们设计并实现了 UFO$^2$ ，一款多智能体 AgentOS，深度嵌入 Windows 操作系统，通过内省、API 访问和细粒度执行控制编排桌面应用。

- Unified GUI-API Action Layer: We propose a hybrid action interface that bridges traditional GUI interactions with application-native API calls, enabling flexible, efficient, and robust automation.

- 统一 GUI-API 操作层: 我们提出了一种混合操作接口，桥接传统 GUI 交互与应用原生 API 调用，支持灵活、高效且稳健的自动化。

- Hybrid Control Detection: We introduce a fusion pipeline combining UIA metadata with vision-based detection to achieve reliable control grounding even in non-standard interfaces.

- 混合控件检测: 我们引入了将 UIA 元数据与基于视觉的检测融合的流水线，以在非标准界面中实现可靠的控件定位。

- Continuous Knowledge Integration: We build a retrieval-augmented memory that integrates documentation and historical execution logs, allowing agents to improve autonomously over time without retraining.

- 持续知识整合: 我们构建了一个检索增强的记忆系统，整合文档与历史执行日志，允许智能体随时间自主改进而无需重训。

- Speculative Multi-Action Execution: We reduce LLM invocation overhead by predicting and validating action sequences ahead of time using UI state signals.

- 推测性多动作执行: 我们通过利用 UI 状态信号提前预测并验证动作序列，减少了对 LLM 调用的开销。

- Non-Disruptive UX: We develop a nested virtual desktop environment that allows automation to proceed in parallel with user activity, avoiding interference and improving adoptability.

- 非干扰性用户体验: 我们开发了一个嵌套虚拟桌面环境，使自动化能与用户活动并行进行，避免干扰并提高可采纳性。

- Comprehensive Evaluation: We evaluate UFO$^2$ across 20+ real-world Windows applications, showing consistent improvements in success rate, execution efficiency, and usability over state-of-the-art CUAs like Operator.

- 全面评估: 我们在 20+ 个真实 Windows 应用上评估了 UFO$^2$，显示在成功率、执行效率和可用性方面相较于如 Operator 等最先进的 CUA 持续提升。

Overall, UFO$^2$ advances the vision of OS-native automation by shifting the paradigm from GUI scripting to structured, programmable application control. Even when paired with general-purpose models like GPT-40, UFO$^2$ outperforms dedicated CUAs by over 10%, highlighting the transformative impact of system-level integration and architectural design.

总体而言，UFO$^2$ 通过将范式从 GUI 脚本转向结构化、可编程的应用控制，推进了原生操作系统自动化的愿景。即便与诸如 GPT-40 之类的通用模型配合，UFO$^2$ 相较专用 CUA 仍领先超过 10%，凸显了系统级集成与架构设计的变革性影响。

# 2 Background

# 2 背景

## 2.1 The Fragility of Traditional Desktop Automation

## 2.1 传统桌面自动化的脆弱性

For decades, desktop automation has relied on brittle techniques to replicate human interactions with GUI-based applications. Commercial RPA (Robotic Process Automation) tools-such as UiPath [1], Automation Anywhere [2], and Microsoft Power Automate [3]-operate by recording and replaying mouse movements, keystrokes, or rule-based scripts. These systems rely heavily on surface-level GUI cues (e.g., pixel regions, window titles), offering little introspection into application state.

数十年来，桌面自动化依赖脆弱的技术来复现人与基于 GUI 的应用的交互。商业 RPA(机器人流程自动化) 工具——如 UiPath [1]、Automation Anywhere [2] 和 Microsoft Power Automate [3]——通过录制和重放鼠标移动、按键或基于规则的脚本来运行。这些系统高度依赖表面级 GUI 线索 (例如像素区域、窗口标题)，对应用状态几乎没有内省能力。

While widely deployed in enterprise settings, traditional RPA systems exhibit poor robustness and scalability [19]. Even minor UI updates-such as reordering buttons or relabeling menus-can silently break automation scripts. Maintaining correctness requires frequent human intervention. Furthermore, these tools lack semantic understanding of application workflows and cannot reason about or adapt to novel tasks. As a result, RPA tools remain constrained to narrow, repetitive workflows in stable environments, far from general-purpose automation.

尽管在企业环境中被广泛部署，传统 RPA 系统在鲁棒性和可扩展性方面表现不佳 [19]。即便是轻微的界面更新——比如按钮重排或菜单重命名——也可能悄然破坏自动化脚本。维护正确性需要频繁的人为干预。此外，这些工具缺乏对应用工作流的语义理解，无法对新任务进行推理或自适应。因此，RPA 工具仍然局限于稳定环境中的狭窄重复性工作流，远非通用自动化。

## 2.2 Rise of Computer-Using Agents

## 2.2 计算机使用型代理的兴起

Recent advances in large language models (LLMs) and multimodal perception have enabled a new class of automation systems, referred to as Computer-Using Agents (CUAs) [7- 9]. CUAs aim to generalize across applications and tasks by leveraging LLMs to interpret user instructions, perceive GUI layouts, and synthesize actions such as clicks and keystrokes. Early CUAs like UFO [10] demonstrated that multimodal models (e.g., GPT-4V [20]) could map natural language requests to sequences of GUI actions with no hand-crafted scripts. More recent industry prototypes, including Claude- 3.5 (Computer Use) [11] and OpenAI Operator [12], have pushed the envelope further, performing realistic desktop workflows across multiple applications.

近期大语言模型 (LLM) 和多模态感知的进展催生了一类新的自动化系统，称为计算机使用型代理 (CUA)[7-9]。CUA 旨在通过利用 LLM 来解释用户指令、感知 GUI 布局并合成点击与键击等操作，从而在应用和任务间实现泛化。早期的 CUA(如 UFO [10]) 展示了多模态模型 (例如 GPT-4V [20]) 能够在无需手工脚本的情况下将自然语言请求映射为一系列 GUI 操作。更近期的工业原型，包括 Claude-3.5 (Computer Use) [11] 与 OpenAI Operator [12]，进一步推进了这一方向，能够跨多应用执行逼真的桌面工作流。

These CUAs represent a promising evolution from static RPA scripts to adaptive, general-purpose automation. However, despite their sophistication, current CUAs largely remain research prototypes, constrained by architectural and systems-level limitations that impede real-world deployment.

这些 CUA 代表了从静态 RPA 脚本向自适应、通用自动化演进的有希望方向。然而，尽管其复杂性日益提高，目前的 CUA 在很大程度上仍是研究原型，受限于架构与系统级的局限，这些限制阻碍了其在现实中的部署。

## 2.3 Systems Challenges in CUAs

## 2.3 CUA 的系统挑战

Current CUAs fall short in three fundamental ways, which we argue stem from missing operating system abstractions:

当前 CUA 在三个根本方面存在不足，我们认为这些问题源于缺失的操作系统抽象:

(1) Lack of OS-Level Integration. Most CUAs interact with the system through screenshots and low-level input emulation (mouse and keyboard events). They ignore rich system interfaces such as accessibility APIs, application process state, and native inter-process communication mechanisms (e.g., shell commands, COM interfaces [21]). This superficial interaction model limits reliability and efficiency-every action must be inferred from pixels rather than structured state.

(1) 缺乏操作系统级集成。大多数 CUA 通过截图和低级输入仿真 (鼠标和键盘事件) 与系统交互。它们忽视了丰富的系统接口，如无障碍 API、应用进程状态和本地进程间通信机制 (例如 shell 命令、COM 接口 [21])。这种表面化的交互模型限制了可靠性与效率——每个动作必须从像素中推断，而不是基于结构化状态。

(2) Absence of Application Introspection. CUAs typically operate as generalists with limited awareness of application-specific capabilities. They treat all interfaces uniformly, lacking the ability to leverage built-in APIs or vendor documentation. As a result, they cannot reason over high-level concepts unless such flows are explicitly embedded in the model. This rigidity limits their generalization and makes maintenance expensive.

(2) 缺乏应用自省能力。CUA 通常作为通用工具运行，对应用特定能力的感知有限。它们对所有界面一视同仁，缺乏利用内建 API 或厂商文档的能力。因此，除非这些流程直接嵌入模型，否则它们无法对高层概念进行推理。这种僵化限制了泛化能力并使维护代价高昂。

(3) Disruptive and Unsafe Execution Model. Most CUAs drive automation directly on the user's desktop session, hijacking the real mouse and keyboard. This design prevents users from interacting with their system during execution, introduces interference risk, and violates isolation principles fundamental to safe system design. Long-running tasks-especially those involving multiple LLM queries-can monopolize the session for minutes at a time.

(3) 具有破坏性和不安全的执行模型。大多数 CUA 直接在用户的桌面会话上驱动自动化，劫持真实的鼠标和键盘。该设计阻止用户在执行期间与系统交互，带来干扰风险，并违背安全系统设计的隔离原则。尤其是涉及多次 LLM 查询的长时间任务，可能会将会话独占数分钟。

## 2.4 Missing Abstraction: OS Support for Automation

## 2.4 缺失的抽象: 对自动化的操作系统支持

Despite growing demand for intelligent, language-driven automation, existing operating systems offer no first-class abstraction for exposing GUI application control to external agents. In contrast to system calls, files, or sockets, GUI work-flows remain opaque and non-programmable. As a result, both RPA and CUA systems are forced to operate as ad-hoc layers atop the GUI, with no unified substrate for execution, coordination, or introspection.

尽管对智能、语言驱动自动化的需求不断增长，现有操作系统并未提供将 GUI 应用控制暴露给外部代理的一级抽象。与系统调用、文件或套接字不同，GUI 工作流仍然是不透明且不可编程的。因此，RPA 与 CUA 系统被迫作为 GUI 之上的临时层运行，缺乏用于执行、协调或自省的统一基底。

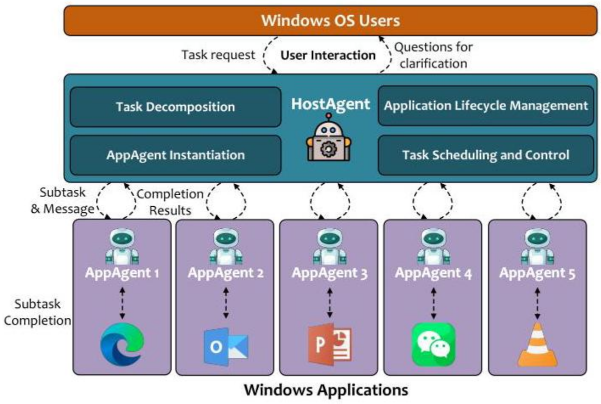


Figure 2. An overview of the architecture of UFO$^2$ .

图 2. UFO$^2$ 架构概览。

This paper argues that automation should be elevated to a system primitive. We present **UFO**$^2$ , a new AgentOS for Windows that addresses these limitations by embedding automation as a deeply integrated OS abstraction-exposing GUI controls, application APIs, and task orchestration as programmable, inspectable, and composable system services.

本文主张应将自动化提升为系统原语。我们提出 **UFO**$^2$，一个面向 Windows 的新 AgentOS，通过将自动化嵌入为深度集成的操作系统抽象——将 GUI 控制、应用 API 与任务编排作为可编程、可检查且可组合的系统服务——来解决这些限制。

# 3 System Design of UFO$^2$

# 3 UFO$^2$ 的系统设计

Motivated by the challenges highlighted in Section 2, UFO2 is designed to seamlessly interpret natural language user requests and reliably automate tasks across a wide range of Windows applications. This section provides an architectural overview of UFO$^2$ (Section 3.1) and explains how each component is deeply integrated with the underlying OS to overcomes the pitfalls of current CUAs, ultimately enabling a practical, robust AgentOS for desktop automation.

针对第 2 节中指出的挑战，UFO2 被设计为能够无缝理解自然语言用户请求并可靠地在各种 Windows 应用间自动执行任务。本节提供 UFO$^2$ 的架构概览 (第 3.1 节)，并解释每个组件如何与底层操作系统深度集成，以克服当前 CUA 的陷阱，最终实现实用且可靠的桌面自动化 AgentOS。

## 3.1 UFO $^2$ as a System Substrate for Automation

## 3.1 将 UFO $^2$ 作为自动化的系统基底

Figure 2 presents the high-level architecture of UFO$^2$ , which provides a structured runtime environment for task-oriented automation on Windows desktops. Deployed as a local daemon, UFO$^2$ enables users to issue natural language requests that are translated into coordinated workflows spanning multiple GUI applications. The system provides core abstractions for orchestration, introspection, control execution, and agent collaboration-exposing these as system-level services analogous to those in traditional OSes.

图 2 展示了 UFO$^2$ 的高层架构，提供了一个面向任务自动化的 Windows 桌面结构化运行时环境。作为本地守护进程部署，UFO$^2$ 使用户能够发出自然语言请求，并将其翻译为跨多个 GUI 应用的协调工作流。该系统提供用于编排、内省、控制执行和代理协作的核心抽象——将这些作为类似传统操作系统中的系统级服务对外公开。

At the heart of UFO$^2$ is a central control plane, the HostA-GENT, responsible for parsing user intent, managing system state, and dispatching subtasks to a collection of specialized runtime modules called AppAGENTS. Each APPAGENT is dedicated to a particular application (e.g., Excel, Outlook, File Explorer) and encapsulates all logic needed to observe and control that application, including API bindings, UI detectors, and knowledge bases. These modules act as isolated execution contexts with application-specific semantics.

UFO$^2$ 的核心是一个中央控制平面 HostA-GENT，负责解析用户意图、管理系统状态并将子任务分派给一组称为 AppAGENTS 的专用运行时模块。每个 APPAGENT 专注于特定应用 (例如 Excel、Outlook、文件资源管理器)，封装了观察和控制该应用所需的全部逻辑，包括 API 绑定、UI 检测器和知识库。这些模块作为具有应用特定语义的隔离执行上下文运行。

Upon receiving a user request, HosTAGENT decomposes it into a series of subtasks, each mapped to the application best suited to fulfill it. If the corresponding application is not already running, HosTAGENT launches it using native Windows APIs and instantiates the corresponding APPA-GENT. Execution proceeds through a structured loop: each APPAGENT continuously observes the application state (via accessibility APIs and vision-based detectors), reasons about the next operation using a ReAct-style planning cycle [22], and invokes the appropriate action-either a GUI event or a native API call. This loop continues until the subtask terminates, either successfully or due to an unrecoverable error.

在接收用户请求后，HosTAGENT 将其分解为一系列子任务，每个子任务映射到最适合执行它的应用。如果相应应用尚未运行，HosTAGENT 会使用原生 Windows API 启动该应用并实例化相应的 APPAGENT。执行按结构化循环进行: 每个 APPAGENT 持续通过可访问性 API 和基于视觉的检测器观察应用状态，使用类似 ReAct 的规划循环 [22] 推理下一步操作，并调用相应动作——要么触发 GUI 事件，要么调用原生 API。该循环持续直到子任务终止，或成功完成，或因不可恢复错误而结束。

UFO$^2$ implements shared memory and control flow via a global blackboard interface, allowing HosTAGENT and APPA-GENTs to exchange intermediate results, dependency states, and execution metadata. This architecture supports complex workflows across application boundaries-for instance, extracting data from a spreadsheet and using it to populate fields in a web form-without requiring hand-crafted scripts or coordination logic. Crucially, all interactions occur within a virtualized, PiP-based desktop environment, ensuring process-level isolation and safe multi-application concurrency.

UFO$^2$ 通过全局黑板接口实现共享内存和控制流，允许 HosTAGENT 与 APPAGENTs 交换中间结果、依赖状态和执行元数据。该架构支持跨应用边界的复杂工作流——例如从电子表格提取数据并用以填充网页表单字段——无需手工脚本或协调逻辑。关键是，所有交互都在虚拟化的 PiP 桌面环境内发生，确保进程级隔离和安全的多应用并发。

Design Rationale: Centralized Multiagent Runtime. UFO$^2$ adopts a centralized multiagent [10,23-25] runtime to support both reliability and extensibility. The centralized HosTAGENT acts as a control plane, simplifying task-level orchestration, error handling, and lifecycle management. Meanwhile, each APPAGENT is architected as a loosely coupled executor that encapsulates deep, application-specific functionality.

设计原理: 集中式多智能体运行时。UFO$^2$ 采用集中式多智能体 [10,23-25] 运行时以支持可靠性和可扩展性。集中式的 HosTAGENT 作为控制平面，简化了任务级编排、错误处理和生命周期管理。与此同时，每个 APPAGENT 被设计为松耦合的执行器，封装了深度的、应用特定的功能。

Modularity at the APPAGENT level allows developers and third-party contributors to incrementally expand UFO2's capabilities by authoring new application interfaces and API bindings. These agents are discoverable, self-contained, and dynamically instantiated by the runtime as needed. From a security and evolvability perspective, this separation of concerns ensures that application logic can evolve independently of the core task orchestration engine.

在 APPAGENT 级别的模块化允许开发者和第三方贡献者通过编写新的应用接口和 API 绑定逐步扩展 UFO2 的能力。这些代理可被发现、自包含，并由运行时按需动态实例化。从安全性和可演化性的角度来看，这种职责分离确保应用逻辑可以独立于核心任务编排引擎演进。

Together, the HosTAGENT -APPAGENT model allows UFO2 to function as a scalable, pluggable runtime substrate for GUI automation-abstracting away the complexity of heterogeneous interfaces and providing a unified system interface to structured application behavior.

总体而言，HosTAGENT–APPAGENT 模式使 UFO2 能够作为可扩展、可插拔的 GUI 自动化运行时基底运行——抽象出异构界面的复杂性，并向结构化的应用行为提供统一的系统接口。

## 3.2 HosTAGENT: System-Level Orchestration and Execution Control

## 3.2 HosTAGENT: 系统级编排与执行控制

The HostAGENT serves as the centralized control plane of UFO$^2$ . It is responsible for interpreting user-specified goals, decomposing them into structured subtasks, instantiating and dispatching APPAGENT modules, and coordinating their progress across the system. HOSTAGENT provides system-level services for introspection, planning, application lifecy-cle management, and multi-agent synchronization.

HostAGENT 是 UFO$^2$ 的集中控制平面。它负责解释用户指定的目标，将其分解为结构化子任务，实例化并分派 APPAGENT 模块，并在系统范围内协调它们的进展。HOSTAGENT 提供用于内省、规划、应用生命周期管理和多智能体同步的系统级服务。

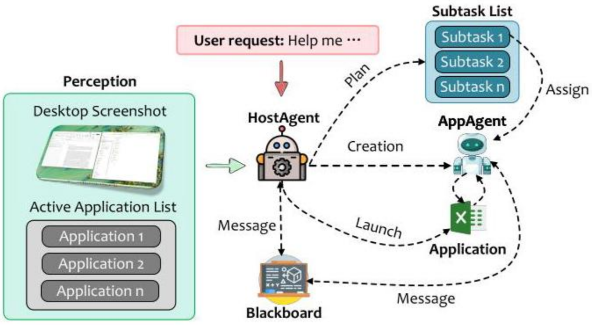


Figure 3. High-level architecture of HOSTAGENT as a control-plane orchestrator.

图 3. 作为控制平面编排器的 HOSTAGENT 的高层架构。

Figure 3 outlines the architecture of HosTAGENT. Operating atop the native Windows substrate, HosTAGENT monitors active applications, issues shell commands to spawn new processes as needed, and manages the creation and teardown of application-specific APPAGENT instances. All coordination occurs through a persistent state machine, which governs the transitions across execution phases.

图 3 概述了 HosTAGENT 的架构。HosTAGENT 在原生 Windows 基础之上运行，监控活动应用，按需发出外壳命令以生成新进程，并管理应用特定 APPAGENT 实例的创建与拆除。所有协调通过一个持久状态机进行，控制执行阶段之间的转换。

Responsibilities and Interfaces. HOSTAGENT exposes the following system services:

职责与接口。HOSTAGENT 暴露以下系统服务:

- Task Decomposition. Given a user's natural language input, HosTAGENT identifies the underlying task goal and decomposes it into a dependency-ordered subtask graph.

- 任务分解。给定用户的自然语言输入，HosTAGENT 识别底层任务目标并将其分解为有依赖顺序的子任务图。

- Application Lifecycle Management. For each subtask, HosTAGENT inspects system process metadata (via UIA APIs) to determine whether the target application is running. If not, it launches the program and registers it with the runtime.

- 应用生命周期管理。对于每个子任务，HosTAGENT 通过 UIA API 检查系统进程元数据以确定目标应用是否正在运行。如未运行，则启动该程序并将其注册到运行时。

- AppAgent Instantiation. HostAgent spawns the corresponding AppAGENT for each active application, providing it with task context, memory references, and relevant toolchains (e.g., APIs, documentation).

- AppAgent 实例化。HostAgent 为每个活动应用生成相应的 AppAGENT，向其提供任务上下文、内存引用和相关工具链 (例如 API、文档)。

- Task Scheduling and Control. The global execution plan is serialized into a finite state machine (FSM), allowing HostAGENT to enforce execution order, detect failures, and resolve dependencies across agents.

- 任务调度与控制。全局执行计划被序列化为一个有限状态机 (FSM)，允许 HostAGENT 强制执行顺序、检测失败并解决跨代理的依赖关系。

- Shared State Communication. HosTAGENT reads from and writes to a global blackboard, enabling interagent communication and system-level observability for debugging and replay.

- 共享状态通信。HosTAGENT 从全局黑板读取并写入数据，支持代理间通信以及用于调试和回放的系统级可观察性。

System Perception and Introspection. To perform its control functions, HosTAGENT fuses two layers of system introspection:

系统感知与自省。为执行控制功能，HosTAGENT 融合了两层系统自省:

1. Visual Layer. Captures pixel-level screenshots of the desktop workspace, enabling coarse-grained layout understanding.

1. 可视层。捕获桌面工作区的像素级截图，实现粗粒度布局理解。

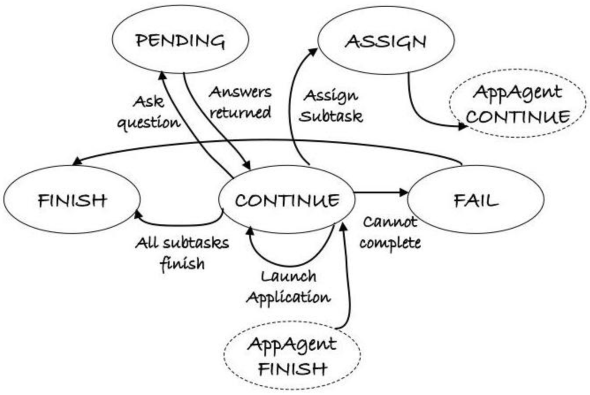


Figure 4. Control-state transitions managed by HosTAGENT.

图 4。HosTAGENT 管理的控制状态转换。

2. Semantic Layer. Queries Windows UIA APIs to extract structural metadata about applications, windows, and control hierarchies.

2. 语义层。查询 Windows UIA API 以提取有关应用、窗口和控件层级的结构化元数据。

This dual perception enables HosTAGENT to resolve ambiguities, detect runtime inconsistencies, and guide agents with context-aware decisions.

这种双重感知使 HosTAGENT 能够消除歧义、检测运行时不一致并以上下文感知的决策指导代理。

Structured Output Interface. HosTAGENT produces structured outputs to drive downstream execution:

结构化输出接口。HosTAGENT 生成结构化输出以驱动下游执行:

- Subtask Plan: A high-level execution plan detailing decomposed subtasks.

- 子任务计划: 详述分解后子任务的高层执行计划。

- Shell Command: A sequence of shell-level invocations for managing application lifecycles.

- Shell 命令: 用于管理应用生命周期的一系列 shell 级调用。

- Assigned Application: The process name and index of the application selected for instantiating the APPA-GENT, which will be used to execute the next subtask.

- 指定应用: 用于实例化 APPAGENT 的被选应用的进程名和索引，该实例将用于执行下一个子任务。

- Agent Messages: Context-specific instructions passed to APPAGENT instances for localized execution.

- 代理消息: 传递给 APPAGENT 实例以进行局部执行的上下文特定指令。

- User Prompts: Interactive clarification requests in cases of ambiguity or failure.

  - 用户提示: 在出现歧义或失败时的交互式澄清请求。

- HosTAGENT State: The current state within the HosTA-GENT's internal FSM.

  - HosTAGENT 状态:HosTAGENT 内部有限状态机的当前状态。

Execution via Finite-State Controller. The core logic of HOSTAGENT is modeled as a finite-state controller (Figure 4), with the following states:

通过有限状态控制器执行。HOSTAGENT 的核心逻辑建模为有限状态控制器 (图 4)，包含以下状态:

- CONTINUE: Main execution loop; evaluates which subtasks are ready to launch or resume.

  - CONTINUE: 主执行循环；评估哪些子任务已准备好启动或恢复。

- ASSIGN: Selects an available application process and spawns the corresponding AppAGENT agent.

  - ASSIGN: 选择可用的应用进程并生成相应的 AppAGENT 代理。

- PENDING: Waits for user input to resolve ambiguity or gather additional task parameters.

  - PENDING: 等待用户输入以消除歧义或收集额外任务参数。

- FINISH: All subtasks complete; cleans up agent instances and finalizes session state.

  - FINISH: 所有子任务完成；清理代理实例并最终确定会话状态。

- FAIL: Enters recovery or abort mode upon irrecoverable failure.

  - FAIL: 在无法恢复的故障时进入恢复或中止模式。

This explicit FSM structure enables HosTAGENT to robustly orchestrate dynamic workflows while maintaining high-level guarantees over task completion and fault isolation.

这种明确的 FSM 结构使 HosTAGENT 能够稳健地协调动态工作流，同时对任务完成和故障隔离保持高级别保证。

Memory and State Management. HosTAGENT maintains two classes of persistent state:

内存与状态管理。HosTAGENT 保持两类持久性状态:

- Private State: Tracks user intent, plan progress, and the control flow of the current session.

- 私有状态: 跟踪用户意图、计划进度和当前会话的控制流。

- Shared Blackboard: A concurrent, append-only memory space that facilitates transparent agent communication by recording key observations, intermediate results, and execution metadata accessible to all AppA-GENT instances.

- 共享黑板: 一个并发的、仅追加的内存空间，通过记录关键观测、中间结果和可被所有 AppA-GENT 实例访问的执行元数据来促进透明的代理通信。

This separation ensures that local context remains encapsulated, while global coordination is visible and consistent across the system.This separation ensures that each agent retains clean, scoped state while benefiting from a globally consistent view for collaborative task execution.

这种分离确保本地上下文保持封装，同时全局协调在系统中是可见且一致的。这种分离确保每个代理保留干净、范围限定的状态，同时受益于用于协作任务执行的全局一致视图。

Overall, HosTAGENT abstracts away the complexity of managing concurrent, stateful, cross-application workflows in desktop environments [26]. Its control-plane role enables modular execution, coordinated progress, and robust task lifecycle management-all critical features in scaling desktop automation to real-world deployments.

总体而言，HosTAGENT 抽象化了在桌面环境中管理并发、有状态、跨应用工作流的复杂性 [26]。其控制平面角色使模块化执行、协调进度和稳健的任务生命周期管理成为可能——这些是在将桌面自动化扩展到实际部署时的关键特性。

## 3.3 APPAGENT: Application-Specialized Execution Runtime

## 3.3 APPAGENT: 应用专用执行运行时

The APPAGENT is the core execution runtime in UFO$^2$ , responsible for carrying out individual subtasks within a specific Windows application. Each APPAGENT functions as an isolated, application-specialized worker process launched and orchestrated by the central HosTAGENT (Section 3.2). Unlike monolithic CUAs that treat all GUI contexts uniformly, each APPAGENT is tailored to a single application and operates with deep knowledge of its API surface, control semantics, and domain logic.

APPAGENT 是 UFO$^2$ 中的核心执行运行时，负责在特定 Windows 应用内执行各个子任务。每个 APPAGENT 作为由中心 HosTAGENT(第 3.2 节) 启动和协调的隔离、应用专用的工作进程运行。与将所有 GUI 上下文一视同仁的单片 CUA 不同，每个 APPAGENT 针对单个应用进行定制，并对其 API 接口、控件语义和领域逻辑具有深入了解。

Figure 5 outlines the architecture of an APPAGENT. Upon receiving a subtask and execution context from the HosTA-GENT, the APPAGENT initializes a ReAct-style control loop [22], where it iteratively senses the current application state, reasons about the next step, and executes either a GUI or API-based action. This hybrid execution layer-implemented via a Puppeteer interface-enables reliable control over dynamic and complex UIs by favoring structured APIs whenever available, while retaining fallback to GUI-based interaction when necessary.

图 5 概述了 APPAGENT 的架构。在从 HosTA-GENT 接收子任务和执行上下文后，APPAGENT 初始化一个 ReAct 风格的控制循环 [22]，在其中迭代感知当前应用状态、推理下一步并执行基于 GUI 或 API 的操作。该混合执行层通过 Puppeteer 接口实现——优先使用结构化 API(若可用) 以实现对动态且复杂 UI 的可靠控制，同时在必要时保留回退到基于 GUI 的交互。

Perception Layer. Each APPAGENT fuses multiple streams of perception:

感知层。每个 APPAGENT 将多路感知流融合:

- Visual Input: Captures GUI screenshots for layout understanding and control grounding.

- 视觉输入: 捕获 GUI 截图以进行布局理解和控件定位。

- Semantic Metadata: Extracted from Windows UIA APIs, including control type, label, hierarchy, and enabled state.

- 语义元数据: 由 Windows UIA API 提取，包括控件类型、标签、层级和启用状态。

- Symbolic Annotation: Uses Set-of-Mark (SoM) techniques [27] to annotate the control on screenshots.

- 符号注释: 使用 Set-of-Mark (SoM) 技术 [27] 在截图上注释控件。

These fused signals are converted into a structured observation object containing both the GUI screenshot and the set of candidate control elements. This multi-modal representation enables a more comprehensive understanding of the application state, going well beyond raw visual input alone.

这些融合的信号被转换为包含 GUI 截图和候选控件集合的结构化观测对象。这种多模态表示比单纯的视觉输入能更全面地理解应用状态。

Structured Output. Based on this state, the APPAGENT produces a structured output:

结构化输出。基于该状态，APPAGENT 生成结构化输出:

- Target control (if applicable)

- 目标控件 (如适用)

- Action type (e.g., click, type, call API)

- 操作类型 (例如，点击、输入、调用 API)

- Arguments or payload

- 参数或载荷

- Reasoning trace and planning with Chain-of-Thought (CoT) [28, 29]

- 带有链式思考 (CoT) 的推理轨迹与规划 [28, 29]

- Current state in local FSM

- 本地有限状态机中的当前状态

This design decouples perception from actuation, enabling deterministic replay, offline debugging, and fine-grained observability.

该设计将感知与执行解耦，支持确定性重放、离线调试和细粒度可观测性。

Execution via Finite-State Controller. Each AppAGENT maintains a local finite-state machine (Figure 6) that governs its behavior within the assigned application context:

通过有限状态控制器执行。每个 AppAGENT 保持一个局部有限状态机 (图 6)，在分配的应用上下文中管理其行为:

- CONTINUE: Default state for action planning and execution.

- CONTINUE: 默认的动作规划与执行状态。

- PENDING: Invoked for safety-critical actions (e.g., destructive operations); requires user confirmation.

- PENDING: 用于安全关键操作 (例如，破坏性操作)；需要用户确认。

- FINISH: Task completed; execution ends.

- FINISH: 任务完成；执行结束。

- FAIL: Irrecoverable failure detected (e.g., application crash, permission error).

- FAIL: 检测到不可恢复的故障 (例如，应用崩溃、权限错误)。

This bounded execution model isolates failures to the current task and enables safe preemption, retry, or delegation. The FSM also supports interruptible workflows, which can resume from intermediate checkpoints.

该有界执行模型将故障限制在当前任务内，并支持安全抢占、重试或委派。该 FSM 还支持可中断的工作流，可从中间检查点恢复。

Memory and State Coordination. To enable stateful execution and maintain contextual awareness, each APPAGENT maintains:

内存与状态协调。为实现有状态执行并保持上下文感知，每个 APPAGENT 保持:

- Private State: A local log of all executed actions, control decisions, and CoT traces.

- 私有状态: 记录所有已执行动作、控制决策和 CoT 轨迹的本地日志。

- Shared State: Updates to the system-wide blackboard, including intermediate outputs, encountered errors, and application-level insights.

- 共享状态: 对系统范围记事板的更新，包括中间输出、遇到的错误和应用级洞见。

This dual-memory design enables APPAGENTs to act autonomously on behalf of HostAgent while remaining synchronized with the broader system. It also supports compos-ability: one AppAGENT's output may become another's input in downstream subtasks.

这种双存储设计使 APPAGENT 能代表 HostAgent 自主行动，同时与更广泛的系统保持同步。它还支持可组合性: 一个 AppAGENT 的输出可成为下游子任务中另一个的输入。

Application-Aware SDK and Extensibility. To support rapid onboarding of new applications, UFO$^2$ exposes an SDK that encapsulates the development and maintenance of AP-PAGENTS. Developers can register application-specific APIs via a declarative interface, including function metadata, argument schemas, and prompt bindings. Domain-specific help documents and patch notes can be ingested into a searchable knowledge base that agents query at runtime.

面向应用的 SDK 与可扩展性。为支持新应用的快速接入，UFO$^2$ 提供了一个封装 AP-PAGENT 开发与维护的 SDK。开发者可通过声明式接口注册应用特定的 API，包括函数元数据、参数模式和提示绑定。领域特定的帮助文档与更新说明可被摄取进可搜索的知识库，供代理在运行时查询。

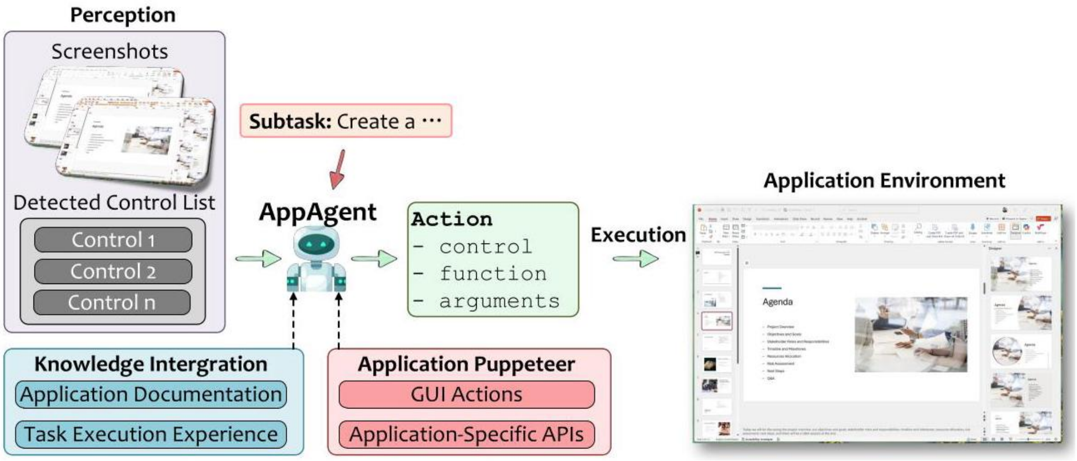


Figure 5. Architecture of an APPAGENT, the per-application execution runtime in UFO$^2$ .

图 5. APPAGENT 架构，即 UFO$^2$ 中的每应用执行运行时。

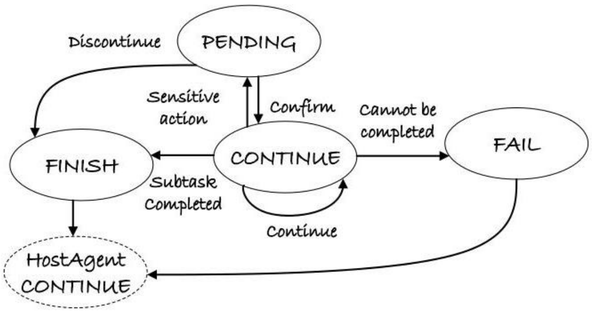


Figure 6. Control-state transitions for an AppAGENT runtime.

图 6. AppAGENT 运行时的控制状态转换。

This modular abstraction allows third-party vendors or power users to extend UFO$^2$ ' s capabilities without retraining any models. New functionality can be integrated by updating the application's APPAGENT module, isolating changes from the rest of the system and minimizing regression risk.

这种模块化抽象允许第三方厂商或高级用户在不重训任何模型的前提下扩展 UFO$^2$ 的能力。通过更新应用的 APPAGENT 模块即可集成新功能，将变更与系统其余部分隔离，最小化回归风险。

Summary. As a per-application execution runtime, each APPAGENT provides modular, domain-aware control that surpasses generic GUI agents in both efficiency and robustness. Its hybrid perception-action loop, plugin-based extensibility, and local fault containment enable UFO $^2$ to scale to large application ecosystems with minimal system-wide disruption.

总结。作为每应用执行运行时，每个 APPAGENT 提供了模块化、领域感知的控制，在效率和鲁棒性上均超越通用 GUI 代理。其混合感知-行动循环、基于插件的可扩展性和本地故障隔离使 UFO $^2$ 能在最小系统级中断下扩展到大型应用生态。

## 3.4 Hybrid Control Detection

## 3.4 混合控件检测

Reliable perception of GUI elements is fundamental to enabling APPAGENTS to interact with application interfaces in a deterministic and safe manner. However, real-world GUI environments exhibit substantial heterogeneity: some applications expose well-structured accessibility data via Windows UI Automation (UIA) APIs, while others-especially legacy or custom applications-render critical controls using non-standard toolkits that bypass UIA entirely.

对 GUI 元素的可靠感知是使 APPAGENT 以确定性和安全方式与应用界面交互的基础。然而，真实世界的 GUI 环境高度异质: 部分应用通过 Windows UI Automation (UIA) API 暴露结构良好的辅助性数据，而其他应用——尤其是遗留或自定义应用——则使用非标准工具包渲染关键控件，完全绕过 UIA。

To address this disparity, UFO$^2$ introduces a hybrid control detection subsystem that fuses UIA-based metadata with vision-based grounding [18, 30, 31] to construct a unified and comprehensive control graph for each application window (Figure 7). This design ensures both coverage and reliability, forming a resilient perceptual foundation for downstream action planning and execution.

> 为解决这一差异，UFO$^2$ 引入了一个混合控件检测子系统，将基于 UIA 的元数据与基于视觉的定位 [18, 30, 31] 融合，以为每个应用窗口构建统一且全面的控件图 (图 7)。该设计兼顾覆盖率与可靠性，为下游动作规划与执行形成稳健的感知基础。

UIA-Layer Detection. When available, UIA offers a semantically rich and high-precision interface for enumerating on-screen controls. The detection pipeline first queries the accessibility tree to extract controls satisfying a set of runtime predicates (e.g., is_visible(), is_enabled()). These controls are annotated with their attributes (type, label, bounding box) and assigned stable identifiers, forming the initial control graph.

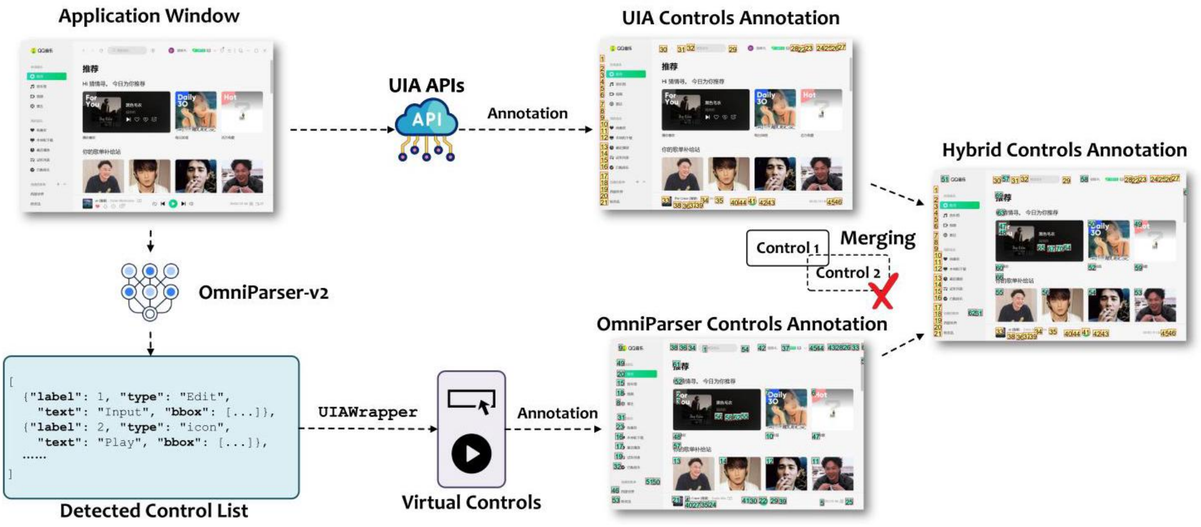
> UIA 层检测。若可用，UIA 提供了语义丰富且高精度的界面枚举控件手段。检测管道首先查询辅助树以提取满足一组运行时谓词 (例如 is_visible(), is_enabled()) 的控件。这些控件被标注其属性 (类型、标签、边界框) 并分配稳定标识符，构成初始控件图。

Vision-Layer Augmentation. To augment the perception pipeline for UI-invisible or custom-rendered controls, we integrate OmniParser-v2 [18], a vision-based grounding model designed for fast and accurate GUI parsing. OmniParser-v2 combines a lightweight YOLO-v8 [32] detector with a fine-tuned Florence-2 (0.23B) [33] encoder to process raw application screenshots and identify additional interactive elements. Each detection includes the control type, confidence score, and spatial bounding box.

> 视觉层增强。为增强对 UI 不可见或自定义渲染控件的感知管道, 我们集成了 OmniParser-v2 [18], 一款为快速且精确的 GUI 解析而设计的基于视觉的定位模型。OmniParser-v2 结合了轻量级的 YOLO-v8 [32] 检测器和微调过的 Florence-2 (0.23B) [33] 编码器，用以处理原始应用截图并识别额外的交互元素。每次检测包含控件类型、置信度分数和空间边界框。

Fusion and Deduplication. We unify these two streams by performing deduplication based on bounding-box overlap. Visual detections with Intersection-over-Union (IoU) greater than 10% against any UIA-derived control are discarded. Remaining visual-only detections are converted into pseudo-UIA objects using a lightweight UIAWrapper abstraction, allowing them to seamlessly integrate into the rest of the APPAGENT pipeline. This fused control set is passed downstream to the SoM-based annotation module [27]. Figure 7 shows a typical scenario involving a hybrid-rendered GUI. Yellow bounding boxes denote standard UIA-detected elements, while blue bounding boxes represent visual-only detections. Both are integrated into a single actionable control graph consumed by the AppAGENT.

> 融合与去重。我们通过基于边界框重叠进行去重来统一这两路信息。对任一 UIA 派生控件，视觉检测若与之的交并比 (IoU) 超过 10% 即被丢弃。剩余的仅视觉检测被转换为伪 UIA 对象，使用轻量级 UIAWrapper 抽象，使其无缝融入 APPAGENT 管道的其余部分。该融合控件集被传递到基于 SoM 的标注模块 [27]。图 7 展示了涉及混合渲染 GUI 的典型场景。黄色边界框表示标准 UIA 检测到的元素，蓝色边界框表示仅视觉检测到的元素。两者被集成到 AppAGENT 消费的单一可操作控件图中。



```
[
  {"label": 1, "type": "Edit",
    "text": "Input", "bbox": [...]},
  {"label": 2, "type": "icon",
    "text": "Play", "bbox": [...]},
  ......
]
```

Figure 7. The hybrid control detection approach employed in UFO $^2$ .

图 7. UFO $^2$ 中采用的混合控件检测方法。

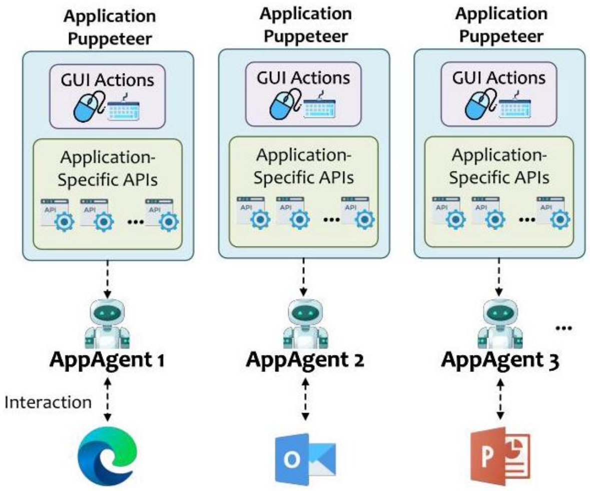


Figure 8. Puppeteer serves as a unified execution engine that harmonizes GUI actions and native API calls.

图 8. Puppeteer 作为统一执行引擎，协调 GUI 操作与原生 API 调用。

## 3.5 Unified GUI-API Action Orchestrator

## 3.5 统一 GUI-API 操作编排器

APPAGENTS can interact with application environments that expose two distinct classes of interfaces: GUI frontends, which are universally observable but often brittle; and native APIs, which are high-fidelity but require explicit integration [17]. To unify these heterogeneous execution backends under a single runtime abstraction, UFO[2] introduces the Puppeteer, a modular execution orchestrator that dynamically selects between GUI-level automation and application-specific APIs for each action step (Figure 8). This design significantly improves task robustness, latency, and maintainability. Tasks that would otherwise require long GUI interaction sequences (e.g., iteratively selecting and formatting cells in Excel) can often be collapsed into a single API call, reducing both execution time and the surface area for failure [17, 34].

APPAGENTS 可以与应用环境交互，这些环境暴露两类不同的接口: 普遍可观测但常脆弱的 GUI 前端；以及需显式集成但高保真度的本地 API[17]。为将这些异构执行后端统一到单一运行时抽象中，UFO[2] 引入了 Puppeteer，一种模块化执行协调器，可在每个动作步骤动态选择 GUI 级自动化或应用特定 API(图 8)。该设计显著提升了任务鲁棒性、延迟和可维护性。原本需长串 GUI 交互的任务 (例如在 Excel 中反复选取并格式化单元格) 往往可合并为单次 API 调用，减少执行时间和故障面 [17, 34]。

```
@ExcelWinCOMReceiver.register
```

```
@ExcelWinCOMReceiver.register
```

```
class SelectTableRangeCommand:
```

```
class SelectTableRangeCommand:
```

```
    def execute(self) -> Dict[str, str]:
```

```
    def execute(self) -> Dict[str, str]:
```

```
        ...
        return {”results”:... ”error”:...}
```

```
        return {”results”:... ”error”:...}
```

```
    @classmethod
```

```
    @classmethod
```

```
    def name(cls) -> str:
```

```
    def name(cls) -> str:
```

```
        return ”select_table_range”
```

```
        return ”select_table_range”
```

Figure 9. Example API registration for Excel.

图 9. Excel 的示例 API 注册。

Puppeteer supports a lightweight API registration mechanism that enables developers to expose high-level operations in target applications. APIs are registered using a simple Python decorator interface, as shown in Figure 9. Each function is wrapped with metadata-name, argument schema, and application binding-and automatically incorporated into the APPAGENT's runtime action space.

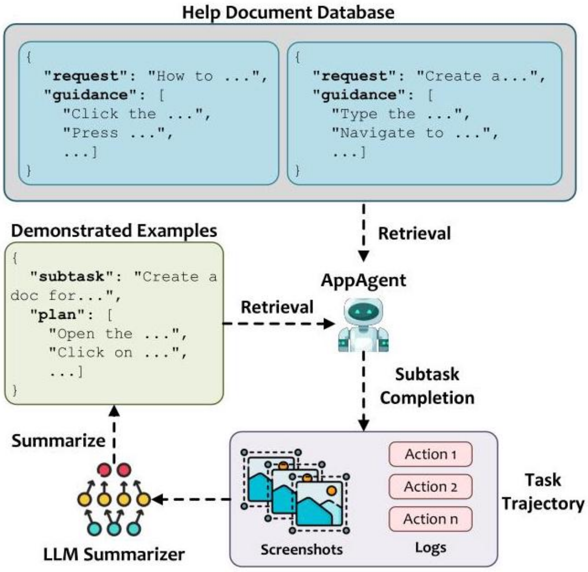
Puppeteer 支持一种轻量级的 API 注册机制，使开发者能在目标应用中公开高阶操作。API 使用如图 9 所示的简单 Python 装饰器接口注册。每个函数被包装以元数据—名称、参数模式和应用绑定—并自动并入 APPAGENT 的运行时动作空间。

At runtime, UFO $^2$ prompts APPAGENT to employs a decision-making policy to choose the most appropriate execution path for each operation. If a semantically equivalent API is available, Puppeteer prefers it over GUI automation for reliability and atomicity.If an API fails or is unavailable (e.g., missing bindings or runtime permission errors), the system gracefully falls back to GUI-based control via simulated clicks or keystrokes. This runtime flexibility allows APPAGENT to preserve robustness across heterogeneous environments without sacrificing generality.

在运行时，UFO $^2$ 提示 APPAGENT 使用决策策略为每个操作选择最合适的执行路径。如果存在语义等价的 API，Puppeteer 优先使用它以提高可靠性与原子性。若 API 失败或不可用 (例如缺失绑定或运行时权限错误)，系统将优雅地回退到通过模拟点击或按键的 GUI 控制。此运行时灵活性允许 APPAGENT 在异构环境中维持鲁棒性而不牺牲通用性。

Puppeteer transforms action execution in UFO$^2$ from a monolithic GUI-centric model into a flexible, OS-integrated control layer that mixes perceptual agility with semantic precision. This hybrid execution model not only improves system performance and stability but also lays the groundwork for sustainable integration of application-specific capabilities in future desktop agents.

Puppeteer 将 UFO$^2$ 中的动作执行从单一的 GUI 中心模型转变为混合的、与操作系统集成的控制层，兼具感知敏捷性与语义精确度。这种混合执行模型不仅提升了系统性能与稳定性，也为未来桌面代理可持续集成应用特定能力奠定了基础。



```
{
  "request": "How to ...",
  "guidance": [
    "Click the ...",
    "Press ...",
    ...]
}
```

```
{
  "request": "Create a...",
  "guidance": [
    "Type the ...",
    "Navigate to ...",
    ...]
}
```

```
{
  "subtask": "Create a
doc for...",
  "plan": [
    "Open the ...",
    "Click on ...",
    ...]
}
```

Figure 10. Overview of the knowledge substrate in UFO$^2$ , combining static documentation with dynamic execution history.

图 10. UFO$^2$ 中知识基底概览，将静态文档与动态执行历史结合。

## 3.6 Continuous Knowledge Integration Substrate

## 3.6 持续知识集成基底

Unlike traditional CUAs, which rely heavily on static training corpora, UFO$^2$ introduces a persistent and extensible knowledge substrate that supports runtime augmentation of application-specific understanding. As illustrated in Figure 10, this substrate enables each APPAGENT to retrieve, interpret, and apply external documentation and prior execution traces without requiring retraining of the underlying models. This hybrid memory design functions analogously to an OS-level metadata manager, abstracting over two key knowledge flows: static references (e.g., user manuals) and dynamic experience (e.g., execution logs).

与高度依赖静态训练语料的传统 CUA 不同，UFO$^2$ 引入了一个持久且可扩展的知识基底，支持在运行时增强应用特定的理解。如图 10 所示，该基底使每个 APPAGENT 能在无需重训练底层模型的情况下检索、解析并应用外部文档和先前执行痕迹。这种混合记忆设计类似于操作系统级的元数据管理器，抽象出两条关键知识流: 静态参考 (例如用户手册) 和动态经验 (例如执行日志)。

Bootstrapping from Documentation. Most real-world desktop applications expose substantial task-level documentation via user guides, help menus, or online tutorials. UFO $^2$ capitalizes on this resource by offering a one-click interface to parse and ingest such documents into an application-specific vector store. Documents are structured as json records with a natural language description in the request field and detailed execution guidance in the guidance field.

从文档自举。大多数真实桌面应用通过用户指南、帮助菜单或在线教程公开大量任务级文档。UFO [2] 利用此资源，提供一键界面以解析并摄取此类文档到应用特定的向量存储中。文档以 json 记录结构化，request 字段为自然语言描述，guidance 字段包含详细执行指导。

At runtime, when an APPAGENT receives a subtask, it queries this indexed store to retrieve relevant guidance, which is then injected into the agent's prompt. This mechanism effectively mitigates cold-start issues-especially when handling novel applications or infrequent operations-by enriching the agent's reasoning context with domain-grounded procedural knowledge.

在运行时，当 APPAGENT 收到子任务时，它查询此索引存储以检索相关指导，然后将其注入到代理的提示中。该机制通过用领域基础的过程性知识丰富代理的推理上下文，有效缓解了冷启动问题——尤其是在处理新颖应用或不常见操作时。

Reinforcing from Experience. Beyond static knowledge, UFO$^2$ continuously learns from its own execution history. Each automation run produces structured logs-including natural language task descriptions, executed action sequences, application screenshots, and final outcomes. Periodically, these logs are mined offline by a summarization module that distills successful trajectories into reusable Example records.

从经验中强化。除了静态知识外，UFO$^2$ 会不断从自身执行历史中学习。每次自动化运行都会产出结构化日志——包括自然语言的任务描述、执行的动作序列、应用程序截图和最终结果。定期地，这些日志由汇总模块离线挖掘，将成功的轨迹提炼为可重用的示例记录。

Each record contains a task signature and an associated step-by-step plan, stored in an application-specific example database. When a similar task is encountered in the future, APPAGENT uses In-Context Learning (ICL) [35-38] to retrieve relevant demonstrations and improve execution fidelity. This dynamic reinforcement pipeline transforms the system into a long-lived agent that improves with use, without introducing the brittleness or operational cost of fine-tuning [39].

每条记录包含任务签名和相应的逐步计划，存储在特定应用的示例数据库中。当将来遇到类似任务时，APPAGENT 使用上下文学习 (ICL)[35-38] 检索相关示范以提高执行准确性。这个动态强化流程将系统转变为随使用而改进的长期代理，同时避免了微调所带来的脆弱性或运营成本 [39]。

Runtime RAG Integration. At the system level, the knowledge substrate acts as a Retrieval-Augmented Generation (RAG) layer [40-43] that bridges the gap between pre-trained language models and application-specific requirements. Because both help documents and examples are indexed with semantic embeddings, the retrieval pipeline is fast, interpretable, and cache-friendly. Additionally, versioned indexing ensures that knowledge artifacts can evolve alongside software updates, preventing model obsolescence and supporting robust execution across long deployment cycles.

运行时 RAG 集成。在系统层面，知识基底充当检索增强生成 (RAG) 层 [40-43]，弥合了预训练语言模型与特定应用需求之间的差距。由于帮助文档和示例都用语义嵌入建立索引，检索流程快速、可解释且易于缓存。此外，版本化索引确保知识工件能随软件更新而演进，防止模型过时并支持在长期部署周期内的稳健执行。

By integrating static and experiential knowledge into a unified RAG pipeline, UFO2 transforms CUAs from brittle, training-time constructs into dynamic, evolving agents. This substrate plays a foundational role in enabling sustainable automation across complex, heterogeneous application ecosystems.

通过将静态与经验性知识整合到统一的 RAG 流水线中，UFO2 将 CUA 从脆弱的训练时构造转变为动态、不断演化的代理。该基底在支持复杂异构应用生态系统中的可持续自动化方面起到基础性作用。

## 3.7 Speculative Multi-Action Execution

## 3.7 推测性多动作执行

Conventional CUAs suffer from a fundamental execution bottleneck: each automation step is executed in isolation, requiring a full LLM inference for every single GUI action. This step-wise inference loop introduces excessive latency, inflates system resource usage, and increases cumulative error rates-especially when interacting with complex or multi-phase workflows [17]. The root cause is the dynamic and uncertain nature of GUI environments, where any single action may alter the interface and invalidate future plans.

传统 CUA 存在一个根本的执行瓶颈: 每个自动化步骤独立执行，每一次 GUI 操作都需要完整的 LLM 推理。这样的逐步推理循环带来过高的延迟、膨胀的系统资源使用并增加累计错误率——尤其在与复杂或多阶段工作流交互时 [17]。根本原因在于 GUI 环境的动态和不确定性，任何单一动作都可能改变界面并使后续计划失效。

To overcome these limitations, UFO$^2$ introduces a system-level optimization called speculative multi-action execution, inspired by classical ideas from speculative execution in processor design and instruction pipelining. Rather than issuing one action per LLM call, UFO$^2$ speculatively generates a batch of likely next steps using a single inference pass and validates their applicability at runtime through tight OS integration. We present an algorithm in 1.

为了克服这些限制，UFO$^2$ 引入了一种系统级优化，称为推测性多动作执行，其灵感来自处理器设计和指令流水线中的推测执行经典思想。UFO$^2$ 不再为每个动作发出一次 LLM 调用，而是通过一次推理传递推测性生成一批可能的下一步，并通过与操作系统的紧密集成在运行时验证其适用性。我们在算法 1 中给出该算法。

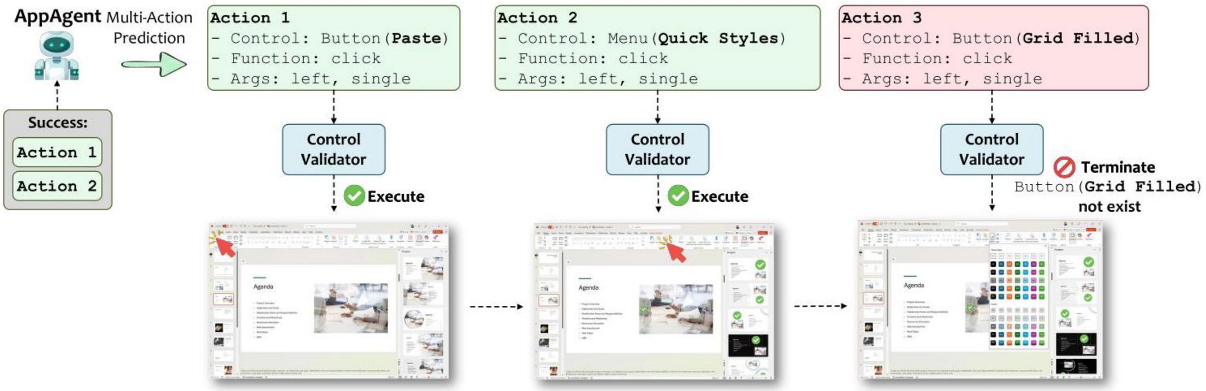


Figure 11. Speculative multi-action execution in UFO$^2$ : batched inference with online validation.

图 11. UFO$^2$ 中的推测性多动作执行: 带有在线验证的批量推理。

Algorithm 1 Speculative Multi-Action Execution in UFO $^2$

算法 1 UFO $^2$ 中的推测性多动作执行

---

Require: Initial UI context $C_0$ , batch size $k$

要求: 初始 UI 上下文 $C_0$， 批量大小 $k$

Ensure: List Executed of actions completed so far

保证: 已完成动作列表 Executed

Stage 1: Batch Prediction

阶段 1: 批量预测

$A \leftarrow \text{LLM\_PreDICT}(C_0, k)$ $\triangleright$ $A = \left[(\text{ctrl}_i, \text{ op}_{\,i}\right]_{i=1}^{k}$
Executed $\leftarrow$ []; $C \leftarrow C_0$

已执行 $\leftarrow$ []; $C \leftarrow C_0$

Stage 2 & 3: Sequential Validate-Execute Loop

阶段 2 & 3: 顺序验证-执行循环

for $i \leftarrow 1$ to $k$ do

for $i \leftarrow 1$ to $k$ do

(ctrl, op,_) $\leftarrow A[i]$

(ctrl, op,_) $\leftarrow A[i]$

// Validate in the current context

// 在当前上下文中验证

if not UIA_IsENABLED(ctrl, C) V not

if not UIA_IsENABLED(ctrl, C) V not

UIA_IsVISIBLE(ctrl, C) then

UIA_IsVISIBLE(ctrl, C) then

```
            break ▷ validation failed → early stop
            break ▷ 验证失败 → 提前停止
        end if
        end if
        //Execute and refresh context
        // 执行并刷新上下文
        EXECUTE(ctrl, op)
        EXECUTE(ctrl, op)
        append (ctrl, op) to Executed
        append (ctrl, op) to Executed
        $C \leftarrow$ UIA_GETCONTEXT()
        $C \leftarrow$ UIA_GETCONTEXT()
    end for
    end for
    if |Executed| < k then
    if |Executed| < k then
        ReportPartial(Executed)
        ReportPartial(Executed)
        Replan(C)
        Replan(C)
    end if
    end if
```

---

The speculative executor operates in three stages:

推测执行器分三阶段运行:

1. Action Prediction: The APPAGENT issues a single LLM query to predict multiple plausible actions under its current context. Each predicted step includes a target control, intended operation, and rationale.

1. 动作预测:APPAGENT 发起一次 LLM 查询，在当前上下文下预测多种可行动作。每个预测步骤包括目标控件、预期操作和理由。

2. Runtime Validation: For each action, the system consults the Windows UIA API to verify the action's preconditions (e.g., is_enabled(), is_visible()). This check ensures that each target control is still valid and interactive.

2. 运行时验证: 对于每个动作，系统查询 Windows UIA API 以核验动作的前置条件 (例如 is_enabled(), is_visible())。此检查确保每个目标控件仍然有效且可交互。

3. Sequential Execution and Early Exit: Actions are executed in order, halting immediately if any validation fails due to interface change (e.g., control no longer exists or is disabled). The executor then reports a partial result set and prompts the agent to replan.

3. 顺序执行与提前退出: 按顺序执行动作，若任一验证因界面变化失败 (例如控件不再存在或被禁用) 则立即停止。执行器随后报告部分结果并提示代理重新规划。

We show an illustrative example of speculative multi-action execution in Figure 11. In this case, the APPAGENT initially plans to execute three actions in a single step: clicking Paste, then Quick Style, and finally Grid Filled. However, after the second action, the control validator detects that the control required for the third action (Grid Filled) is no longer present-likely because the GUI layout changed as a result of the previous step. The Puppeteer then terminates execution at that point and returns the partial results. This example highlights how UFO$^2$ safely handles speculative execution by validating each control before acting, ensuring robustness even in the face of dynamic interface changes.

我们在图 11 中给出一个示例，说明推测性多动作执行。在此情形下，APPAGENT 最初计划在一步中执行三次操作: 先点击 Paste，再 Quick Style，最后 Grid Filled。然而在第二个动作之后，控件验证器检测到第三个动作所需的控件 (Grid Filled) 不再存在——很可能是前一步导致的 GUI 布局变化。Puppeteer 随即在该点终止执行并返回部分结果。该示例凸显了 UFO$^2$ 通过在执行前验证每个控件来安全地处理推测性执行，从而在面对动态界面变化时仍能保证稳健性。

Overall, this strategy drastically reduces LLM invocation frequency and amortizes the cost of action planning across multiple steps, while preserving the correctness guarantees of per-step validation. Critically, validation is performed by trusted OS-level APIs instead of vision models, ensuring high reliability and eliminating spurious interactions.

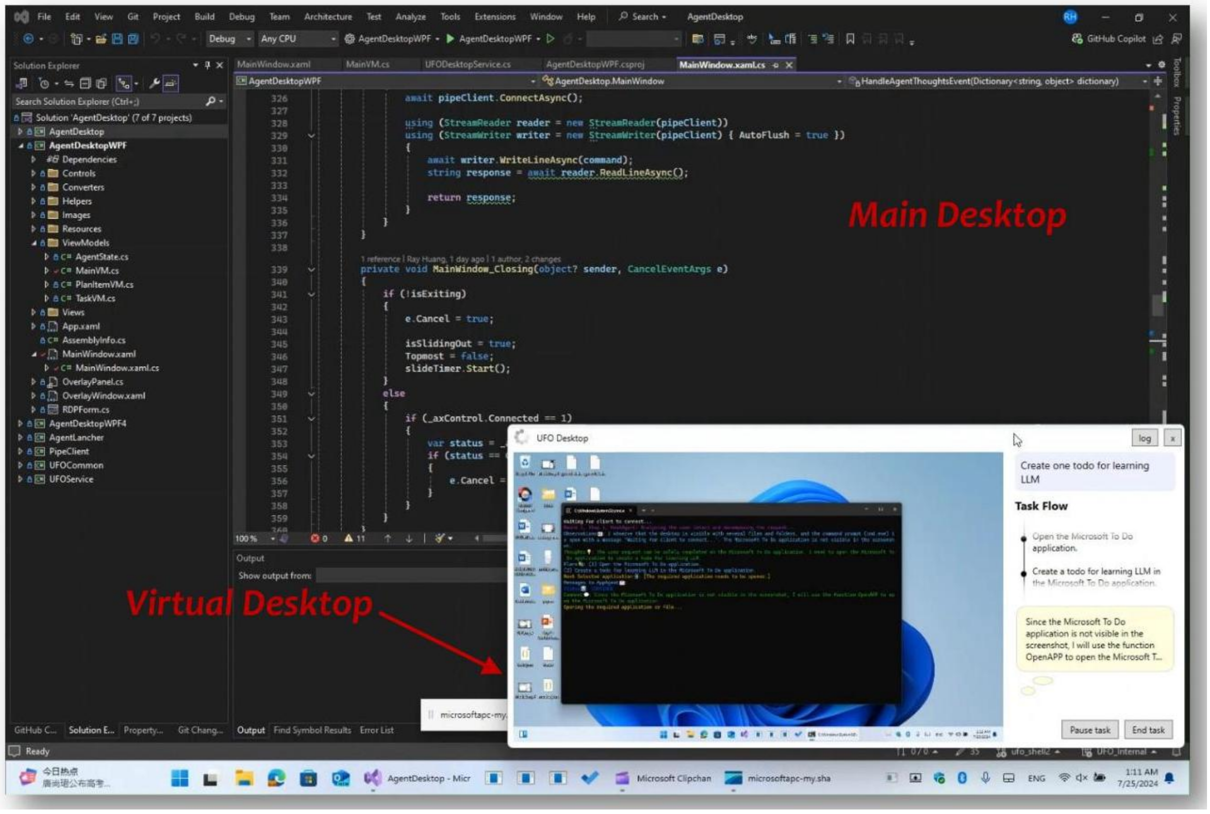
总体上，该策略大幅降低了对 LLM 的调用频率并将动作规划成本分摊到多步上，同时保留每步验证的正确性保证。关键在于，验证由受信任的操作系统级 API 执行，而非视觉模型，确保高可靠性并消除误触发交互。

# 4 Picture-in-Picture Interface

# 4 画中画界面

A key design objective of UFO$^2$ is to deliver high-throughput automation while preserving the responsiveness and usability of the primary desktop environment. Existing CUAs often monopolize the user's workspace, seizing mouse and keyboard control for extended periods and making the system effectively unusable during task execution. To overcome this, UFO$^2$ introduces a Picture-in-Picture (PiP) interface: a lightweight, virtualized desktop window powered by Remote Desktop loopback, enabling fully isolated agent execution in parallel with active user workflows, as illustrated in Figure 12.

UFO$^2$ 的一个关键设计目标是实现高吞吐量自动化，同时保留主桌面环境的响应性与可用性。现有的 CUA 往往会独占用户工作区，长时间控制鼠标和键盘，使系统在任务执行期间实际上不可用。为解决此问题，UFO$^2$ 引入了画中画 (PiP) 界面: 一个由远程桌面回环驱动的轻量虚拟桌面窗口，允许在不干扰用户活动的情况下并行进行完全隔离的代理执行，如图 12 所示。



```
                    await pipeClient.ConnectAsync();

                    using (StreamReader reader = new StreamReader(pipeClient))
                    using (StreamWriter writer = new StreamWriter(pipeClient) { AutoFlush = true })
                    {
                        await writer.WriteLineAsync(command);
                        string response = await reader.ReadLineAsync();

                        return response;
                    }
                }
            }

            1 reference | Ray Huang, 1 day ago | 1 author, 2 changes
            private void MainWindow_Closing(object? sender, CancelEventArgs e)
            {
                if (!isExiting)
                {
                    e.Cancel = true;

                    isSlidingOut = true;
                    Topmost = false;
                    slideTimer.Start();
                }
                else
                {
                    if (_axControl.Connected == 1)
                    {
                        var status = _
                        if (status ==
                        {
                            e.Cancel =
                        }
                    }
                }
```

Create one todo for learning LLM

Task Flow

- Open the Microsoft To Do application.
- Create a todo for learning LLM in the Microsoft To Do application.

Since the Microsoft To Do application is not visible in the screenshot, I will use the function OpenAPP to open the Microsoft T...

Figure 12. The Picture-in-Picture interface: a virtual desktop window for non-disruptive automation.

图 12。画中画界面: 用于非扰动式自动化的虚拟桌面窗口。

## 4.1 Virtualized User Environment with Minimal Disruption

## 4.1 最小干扰的虚拟化用户环境

Unlike conventional CUAs that operate in the main desktop session, the PiP interface presents a resizable, movable window containing a fully functional replica of the user's desktop. Internally, this is implemented via Windows' native Remote Desktop Protocol (RDP) loopback [44], creating a distinct virtual session hosted on the same machine. Applications launched within the PiP session inherit the user's identity, credentials, settings, and network context, ensuring consistency with foreground operations.

与在主桌面会话中运行的传统 CUA 不同，PiP 窗口呈现一个可调整大小、可移动的窗口，包含用户桌面的完整功能副本。其内部通过 Windows 原生的远程桌面协议 (RDP) 回环实现 [44]，在同一台机器上创建一个独立的虚拟会话。PiP 会话中启动的应用继承用户身份、凭据、设置和网络上下文，确保与前台操作一致。

From the user's perspective, the PiP window behaves like a sandboxed workspace: the automation executes in the background, visible but unobtrusive. The user retains full control of the primary desktop and can minimize or reposition the PiP window at will. This enables UFO$^2$ to perform long-running or repetitive workflows (e.g., data entry, batch file processing) without blocking user interaction or degrading responsiveness.

从用户视角看，PiP 窗口表现得像一个沙盒工作区: 自动化在后台执行，可见但不显眼。用户保持对主桌面的完全控制，并可随意最小化或重新定位 PiP 窗口。这样 UFO$^2$ 能在不阻塞用户交互或降低响应性的情况下执行长时间或重复性的流程 (例如数据录入、批量文件处理)。

## 4.2 Robust Input and State Isolation

## 4.2 强健的输入与状态隔离

To ensure robust separation between agent actions and user activities, UFO$^2$ leverages the RDP subsystem to maintain distinct input queues and device contexts across sessions. Mouse and keyboard events generated within the PiP desktop are fully scoped to that session and cannot interfere with the primary desktop. Similarly, GUI changes and focus transitions are restricted to the virtual environment.

为确保代理动作与用户活动之间的稳健隔离，UFO$^2$ 利用 RDP 子系统在会话间维持独立的输入队列与设备上下文。PiP 桌面内产生的鼠标和键盘事件完全限定在该会话内，无法干扰主桌面。同样，GUI 变化和焦点切换也被限制在虚拟环境中。

This level of input isolation is critical for preventing accidental interference-either by the user or the agent-and ensures that automation sequences remain stable, even during simultaneous foreground activity. The architecture also supports controlled error recovery: failures or unexpected UI states within the PiP session do not propagate to the primary desktop, preserving the integrity of the user's environment.

这种级别的输入隔离对于防止用户或代理的意外干扰至关重要, 并确保即使在前台活动同时进行时自动化序列仍能稳定运行。该架构还支持可控的错误恢复:PiP 会话内的失败或意外 UI 状态不会传播到主桌面，保全用户环境的完整性。

## 4.3 Secure Cross-Session Coordination

## 4.3 跨会话安全协调

Although visually and operationally distinct, the PiP session must remain logically connected to the host environment. To enable this, UFO$^2$ establishes a secure inter-process communication (IPC) channel between the PiP agent runtime and a host-side coordinator. We implement this using Windows

尽管在视觉和操作上是独立的，PiP 会话必须与主机环境保持逻辑连接。为此，UFO$^2$ 在 PiP 代理运行时与主机端协调器之间建立了安全的进程间通信 (IPC) 通道。我们使用 Windows 来实现此通信

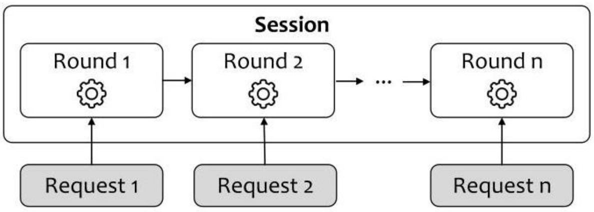


Figure 13. The interactive Session model in UFO$^2$ supports multi-round refinement.

图 13。UFO$^2$ 中的交互式会话模型支持多轮细化。

Named Pipes, authenticated and encrypted using per-session credentials [45].

命名管道，使用每会话凭据进行身份验证和加密 [45]。

This IPC layer supports two-way messaging:

该 IPC 层支持双向消息传递:

- From the host to the PiP: task assignment, progress polling, cancellation, and user clarifications.

- 从主机到 PiP: 任务分配、进度轮询、取消和用户澄清。

- From the PiP to the host: status updates, completion reports, and exception notifications.

- 从 PiP 到主机: 状态更新、完成报告和异常通知。

Users interact with the automation pipeline through a lightweight frontend panel on the host desktop, enabling real-time visibility and partial control without needing to directly access the PiP window. This transparent yet secure communication channel ensures trust and usability, particularly in long-running or partially supervised workflows.

用户通过主机桌面上的轻量前端面板与自动化流水线交互，能够实时查看并部分控制，而无需直接访问 PiP 窗口。这种透明但安全的通信通道确保了信任性和可用性，尤其适用于长时运行或部分监督的工作流。

## 4.4 System-Level Implications

## 4.4 系统层面的影响

The PiP interface represents more than a UX refinement-it is a system-level abstraction that reconciles concurrency, usability, and safety. It decouples automation execution from foreground interactivity, introduces a new isolation primitive for GUI-based agents, and simplifies failure recovery by sandboxing side effects. By exploiting existing RDP capabilities with minimal system overhead, the PiP interface offers a practical and backwards-compatible approach to scalable desktop automation.

PiP 界面不只是 UX 的改进——它是一个在系统层面调和并发性、可用性与安全性的抽象。它将自动化执行与前台交互解耦，引入了针对基于 GUI 的代理的新隔离原语，并通过将副作用沙箱化简化了故障恢复。通过以极低系统开销利用现有的 RDP 能力，PiP 界面为可扩展桌面自动化提供了一种实用且向后兼容的方法。

# 5 Implementation and Specialized Engineering Design

# 5 实现与专门工程设计

We implement UFO$^2$ as a full-stack desktop automation framework spanning over 30,000 lines of Python and C# code. Python serves as the core runtime environment for agent orchestration, control logic, and API integration, while C# supports GUI development, debugging interfaces, and Windows-specific operations such as the Picture-in-Picture desktop. To support retrieval-augmented reasoning, UFO$^2$ leverages Sentence Transformers [46] for embedding-based document and experience retrieval.

我们将 UFO$^2$ 实现为一个完整的桌面自动化框架, 包含 30,000 多行 Python 和 C# 代码。Python 作为代理编排、控制逻辑与 API 集成的核心运行时环境，而 C# 支持 GUI 开发、调试接口以及 Windows 特有操作 (例如画中画桌面)。为支持增强检索推理，UFO$^2$ 利用 Sentence Transformers [46] 进行基于嵌入的文档与经验检索。

Beyond its core functionality, UFO$^2$ incorporates multiple specialized engineering components that target critical systems goals: composability, interactivity, debuggability, and scalable deployment. We highlight several key mechanisms below.

除了核心功能外，UFO$^2$ 还包含多个面向关键系统目标的专门工程组件: 可组合性、交互性、可调试性与可扩展部署。下面我们强调若干关键机制。

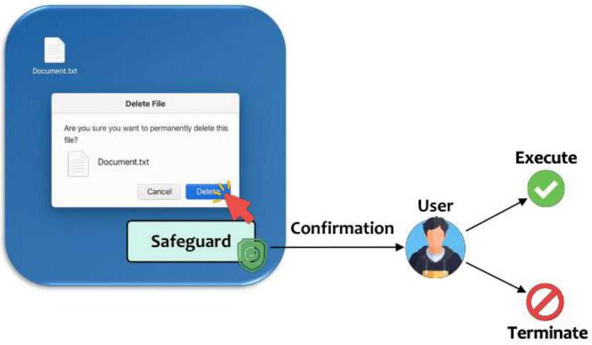


Figure 14. The safeguard mechanism employed in UFO$^2$ .

图 14. UFO$^2$ 中使用的保障机制。

## 5.1 Multi-Round Task Execution

## 5.1 多轮任务执行

Unlike stateless one-shot agents, UFO$^2$ adopts a session-based execution model to support iterative, interactive work-flows (Figure 13). Each Session maintains persistent contextual memory-including intermediate results, task progress, and application state-across multiple Rounds of execution. Users can refine prior instructions, launch follow-up tasks, or intervene when agents encounter ambiguous or unsafe operations.

不同于无状态的一次性代理，UFO$^2$ 采用基于会话的执行模型以支持迭代、交互式工作流 (见图 13)。每个会话在多个执行轮次中保持持久的上下文记忆——包括中间结果、任务进度与应用状态。用户可以细化先前指令、发起后续任务，或在代理遇到模糊或不安全操作时介入。

This multi-round interaction paradigm facilitates progressive convergence on complex tasks while preserving transparency and human oversight. It enables UFO$^2$ to support human-in-the-loop refinement strategies, bridging static LLM workflows with dynamic user guidance.

这种多轮交互范式有助于在复杂任务上逐步收敛，同时保持透明性和人工监督。它使 UFO$^2$ 能够支持人机协同的精炼策略，弥合静态 LLM 工作流与动态用户指导之间的差距。

## 5.2 Safeguard Mechanism

## 5.2 保障机制

While automation substantially boosts productivity, any CUA carries inherent risks of executing unsafe actions that may adversely affect user data or system stability [10, 47]. Examples include deleting critical files, terminating applications prematurely (resulting in unsaved data loss), or activating sensitive devices such as webcams without explicit consent. These actions pose severe risks, potentially causing irrecoverable damage or security breaches.

尽管自动化大幅提高了生产力，任何 CUA 都存在执行可能对用户数据或系统稳定性产生不利影响的不安全操作的内在风险 [10, 47]。示例包括删除关键文件、过早终止应用 (导致未保存数据丢失)，或在未获明确同意下激活诸如摄像头等敏感设备。这些行为带来严重风险，可能造成不可挽回的损害或安全漏洞。

To mitigate such risks, UFO$^2$ incorporates an explicit safeguard mechanism, designed to actively detect potentially dangerous actions, as shown in Figure 14. Specifically, whenever an APPAGENT identifies an action matching predefined risk criteria, it transitions into a dedicated PENDING state, pausing execution and actively prompting the user for confirmation. Only upon receiving explicit user consent does the agent proceed; otherwise, the action is aborted to prevent harm. The definition and scope of what constitutes a risky action are fully customizable through a straightforward prompt-based interface, enabling users and system administrators to precisely tailor safeguard behavior according to their organization's specific risk policies. This flexibility allows the safeguard system to be dynamically adapted as automation requirements evolve.

为缓解此类风险，UFO$^2$ 引入了明确的保护机制，用于主动检测潜在危险操作 (见图 14)。具体而言，每当 APPAGENT 识别到与预定义风险标准匹配的操作时，会转入专门的 PENDING 状态，暂停执行并主动向用户请求确认。只有在得到用户明确同意后，代理才会继续；否则该操作将被中止以防止造成危害。何为危险操作及其范围可通过简单的基于提示的界面完全自定义，使用户和系统管理员能够根据组织的具体风险策略精确调整保护行为。这种灵活性使保护系统能随着自动化需求的演进动态适配。

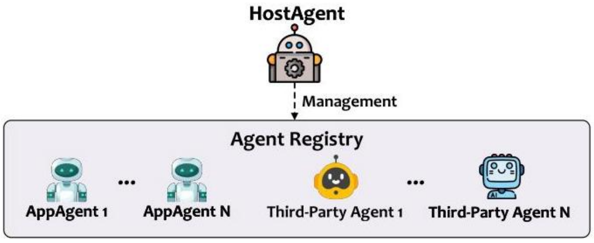


Figure 15. The agent registry supports seamless wrapping of third-party components into the APPAGENT framework.

图 15。代理注册表支持将第三方组件无缝包装进 APPAGENT 框架。

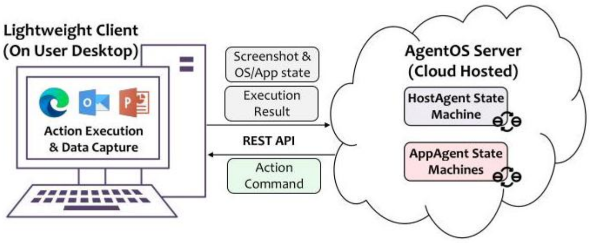


Figure 16. The client-server deployment model used in AgentOS-as-a-Service.

图 16。AgentOS-as-a-Service 使用的客户端-服务器部署模型。

Through this proactive safety-checking framework, UFO$^2$ significantly reduces the likelihood of executing harmful operations, thus enhancing overall system safety, user trust, and robustness in real-world deployments.

通过这一主动的安全检查框架，UFO$^2$ 大幅降低了执行有害操作的可能性，从而提升了系统总体安全性、用户信任度与在真实部署中的鲁棒性。

## 5.3 Everything-as-an-APPAGENT

## 5.3 一切皆为 APPAGENT

To support ecosystem extensibility, UFO$^2$ introduces an agent registry mechanism that encapsulates arbitrary third-party components as pluggable APPAGENTS (Figure 15). Through a simple registration API, external automation solutions-such as domain-specific copilots or proprietary tools-can be wrapped with lightweight compatibility shims that expose a unified interface to the HosTAGENT.

为支持生态可扩展性，UFO$^2$ 引入了代理注册表机制，可将任意第三方组件封装为可插拔的 APPAGENT(见图 15)。通过简单的注册 API，外部自动化解决方案——如领域专用协助工具或专有工具——可以用轻量兼容层进行包装，向 HosTAGENT 暴露统一接口。

This design enables HosTAGENT to treat native and external AppAGENTS interchangeably, dispatching subtasks based on capability and specialization. We find that even minimal wrappers (e.g., for OpenAI Operator [12]) lead to tangible performance gains, highlighting the system's modularity and its ability to incorporate diverse execution backends with minimal engineering overhead.

此设计使 HosTAGENT 能将本地与外部 AppAGENT 同等对待，基于能力与专长分派子任务。我们发现即使是最小的包装 (如用于 OpenAI Operator [12]) 也能带来明显的性能提升，凸显了系统的模块化及以最小工程代价整合多样执行后端的能力。

## 5.4 AgentOS-as-a-Service

## 5.4 AgentOS-as-a-Service

UFO$^2$ adopts a client-server architecture to support practical deployment at scale (Figure 16). A lightweight client resides on the user's machine and is responsible for GUI operations and application-side sensing. Meanwhile, a centralized server (running on-premises or in the cloud) hosts the HosTAGENT/APPAGENT logic, orchestrates workflows, and handles LLM queries.

UFO$^2$ 采用客户端-服务器架构以支持大规模的实际部署 (见图 16)。轻量客户端驻留在用户机器上,负责 GUI 操作与应用侧感知; 同时, 集中式服务器 (可驻留本地或云端) 承载 HosTAGENT/APPAGENT 逻辑、编排工作流并处理 LLM 查询。

This separation of control and execution offers several systems-level benefits:

这种控制与执行的分离带来若干系统层面的优势:

- Security: Sensitive orchestration and model execution are isolated from user devices.

- 安全性: 敏感的编排与模型执行与用户设备隔离。

- Maintainability: Server-side updates propagate without modifying the client.

- 可维护性: 服务器端更新可在不修改客户端的情况下传播。

- Scalability: The system can support multiple concurrent clients with centralized scheduling and load management.

- 可扩展性: 系统可通过集中调度与负载管理支持多个并发客户端。

The client-server boundary enforces a clean service abstraction, promoting modularity and simplifying rollout in enterprise environments.

客户端-服务器边界强制实施清晰的服务抽象，促进模块化并简化企业环境中的部署推广。

## 5.5 Comprehensive Logging and Debugging Infrastructure

## 5.5 全面日志与调试基础设施

Robust observability is essential for diagnosing failures and supporting ongoing system improvement. To this end, UFO$^2$ implements a comprehensive logging and debugging framework. Each session captures fine-grained traces of execution: prompts, LLM outputs, control metadata, UI state snapshots, and error events.

健全的可观测性对诊断故障和支持持续系统改进至关重要。为此，UFO$^2$ 实现了全面的日志与调试框架。每次会话都会捕获细粒度的执行追踪: 提示、LLM 输出、控制元数据、UI 状态快照和错误事件。

At the end of each session, UFO$^2$ compiles these artifacts into a structured, Markdown-formatted execution log. Developers can inspect action-by-action agent decisions, visualize interface state transitions, and replay behavior for debugging. The framework also supports prompt editing and selective replay for targeted hypothesis testing, significantly accelerating the debugging cycle. We show an example of these tools in Figure 17.

在每次会话结束时，UFO$^2$ 将这些产物汇编为结构化的 Markdown 格式执行日志。开发者可以逐步检查代理的决策、可视化界面状态转换并回放行为以便调试。该框架还支持提示编辑和选择性回放以进行有针对性的假设检验，从而显著加速调试周期。我们在图 17 中展示了这些工具的示例。

This observability layer functions as a lightweight provenance system for agent behavior, fostering transparency, accountability, and rapid iteration during deployment.

该可观测性层充当代理行为的轻量级溯源系统，促进透明性、问责制和部署期间的快速迭代。

## 5.6 Automated Task Evaluator

## 5.6 自动化任务评估器

To provide structured feedback and facilitate continuous improvement, UFO$^2$ includes an automated task evaluation engine based on LLM-as-a-judge [48]. As shown in Figure 18, the evaluator parses session traces-including actions, rationales, and screenshots-and applies CoT reasoning to decompose tasks into evaluation criteria.

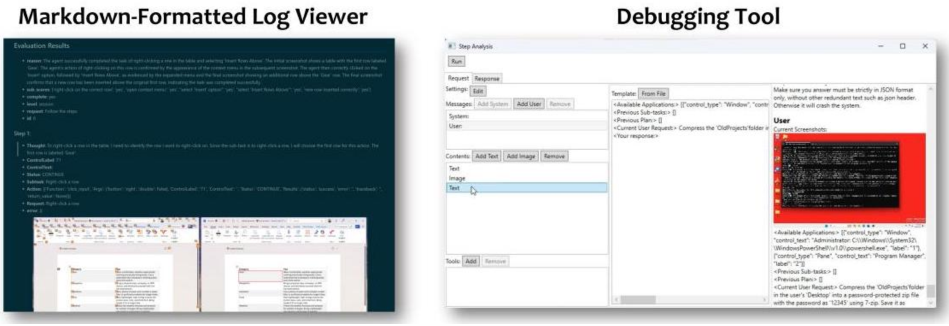
为了提供结构化反馈并促进持续改进，UFO$^2$ 包含基于 LLM 作为裁判的自动化任务评估引擎 [48]。如图 18 所示，评估器解析会话轨迹——包括操作、推理依据和截图——并应用链式思维 (CoT) 推理将任务分解为评估标准。

It assigns partial scores and synthesizes an overall result: success, partial, or failure. This structured outcome feeds into downstream dashboards and debugging tools. It also supports self-monitoring and offline analysis of failure cases, closing the loop between execution, diagnosis, and improvement.

它分配部分分数并综合生成总体结果: 成功、部分成功或失败。该结构化结果会被送入下游仪表板和调试工具。它还支持自我监控和故障案例的离线分析，闭合执行、诊断与改进之间的循环。

Summary. These engineering components demonstrate UFO2's commitment to operational robustness and extensibility. From session-based execution and pluggable agents to service-oriented deployment and observability infrastructure, each module reflects a design focused on bridging conceptual LLM agent architectures with the systems realities of deployment at scale.

总结。这些工程组件展示了 UFO2 对运行健壮性和可扩展性的承诺。从基于会话的执行和可插拔代理到面向服务的部署与可观测性基础设施，每个模块都体现了将概念性 LLM 代理架构与大规模部署的系统现实相结合的设计取向。



Evaluation Results

- reason: The agent successfully completed the task of right-clicking a row in the table and selecting 'Insert Rows Above'. [illegible]
- sub_scores: [illegible]
- complete: yes
- level: session
- request: Follow the steps
- id: 0

Step 1:

- Thought: To right-click a row in the table, I need to identify the row I want to right-click on. [illegible]
- ControlLabel: 71
- ControlText:
- Status: CONTINUE
- Subtask: Right-click a row
- Action: [illegible]
- Request: Right-click a row
- error: []

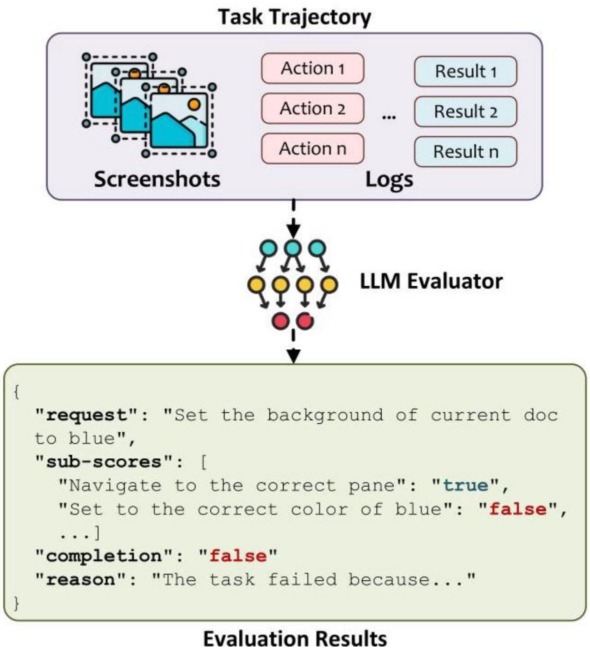
```
<Available Applications:> [{"control_type": "Window", "contr
<Previous Sub-tasks:> []
<Previous Plan:> []
<Current User Request:> Compress the 'OldProjects'folder i
<Your response:>
```

Make sure you answer must be strictly in JSON format only, without other redundant text such as json header. Otherwise it will crash the system.

```
<Available Applications:> [{"control_type": "Window", "control_text": "Administrator: C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe", "label": "1"}, {"control_type": "Pane", "control_text": "Program Manager", "label": "2"}]
<Previous Sub-tasks:> []
<Previous Plan:> []
<Current User Request:> Compress the 'OldProjects'folder in the user's 'Desktop' into a password-protected zip file with the password as '12345' using 7-zip. Save it as
```

Figure 17. An illustration of the markdown-formatted log viewer and debugging tool in UFO $^2$ .

图 17。UFO $^2$ 中 Markdown 格式日志查看器和调试工具的示意图。



```
{
  "request": "Set the background of current doc
  to blue",
  "sub-scores": [
    "Navigate to the correct pane": "true",
    "Set to the correct color of blue": "false",
    ...]
  "completion": "false"
  "reason": "The task failed because..."
}
```

Figure 18. The LLM-based task evaluator applies CoT reasoning to structured session logs.

图 18。基于 LLM 的任务评估器对结构化会话日志应用链式思维推理。

# 6 Evaluation

# 6 评估

We tested UFO$^2$ rigorously across more than 20 Windows applications, including office suites, file explorers, and custom enterprise tools to assess performance, efficiency, and robustness. Our experiments show:

我们在 20 多款 Windows 应用上对 UFO$^2$ 进行了严格测试，包括办公套件、文件管理器和定制企业工具，以评估性能、效率和鲁棒性。我们的实验表明:

1. UFO$^2$ achieves a 10% higher task completion rate, a 50% relative improvement-over the best-performing current CUA Operator, enabled by deeper OS-level integration.

1. UFO$^2$ 实现了 10% 更高的任务完成率，相较表现最好的现有 CUA Operator 提供了 50% 的相对提升，这得益于更深层级的操作系统集成。

2. The hybrid UIA-vision approach identifies custom or nonstandard GUI elements missed by UIA alone, boosting success in interfaces with proprietary widgets.

2. 混合 UIA-视觉方法识别了单靠 UIA 无法检测的自定义或非标准 GUI 元素，在包含专有控件的界面中提升了成功率。

3. Allowing AppAGENTS to invoke native APIs or GUI interactions to improve completion rate by over 8%, cuts latency and reduces the fragility seen in purely click-based workflows.

3. 允许 AppAGENTS 调用本地 API 或进行 GUI 交互将完成率提高了超过 8%，并降低了延迟及纯点击式工作流中的脆弱性。

4. Leveraging external documents and execution logs increases UFO2's ability to handle unfamiliar features without retraining.

4. 利用外部文档和执行日志提高了 UFO2 在无需再训练的情况下处理不熟悉功能的能力。

5. Speculative multi-action execution consolidates multiple steps into a single LLM call, lowering inference cost by up to 51.5% without compromising reliability.

5. 推测性多动作执行将多个步骤合并为一次 LLM 调用，在不损害可靠性的前提下将推理成本降低了最多 51.5%。

6. By enabling Everything-as-an-AppAGENT (e.g., Operator), UFO$^2$ both boosts overall performance and uncovers the full potential of each individual agent.

6. 通过将万物作为 AppAGENT(例如 Operator)，UFO$^2$ 不仅提升了整体性能，还发掘了每个单独代理的全部潜力。

Overall, these results confirm that UFO2's deeper integration with Windows and application-level APIs yields both higher performance and reduced overhead, making a compelling case for an OS-native approach to desktop automation.

总体而言，这些结果证实了 UFO2 与 Windows 及应用级 API 更深的整合带来了更高的性能和更低的开销，为面向操作系统的桌面自动化方法提供了有力论证。

## 6.1 Experimental Setup

## 6.1 实验设置

Deployment Environment. The benchmark environments are hosted on isolated VMs with 8 AMD Ryzen 7 CPU cores and 8 GB of memory, matching typical deployment conditions. All GPT-family models (GPT-4V, GPT-40, o1, and Operator) are accessed via Azure OpenAI services, while the OmniParser-v2 vision model operates on a separate virtual machine provisioned with an NVIDIA A100 80GB GPU to support efficient and high-throughput visual grounding.

部署环境。基准测试环境运行在隔离虚拟机上，配备 8 核 AMD Ryzen 7 CPU 和 8 GB 内存，符合典型部署条件。所有 GPT 系列模型 (GPT-4V、GPT-40、o1 和 Operator) 通过 Azure OpenAI 服务访问，而 OmniParser-v2 视觉模型在另一台配备 NVIDIA A100 80GB GPU 的虚拟机上运行，以支持高效且高吞吐的视觉定位。

Benchmarks. We evaluate UFO$^2$ using two established Windows-centric automation benchmarks:

基准。我们使用两项成熟的以 Windows 为中心的自动化基准来评估 UFO$^2$：

- Windows Agent Arena (WAA) [49]: Consists of 154 live automation tasks across 15 commonly used Windows applications, including office productivity tools, web browsers, system utilities, development environments, and multimedia apps. Each task includes a custom verification script for automated correctness checking.

- Windows Agent Arena (WAA) [49]: 包含 15 个常用 Windows 应用中 154 个实时自动化任务，涵盖办公生产力工具、网页浏览器、系统实用程序、开发环境和多媒体应用。每个任务都包括用于自动化正确性检查的自定义验证脚本。

- OSWorld-W [50]: A targeted subset of the OSWorld benchmark specifically tailored for Windows, comprising 49 live tasks across office applications, browser interactions, and file-system operations. Tasks are similarly equipped with handcrafted verification scripts for reliable outcome validation.

- OSWorld-W [50]: 针对 Windows 的 OSWorld 基准的定向子集，由 49 个涉及办公应用、浏览器交互和文件系统操作的实时任务组成。任务同样配备手工制作的验证脚本以可靠地验证结果。

Each task runs independently, and verification strictly follows the original scripts provided by each benchmark. [1]

每个任务独立运行，验证严格遵循各基准提供的原始脚本。[1]

Baselines. We compare UFO $^2$ with five representative state-of-the-art CUAs, each leveraging GPT-40 as the inference engine:

基线。我们将 UFO $^2$ 与五个代表性的最先进 CUA 进行比较，均以 GPT-40 作为推理引擎:

- UFO [10]: A pioneering multiagent, GUI-focused automation system designed explicitly for Windows, integrating UIA and visual perception.

- UFO [10]: 一个开创性的多代理、聚焦 GUI 的自动化系统，专为 Windows 设计，整合了 UIA 与视觉感知。

- NAVI [49]: A single-agent baseline from WAA, utilizing screenshots and accessibility data for GUI understanding.

- NAVI [49]:WAA 中的单代理基线，利用屏幕截图和无障碍数据进行 GUI 理解。

- OmniAgent [18]: Employs OmniParser for visual grounding combined with GPT-based action planning.

- OmniAgent [18]: 结合 OmniParser 用于视觉定位与基于 GPT 的动作规划。

- Agent S [51]: Features a multiagent architecture with experience-driven hierarchical planning, optimized for complex, multi-step tasks.

- Agent S [51]: 采用多代理架构并具备经验驱动的分层规划，针对复杂多步骤任务进行优化。

- Operator [12]: A recent, high-performance CUA from OpenAI, simulating human-like mouse and keyboard interactions via screenshots.

- Operator [12]:OpenAI 的近期高性能 CUA，通过屏幕截图模拟类人鼠标和键盘交互。

These baselines were selected for their representativeness of diverse architectural and design paradigms (e.g., single-agent vs. multiagent, GUI-only vs. hybrid approaches). To ensure fairness, each agent is restricted to a maximum of 30 execution steps per task, reflecting practical user expectations and preventing excessively long task executions. Additionally, we evaluate a base version of UFO$^2$ (termed UFO$^2$ -base) using only UIA detection, GUI-based interactions, and without dynamic knowledge integration, alongside the full implementation of UFO$^2$ featuring hybrid control detection, combined GUI-API interactions, and continuous knowledge augmentation. API integrations were selectively implemented for three office applications within OSWorld-W as illustrative examples; no APIs were introduced for the WAA tasks. Further implementation details are available in Section 6.4.

选择这些基线是为了代表不同的架构与设计范式 (例如单代理 vs. 多代理、仅 GUI vs. 混合方法)。为确保公平，每个代理在每个任务上最多允许 30 步执行，反映实际用户期望并防止过长的任务执行。此外，我们评估了 UFO$^2$ 的基础版本 (称为 UFO$^2$ -base)，仅使用 UIA 检测、基于 GUI 的交互且不包含动态知识整合，以及包含混合控件检测、结合 GUI-API 交互与持续知识增强的完整 UFO$^2$ 实现。作为示例性说明，API 集成在 OSWorld-W 的三个办公应用中有选择性地实现；WAA 任务未引入任何 API。更多实现细节见 6.4 节。

Evaluation Metrics. We utilize two primary metrics for performance evaluation:

评估指标。我们使用两个主要指标来评估性能:

- Success Rate (SR): Defined as the percentage of tasks successfully completed, validated via the benchmarks' own verification scripts.

- 成功率 (SR): 定义为成功完成的任务百分比，通过基准自带的验证脚本进行验证。

- Average Completion Steps (ACS): Measures the average number of LLM-involved action inference steps required per task. Fewer steps correspond to higher efficiency, directly correlating with lower inference latency and reduced computational overhead.

- 平均完成步骤数 (ACS): 衡量每个任务中涉及 LLM 的动作推断平均步骤数。步骤越少表示效率越高，直接对应更低的推理延迟和更少的计算开销。

Table 1. Comparison of success rates (SR) across agents on WAA and OSWorld-W benchmarks.

表 1. 各智能体在 WAA 与 OSWorld-W 基准上的成功率 (SR) 比较。

| Agent | Model | WAA | OSWorld-W |
|---|---|---|---|
| UFO | GPT-40 | 19.5% | 12.2% |
| NAVI | GPT-40 | 13.3% | 10.2% |
| OmniAgent | GPT-40 | 19.5% | 8.2% |
| Agent S | GPT-40 | 18.2% | 12.2% |
| Operator | computer-use | 20.8% | 14.3% |
| UFO$^2$ -base | GPT-40 | 23.4% | 16.3% |
| UFO$^2$ -base | o1 | 25.3% | 16.3% |
| UFO2 | GPT-40 | 27.9% | 28.6% |
| UFO2 | o1 | 30.5% | 32.7% |

| 特工 | 模型 | WAA | OSWorld-W |
|---|---|---|---|
| UFO | GPT-40 | 19.5% | 12.2% |
| NAVI | GPT-40 | 13.3% | 10.2% |
| 全能特工 | GPT-40 | 19.5% | 8.2% |
| 特工S | GPT-40 | 18.2% | 12.2% |
| 操作员 | 计算机使用 | 20.8% | 14.3% |
| UFO$^2$ -base | GPT-40 | 23.4% | 16.3% |
| UFO$^2$ -base | o1 | 25.3% | 16.3% |
| UFO2 | GPT-40 | 27.9% | 28.6% |
| UFO2 | o1 | 30.5% | 32.7% |

These metrics effectively reflect both functional effectiveness and practical efficiency, providing clear indicators of real-world automation performance.

这些指标有效反映了功能有效性与实际效率，为现实世界的自动化表现提供了清晰指标。

## 6.2 Success Rate Comparison

## 6.2 成功率比较

Table 1 summarizes the success rates (SR) of all evaluated agents across the WAA and OSWorld-W benchmarks, as verified by each benchmark's automated validation scripts. Notably, even the basic configuration (UFO$^2$ -base )– which relies solely on standard UI Automation and GUI-driven actions-consistently surpasses prior state-of-the-art CUAs. Specifically, with GPT-40, UFO$^2$ -base achieves an SR of 23.4% on WAA, outperforming the best existing baseline, Operator (20.8%), by 2.6%. This margin widens significantly when employing the stronger o1 LLM, lifting UFO$^2$ -base's performance to 25.3%.

表1汇总了在WAA和OSWorld-W基准上经各自自动验证脚本核验的所有被评估代理的成功率(SR)。值得注意的是，即便只依赖标准UI自动化和GUI驱动操作的基础配置(UFO$^2$ -base )– 也持续超越以往最先进的CUA。具体来说，使用GPT-40时，UFO$^2$ -base在WAA上取得23.4%的SR，优于现有最佳基线Operator(20.8%)2.6个百分点。采用更强的o1 LLM时，这一差距显著扩大，将UFO$^2$ -base的表现提升到25.3%。

Moreover, the complete version of UFO$^2$ , incorporating hybrid GUI-API action execution, advanced visual grounding, and continuous knowledge integration, further amplifies these performance gains. With GPT-40, UFO$^2$ achieves a 27.9% SR on WAA, exceeding Operator by a substantial 7.1%. The performance gap becomes even more pronounced on OSWorld-W, where UFO$^2$ achieves a 28.6% SR compared to Operator's 14.3%, effectively doubling its success rate. Utilizing the stronger o1 model further improves UFO2's performance to 30.5% (WAA) and 32.7% (OSWorld-W), solidifying its leading position.

此外，完整版本的 UFO$^2$ 融合了混合 GUI-API 操作执行、先进的视觉定位和持续知识整合，进一步放大了这些性能提升。使用 GPT-40 时，UFO$^2$ 在 WAA 上达到 27.9% 的 SR，超出 Operator 7.1 个百分点。在 OSWorld-W 上，UFO$^2$ 达到 28.6% 的 SR，而 Operator 为 14.3%，成功率几乎翻倍。采用更强的 o1 模型后，UFO2 在 WAA 和 OSWorld-W 的表现分别提升至 30.5% 和 32.7%，巩固了其领先地位。

These significant performance improvements clearly underscore the advantages of UFO$^{2,s}$ s deep integration with OS-level mechanisms and its unified system architecture. While prior CUAs primarily emphasize model-level optimization or singular reliance on visual interfaces, our results demonstrate that robust, system-level orchestration-combining structured OS APIs, specialized application knowledge, and hybrid GUI-API interaction-is instrumental in achieving higher task reliability and broader automation coverage. Crucially, even a general-purpose, less-specialized model like GPT-40 can surpass highly specialized CUAs (such as Operator) when integrated within the comprehensive UFO$^2$ framework. This insight reinforces the value of architectural design and OS integration as key drivers of practical, deployable desktop automation solutions.

这些显著的性能提升清楚地凸显了 UFO$^{2,s}$ 与操作系统级机制的深度集成及其统一系统架构的优势。以往的 CUA 主要侧重模型层优化或单一依赖视觉界面，而我们的结果表明，结合结构化操作系统 API、专门应用知识与混合 GUI-API 交互的健壮系统级协调，对于实现更高任务可靠性和更广泛的自动化覆盖至关重要。关键是，即便像 GPT-40 这样通用且不太专门化的模型，在置入完整的 UFO$^2$ 框架后，也能超越高度专门化的 CUA(如 Operator)。这一洞见强化了架构设计与操作系统集成作为可部署桌面自动化解决方案关键驱动因素的价值。

[1] Reported baseline scores in OSWorld differ slightly from prior results focused on Ubuntu due to corrections in verification scripts and alignment with Windows-specific tasks (OSWorld-W).

[1] 在 OSWorld 报告的基线分数与此前针对 Ubuntu 的结果略有不同，原因是验证脚本的修正及与特定于 Windows 的任务 (OSWorld-W) 的对齐。

Performance Breakdown. Table 2 presents a detailed breakdown of success rates (SR) by application type on the WAA and OSWorld-W benchmarks, enabling deeper understanding of where UFO$^2$ achieves particularly strong results and identifying areas for further system-level improvements. Across multiple categories, UFO$^2$ consistently demonstrates superior performance compared to baseline CUAs, particularly in application scenarios demanding deeper OS integration or sophisticated multi-step task execution.

性能细分。表 2 按应用类型在 WAA 和 OSWorld-W 基准上详细列出成功率 (SR) 的分解，便于深入理解 UFO$^2$ 在何处取得特别强劲的结果并识别需要进一步系统级改进的领域。在多个类别中，UFO$^2$ 相较基线 CUA 一直表现更优，尤其是在需要更深 OS 集成或复杂多步任务执行的应用场景中。

Notably, UFO$^2$ excels in tasks involving web browsers and coding environments. For instance, the strongest configuration (UFO $^2$ with o1) attains an impressive 40.0% SR for web browser tasks-markedly outperforming the next-best baseline (OmniAgent) by over 12%. Similarly, in coding-related workflows, UFO $^2$ (GPT-40) achieves the highest SR of 58.3%, significantly exceeding all competing CUAs. These results underscore the effectiveness of UFO2's hybrid GUI-API approach and continuous knowledge integration, which enable more precise action inference, reduce brittleness due to GUI changes, and substantially elevate reliability in multi-step workflows.

值得注意的是，UFO$^2$ 在涉及网页浏览器和编码环境的任务中表现尤为突出。例如，最强配置 (UFO $^2$ 使用 o1) 在网页浏览器任务上取得了令人印象深刻的 40.0% SR，明显领先于次优基线 OmniAgent 超过 12%。同样，在与编码相关的工作流中，UFO $^2$ (GPT-40) 达到最高 58.3% 的 SR，显著超越所有竞争的 CUA。这些结果凸显了 UFO2 的混合 GUI-API 方法和持续知识整合的有效性，使得动作推断更精确、因 GUI 变化导致的脆弱性降低，并在多步工作流中大幅提高可靠性。

The breakdown further reveals a clear correlation between application complexity, popularity, and system-level support. Tasks involving LibreOffice (in the Office category of WAA) uniformly yield lower SRs across all evaluated CUAs, largely due to inadequate adherence to accessibility standards and incomplete UIA support. Conversely, OSWorld-W tasks predominantly utilize Microsoft 365 Office applications, which offer richer OS-native APIs and structured accessibility data, resulting in improved SRs (up to 51.9% for UFO $^2$ -o1). This discrepancy highlights the critical role that robust OS-level integration and API availability play in achieving high-quality desktop automation.

分解结果还显示应用复杂性、流行度与系统级支持之间存在明显相关性。涉及 LibreOffice(WAA 的 Office 类别) 的任务在所有评估的 CUA 中普遍获得较低的 SR，主要因为对无障碍标准的支持不足和不完整的 UIA 支持。相反，OSWorld-W 任务主要使用 Microsoft 365 Office 应用，这些应用提供更丰富的操作系统原生 API 和结构化的可访问性数据，导致 SR 提升 (UFO $^2$ -o1 可达 51.9%)。这种差异凸显了稳健的操作系统级集成和 API 可用性在实现高质量桌面自动化中的关键作用。

Cross-application tasks, especially prominent in OSWorld-W, present an even greater challenge. Such tasks inherently require sophisticated task decomposition and robust interagent coordination, pushing CUAs-and even human users-to their limits. Here, UFO2's multiagent architecture, led by the centralized HosTAGENT and specialized APPAGENTs, demonstrates notable promise, outperforming other baselines with a 9.1% SR. Although performance remains relatively modest, it clearly illustrates the strength of systematic multiagent collaboration and centralized orchestration in addressing complex scenarios that cross traditional application boundaries.

跨应用任务，在 OSWorld-W 中尤为突出，挑战更大。此类任务本质上要求复杂的任务拆解和稳健的代理间协调，逼近 CUAs——甚至人类用户——的极限。在此情形下，UFO2 的多智能体架构，由集中式 HosTAGENT 和专门的 APPAGENTs 领导，展现出显著潜力，以 9.1% 的 SR 优于其他基线。尽管性能仍然相对有限，但它清楚地说明了系统化多智能体协作与集中协调在处理跨传统应用边界的复杂场景中的优势。

Overall, these detailed breakdown results validate the system-level design principles of UFO$^2$ , particularly its emphasis on deep OS and application-specific integration, mul-tiagent coordination, and flexible action orchestration. While there remains significant potential for further enhancements in niche or less-supported application domains (e.g., custom or legacy software with limited API availability), UFO2's current architecture already provides a substantial, measurable improvement in practical, real-world desktop automation tasks.

总体而言，这些详细的拆分结果验证了 UFO$^2$ 的系统级设计原则，特别是其对深入操作系统与应用特定集成、多智能体协调和灵活动作编排的重视。尽管在某些利基或支持较少的应用领域 (例如具有有限 API 的定制或遗留软件) 仍有显著提升空间，UFO2 的当前架构已在现实桌面自动化任务中带来可观且可量化的改进。

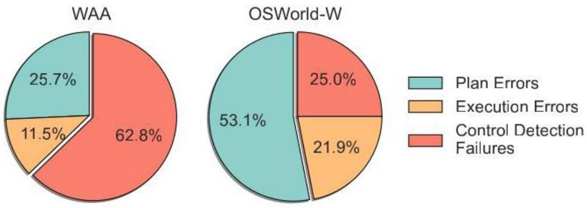


Figure 19. Error analysis of UFO$^2$ -base (GPT-40) on the two benchmarks.

图 19. UFO$^2$ -base(GPT-40) 在两项基准上的错误分析。

Error Analysis. To systematically understand the limitations of UFO$^2$ and identify opportunities for further improvement, we conducted a detailed manual review of all failure cases for UFO $^2$ -base (GPT-4o) on both benchmarks. Following a classification framework similar to Agashe et al., [51], each failure was categorized into one of three distinct system-level categories:

错误分析。为系统性地理解 UFO$^2$ 的局限并识别改进机会，我们对 UFO $^2$ -base(GPT-4o) 在两项基准上的所有失败案例进行了详尽人工复审。参照类似 Agashe 等人的分类框架 [51]，每个失败被归入以下三类系统级别类别之一:

- Plan Errors: Failures arising from inadequate high-level task understanding, typically reflected by incomplete or incorrect action plans. These errors indicate gaps in the agent's task comprehension or insufficient grounding in application-specific workflows.

- 计划错误: 源于高层任务理解不足的失败，通常表现为不完整或错误的动作计划。这类错误表明代理在任务理解上存在缺口，或缺乏对应用特定工作流的充分落地。

- Execution Errors: Cases where the high-level plan is reasonable but the execution is flawed (e.g., selecting an incorrect control, performing unintended actions). Execution errors often stem from inaccurate visual reasoning, incorrect associations between GUI elements and actions, or erroneous inference by the LLM.

- 执行错误: 高层计划合理但执行存在缺陷的情况 (例如选错控件、执行了非预期操作)。执行错误常由视觉推理不准确、GUI 元素与动作之间的错误关联或 LLM 的错误推断造成。

- Control Detection Failures: Instances where the agent fails to detect or identify critical GUI controls required to complete a task, usually due to non-standard or custom-rendered UI elements that are not fully accessible via standard OS APIs.

- 控件检测失败: 代理未能检测或识别完成任务所需的关键 GUI 控件的情况，通常由于非标准或自定义渲染的 UI 元素无法通过标准操作系统 API 完全访问所致。

Figure 19 summarizes our findings for UFO$^2$ -base. On the WAA benchmark, more than 62% of failures were attributed to Control Detection Failures, highlighting significant gaps in standard UIA API coverage-especially for third-party applications (e.g., LibreOffice) that do not strictly adhere to accessibility standards. Conversely, the OSWorld-W benchmark exhibited a higher incidence of Plan Errors, underscoring that tasks in this set frequently involve more complex workflows, necessitating deeper domain knowledge or advanced contextual reasoning capabilities beyond simple visual recognition.

图 19 汇总了我们对 UFO$^2$ -base 的发现。在 WAA 基准上，超过 62% 的失败归因于控件检测失败，凸显了标准 UIA API 覆盖的显著缺口——尤其是对不严格遵循无障碍标准的第三方应用 (例如 LibreOffice)。相反，OSWorld-W 基准中计划错误的发生率更高，表明该集合中的任务往往涉及更复杂的工作流，需要更深的领域知识或超越简单视觉识别的高级上下文推理能力。

These observations provide concrete evidence of specific system-level shortcomings, directly motivating the enhancements incorporated into the complete version of UFO$^2$ . The high frequency of Control Detection Failures validates our choice of adopting a hybrid GUI detection pipeline that supplements standard UIA data with advanced visual grounding techniques. Similarly, the prevalence of Plan Errors underscores the critical role of integrating richer external documentation, domain-specific knowledge bases, and application-level APIs to strengthen task understanding and action inference. In the subsequent sections, we explicitly demonstrate how these incremental system-level improvements progressively mitigate each identified category of errors, thereby substantially boosting UFO2's overall task completion effectiveness.

这些观察为具体的系统级短板提供了有力证据，直接促成了完整版本 UFO$^2$ 中所采纳的改进。控件检测失败高频率验证了我们采用混合 GUI 检测流程的选择，该流程以高级视觉定位技术补充标准 UIA 数据。同样，计划错误的普遍存在强调了整合更丰富的外部文档、领域特定知识库和应用级 API，以增强任务理解与动作推断的关键作用。在后续章节中，我们将明确展示这些渐进的系统级改进如何逐步缓解每一类识别出的错误，从而显著提升 UFO2 的整体任务完成效能。

Table 2. SR breakdown by application type on WAA and OSWorld-W.

表 2. WAA 与 OSWorld-W 上按应用类型划分的 SR 明细。

| Agent | Model | WAA | | | | | | OSWorld-W | |
|---|---|---|---|---|---|---|---|---|---|
| | | Office | Web Browser | Windows System | Coding | Media & Video | Windows Utils | Office | Cross-App |
| UFO | GPT-40 | 0.0% | 23.3% | 33.3% | 29.2% | 33.3% | 8.3% | 18.5% | 4.5% |
| NAVI | GPT-40 | 0.0% | 20.0% | 29.2% | 9.1% | 25.3% | 0.0% | 18.5% | 0.0% |
| OmniAgent | GPT-40 | 0.0% | 27.3% | 33.3% | 27.3% | 30.3% | 8.3% | 14.8% | 0.0% |
| Agent S | GPT-40 | 0.0% | 13.3% | 45.8% | 29.2% | 19.1% | 22.2% | 22.2% | 0.0% |
| Operator | computer-use | 7% | 26.7% | 29.2% | 29.2% | 28.6% | 8.3% | 22.2% | 4.5% |
| UFO$^2$ -base | GPT-40 | 2.3% | 36.7% | 29.2% | 41.7% | 33.3% | 0.0% | 22.2% | 9.1% |
| UFO$^2$ -base | o1 | 2.3% | 30.0% | 37.5% | 50.0% | 33.3% | 8.3% | 22.2% | 9.1% |
| UFO$^2$ | GPT-40 | 4.7% | 30.0% | 41.7% | 58.3% | 33.3% | 8.3% | 44.4% | 9.1% |
| UFO2 | o1 | 4.7% | 40.0% | 45.8% | 50.0% | 38.1% | 16.7% | 51.9% | 9.1% |

| 代理 | 模型 | WAA | | | | | | OSWorld-W | |
|---|---|---|---|---|---|---|---|---|---|
| | | 办公 | 网页浏览器 | Windows 系统 | 编程 | 媒体与视频 | Windows 工具 | 办公 | 跨应用 |
| UFO | GPT-40 | 0.0% | 23.3% | 33.3% | 29.2% | 33.3% | 8.3% | 18.5% | 4.5% |
| NAVI | GPT-40 | 0.0% | 20.0% | 29.2% | 9.1% | 25.3% | 0.0% | 18.5% | 0.0% |
| 全能代理 | GPT-40 | 0.0% | 27.3% | 33.3% | 27.3% | 30.3% | 8.3% | 14.8% | 0.0% |
| 代理 S | GPT-40 | 0.0% | 13.3% | 45.8% | 29.2% | 19.1% | 22.2% | 22.2% | 0.0% |
| 操作者 | 计算机使用 | 7% | 26.7% | 29.2% | 29.2% | 28.6% | 8.3% | 22.2% | 4.5% |
| UFO$^2$ -base | GPT-40 | 2.3% | 36.7% | 29.2% | 41.7% | 33.3% | 0.0% | 22.2% | 9.1% |
| UFO$^2$ -base | o1 | 2.3% | 30.0% | 37.5% | 50.0% | 33.3% | 8.3% | 22.2% | 9.1% |
| UFO$^2$ | GPT-40 | 4.7% | 30.0% | 41.7% | 58.3% | 33.3% | 8.3% | 44.4% | 9.1% |
| UFO2 | o1 | 4.7% | 40.0% | 45.8% | 50.0% | 38.1% | 16.7% | 51.9% | 9.1% |

Table 3. Comparison of SR and CRR across control detection mechanisms.

表 3. 不同控制检测机制下 SR 与 CRR 的比较。

| Control Detector | Model | WAA | | OSWorld-W | |
|---|---|---|---|---|---|
| | | SR | CRR | SR | CRR |
| UIA | GPT-40 | 23.4% | - | 22.4% | - |
| OmniParser-v2 | GPT-40 | 26.6% | 7.0% | 14.3% | 0% |
| Hybrid | GPT-40 | 26.6% | 9.9% | 22.4% | 12.5% |
| UIA | o1 | 25.3% | - | 24.5% | - |
| OmniParser-v2 | o1 | 20.8% | 7.0% | 14.3% | 0% |
| Hybrid | o1 | 27.9% | 9.9% | 28.6% | 25.0% |

| 控制检测器 | 型号 | WAA | | OSWorld-W | |
|---|---|---|---|---|---|
| | | SR | CRR | SR | CRR |
| UIA | GPT-40 | 23.4% | - | 22.4% | - |
| OmniParser-v2 | GPT-40 | 26.6% | 7.0% | 14.3% | 0% |
| 混合 | GPT-40 | 26.6% | 9.9% | 22.4% | 12.5% |
| UIA | o1 | 25.3% | - | 24.5% | - |
| OmniParser-v2 | o1 | 20.8% | 7.0% | 14.3% | 0% |
| 混合 | o1 | 27.9% | 9.9% | 28.6% | 25.0% |

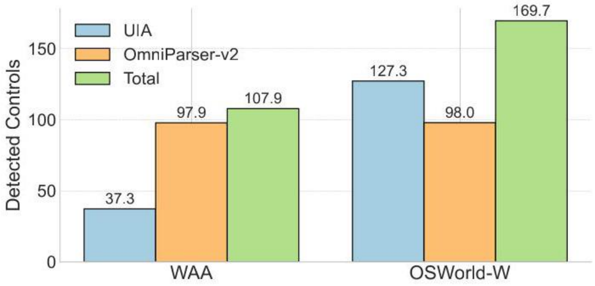


Figure 20. The number of detected controls of different approaches.

图 20。不同方法检测到的控件数量。

## 6.3 Evaluation on Hybrid Control Detection

## 6.3 对混合控件检测的评估

As shown in Figure 19, a considerable fraction of failures arise from Control Detection Failures, where non-standard UI elements do not comply with UIA guidelines. To quantify the effectiveness of different detection strategies, we compare UIA-only, OmniParser-v2-only, and our hybrid method (Section 3.4). We introduce a Control Recovery Ratio (CRR) to measure how many UIA-only failures are ”recovered” (i.e., become successful completions) under OmniParser or the hybrid approach.

如图 19 所示，相当一部分失败来自控件检测失败，即非标准 UI 元素不符合 UIA 指南。为量化不同检测策略的有效性，我们比较了仅 UIA、仅 OmniParser-v2 与我们的混合方法 (第 3.4 节)。我们引入控件恢复比 (CRR) 来衡量在 OmniParser 或混合方法下，有多少 UIA-only 的失败被“恢复”(即变为成功完成)。

Table 3 presents the results on both benchmarks, across multiple model configurations. The hybrid method consistently outperforms either UIA-only or OmniParser-only settings, raising the overall success rate and converting up to 9.86% of previously irrecoverable cases into completions. This gain highlights the complementary strengths of the two detection pipelines, as the hybrid approach bridges coverage gaps in UIA while avoiding OmniParser’s limitations in more standardized GUIs.

表 3 给出了在两个基准和多种模型配置上的结果。混合方法始终优于仅 UIA 或仅 OmniParser 的设置，提高了整体成功率，并将最多 9.86% 的先前不可恢复的案例转化为完成。这一增益凸显了两条检测管线的互补优势: 混合方法弥补了 UIA 的覆盖空白，同时避免了 OmniParser 在更标准化 GUI 中的局限性。

In Figure 20, we report the average number of controls detected from each source (UIA, OmniParser-v2, and the merged set) under the hybrid approach. Owing to differences in application coverage, the total number of detected controls is generally higher in OSWorld-W than in WAA. Notably, both UIA and OmniParser-v2 identify substantial subsets of controls, and after merging, 27.9% and 56.7% of OmniParser-v2 detections are discarded due to overlap with UIA. These observations indicate that OmniParser-v2 provides a valuable complement to UIA by recovering nonstandard or custom elements. At the same time, the merging step removes redundancies and prevents double-counting, ultimately reducing control detection failures in the hybrid scheme.

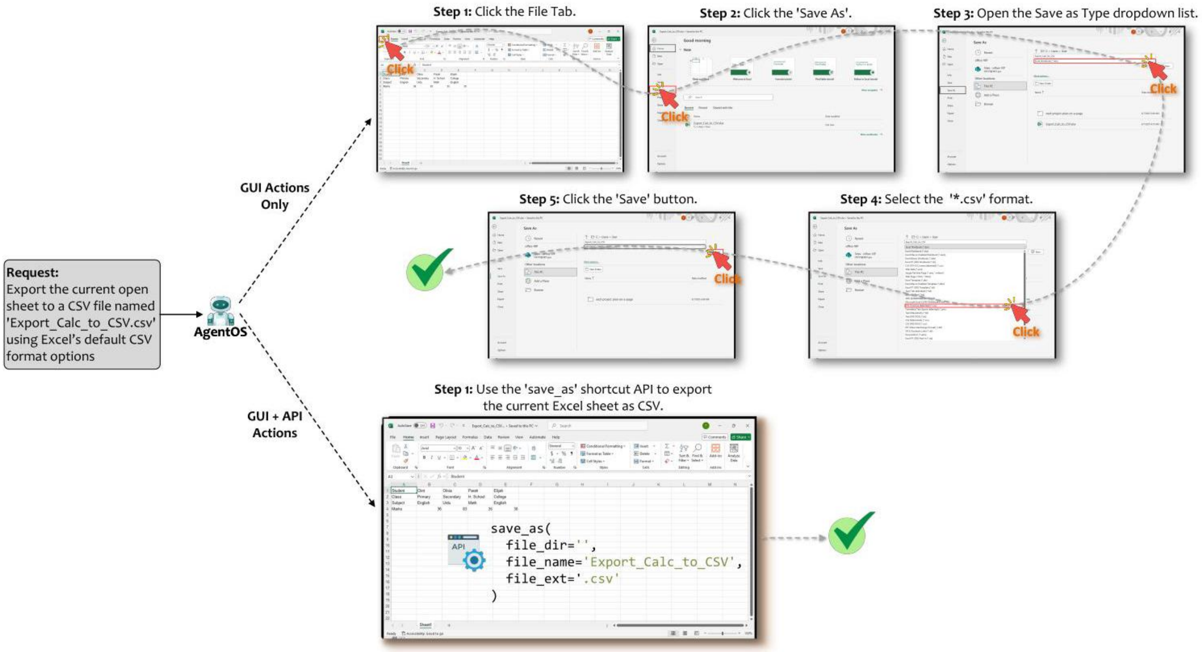
在图 20 中，我们报告了混合方法下从每个来源 (UIA、OmniParser-v2 及合并集) 检测到的控件平均数量。由于应用覆盖差异，OSWorld-W 中检测到的控件总数通常高于 WAA。值得注意的是，UIA 和 OmniParser-v2 均识别出大量控件子集，合并后有 27.9% 和 56.7% 的 OmniParser-v2 检测因为与 UIA 重叠而被丢弃。这些观察表明 OmniParser-v2 通过恢复非标准或自定义元素，为 UIA 提供了有价值的补充。同时，合并步骤移除冗余并防止重复计数，最终在混合方案中减少了控件检测失败。

## 6.4 Effectiveness of GUI + API Integration

## 6.4 GUI 与 API 集成的有效性

We now evaluate how unifying API-based actions with standard GUI interactions in the Puppeteer impacts performance

我们现在评估在 Puppeteer 中将基于 API 的操作与标准 GUI 交互统一如何影响性能



**Request:** Export the current open sheet to a CSV file named 'Export_Calc_to_CSV.csv' using Excel's default CSV format options

**Step 1:** Click the File Tab.

**Step 2:** Click the 'Save As'.

**Step 3:** Open the Save as Type dropdown list.

**Step 4:** Select the '*.csv' format.

**Step 5:** Click the 'Save' button.

**Step 1:** Use the 'save_as' shortcut API to export the current Excel sheet as CSV.

```
save_as(
  file_dir='',
  file_name='Export_Calc_to_CSV',
  file_ext='.csv'
)
```

Figure 21. A case study comparing the completion of the same task using GUI-only actions vs. GUI + API actions.

图 21。对比案例研究: 使用仅 GUI 操作与 GUI+API 操作完成相同任务的比较。

Table 4. APIs supported across Office applications.

表 4。Office 应用中支持的 API 一览。

| API | Application | Description |
|---|---|---|
| select_text | Word | Select matched text in the document. |
| select_paragraph | Word | Select a paragraph in the document. |
| set_font | Word | Set the font size and style of selected text. |
| save_as | Word | Save the current document to a desired format. |
| insert_excel_table | Excel | Insert a table at the desired position. |
| select_table_range | Excel | Select a range within a table. |
| reorder_column | Excel | Reorder columns of a table. |
| save_as | Excel | Save the current sheet to a desired format. |
| set_background_color | PowerPoint | Set the background color of slide(s). |
| save_as | PowerPoint | Save the current presentation to a desired format. |

| API | 应用 | 描述 |
|---|---|---|
| select_text | 单词 | 在文档中选择匹配的文本。 |
| select_paragraph | 单词 | 在文档中选择一个段落。 |
| set_font | 单词 | 设置所选文本的字体大小和样式。 |
| save_as | 单词 | 将当前文档另存为所需格式。 |
| insert_excel_table | Excel | 在指定位置插入表格。 |
| select_table_range | Excel | 选择表格中的一个范围。 |
| reorder_column | Excel | 重新排序表格列。 |
| save_as | Excel | 将当前工作表另存为所需格式。 |
| set_background_color | PowerPoint | 设置幻灯片的背景颜色。 |
| save_as | PowerPoint | 将当前演示文稿另存为所需格式。 |

(Section 3.5). To do so, we focus on the 27 office-related tasks in OSWorld and manually develop 12 APIs for Word, Excel, and PowerPoint. These applications provide COM interfaces that facilitate the creation of custom functions, making them ideal exemplars for deeper OS- and application-level integration. Importantly, many of these operations would require cumbersome multi-step GUI procedures but become straightforward single calls via these APIs (e.g., select paragraphs). Table 4 details the implemented APIs.

(第 3.5 节)。为此，我们聚焦于 OSWorld 中与办公相关的 27 项任务，并为 Word、Excel 和 PowerPoint 手工开发了 12 个 API。这些应用提供 COM 接口，便于创建自定义函数，是进行更深层次操作系统与应用级集成的典型示例。重要的是，许多操作在 GUI 中需通过繁琐的多步流程完成，但通过这些 API 可简化为单次调用 (例如，选择段落)。表 4 详述了已实现的 API。

Table 5. Performance comparison of GUI-only vs. GUI + API actions.

表 5。仅 GUI 与 GUI + API 操作的性能比较。

| Action | Model | SR | PRR | ERR | CRR | ACS |
|---|---|---|---|---|---|---|
| GUI-only | GPT-40 | 16.3% | - | - | - | 13.8 |
| GUI+API | GPT-40 | 22.4% | 5.9% | 14.3% | 25.0% | 12.9 |
| GUI-only | o1 | 16.3% | - | - | - | 16.0 |
| GUI+API | o1 | 24.5% | 17.7% | 0.0% | 12.5% | 6.6 |

| 操作 | 模型 | SR | PRR | ERR | CRR | ACS |
|---|---|---|---|---|---|---|
| 仅限 GUI | GPT-40 | 16.3% | - | - | - | 13.8 |
| GUI+API | GPT-40 | 22.4% | 5.9% | 14.3% | 25.0% | 12.9 |
| 仅限 GUI | o1 | 16.3% | - | - | - | 16.0 |
| GUI+API | o1 | 24.5% | 17.7% | 0.0% | 12.5% | 6.6 |

Table 5 compares (i) overall Success Rate (SR), (ii) Plan Error Recovery rate (PRR), (iii) Execution Error Recovery rate (ERR), (iv) Control Detection Failure Recovery rate (CRR), and (v) Average Completion Steps (ACS) for two configurations: GUI-only versus GUI + API. We calculate ACS on the subset of tasks that both configurations successfully complete, ensuring a fair comparison.

表 5 比较了两种配置 (仅 GUI 与 GUI + API) 的 (i) 总成功率 (SR)、(ii) 计划错误恢复率 (PRR)、(iii) 执行错误恢复率 (ERR)、(iv) 控件检测失败恢复率 (CRR) 和 (v) 平均完成步数 (ACS)。我们在两种配置均成功完成的任务子集上计算 ACS，以确保公平比较。

The results show that integrating API actions boosts SR for both GPT-40 (+6.1%) and o1 (+8.2%), underscoring the effectiveness of mixing GUI and API interactions. Notably, GPT-40 benefits most from APIs in recovering from Control Detection Failures by circumventing unannotated GUI elements. In contrast, o1 more frequently addresses Plan Errors through API ”shortcuts”, reflecting the model’s stronger reasoning capabilities and preference for concise solutions.

结果表明，加入 API 操作提高了 GPT-40(+6.1%) 和 o1(+8.2%) 的 SR，强调了混合 GUI 与 API 交互的有效性。值得注意的是，GPT-40 在通过绕过未标注的 GUI 元素来从控件检测失败中恢复方面受益最大。相比之下，o1 更常通过 API “捷径”解决计划错误，反映出该模型更强的推理能力和对简洁方案的偏好。

Table 6. Performance comparison with and without knowledge integration.

表 6。有无知识整合的性能比较。

| Knowledge Enhancement | Model | WAA | | OSWorld-W | |
|---|---|---|---|---|---|
| | | SR | PRR | SR | PRR |
| None | GPT-40 | 23.4% | - | 22.4% | - |
| Help Document | GPT-40 | 26.6% | 10.34% | 26.5% | 11.8% |
| Self-Experience | GPT-40 | 26.6% | 13.79% | 24.5% | 11.8% |
| None | o1 | 25.3% | - | 24.5% | - |
| Help Document | o1 | 27.9% | 3.5% | 28.5% | 17.7% |
| Self-Experience | o1 | 20.8% | 13.79% | 26.5% | 17.7% |

| 知识增强 | 模型 | WAA | | OSWorld-W | |
|---|---|---|---|---|---|
| | | SR | PRR | SR | PRR |
| 无 | GPT-40 | 23.4% | - | 22.4% | - |
| 帮助文档 | GPT-40 | 26.6% | 10.34% | 26.5% | 11.8% |
| 自我体验 | GPT-40 | 26.6% | 13.79% | 24.5% | 11.8% |
| 无 | o1 | 25.3% | - | 24.5% | - |
| 帮助文档 | o1 | 27.9% | 3.5% | 28.5% | 17.7% |
| 自我体验 | o1 | 20.8% | 13.79% | 26.5% | 17.7% |

Beyond higher success rates, GUI + API also reduces the effort required to complete tasks. UFO $^2$ achieves a 6.5% step savings with GPT-40 and an impressive 58.5% reduction for o1 on identical tasks. The latter improvement stems from o1's ability to strategically call API functions, bypassing multiple GUI-based steps. Overall, these findings confirm the advantages of mixing GUI automation with API calls, both in terms of robustness and efficiency, and showcase the importance of deep system integration for desktop automation.

除了更高的成功率外，GUI + API 还减少了完成任务所需的工作量。UFO $^2$ 在相同任务上使用 GPT-40 实现了 6.5% 的步骤节省，而在 o1 上则达到了令人印象深刻的 58.5% 减少。后者的改进源于 o1 能够策略性地调用 API 函数，从而绕过多个基于 GUI 的步骤。总体而言，这些发现证实了将 GUI 自动化与 API 调用混合使用在鲁棒性和效率方面的优势，并展示了深度系统集成对桌面自动化的重要性。

Case Study. To illustrate how the GUI + API approach streamlines task execution, Figure 21 shows the completion trajectory for exporting an Excel file to CSV format in a case of OSWorld-W, using either GUI-only or GUI + API interactions. Although both configurations eventually succeed, the GUI-only setting requires five steps to open the Save dialog, select the file format, and confirm the action. In contrast, a single call to the save_as API completes the task immediately. Beyond improving efficiency, this one-step solution also reduces the risk of compounding errors across multiple GUI interactions-a clear demonstration of the advantages of deeper OS and application-level integration.

案例研究。为说明 GUI + API 方法如何简化任务执行，图 21 展示了在 OSWorld-W 中将 Excel 文件导出为 CSV 格式时，使用仅 GUI 或 GUI + API 交互的完成轨迹。尽管两种配置最终都能成功，纯 GUI 设置需要五个步骤才能打开“另存为”对话框、选择文件格式并确认操作。相比之下，调用一次 save_as API 即可立即完成任务。除了提升效率外，这一步解决方案还降低了多次 GUI 交互中错误累积的风险——清晰地证明了更深入的操作系统与应用级集成的优势。

## 6.5 Continuous Knowledge Integration Evaluation

## 6.5 持续知识整合评估

We next evaluate the impact of continuous knowledge integration (Section 3.6) on UFO2's performance. Specifically, we augment UFO $^2$ with external documentation and execution-derived insights to dynamically improve its domain understanding without retraining. We create 34 help documents tailored to benchmark tasks, each containing precise step-by-step instructions, enabling UFO$^2$ to retrieve the most relevant guidance (maximum of one per task) at runtime. Additionally, we implement an automated pipeline that summarizes successful execution trajectories-validated by our Task Evaluator and archives them into a retrievable knowledge database. For subsequent tasks, UFO$^2$ dynamically retrieves up to three relevant past execution logs to guide task planning and execution. Given that knowledge integration primarily addresses failures arising from insufficient planning (Plan Errors), we employ the Plan Recovery Ratio (PRR) to measure the proportion of previously failed planning cases successfully resolved by integrating new knowledge.

接下来我们评估持续知识整合 (第 3.6 节) 对 UFO2 性能的影响。具体来说，我们将外部文档和执行中得出的洞见加入到 UFO$^2$ 中，以在无需再训练的情况下动态提升其领域理解。我们为基准任务创建了 34 份帮助文档，每份包含精确的逐步指令，使 UFO$^2$ 能在运行时检索到最相关的指导 (每个任务最多一条)。此外，我们实现了一个自动化流程，汇总经任务评估器验证的成功执行轨迹，并将其存档到可检索的知识库中。对于后续任务，UFO$^2$ 会动态检索最多三条相关的历史执行日志以指导任务规划和执行。鉴于知识整合主要解决因规划不足 (规划错误) 导致的失败，我们采用规划恢复率 (PRR) 来衡量通过整合新知识成功解决的先前失败规划案例的比例。

Table 7. The SR and ACS comparison between single action and speculative multi-action mode.

表 7。单动作模式与推测性多动作模式之间的 SR 与 ACS 比较。

| Action Execution | Model | WAA | | | OSWorld-W | | |
|---|---|---|---|---|---|---|---|
| | | SR | ACS | Success Subset | SR | ACS | Success Subset |
| Single | GPT-4o | 23.4% | 10.00 | 30 | 22.4% | 13.30 | 10 |
| Speculative | GPT-4o | 23.4% | 8.78 | | 24.5% | 7.40 | |
| Single | o1 | 25.3% | 9.95 | 32 | 24.5% | 6.80 | 10 |
| Speculative | o1 | 24.7% | 8.85 | | 26.5% | 3.30 | |

| 动作执行 | 模型 | WAA | | | OSWorld-W | | |
|---|---|---|---|---|---|---|---|
| | | SR | ACS | 成功子集 | SR | ACS | 成功子集 |
| 单一 | GPT-4o | 23.4% | 10.00 | 30 | 22.4% | 13.30 | 10 |
| 推测性 | GPT-4o | 23.4% | 8.78 | | 24.5% | 7.40 | |
| 单一 | o1 | 25.3% | 9.95 | 32 | 24.5% | 6.80 | 10 |
| 推测性 | o1 | 24.7% | 8.85 | | 26.5% | 3.30 | |

Table 6 compares the overall SR and PRRs across two benchmarks, highlighting significant performance improvements attributable to knowledge integration. Both live help-document retrieval and self-experience summarization yield noticeable gains, reducing planning failures by up to 17.7%. Notably, self-experience enhancements using the stronger model (o1) achieve consistent improvements across both benchmarks, underscoring the efficacy of leveraging prior successes for adaptive improvement. While help documents occasionally result in modest gains, their effectiveness depends on task complexity and document specificity.

表 6 比较了两个基准上的整体 SR 和 PRR，突出了归因于知识整合的显著性能提升。无论是实时帮助文档检索还是自我经验摘要，都带来了显著增益，将规划失败率最多减少 17.7%。值得注意的是，使用更强模型 (o1) 的自我经验增强在两个基准上都取得了一致改进，强调了利用既有成功经验实现自适应改进的有效性。虽然帮助文档有时只带来适度提升，但其效果取决于任务复杂性和文档的具体性。

These findings underscore the value of systematic knowledge integration, demonstrating that continuous augmentation of the agent's knowledge base can substantially enhance its robustness, scalability, and adaptability in real-world deployments. Moreover, as UFO$^2$ continues to accumulate execution experience and documentation over time, it inherently evolves toward higher reliability and improved autonomy, marking a clear path for ongoing enhancement in desktop automation.

这些发现强调了系统化知识整合的价值，表明持续扩充智能体知识库可以显著增强其在真实部署中的鲁棒性、可扩展性和适应性。此外，随着 UFO$^2$ 随时间不断积累执行经验和文档，它本身会朝着更高可靠性和更强自治性演进，为桌面自动化的持续改进指明了明确路径。

## 6.6 Effectiveness of Speculative Multi-Action Execution

## 6.6 推测性多动作执行的有效性

Next, we evaluate how speculative multi-action execution (Section 3.7) affects task completion rates and efficiency. Table 7 compares two modes for UFO$^2$ : generating and executing one action per inference (single-action) versus inferring multiple consecutive actions in one step (speculative multi-action). To ensure a fair comparison, we compute the Average Completion Steps (ACS) only on the subset of tasks that both modes complete successfully.

接下来，我们评估推测性多动作执行 (第 3.7 节) 如何影响任务完成率和效率。表 7 比较了 UFO$^2$ 的两种模式: 每次推理生成并执行一个动作 (单动作) 与在一步中推断多次连续动作 (推测性多动作)。为确保公平比较，我们仅在两种模式都成功完成的任务子集中计算平均完成步数 (ACS)。

The results show that speculative multi-action execution retains a comparable Success Rate (SR) to single-action mode while notably cutting the average steps-by up to 10% on WAA and an impressive 51.5% on OSWorld-W. Because each step requires an LLM call, reducing the number of steps significantly lowers both latency and cost. This finding confirms that speculative multi-action planning enhances efficiency without compromising reliability, further highlighting UFO2's ability to optimize resource utilization in practical desktop automation.

结果表明，推测性多动作执行在保持与单动作模式相当的成功率 (SR) 的同时，明显减少了平均步骤——在 WAA 上最多减少 10%，在 OSWorld-W 上则高达 51.5%。由于每一步都需要一次 LLM 调用，减少步骤数能显著降低延迟和成本。该发现证实推测性多动作规划在不牺牲可靠性的前提下提升了效率，进一步凸显了 UFO2 在实际桌面自动化中优化资源利用的能力。

Case Study. Figure 22 illustrates how speculative multi-action execution operates in practice. When the user requests that UFO$^2$ center-align a heading in a Word document, the sequence of steps would typically require selecting the heading text and then clicking the Center icon. These actions that are sequentially dependent but do not interfere with each other. Instead of treating each action as a separate LLM inference, UFO$^2$ predicts both actions in a single step, leveraging speculative multi-action planning. Consequently, it completes the task with just one LLM call, significantly enhancing efficiency while maintaining accuracy.

案例研究。图 22 展示了推测性多动作执行的实际操作。当用户请求 UFO$^2$ 在 Word 文档中将标题居中对齐时，通常的步骤序列需要先选中文本然后点击“居中”图标。这些动作在顺序上相互依赖但互不干扰。UFO$^2$ 并不将每个动作视为单独的 LLM 推理，而是在一步中预测出两个动作，利用推测性多动作规划。因此，它仅用一次 LLM 调用就完成任务，大幅提高了效率并保持准确性。

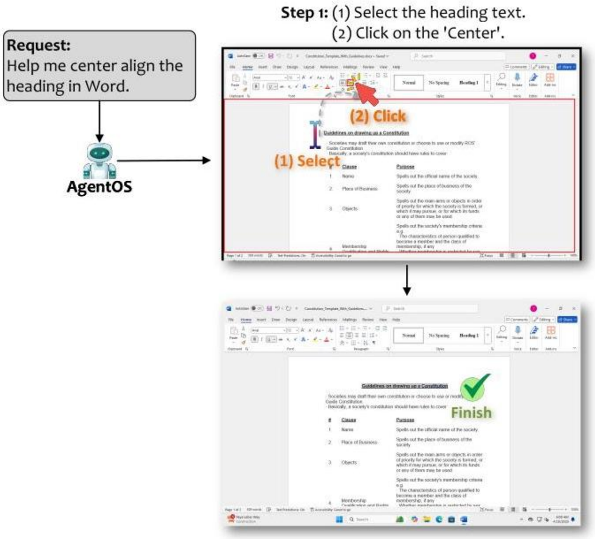

Step 1: (1) Select the heading text.
(2) Click on the 'Center'.
Request:
Help me center align the heading in Word.
AgentOS
(2) Click
(1) Select
Finish

Figure 22. A case study of the successful speculative multi-action execution.

图 22。一次成功的推测性多动作执行的案例研究。

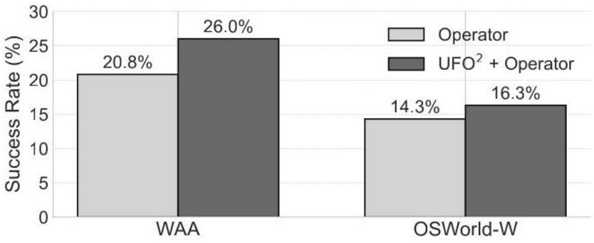


Figure 23. Comparison of Operator vs. UFO$^2$+ Operator on WAA and OSWorld-W.

图 23。WAA 和 OSWorld-W 上 Operator 与 UFO$^2$+ Operator 的比较。

## 6.7 Operator as an AppAGENT

## 6.7 将 Operator 作为 AppAGENT

To demonstrate the ”Everything-as-an-AppAGENT” capability (Section 5.3), we conducted an experiment where UFO2’s HosTAGENT orchestrator uses only Operator as the AppA-GENT. In other words, all native AppAGENTS were disabled, leaving Operator to accept subtasks and communicate via the standard

UFO$^2$ messaging protocol. The only adjustment to Operator's perception layer was restricting it to screenshots of the selected application window, rather than the full desktop.

为展示“万物皆 AppAGENT”能力 (第 5.3 节)，我们进行了一个实验，UFO2 的 HosTAGENT 协调器仅使用 Operator 作为 AppA-GENT。换言之，所有原生 AppAGENT 被禁用，仅由 Operator 接受子任务并通过标准 UFO$^2$ 消息协议通信。对 Operator 感知层的唯一下调是将其限制为所选应用窗口的屏幕截图，而非整个桌面。

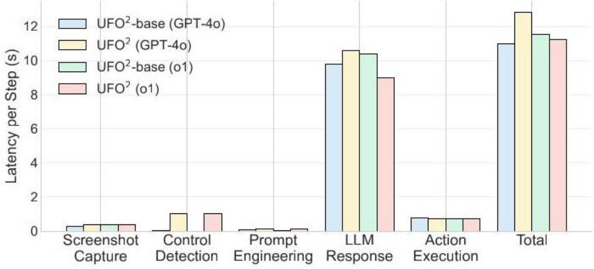


Figure 24. Average time cost per-stage of a single execution step.

图 24。单次执行步骤各阶段的平均时间成本。

Figure 23 shows that UFO$^2$+ Operator achieves higher success rates than running Operator alone, particularly on WAA (26.0% vs. 20.8%). We attribute these gains to three key factors. First, the HosTAGENT breaks down complex user instructions into clearer, single-application subtasks, reducing ambiguity. Second, HosTAGENT messages include additional tips that improve Operator's decision making. Finally, limiting Operator's view to a single, active application window reduces visual noise and simplifies control detection. Taken together, these results underscore the benefits of UFO$^{2,}$ s multiagent design, while demonstrating how ”Everything-as-an-APPAGENT” can elevate the performance of an existing CUA.

图 23 显示 UFO$^2$+ Operator 的成功率高于单独运行的 Operator，尤其在 WAA 上 (26.0% 对 20.8%)。我们将这些提升归因于三点关键因素。首先，HosTAGENT 将复杂的用户指令细分为更清晰的单应用子任务，减少了歧义。其次，HosTAGENT 的消息包含额外提示，改善了 Operator 的决策。最后，将 Operator 的视野限定在单一活动应用窗口减少了视觉噪声并简化了控件检测。综上，这些结果强调了 UFO$^{2,}$ s 多智能体设计的优势，同时展示了“万物皆 APPAGENT”如何提升现有 CUA 的性能。

## 6.8 Efficiency Analysis

## 6.8 效率分析

To comprehensively understand the performance characteristics of UFO$^2$ , we conducted detailed profiling of task execution efficiency, focusing specifically on step count and latency.

为全面了解 UFO$^2$ 的性能特征，我们对任务执行效率进行了详细剖析，重点关注步数和延迟。

Step Count Profiling. Table 8 summarizes the average number of execution steps performed by the HosTAGENT and AppAGENTS across both benchmark suites. The steps reported are computed on tasks that were successfully completed across all model configurations, ensuring fair comparisons. Two key insights emerge:

步数分析。表 8 汇总了 HosTAGENT 和 AppAGENTS 在两个基准套件中执行的平均步骤数。所报告的步骤基于在所有模型配置下成功完成的任务，确保比较公正。由此得出两点重要见解:

First, the fully integrated UFO$^2$ configuration consistently reduces the average number of steps required compared to the baseline (UFO $^2$ -base), achieving reductions of up to 50% . This substantial efficiency gain demonstrates how deep OS integration, specifically the hybrid GUI-API action orchestration and advanced control detection strategies, significantly streamline execution paths.

首先，完全集成的 UFO$^2$ 配置相比基线 (UFO $^2$ -base) 持续减少了平均步骤数，最多可降低 50% 。这一显著的效率提升表明，深度操作系统集成，尤其是混合 GUI-API 操作编排和高级控制检测策略，大幅简化了执行路径。

Second, utilizing a more powerful reasoning model (e.g., o1 versus GPT-40) further reduces step counts, indicating that enhanced reasoning capability enables the agent to identify and exploit more efficient action sequences. For instance, stronger models can better leverage direct API interactions or avoid unnecessary intermediate GUI interactions. This underscores the complementary role of both robust system integration and advanced LLM reasoning in minimizing execution overhead.

其次，使用更强大的推理模型 (例如 o1 对比 GPT-40) 进一步减少了步骤数，表明增强的推理能力使代理能够识别并利用更高效的动作序列。例如，能力更强的模型能更好地利用直接 API 交互或避免不必要的中间 GUI 交互。这强调了稳健系统集成与高级 LLM 推理在最小化执行开销方面的互补作用。

Table 8. Step count statistic for UFO$^2$ .

表 8. UFO$^2$ 的步骤数统计。

| Agent | Model | WAA | | | | OSWorld-W | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | HostAgent | AppAGENT | Total | Success Subset | HostAgent | AppAgent | Total | Success Subset |
| UFO$^2$ -base | GPT-40 | 2.21 | 8.11 | 10.32 | 31 | 1.80 | 10.80 | 12.60 | 7 |
| UFO$^2$ | GPT-40 | 2.32 | 7.89 | 10.21 | | 2.80 | 7.20 | 10.00 | |
| UFO$^2$ -base | o1 | 2.14 | 7.00 | 9.14 | 34 | 2.50 | 8.83 | 11.33 | 8 |
| UFO$^2$ | o1 | 2.00 | 4.05 | 6.05 | | 2.00 | 3.50 | 5.50 | |

| 代理 | 模型 | WAA | | | | OSWorld-W | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 主机代理 | 应用代理 | 总计 | 成功子集 | 主机代理 | 应用代理 | 总计 | 成功子集 |
| UFO$^2$ -基础 | GPT-40 | 2.21 | 8.11 | 10.32 | 31 | 1.80 | 10.80 | 12.60 | 7 |
| UFO$^2$ | GPT-40 | 2.32 | 7.89 | 10.21 | | 2.80 | 7.20 | 10.00 | |
| UFO$^2$ -基础 | o1 | 2.14 | 7.00 | 9.14 | 34 | 2.50 | 8.83 | 11.33 | 8 |
| UFO$^2$ | o1 | 2.00 | 4.05 | 6.05 | | 2.00 | 3.50 | 5.50 | |

Latency Breakdown. Figure 24 provides a detailed breakdown of average latency per execution step in UFO$^2$ , separated into five key phases: (i) Screenshot Capture, (ii) Control Detection via UIA APIs (and) OmniParser-v2, (iii) Prompt Preparation, including retrieval of relevant help documents and historical execution experiences, (iv) LLM Inference, and (v) Action Execution on target applications.

延迟构成。图 24 展示了在 UFO$^2$ 中每个执行步骤平均延迟的详细分解，分为五个关键阶段:(i) 截图采集，(ii) 通过 UIA API 和 OmniParser-v2 进行控件检测，(iii) 提示准备，包括检索相关帮助文档和历史执行经验，(iv) LLM 推理，和 (v) 在目标应用上的动作执行。

Across all configurations, the LLM inference phase dominates total latency, averaging around 10 seconds per inference. This bottleneck highlights a clear opportunity for optimization by deploying smaller, specialized models or employing more powerful inference hardware-strategies that remain viable but are beyond our current evaluation scope.

在所有配置中，LLM 推理阶段主导总体延迟，平均每次推理约为 10 秒。该瓶颈指出了通过部署更小型的专用模型或采用更强大的推理硬件策略进行优化的明显机会，但这些策略仍超出我们当前评估范围。

Excluding LLM inference overhead, the baseline system (UFO2-base) achieves highly efficient execution, incurring around 10 seconds per step on average. In contrast, the fully integrated UFO$^2$ incurs only an additional 1 second per step for its hybrid control detection pipeline, largely due to OmniParser-v2 visual parsing. This added overhead represents a deliberate trade-off, significantly enhancing the robustness and accuracy of GUI control detection at a modest latency cost.

不计 LLM 推理开销，基线系统 (UFO2-base) 实现了高效执行，平均每步约耗时 10 秒。相比之下，完全集成的 UFO$^2$ 为其混合控件检测管线仅增加了每步约 1 秒的开销，这主要来自 OmniParser-v2 的视觉解析。该额外开销是一个有意的权衡，以适度的延迟成本显著提升 GUI 控件检测的鲁棒性和准确性。

Taken together, these results indicate that the substantial reduction in total steps required per task ensures overall task completion times remain practical (approximately 1 minute per task). These profiling insights reinforce that UFO$^2$ ’ s comprehensive system-level integration balances latency, accuracy, and efficiency, offering a scalable and performant solution for real-world desktop automation.

综合来看，这些结果表明每个任务所需总步骤的大幅减少确保了整体任务完成时间保持在可行范围内 (约每任务 1 分钟)。这些性能剖析表明 UFO$^2$ 的全系统级集成在延迟、准确性和效率之间达成平衡，为现实桌面自动化提供了可扩展且高效的解决方案。

## 6.9 Model Ablation

## 6.9 模型消融

Table 25 compares the performance of UFO$^2$ and UFO$^2$ -base across four large language models. GPT-4V and GPT-4o generate direct answers without exposing an explicit CoT, whereas Gemini 2.0 (Flash Thinking) and o1 embed reasoning steps internally before producing final outputs. Overall, models with built-in reasoning typically achieve higher success rates (SR), highlighting the value of more deliberative or CoT-driven processes in desktop automation [52].

表 25 比较了 UFO$^2$ 与 UFO$^2$ -base 在四种大型语言模型上的表现。GPT-4V 和 GPT-4o 在不展示显式思考链 (CoT) 的情况下直接生成答案，而 Gemini 2.0(Flash Thinking) 和 o1 在输出最终结果前将推理步骤内嵌。总体上，内置推理的模型通常取得更高的成功率 (SR)，凸显了更审慎或基于 CoT 的流程在桌面自动化中的价值 [52]。

This result underscores a promising direction for CUAs: fine-tuning advanced reasoning models specifically for desktop automation tasks. By allowing agents to formulate and refine multi-step plans-especially when integrated with deeper OS-level signals-UFO $^2$ can address complex or ambiguous situations more reliably. As LLM-based reasoning continues to mature, we expect further gains in both accuracy and generality from UFO2's model-agnostic design.

该结果强调了 CUAs 的一条有前景方向: 针对桌面自动化任务对先进推理模型进行微调。允许智能体制定并优化多步计划——尤其是与更深层的操作系统级信号集成时——UFO $^2$ 能更可靠地应对复杂或模糊情形。随着基于 LLM 的推理持续成熟，我们预计来自 UFO2 的模型无关设计将在准确性和通用性方面带来进一步提升。

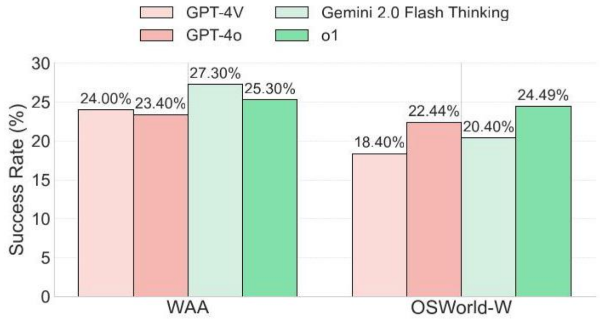


Figure 25. Comparison of different LLMs used in UFO$^2$ and UFO$^2$ -base on WAA and OSWorld-W.

图 25。比较在 WAA 和 OSWorld-W 上用于 UFO$^2$ 与 UFO$^2$ -base 的不同 LLM。

# 7 Discussion & Future Work

# 7 讨论与未来工作

Latency and Responsiveness. UFO$^2$ currently invokes LLM inference at each decision step, incurring a latency typically ranging from several seconds to tens of seconds per action. Despite various engineering optimizations, complex tasks comprising multiple sequential actions can accumulate to execution times of 1-2 minutes, which remains acceptable but still inferior to skilled human performance. To alleviate user-perceived delay, we introduced the Picture-in-Picture (PiP) interface, enabling UFO$^2$ to execute tasks unobtrusively within an isolated virtual desktop, thus substantially reducing user inconvenience during longer-running automations. In future work, we aim to further lower latency by investigating the deployment of specialized, lightweight Large Action Models (LAMs) [13], optimized for task-specific inference to enhance both responsiveness and scalability.

延迟与响应性。UFO$^2$ 目前在每个决策步骤调用 LLM 推理，导致每次动作通常需数秒至数十秒的延迟。尽管进行了各种工程优化，包含多步顺序动作的复杂任务的执行时间仍可累积到 1–2 分钟，虽可接受但仍劣于熟练的人类表现。为减少用户感知延迟，我们引入了画中画 (PiP) 界面，使 UFO$^2$ 能在隔离的虚拟桌面内不干扰地执行任务，从而在长时间运行的自动化过程中显著减少用户不便。未来工作中，我们计划通过研究部署专用的轻量级大动作模型 (LAMs)[13] 来进一步降低延迟，这些模型为特定任务优化推理，以提升响应性和可扩展性。

Closing the Gap with Human-Level Performance. Our comprehensive evaluations indicate that UFO$^2$ , while robust and effective, does not yet consistently achieve human-level performance across all Windows applications. Bridging this gap will necessitate advances primarily along two critical dimensions. First, enhancing foundational visual-language models through fine-tuning on extensive, diverse GUI interaction datasets will significantly improve agents' capabilities and generalization across varied applications. Second, tighter integration with OS-level APIs, native application interfaces, and comprehensive, structured documentation sources will deepen contextual understanding and bolster execution reliability. Given UFO2's modular architecture, these enhancements can be incrementally adopted, continuously refining performance towards human-equivalent proficiency across diverse application scenarios.

缩小与人类水平的差距。我们的综合评估表明，UFO$^2$ 虽然稳健且有效，但尚未在所有 Windows 应用上持续达到人类水平的表现。缩小此差距主要需要沿两个关键方向取得进展。首先，通过在大规模、多样化的 GUI 交互数据集上微调基础的视觉-语言模型，将显著提升智能体的能力及跨应用的泛化性。其次，与操作系统级 API、本地应用接口和全面的结构化文档源进行更紧密的集成，将加深上下文理解并增强执行可靠性。鉴于 UFO2 的模块化架构，这些改进可逐步采用，持续提升在不同应用场景下达到人类等效熟练度的表现。

Generalization Across Operating Systems. While UFO $^2$ targets Windows OS due to its widespread market adoption (over 70% market share $^2$ ), the modular, layered system design facilitates straightforward adaptation to other desktop platforms, such as Linux and macOS. The core approach-leveraging accessibility frameworks like Windows UI Automation (UIA)-has direct analogs on Linux (AT-SPI [53]) and macOS (Accessibility API [54]). Consequently, the existing design principles, agent decomposition strategy, and unified GUI-API orchestration model generalize naturally, enabling rapid, platform-specific customizations. Exploring cross-platform deployments will be an important area of future work, potentially laying the groundwork for a unified ecosystem of desktop automation solutions spanning diverse operating environments.

跨操作系统的泛化。虽然 UFO $^2$ 针对 Windows 操作系统，因为其市场占有率高 (超过 70% 市场份额 $^2$ )，但其模块化、分层的系统设计便于向其他桌面平台 (如 Linux 和 macOS) 进行直接适配。核心方法——利用无障碍框架如 Windows UI Automation (UIA)——在 Linux(AT-SPI [53]) 和 macOS(Accessibility API [54]) 上有直接对应。因此，现有的设计原则、代理分解策略以及统一的 GUI-API 协调模型可以自然地推广，从而实现快速的、平台特定的定制。探索跨平台部署将是未来工作的重要方向，可能为涵盖多样操作环境的统一桌面自动化生态系统奠定基础。

# 8 Related Work

# 8 相关工作

Integrating LLMs into OS represents a growing, yet nascent area of research. In this section, we discuss prior work that intersects with our research on system-level integration of multimodal LLM-based desktop automation agents.

将 LLM 集成到操作系统是一个不断发展且尚处早期的研究领域。本节讨论与我们关于多模态 LLM 驱动的桌面自动化代理的系统级集成研究相关的既往工作。

## 8.1 Computer-Using Agents (CUAs)

## 8.1 计算机使用代理 (CUAs)

Recent advancements in multimodal LLMs have significantly accelerated the development of Computer-Using Agents (CUAs), which automate desktop workflows by simulating GUI interactions at the OS level. Early pioneering systems, such as UFO [10], leveraged multimodal models (e.g., GPT-4V [20]) alongside UIA APIs to interpret graphical interfaces and execute complex tasks via natural language instructions. UFO notably introduced a multi-agent architecture, enhancing the reliability and capability of CUAs to handle cross-application and long-term workflows.

多模态 LLM 的近期进展显著加速了计算机使用代理 (CUAs) 的发展，CUAs 通过在操作系统级别模拟 GUI 交互来自动化桌面工作流。早期开创性系统如 UFO [10] 利用多模态模型 (例如 GPT-4V [20]) 并结合 UIA API 来理解图形界面并通过自然语言指令执行复杂任务。UFO 显著引入了多代理架构，提高了 CUA 在跨应用和长期工作流中的可靠性和能力。

Subsequent efforts have focused primarily on refining underlying multimodal models and extending platform capabilities. For example, CogAgent [55], built upon the vision-language model CogVLM [56], specialized in GUI understanding across multiple platforms (PC, web, Android), representing one of the earliest dedicated multimodal CUAs. Industry interest has similarly accelerated with Anthropic's Claude-3.5 (Computer Use) [11], an agent relying entirely on screenshot-based GUI interactions, and OpenAI's Operator [12], which significantly improved desktop automation performance through advanced multimodal reasoning.

随后的工作主要集中在改进底层多模态模型和扩展平台能力。例如，基于视觉-语言模型 CogVLM [56] 构建的 CogAgent [55] 专注于跨多平台 (PC、网页、Android) 的 GUI 理解，代表了最早的专用多模态 CUA 之一。行业兴趣也随之加速，Anthropic 的 Claude-3.5 (Computer Use) [11] 完全依赖截屏的 GUI 交互，而 OpenAI 的 Operator [12] 通过高级多模态推理显著提升了桌面自动化性能。

However, these existing CUAs remain largely prototype demonstrations, often lacking deep integration with the OS and native application capabilities. In contrast, our work in UFO$^2$ directly addresses these fundamental system-level limitations through a modular AgentOS architecture, deep OS and API integration, hybrid GUI detection, and a nondisruptive execution model, bridging the gap between conceptual CUAs and practical desktop automation.

然而，这些现有的 CUA 多为原型演示，通常缺乏与操作系统和本地应用的深度集成。相比之下，我们在 UFO$^2$ 中通过模块化的 AgentOS 架构、深度的操作系统与 API 集成、混合 GUI 检测和非侵入性的执行模型直接解决了这些系统级的根本限制，弥合了概念性 CUA 与实用桌面自动化之间的差距。

## 8.2 LLMs for Operating Systems

## 8.2 将 LLM 应用于操作系统

Another promising research direction involves embedding LLMs directly within OS architectures, aiming to substantially enhance automation, adaptability, and usability. Ge et al., first proposed the conceptual framework AIOS [57], positioning an LLM at the center of OS design to orchestrate high-level user interactions and automated decision-making. In their vision, agents resemble OS applications, each exposing specialized capabilities accessible via natural language, effectively enabling users to ”program” their OS intuitively.

另一条有前景的研究方向是将 LLM 直接嵌入操作系统架构，旨在大幅增强自动化、适应性和可用性。Ge 等人首次提出了概念框架 AIOS [57]，将 LLM 置于操作系统设计的中心以协调高层用户交互和自动化决策。在他们的设想中，代理类似于操作系统应用，每个代理暴露可通过自然语言访问的专门能力，实质上使用户能够直观地“编程”他们的操作系统。

Building on this conceptual foundation, Mei et al., [58] realized AIOS as a concrete prototype, encapsulating LLM interactions and tool APIs within a privileged OS kernel. This design provides core OS functionalities such as process scheduling, memory management, I/O handling, and access control, leveraging LLMs to simplify agent development through a dedicated SDK. Rama et al., [59] extended this paradigm, introducing semantic file management capabilities directly within traditional OS environments through AIOS-based agents, further demonstrating practical system-level integration.

在此概念基础上，Mei 等人 [58] 将 AIOS 实现为具体原型，将 LLM 交互和工具 API 封装在特权操作系统内核中。该设计提供了进程调度、内存管理、I/O 处理和访问控制等核心操作系统功能，并通过专用 SDK 利用 LLM 简化代理开发。Rama 等人 [59] 扩展了这一范式，通过基于 AIOS 的代理在传统操作系统环境中直接引入语义文件管理能力，进一步展示了实际的系统级集成。

Complementing these high-level OS integrations, AutoOS [60] applied LLMs for automatic tuning of kernel-level parameters in Linux, achieving substantial efficiency gains through autonomous exploration and optimization. This highlights another dimension where LLM integration can directly enhance core system performance and management.

补充这些高层操作系统集成的是 AutoOS [60]，它将 LLM 应用于 Linux 内核级参数的自动调优，通过自主探索与优化实现了显著的效率提升。这凸显了 LLM 集成可直接增强核心系统性能与管理的另一个维度。

Collectively, these research efforts illustrate an emerging paradigm shift where LLMs become integral components of operating systems, enabling powerful automation, enhanced user interaction, and adaptive system behavior. Our work with UFO$^2$ extends this line of research specifically to desktop automation, offering a deeply integrated, scalable, and practical AgentOS that leverages multimodal LLMs in conjunction with robust OS-level mechanisms.

综上，这些研究工作展示了一种新兴范式转变，即 LLM 成为操作系统的核心组件，从而实现强大的自动化、增强的用户交互和自适应系统行为。我们在 UFO$^2$ 的工作将该研究推进到桌面自动化领域，提供了一个深度集成、可扩展且实用的 AgentOS，将多模态 LLM 与稳健的操作系统级机制相结合。

## 9 Conclusion

## 9 结论

We introduced **UFO$^2$** , a practical, OS-integrated Windows desktop automation AgentOS that transforms CUAs from conceptual prototypes into robust, user-oriented solutions. Unlike prior CUAs, UFO$^2$ leverages deep system-level integration through a modular, multiagent architecture consisting of a centralized HosTAGENT and application-specialized APPAGENTS. Each APPAGENT seamlessly combines GUI interactions with native APIs and continually integrates application-specific knowledge, substantially improving reliability and execution efficiency. UFO$^2$ can operate on a PiP virtual desktop interface further enhances usability, enabling concurrent user-agent workflows without interference.

我们提出了 **UFO$^2$** ，一个实用的、与操作系统集成的 Windows 桌面自动化 AgentOS，将 CUA 从概念原型转变为面向用户的稳健解决方案。与以往的 CUA 不同，UFO$^2$ 通过模块化多智能体架构实现深度系统级集成，包含一个集中式 HosTAGENT 和若干面向应用的 APPAGENT。每个 APPAGENT 无缝结合 GUI 交互与原生 API，并持续整合应用特定知识，大幅提升了可靠性和执行效率。UFO$^2$ 还能在 PiP 虚拟桌面界面上运行，进一步增强可用性，使用户与智能体能并行工作而互不干扰。

Our comprehensive evaluation across over 20 real-world Windows applications demonstrated that UFO$^2$ achieves significant improvements in robustness, accuracy, and scalabil-ity compared to state-of-the-art CUAs. Notably, by coupling our integrated framework with robust OS-level features, even less specialized foundation models (e.g., GPT-40) surpass specialized CUAs such as Operator.

我们在 20 多款真实 Windows 应用上的全面评估表明，与最先进的 CUA 相比，UFO$^2$ 在鲁棒性、准确性和可扩展性方面均有显著提升。值得注意的是，通过将我们的集成框架与稳健的操作系统级功能结合，即使是较不专门的基础模型 (例如 GPT-40) 也能超越诸如 Operator 等专门化 CUA。

[2] https://gs.statcounter.com/os-market-share/desktop/worldwide/ #monthly-202301-202301-bar

[2] https://gs.statcounter.com/os-market-share/desktop/worldwide/ #monthly-202301-202301-bar

# References

## References

[1] UiPath. Uipath: Automation platform, 2025. Accessed: 2025-03-18.

[1] UiPath. Uipath: Automation platform, 2025. Accessed: 2025-03-18.

[2] Automation Anywhere. Automation anywhere: Automation 360 platform, 2025. Accessed: 2025-03-18.

[2] Automation Anywhere. Automation anywhere: Automation 360 platform, 2025. Accessed: 2025-03-18.

[3] Microsoft. Microsoft power automate, 2025. Accessed: 2025-03-18.

[3] Microsoft. Microsoft power automate, 2025. Accessed: 2025-03-18.

[4] Peter Hofmann, Caroline Samp, and Nils Urbach. Robotic process automation. Electronic markets, 30(1):99-106, 2020.

[4] Peter Hofmann, Caroline Samp, and Nils Urbach. Robotic process automation. Electronic markets, 30(1):99-106, 2020.

[5] Somayya Madakam, Rajesh M Holmukhe, and Durgesh Kumar Jaiswal. The future digital work force: robotic process automation (rpa). JISTEM-Journal of Information Systems and Technology Management, 16:e201916001, 2019.

[5] Somayya Madakam, Rajesh M Holmukhe, and Durgesh Kumar Jaiswal. The future digital work force: robotic process automation (rpa). JISTEM-Journal of Information Systems and Technology Management, 16:e201916001, 2019.

[6] Dhanya Pramod. Robotic process automation for industry: adoption status, benefits, challenges and research agenda. Benchmarking: an international journal, 29(5):1562-1586, 2022.

[6] Dhanya Pramod. Robotic process automation for industry: adoption status, benefits, challenges and research agenda. Benchmarking: an international journal, 29(5):1562-1586, 2022.

[7] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, et al. Large language model-brained gui agents: A survey. arXiv preprint arXiv:2411.18279, 2024.

[7] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, et al. Large language model-brained gui agents: A survey. arXiv preprint arXiv:2411.18279, 2024.

[8] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 1(2), 2023.

[8] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 1(2), 2023.

[9] Duzhen Zhang, Yahan Yu, Jiahua Dong, Chenxing Li, Dan Su, Chenhui Chu, and Dong Yu. Mm-llms: Recent advances in multimodal large language models. arXiv preprint arXiv:2401.13601, 2024.

[9] Duzhen Zhang, Yahan Yu, Jiahua Dong, Chenxing Li, Dan Su, Chenhui Chu, and Dong Yu. Mm-llms: Recent advances in multimodal large language models. arXiv preprint arXiv:2401.13601, 2024.

[10] Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, et al. Ufo: A ui-focused agent for windows os interaction. arXiv preprint arXiv:2402.07939, 2024.

[10] Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, et al. Ufo: A ui-focused agent for windows os interaction. arXiv preprint arXiv:2402.07939, 2024.

[11] Anthropic. Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku, 2024. Accessed: 2024-10-26.

[11] Anthropic. Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku, 2024. Accessed: 2024-10-26.

[12] OpenAI. Computer-using agent: Introducing a universal interface for ai to interact with the digital world. 2025.

[12] OpenAI. Computer-using agent: Introducing a universal interface for ai to interact with the digital world. 2025.

[13] Lu Wang, Fangkai Yang, Chaoyun Zhang, Junting Lu, Jiaxu Qian, Shilin He, Pu Zhao, Bo Qiao, Ray Huang, Si Qin, et al. Large action models: From inception to implementation. arXiv preprint arXiv:2412.10047, 2024.

[13] Lu Wang, Fangkai Yang, Chaoyun Zhang, Junting Lu, Jiaxu Qian, Shilin He, Pu Zhao, Bo Qiao, Ray Huang, Si Qin, et al. Large action models: From inception to implementation. arXiv preprint arXiv:2412.10047, 2024.

[14] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. arXiv preprint arXiv:2501.12326, 2025.

[14] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. arXiv preprint arXiv:2501.12326, 2025.

[15] Jiani Zheng, Lu Wang, Fangkai Yang, Chaoyun Zhang, Lingrui Mei, Wenjie Yin, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. Vem: Environment-free exploration for training gui agent with value environment model. arXiv preprint arXiv:2502.18906, 2025.

[15] Jiani Zheng, Lu Wang, Fangkai Yang, Chaoyun Zhang, Lingrui Mei, Wenjie Yin, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. Vem: Environment-free exploration for training gui agent with value environment model. arXiv preprint arXiv:2502.18906, 2025.

[16] 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-Third International Joint Conference on Artificial Intelligence, pages 6433-6441, 2024.

[16] 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-Third International Joint Conference on Artificial Intelligence, pages 6433-6441, 2024.

[17] Chaoyun Zhang, Shilin He, Liqun Li, Si Qin, Yu Kang, Qingwei Lin, and Dongmei Zhang. Api agents vs. gui agents: Divergence and convergence. arXiv preprint arXiv:2503.11069, 2025.

[17] Chaoyun Zhang, Shilin He, Liqun Li, Si Qin, Yu Kang, Qingwei Lin, and Dongmei Zhang. Api agents vs. gui agents: Divergence and convergence. arXiv preprint arXiv:2503.11069, 2025.

[18] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omni-parser for pure vision based gui agent. arXiv preprint arXiv:2408.00203, 2024.

[18] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omni-parser for pure vision based gui agent. arXiv preprint arXiv:2408.00203, 2024.

[19] Julia Siderska, Lili Aunimo, Thomas Süße, John von Stamm, Damian Kedziora, and Suraya Nabilah Binti Mohd Aini. Towards intelligent automation (ia): literature review on the evolution of robotic process automation (rpa), its challenges, and future trends. Engineering Management in Production and Services, 15(4), 2023.

[19] Julia Siderska, Lili Aunimo, Thomas Süße, John von Stamm, Damian Kedziora, and Suraya Nabilah Binti Mohd Aini. Towards intelligent automation (ia): literature review on the evolution of robotic process automation (rpa), its challenges, and future trends. Engineering Management in Production and Services, 15(4), 2023.

[20] Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. The dawn of lmms: Preliminary explorations with gpt-4v (ision). arXiv preprint arXiv:2309.17421, 9(1):1, 2023.

[20] Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. The dawn of lmms: Preliminary explorations with gpt-4v (ision). arXiv preprint arXiv:2309.17421, 9(1):1, 2023.

[21] David N Gray, John Hotchkiss, Seth LaForge, Andrew Shalit, and Toby Weinberg. Modern languages and microsoft's component object model. Communications of the ACM, 41(5):55-65, 1998.

[21] David N Gray, John Hotchkiss, Seth LaForge, Andrew Shalit, and Toby Weinberg. Modern languages and microsoft's component object model. Communications of the ACM, 41(5):55-65, 1998.

[22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In International Conference on Learning Representations (ICLR), 2023.

[22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In International Conference on Learning Representations (ICLR), 2023.

[23] Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. Taskweaver: A code-first agent framework. arXiv preprint arXiv:2311.17541, 2023.

[23] Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. Taskweaver: A code-first agent framework. arXiv preprint arXiv:2311.17541, 2023.

[24] Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhuo Xu, and Chaoyang He. Llm multi-agent systems: Challenges and open problems. arXiv preprint arXiv:2402.03578, 2024.

[24] Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhuo Xu, and Chaoyang He. Llm multi-agent systems: Challenges and open problems. arXiv preprint arXiv:2402.03578, 2024.

[25] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. Frontiers of Computer Science, 18(6):186345, 2024.

[25] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. Frontiers of Computer Science, 18(6):186345, 2024.

[26] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. arXiv preprint arXiv:2404.13501, 2024.

[26] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. arXiv preprint arXiv:2404.13501, 2024.

[27] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. arXiv preprint arXiv:2310.11441, 2023.

[27] 杨建伟, 张昊, 李锋, 邹雪燕, 李春元, 高建峰. Set-of-mark prompting在gpt-4v中释放了非凡的视觉定位能力. arXiv预印本arXiv:2310.11441, 2023.

[28] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824-24837, 2022.

[28] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, 等. Chain-of-thought prompting在大型语言模型中引发推理. Advances in Neural Information Processing Systems, 35:24824-24837, 2022.

[29] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Everything of thoughts: Defying the law of penrose triangle for thought generation. In Findings of the Association for Computational Linguistics ACL 2024, pages 1638-1662, 2024.

[29] 丁若梦, 张朝云, 王露, 徐勇, 马明华, 张伟, 秦思, Saravan Rajmohan, 林清伟, 张冬梅. Everything of thoughts: 违背彭罗斯三角法则的思维生成. 在ACL 2024 Findings, 页1638-1662, 2024.

[30] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 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), pages 9313-9332, 2024.

[30] 程坤志, 孙秋实, 褚友刚, 徐方志, 闫涛利, 张建兵, 伍志勇. Seeclick: 利用GUI定位推动高级视觉GUI代理. 在第62届ACL年会论文集 (长文卷), 页9313-9332, 2024.

[31] 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. arXiv preprint arXiv:2410.05243, 2024.

[31] 苟博宇, 王若涵, 郑博元, 谢雅楠, 常诚, 舒意恒, 孙焕, 苏煜. 像人类一样导航数字世界: 面向GUI代理的通用视觉定位. arXiv预印本arXiv:2410.05243, 2024.

[32] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. Real-time flying object detection with yolov8. arXiv preprint arXiv:2305.09972, 2023.

[32] Dillon Reis, Jordan Kupec, Jacqueline Hong, Ahmad Daoudi. 使用YOLOv8的实时飞行物体检测. arXiv预印本arXiv:2305.09972, 2023.

[33] Bin Xiao, Haiping Wu, Weijian Xu, Xiyang Dai, Houdong Hu, Yumao Lu, Michael Zeng, Ce Liu, and Lu Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4818-4829, 2024.

[33] 肖斌, 吴海平, 徐伟坚, 戴熙洋, 胡浩东, 卢裕茂, Michael Zeng, 刘策, 袁璐. Florence-2: 为多样视觉任务推进统一表征. 在 IEEE/CVF 计算机视觉与模式识别会议论文集, 页 4818-4829, 2024.

[34] Yueqi Song, Frank Xu, Shuyan Zhou, and Graham Neubig. Beyond browsing: Api-based web agents. arXiv preprint arXiv:2410.16464, 2024.

[34] 宋越琦, 徐骏, 周舒妍, Graham Neubig. 超越浏览: 基于 API 的网络代理. arXiv 预印本 arXiv:2410.16464, 2024.

[35] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Hem-ing Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, et al. A survey on in-context learning. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 1107-1128, 2024.

[35] 董庆秀, 李磊, 戴大迈, 郑策, 马靖远, 李锐, 夏和明, 徐婧婧, 吴志勇, 昌宝宝, 等. 关于上下文学习的综述. 在 2024 年 EMNLP 会议论文集, 页 1107-1128, 2024.

[36] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 11048-11064, 2022.

[36] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, Luke Zettlemoyer. 重新思考示例的作用: 是什么使得上下文学习有效? 在 2022 年 EMNLP 会议论文集, 页 11048-11064, 2022.

[37] Chaoyun Zhang, Zicheng Ma, Yuhao Wu, Shilin He, Si Qin, Minghua Ma, Xiaoting Qin, Yu Kang, Yuyi Liang, Xiaoyu Gou, et al. Allhands: Ask me anything on large-scale verbatim feedback via large language models. arXiv preprint arXiv:2403.15157, 2024.

[37] 张朝云, 马子成, 吴宇昊, 何世林, 秦思, 马明华, 秦晓婷, 康宇, 梁玉怡, 苟晓宇, 等. Allhands: 通过大规模逐字反馈基于大语言模型的随问即答. arXiv 预印本 arXiv:2403.15157, 2024.

[38] Man Luo, Xin Xu, Yue Liu, Panupong Pasupat, and Mehran Kazemi. In-context learning with retrieved demonstrations for language models: A survey. Transactions on Machine Learning Research.

[38] 罗曼, 徐昕, 刘岳, Panupong Pasupat, Mehran Kazemi. 使用检索示例的上下文学习: 一项综述. Transactions on Machine Learning Research.

[39] Siyuan Wang, Zhuohan Long, Zhihao Fan, Xuan-Jing Huang, and Zhongyu Wei. Benchmark self-evolving: A multi-agent framework for dynamic llm evaluation. In Proceedings of the 31st International Conference on Computational Linguistics, pages 3310-3328, 2025.

[39] 王思远, 龙卓瀚, 范志豪, 黄轩敬, 魏仲宇. 基准自我进化: 用于动态大模型评估的多智能体框架. 在第 31 届国际计算语言学大会论文集, 页 3310-3328, 2025.

[40] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for

knowledge-intensive nlp tasks. Advances in neural information processing systems, 33:9459-9474, 2020.

[40] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 检索增强生成用于知识密集型 NLP 任务。Advances in neural information processing systems, 33:9459-9474, 2020.

[41] Jun Liu, Chaoyun Zhang, Jiaxu Qian, Minghua Ma, Si Qin, Chetan Bansal, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Large language models can deliver accurate and interpretable time series anomaly detection. arXiv preprint arXiv:2405.15370, 2024.

[41] Jun Liu, Chaoyun Zhang, Jiaxu Qian, Minghua Ma, Si Qin, Chetan Bansal, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 大型语言模型可用于提供准确且可解释的时序异常检测。arXiv preprint arXiv:2405.15370, 2024.

[42] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997, 2, 2023.

[42] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 大型语言模型的检索增强生成: 一项综述。arXiv preprint arXiv:2312.10997, 2, 2023.

[43] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, et al. Xpert: Empowering incident management with query recommendations via large language models. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pages 1-13, 2024.

[43] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, et al. Xpert: 通过大型语言模型的查询推荐赋能事件管理。In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pages 1-13, 2024.

[44] Karissa Miller and Mahmoud Pegah. Virtualization: virtually at the desktop. In Proceedings of the 35th annual ACM SIGUCCS fall conference, pages 255-260, 2007.

[44] Karissa Miller and Mahmoud Pegah. 虚拟化: 桌面上的虚拟体验。In Proceedings of the 35th annual ACM SIGUCCS fall conference, pages 255-260, 2007.

[45] Aditya Venkataraman and Kishore Kumar Jagadeesha. Evaluation of inter-process communication mechanisms. Architecture, 86(64), 2015.

[45] Aditya Venkataraman and Kishore Kumar Jagadeesha. 进程间通信机制的评估。Architecture, 86(64), 2015.

[46] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embed-dings using siamese bert-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3982-3992, 2019.

[46] Nils Reimers and Iryna Gurevych. Sentence-bert: 使用暹罗 BERT 网络的句子嵌入。In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3982-3992, 2019.

[47] Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, and Segev Shlomov. St-webagentbench: A benchmark for evaluating safety and trustworthiness in web agents. arXiv preprint arXiv:2410.06703, 2024.

[47] Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, and Segev Shlomov. St-webagentbench: 用于评估网络代理安全性与可信度的基准。arXiv preprint arXiv:2410.06703, 2024.

[48] Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. Mllm-as-a-judge: Assessing multimodal llm-as-a-judge with vision-language benchmark. In Forty-first International Conference on Machine Learning, 2024.

[48] Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. MLLM-as-a-judge: 用视觉-语言基准评估多模态大型语言模型作为评判者。In Forty-first International Conference on Machine Learning, 2024.

[49] Rogerio Bonatti, Dan Zhao, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Keunho Jang, et al. Windows agent arena: Evaluating multimodal os agents at scale. In NeurIPS 2024 Workshop on Open-World Agents.

[49] Rogerio Bonatti, Dan Zhao, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Keunho Jang, et al. Windows agent arena: 大规模评估多模态操作系统代理。In NeurIPS 2024 Workshop on Open-World Agents.

[50] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. Advances in Neural Information Processing Systems, 37:52040-52094, 2024.

[50] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: 在真实计算环境中为开放式任务评测多模态代理。Advances in Neural Information Processing Systems, 37:52040-52094, 2024.

[51] Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human. In The Thirteenth International Conference on Learning Representations.

[51] Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: 一种开放的代理框架，使计算机像人类一样被使用。In The Thirteenth International Conference on Learning Representations.

[52] Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Guanjing Xiong, and Hongsheng Li. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning, 2025.

[52] Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Guanjing Xiong, and Hongsheng Li. Ui-r1: 通过强化学习增强 GUI 代理的动作预测, 2025.

[53] Linux From Scratch. At-spi - assistive technology service provider interface. https://www.linuxfromscratch.org/blfs/vi
spi.html.Accessed: 2025-03-31.

[53] Linux From Scratch. At-spi - assistive technology service provider interface. https://www.linuxfromscratch.org/blfs/view/5.1/gnome/at-spi.html.Accessed: 2025-03-31.

[54] Apple Inc. Accessibility api. https://developer.apple.com/ documentation/accessibility/accessibility-api, 2025. Accessed: 2025- 03-31.

[54] Apple Inc. 无障碍访问性 API。https://developer.apple.com/ documentation/accessibility/accessibility-api，2025。访问日期:2025-03-31。

[55] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 14281-14290, 2024.

[55] 洪文怡、王伟涵、吕庆松、徐家铮、于文萌、纪俊辉、王妍、王梓涵、董宇霄、丁明等。Cogagent: 面向 GUI 代理的视觉语言模型。载于 IEEE/CVF 计算机视觉与模式识别会议论文集，页 14281-14290, 2024。

[56] Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Song XiXuan, et al. Cogvlm: Visual expert for pretrained language models. Advances in Neural Information Processing Systems, 37:121475-121499, 2024.

[56] 王伟涵、吕庆松、于文萌、洪文怡、齐纪、王妍、纪俊辉、杨卓意、赵磊、席珅等。CogVLM: 为预训练语言模型提供视觉专家。神经信息处理系统进展，37:121475-121499，2024。

[57] Yingqiang Ge, Yujie Ren, Wenyue Hua, Shuyuan Xu, Juntao Tan, and Yongfeng Zhang. Llm as os, agents as apps: Envisioning aios, agents and the aios-agent ecosystem. arXiv preprint arXiv:2312.03815, 2023.

[57] 葛英强、任宇杰、华文岳、许书远、谭军涛、张永锋。LLM 作为操作系统，代理作为应用: 展望 AIOS、代理与 AIOS-代理生态。arXiv 预印本 arXiv:2312.03815，2023。

[58] Kai Mei, Xi Zhu, Wujiang Xu, Wenyue Hua, Mingyu Jin, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. Aios: Llm agent operating system. arXiv preprint arXiv:2403.16971, 2024.

[58] 梅凯、朱熙、徐武江、华文岳、金明宇、李泽龙、许书远、叶若松、葛英强、张永锋。AIOS:LLM 代理操作系统。arXiv 预印本 arXiv:2403.16971，2024。

[59] Balaji Rama, Kai Mei, and Yongfeng Zhang. Cerebrum (aios sdk): A platform for agent development, deployment, distribution, and discovery, 2025.

[59] 巴拉吉·拉玛、梅凯、张永锋。Cerebrum (AIOS SDK): 用于代理开发、部署、分发与发现的平台，2025。

[60] Huilai Chen, Yuanbo Wen, Limin Cheng, Shouxu Kuang, Yumeng Liu, Weijia Li, Ling Li, Rui Zhang, Xinkai Song, Wei Li, et al. Autoos: make your os more powerful by exploiting large language models. In Forty-first International Conference on Machine Learning, 2024.

[60] 陈惠莱、温远博、程利民、匡守旭、刘雨萌、李伟佳、李玲、张睿、宋新凯、李威等。AutoOS: 通过利用大型语言模型增强你的操作系统。在第四十一届国际机器学习会议上，2024。