Large Language Model-Brained GUI Agents: A Survey

大语言模型驱动的图形用户界面代理: 综述

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, Qi Zhang

张超云、何世林、钱佳旭、李博文、李立群、秦思、康宇、马明华、刘谷雨、林庆伟、萨拉万·拉杰莫汉、张冬梅、张琦

Abstract-Graphical User Interfaces (GUIs) have long been central to human-computer interaction, providing an intuitive and visually-driven way to access and interact with digital systems. Traditionally, automating GUI interactions relied on script-based or rule-based approaches, which, while effective for fixed workflows, lacked the flexibility and adaptability required for dynamic, real-world applications. The advent of Large Language Models (LLMs), particularly multimodal models, has ushered in a new era of GUI automation. They have demonstrated exceptional capabilities in natural language understanding, code generation, task generalization, and visual processing. This has paved the way for a new generation of ”LLM-brained” GUI agents capable of interpreting complex GUI elements and autonomously executing actions based on natural language instructions. These agents represent a paradigm shift, enabling users to perform intricate, multi-step tasks through simple conversational commands. Their applications span across web navigation, mobile app interactions, and desktop automation, offering a transformative user experience that revolutionizes how individuals interact with software. This emerging field is rapidly advancing, with significant progress in both research and industry.

摘要—图形用户界面 (Graphical User Interfaces，GUIs) 长期以来一直是人机交互的核心，为访问和与数字系统交互提供了一种直观且可视化驱动的方式。传统上，图形用户界面交互的自动化依赖于基于脚本或基于规则的方法，虽然这些方法对于固定工作流程有效，但缺乏动态现实应用所需的灵活性和适应性。大语言模型 (Large Language Models，LLMs)，尤其是多模态模型的出现，开启了图形用户界面自动化的新纪元。它们在自然语言理解、代码生成、任务泛化和视觉处理方面展现出卓越的能力。这为新一代“由大语言模型驱动”的图形用户界面代理铺平了道路，这些代理能够解释复杂的图形用户界面元素，并根据自然语言指令自主执行操作。这些代理代表了一种范式转变，使用户能够通过简单的对话式命令执行复杂的多步骤任务。它们的应用涵盖网页导航、移动应用交互和桌面自动化，提供了一种变革性的用户体验，彻底改变了个人与软件交互的方式。这个新兴领域正在迅速发展，在研究和行业领域都取得了重大进展。

To provide a structured understanding of this trend, this paper presents a comprehensive survey of LLM-brained GUI agents, exploring their historical evolution, core components, and advanced techniques. We address critical research questions such as existing GUI agent frameworks, the collection and utilization of data for training specialized GUI agents, the development of large action models tailored for GUI tasks, and the evaluation metrics and benchmarks necessary to assess their effectiveness. Additionally, we examine emerging applications powered by these agents. Through a detailed analysis, this survey identifies key research gaps and outlines a roadmap for future advancements in the field. By consolidating foundational knowledge and state-of-the-art developments, this work aims to guide both researchers and practitioners in overcoming challenges and unlocking the full potential of LLM-brained GUI agents. We anticipate that this survey will serve both as a practical cookbook for constructing LLM-powered GUI agents, and as a definitive reference for advancing research in this rapidly evolving domain.

为了对这一趋势进行系统性的理解，本文对由大语言模型驱动的图形用户界面代理进行了全面综述，探讨了它们的历史演变、核心组件和先进技术。我们探讨了关键的研究问题，例如现有的图形用户界面代理框架、用于训练专门图形用户界面代理的数据收集和利用、为图形用户界面任务量身定制的大型动作模型的开发，以及评估其有效性所需的评估指标和基准。此外，我们还研究了由这些代理驱动的新兴应用。通过详细分析，本综述确定了关键的研究空白，并为该领域未来的发展勾勒了路线图。通过整合基础知识和最新发展，这项工作旨在指导研究人员和从业者克服挑战，充分发挥由大语言模型驱动的图形用户界面代理的潜力。我们预计，本综述既可以作为构建由大语言模型驱动的图形用户界面代理的实用指南，也可以作为该快速发展领域推进研究的权威参考。

The collection of papers reviewed in this survey will be hosted and regularly updated on the GitHub repository: https://github.com/vyokky/LLM-Brained-GUI-Agents-Survey Additionally, a searchable webpage is available at https://aka.ms/gui-agent for easier access and exploration.

本综述中所引用论文的集合将托管在 GitHub 仓库中并定期更新:https://github.com/vyokky/LLM-Brained-GUI-Agents-Survey 此外，还可以通过可搜索的网页 https://aka.ms/gui-agent 更方便地访问和探索。

Index Terms-Large Language Model, Graphical User Interface, AI Agent, Automation, Human-Computer Interaction

关键词—大语言模型、图形用户界面、人工智能代理、自动化、人机交互

1 INTRODUCTION

1 引言

Graphical User Interfaces (GUIs) have been a cornerstone of human-computer interaction, fundamentally transforming how users navigate and operate within digital systems [1]. Designed to make computing more intuitive and accessible, GUIs replaced command-line interfaces (CLIs) [2] with visually driven, user-friendly environments. Through the use of icons, buttons, windows, and menus, GUIs empowered a broader range of users to interact with computers using simple actions such as clicks, typing, and gestures. This shift democratized access to computing, allowing even non-technical users to effectively engage with complex systems. However, GUIs often sacrifice efficiency for usability, particularly in workflows requiring repetitive or multi-step interactions, where CLIs can remain more streamlined [3].

图形用户界面 (Graphical User Interfaces，GUIs) 一直是人机交互的基石，从根本上改变了用户在数字系统中的导航和操作方式 [1]。图形用户界面旨在使计算更加直观和易于使用，它用可视化驱动的、用户友好的环境取代了命令行界面 (Command - Line Interfaces，CLIs)[2]。通过使用图标、按钮、窗口和菜单，图形用户界面使更广泛的用户能够通过点击、打字和手势等简单操作与计算机进行交互。这一转变使计算的使用更加普及，即使是非技术用户也能有效地与复杂系统进行交互。然而，图形用户界面通常为了可用性而牺牲了效率，特别是在需要重复或多步骤交互的工作流程中，命令行界面可能仍然更加高效 [3]。

While GUIs revolutionized usability, their design, primarily tailored for human visual interaction, poses significant challenges for automation. The diversity, dynamism, and platform-specific nature of GUI layouts make

it difficult to develop flexible and intelligent automation tools capable of adapting to various environments. Early efforts to automate GUI interactions predominantly relied on script-based or rule-based methods [4], [5]. Although effective for predefined workflows, these methods were inherently narrow in scope, focusing primarily on tasks such as software testing and robotic process automation (RPA) [6]. Their rigidity required frequent manual updates to accommodate new tasks, changes in GUI layouts, or evolving workflows, limiting their scalability and versatility. Moreover, these approaches lacked the sophistication needed to support dynamic, human-like interactions, thereby constraining their applicability in complex or unpredictable scenarios.

虽然图形用户界面彻底改变了可用性，但其主要为人类视觉交互设计的特点给自动化带来了重大挑战。图形用户界面布局的多样性、动态性和特定平台性质使得开发能够适应各种环境的灵活智能自动化工具变得困难。早期实现图形用户界面交互自动化的努力主要依赖于基于脚本或基于规则的方法 [4]、[5]。虽然这些方法对于预定义的工作流程有效，但本质上适用范围狭窄，主要侧重于软件测试和机器人流程自动化 (Robotic Process Automation，RPA) 等任务 [6]。它们的僵化性要求频繁手动更新以适应新任务、图形用户界面布局的变化或不断演变的工作流程，限制了它们的可扩展性和通用性。此外，这些方法缺乏支持动态、类人交互所需的复杂性，从而限制了它们在复杂或不可预测场景中的适用性。

The rise of Large Language Models (LLMs) [8, 9], especially those augmented with multimodal capabilities [10], has emerged as a game changer for GUI automation, redefining the the way agents interact with graphical user interfaces. Beginning with models like ChatGPT [11], LLMs have demonstrated extraordinary proficiency in natural language understanding, code generation, and generalization across diverse tasks [8], 12-14. The integration of visual language models (VLMs) has further extended these capabilities, enabling these models to process visual data, such as the intricate layouts of GUIs [15]. This evolution bridges the gap between linguistic and visual comprehension, empowering intelligent agents to interact with GUIs in a more human-like and adaptive manner. By leveraging these advancements, LLMs and VLMs offer transformative potential, enabling agents to navigate complex digital environments, execute tasks dynamically, and revolutionize the field of GUI automation.

大语言模型 (LLMs)[8, 9] 的兴起，尤其是那些具备多模态能力的模型 [10]，已成为图形用户界面 (GUI) 自动化领域的变革性力量，重新定义了智能体与图形用户界面交互的方式。从 ChatGPT[11] 等模型开始，大语言模型在自然语言理解、代码生成以及跨多种任务的泛化能力方面展现出了非凡的实力 [8, 12 - 14]。视觉语言模型 (VLMs) 的融入进一步拓展了这些能力，使这些模型能够处理视觉数据，例如图形用户界面的复杂布局 [15]。这一发展弥合了语言理解和视觉理解之间的差距，使智能体能够以更接近人类且自适应的方式与图形用户界面进行交互。借助这些进展，大语言模型和视觉语言模型具有变革性潜力，使智能体能够在复杂的数字环境中导航、动态执行任务，并彻底改变图形用户界面自动化领域。

Version: v8 (major update on May 2, 2025)

版本:v8(2025 年 5 月 2 日重大更新)

Chaoyun Zhang, Shilin He, Jiaxu Qian, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang and Qi Zhang are with Microsoft. e-mail: {chaoyun.zhang, shilin.he, v-jiaxuqian, liqun.li, si.gin, yu.kang, minghuama, qlin, saravan.rajmohan, dongmeiz, zhang.qi}@microsoft.com.

张朝云、何仕林、钱佳旭、李立群、秦思、康宇、马明华、林庆伟、萨拉万·拉杰莫汉、张冬梅和张琦就职于微软。电子邮件:{chaoyun.zhang, shilin.he, v - jiaxuqian, liqun.li, si.gin, yu.kang, minghuama, qlin, saravan.rajmohan, dongmeiz, zhang.qi}@microsoft.com。

Bowen Li is with Shanghai Artificial Intelligence Laboratory, China. e-mail: libowen.ne@gmail.com.

李博文就职于中国上海人工智能实验室。电子邮件:libowen.ne@gmail.com。

Guyue Liu is with Peking University, China. e-mail: guyue.liu@gmail.com.For any inquiries or discussions, please contact Chaoyun Zhang and Shilin He.

刘谷雨就职于中国北京大学。电子邮件:guyue.liu@gmail.com。如有任何咨询或讨论，请联系张朝云和何仕林。

1. By LLMs, we refer to the general concept of foundation models capable of accepting various input modalities (e.g., visual language models (VLMs), multimodal LLMs (MLLMs)) while producing output exclusively in textual sequences [7].

1. 我们所说的大语言模型，是指能够接受各种输入模态 (例如视觉语言模型 (VLMs)、多模态大语言模型 (MLLMs))，同时仅以文本序列形式输出的基础模型的通用概念 [7]。

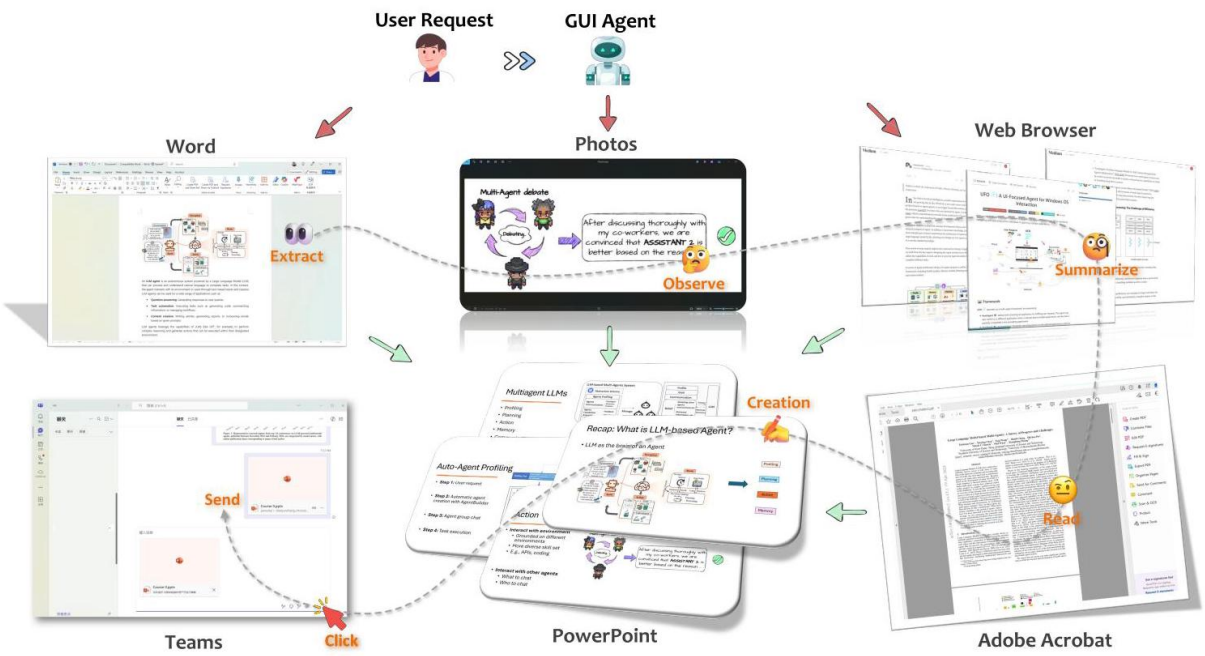


Fig. 1: Illustration of the high-level concept of an LLM-powered GUI agent. The agent receives a user's natural language request and orchestrates actions seamlessly across multiple applications. It extracts information from Word documents, observes content in Photos, summarizes web pages in the browser, reads PDFs in Adobe Acrobat, and creates slides in PowerPoint before sending them through Teams.

图 1: 由大语言模型驱动的图形用户界面智能体的高层概念示意图。该智能体接收用户的自然语言请求，并在多个应用程序之间无缝协调操作。它从 Word 文档中提取信息，查看照片中的内容，总结浏览器中的网页内容，在 Adobe Acrobat 中阅读 PDF 文件，并在 PowerPoint 中创建幻灯片，然后通过 Teams 发送。

1.1 Motivation for LLM-Brained GUI agents

1.1 基于大语言模型的图形用户界面智能体的动机

With an LLM serving as its ”brain”, LLM-powered GUI automation introduces a new class of intelligent agents capable of interpreting a user's natural language requests, analyzing GUI screens and their elements, and autonomously executing appropriate actions. Importantly, these capabilities are achieved without reliance on complex, platform-specific scripts or predefined workflows. These agents, referred to as ”LLM-brained GUI agents”, can be formally defined as:

以大语言模型作为“大脑”的图形用户界面自动化引入了一类新型智能体，它们能够解读用户的自然语言请求、分析图形用户界面屏幕及其元素，并自主执行适当的操作。重要的是，实现这些能力无需依赖复杂的、特定平台的脚本或预定义的工作流程。这些智能体，即“基于大语言模型的图形用户界面智能体”，可以正式定义为:

Intelligent agents that operate within GUI environments, leveraging LLMs as their core inference and cognitive engine to generate, plan, and execute actions in a flexible and adaptive manner.

在图形用户界面环境中运行的智能体，利用大语言模型作为其核心推理和认知引擎，以灵活和自适应的方式生成、规划和执行操作。

This paradigm represents a transformative leap in GUI automation, fostering dynamic, human-like interactions across diverse platforms. It enables the creation of intelligent, adaptive systems that can reason, make decisions in real-time, and respond flexibly to evolving tasks and environments. We illustrate this high-level concept in Figure 1

这种范式代表了图形用户界面自动化领域的一次变革性飞跃，促进了跨不同平台的动态、类人交互。它使得能够创建智能、自适应的系统，这些系统可以进行推理、实时决策，并灵活应对不断变化的任务和环境。我们在图 1 中展示了这一高层概念。

Traditional GUI automation are often limited by predefined rules or narrowly focused on specific tasks, constraining their ability to adapt to dynamic environments and diverse applications. In contrast, LLM-powered GUI agents bring a paradigm shift by integrating natural language understanding, visual recognition, and decision-making into a unified framework. This enables them to generalize across a wide range of use cases, transforming task automation and significantly enhancing the intuitiveness and efficiency of human-computer interaction. Moreover, unlike the emerging trend of pure Application Programming Interface (API)-based agents—which depend on APIs that may not always be exposed or accessible-GUI agents leverage the universal nature of graphical interfaces. GUIs offer a general mechanism to control most software applications, enabling agents to operate in a nonintrusive manner without requiring internal API access. This capability not only broadens the applicability of GUI agents

but also empowers external developers to build advanced functionality on top of existing software across diverse platforms and ecosystems. Together, these innovations position GUI agents as a versatile and transformative technology for the future of intelligent automation.

传统的图形用户界面自动化通常受限于预定义规则，或者仅专注于特定任务，这限制了它们适应动态环境和多样化应用的能力。相比之下，基于大语言模型的图形用户界面智能体通过将自然语言理解、视觉识别和决策制定集成到一个统一的框架中，带来了范式转变。这使它们能够在广泛的用例中进行泛化，改变任务自动化方式，并显著提高人机交互的直观性和效率。此外，与新兴的纯基于应用程序编程接口 (API) 的智能体趋势不同——这类智能体依赖的 API 可能并不总是公开或可访问的——图形用户界面智能体利用了图形界面的通用性。图形用户界面提供了一种通用机制来控制大多数软件应用程序，使智能体能够以非侵入性的方式运行，而无需访问内部 API。这种能力不仅拓宽了图形用户界面智能体的适用性，还使外部开发人员能够在不同平台和生态系统的现有软件基础上构建高级功能。总之，这些创新使图形用户界面智能体成为未来智能自动化领域一种通用且具有变革性的技术。

This new paradigm enables users to control general software systems with conversational commands [16]. By reducing the cognitive load of multi-step GUI operations, LLM-powered agents make complex systems accessible to non-technical users and streamline workflows across diverse domains. Notable examples include SeeAct [17] for web navigation, AppAgent [18] for mobile interactions, and UFO [19] for Windows OS applications. These agents resemble a ”virtual assistant” [20] akin to J.A.R.V.I.S. from Iron Man-an intuitive, adaptive system capable of understanding user goals and autonomously performing actions across applications. The futuristic concept of an AI-powered operating system that executes cross-application tasks with fluidity and precision is rapidly becoming a reality [21], [22].

这种新范式使用户能够通过对话式命令控制通用软件系统 [16]。通过减轻多步骤图形用户界面 (GUI) 操作的认知负担，由大语言模型 (LLM) 驱动的智能体使非技术用户也能使用复杂系统，并简化了不同领域的工作流程。值得注意的例子包括用于网页导航的 SeeAct [17]、用于移动交互的 AppAgent [18] 以及用于 Windows 操作系统应用程序的 UFO [19]。这些智能体类似于《钢铁侠》中的贾维斯 (J.A.R.V.I.S.) 那样的“虚拟助手”[20] ——一种直观、自适应的系统，能够理解用户目标并自动跨应用程序执行操作。由人工智能驱动的操作系统能够流畅、精确地执行跨应用程序任务的未来概念正在迅速成为现实 [21]、[22]。

Real-world applications of LLM-powered GUI agents are already emerging. For example, Microsoft Power Automate utilizes LLMs to streamline low-code/no-code automation [2] . allowing users to design workflows across Microsoft applications with minimal technical expertise. Integrated AI assistants in productivity software, like Microsoft Copilot [3] are bridging the gap between natural language instructions and operations on application. Additionally, LLM-powered agents show promise for enhancing accessibility [23], potentially allowing visually impaired users to navigate GUIs more effectively by converting natural language commands into executable steps. These developments underscore the timeliness and transformative potential of LLM-powered GUI agents across diverse applications.

由大语言模型驱动的图形用户界面智能体在现实世界中的应用已经开始出现。例如，微软 Power Automate 利用大语言模型简化低代码/无代码自动化 [2]，使用户只需具备最少的技术知识就能设计跨微软应用程序的工作流程。生产力软件中的集成式人工智能助手，如微软 Copilot [3]，正在弥合自然语言指令与应用程序操作之间的差距。此外，由大语言模型驱动的智能体在提高可访问性方面显示出潜力 [23]，有可能通过将自然语言命令转换为可执行步骤，让视障用户更有效地浏览图形用户界面。这些发展凸显了由大语言模型驱动的图形用户界面智能体在不同应用中的及时性和变革潜力。

The convergence of LLMs and GUI automation addresses longstanding challenges in human-computer interaction and introduces new opportunities for intelligent GUI control [24]. This integration has catalyzed a surge in research activity, spanning application frameworks [19], data collection [25], model optimization [15], and evaluation benchmarks [26]. Despite these advancements, key challenges and limitations persist, and many foundational questions remain unexplored. However, a systematic review of this rapidly evolving area is notably absent, leaving a critical gap in understanding.

大语言模型与图形用户界面自动化的融合解决了人机交互中长期存在的挑战，并为智能图形用户界面控制带来了新机遇 [24]。这种融合促使研究活动激增，涵盖了应用框架 [19]、数据收集 [25]、模型优化 [15] 和评估基准 [26] 等方面。尽管取得了这些进展，但关键挑战和局限性仍然存在，许多基础性问题仍未得到探索。然而，对这一快速发展领域的系统综述明显缺失，这在理解上造成了重大空白。

1.2 Scope of the Survey

1.2 调查范围

To address this gap, this paper provides a pioneering, comprehensive survey of LLM-brained GUI agents. We cover the historical evolution of GUI agents, provide a step-by-step guide to building these agents, summarize essential and advanced techniques, review notable tools and research related to frameworks, data and models, showcase representative applications, and outline future directions. Specifically, this survey aims to answer the following research questions (RQs):

为了填补这一空白，本文对由大语言模型驱动的图形用户界面智能体进行了开创性的全面调查。我们涵盖了图形用户界面智能体的历史演变，提供了构建这些智能体的分步指南，总结了基本和高级技术，回顾了与框架、数据和模型相关的重要工具和研究，展示了代表性应用，并概述了未来方向。具体而言，本次调查旨在回答以下研究问题 (RQs):

1) RQ1: What is the historical development trajectory of LLM-powered GUI agents? (Section 4)

1) 研究问题 1(RQ1): 由大语言模型驱动的图形用户界面智能体的历史发展轨迹是怎样的？(第 4 节)

2) RQ2: What are the essential components and advanced technologies that form the foundation of LLM-brained GUI agents? (Section 5)

2) 研究问题 2(RQ2): 构成由大语言模型驱动的图形用户界面智能体基础的基本组件和先进技术有哪些？(第 5 节)

3) RQ3: What are the principal frameworks for LLM GUI agents, and what are their defining characteristics? (Section 6)

3) 研究问题 3(RQ3): 大语言模型图形用户界面智能体的主要框架有哪些，它们的定义特征是什么？(第 6 节)

4) RQ4: What are the existing datasets, and how can comprehensive datasets be collected to train optimized LLMs for GUI agents? (Section 7)

4) 研究问题 4(RQ4): 现有的数据集有哪些，如何收集全面的数据集来训练适用于图形用户界面智能体的优化大语言模型？(第 7 节)

5) RQ5: How can the collected data be used to train purpose-built Large Action Models (LAMs) for GUI agents, and what are the current leading models in the field? (Section 8)

5) 研究问题 5(RQ5): 如何利用收集到的数据为图形用户界面智能体训练专用的大型动作模型 (LAMs)，该领域目前的领先模型有哪些？(第 8 节)

6) RQ6: What metrics and benchmarks are used to evaluate the capability and performance of GUI agents? (Section 9)

6) 研究问题 6(RQ6): 用于评估图形用户界面智能体能力和性能的指标和基准有哪些？(第 9 节)

7) RQ7: What are the most significant real-world applications of LLM-powered GUI agents, and how have they been adapted for practical use? (Section 10)

7) 研究问题 7(RQ7): 由大语言模型驱动的图形用户界面智能体在现实世界中最重要的应用有哪些，它们是如何适应实际应用的？(第 10 节)

8) RQ8: What are the major challenges, limitations, and future research directions for developing robust and intelligent GUI agents? (Section 11)

8) 研究问题 8(RQ8): 开发强大而智能的图形用户界面智能体面临的主要挑战、局限性和未来研究方向有哪些？(第 11 节)

Through these questions, this survey aims to provide a comprehensive overview of the current state of the field, offer a guide for building LLM-brained GUI agents, identify key research gaps, and propose directions for future work. This survey is one of the pioneers to systematically examine the domain of LLM-brained GUI agents, integrating perspectives from LLM advancements, GUI automation, and human-computer interaction.

通过这些问题，本次调查旨在全面概述该领域的当前状态，为构建由大语言模型驱动的图形用户界面智能体提供指南，确定关键研究空白，并提出未来工作的方向。本次调查是系统研究由大语言模型驱动的图形用户界面智能体领域的先驱之一，整合了大语言模型进展、图形用户界面自动化和人机交互等方面的观点。

1.3 Survey Structure

1.3 调查结构

The survey is organized as follows, with a structural illustration provided in Figure 2. Section 2 reviews related survey and review literature on LLM agents and GUI automation. Section 3 provides preliminary background on LLMs, LLM agents, and GUI automation. Section 2 traces the evolution of LLM-powered GUI agents. Section 5 introduces key components and advanced technologies within LLM-powered GUI agents, serving as a comprehensive guide. Section 6 presents representative frameworks for LLM-powered GUI agents. Section 7 discusses dataset collection and related data-centric research for optimizing LLMs in GUI agent. Section 8 covers foundational and optimized models for GUI agents. Section 9 outlines evaluation metrics and benchmarks. Section 10 explores real-world applications and use cases. Finally, Section 11 examines current limitations, challenges, and potential future directions, and section 12 conclude this survey. For clarity, a list of abbreviations is provided in Table 1

本次调查的组织如下，图 2 提供了结构示意图。第 2 节回顾了关于大语言模型智能体和图形用户界面自动化的相关调查和综述文献。第 3 节提供了大语言模型、大语言模型智能体和图形用户界面自动化的初步背景知识。第 4 节追溯了由大语言模型驱动的图形用户界面智能体的演变。第 5 节介绍了由大语言模型驱动的图形用户界面智能体中的关键组件和先进技术，作为全面指南。第 6 节介绍了由大语言模型驱动的图形用户界面智能体的代表性框架。第 7 节讨论了数据集收集以及以数据为中心的相关研究，以优化图形用户界面智能体中的大语言模型。第 8 节涵盖了图形用户界面智能体的基础模型和优化模型。第 9 节概述了评估指标和基准。第 10 节探索了现实世界中的应用和用例。最后，第 11 节探讨了当前的局限性、挑战和潜在的未来方向，第 12 节对本次调查进行总结。为清晰起见，表 1 提供了缩略语列表

2 RELATED WORK

2 相关工作

The integration of LLMs with GUI agents is an emerging and rapidly growing field of research. Several related surveys and tutorials provide foundational insights and guidance. We provide a brief review of existing overview articles on GUI automation and LLM agents, as these topics closely relate to and inform our research focus. To begin, we provide an overview of representative surveys and books on GUI automation, LLM agents, and their integration, as summarized in Table 2 These works either directly tackle one or two core areas in GUI automation and LLM-driven agents, or provide valuable insights that, while not directly addressing the topic, contribute indirectly to advancing the field. GUI agents, application UI screenshots are equally essential, serving as key inputs for reliable task comprehension and execution.

大语言模型 (LLM) 与图形用户界面 (GUI) 代理的集成是一个新兴且快速发展的研究领域。一些相关的综述和教程提供了基础见解和指导。我们简要回顾了现有的关于 GUI 自动化和大语言模型代理的概述文章，因为这些主题与我们的研究重点密切相关，并为其提供了参考。首先，我们概述了关于 GUI 自动化、大语言模型代理及其集成的代表性综述和书籍，如表 2 所示。这些工作要么直接涉及 GUI 自动化和大语言模型驱动代理的一两个核心领域，要么提供了有价值的见解，虽然没有直接涉及该主题，但间接推动了该领域的发展。对于 GUI 代理来说，应用程序的用户界面截图同样至关重要，它们是可靠地理解和执行任务的关键输入。

---

2. https://www.microsoft.com/en-us/power-platform/blog/

2. https://www.microsoft.com/en-us/power-platform/blog/

power-automate/revolutionize-the-way-you-work-with-automation-and-ai/ 3. https://copilot.microsoft.com/

power-automate/revolutionize-the-way-you-work-with-automation-and-ai/ 3. https://copilot.microsoft.com/

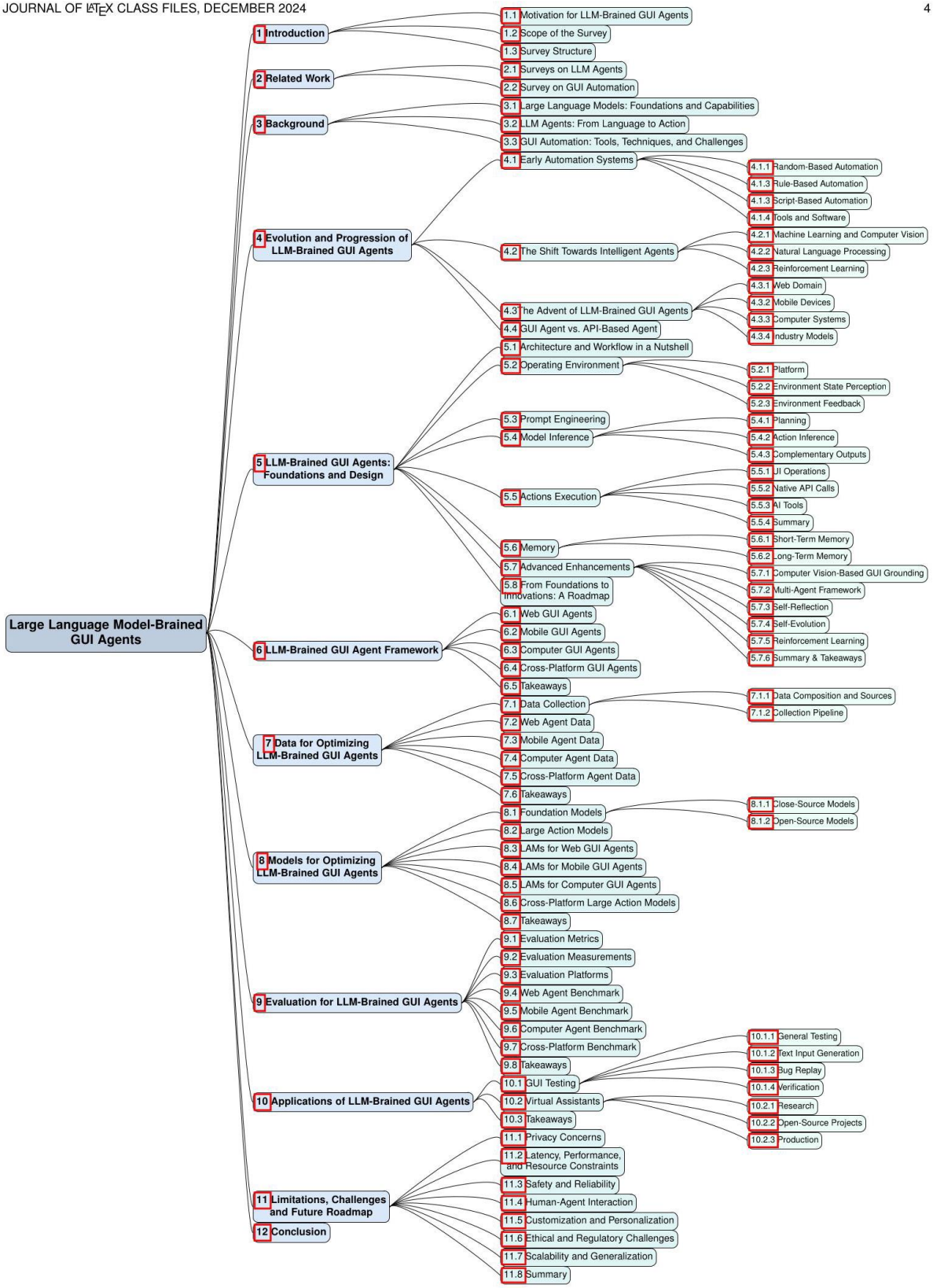


Fig. 2: The structure of the survey on LLM-brained GUI agents.

图 2: 基于大语言模型的 GUI 代理综述的结构。

TABLE 1: List of abbreviations in alphabetical order.

表 1: 按字母顺序排列的缩写列表。

Acronym	Explanation
AI	Artificial Intelligence
AITW	Android in the Wild
AITZ	Android in The Zoo
API	Application Programming Interface
CLI	Command-Line Interface
CLIP	Contrastive Language-Image Pre-Training
CoT	Chain-of-Thought
CSS	Cascading Style Sheets
CUA	Computer-Using Agent
CuP	Completion under Policy
CV	Computer Vision
DOM	Document Object Model
DPO	Direct Preference Optimization
GCC	General Computer Control
GPT	Generative Pre-trained Transformers
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HTML	Hypertext Markup Language
ICL	In-Context Learning
IoU	Intersection over Union
LAM	Large Action Model
LLM	Large Language Model
LSTM	Long Short-Term Memory
LTM	Long-Term Memory
MCTS	Monte Carlo Tree Search
MoE	Mixture of Experts
MDP	Markov Decision Process
MLLM	Multimodal Large Language Model
OCR	Optical Character Recognition
OS	Operation System
RAG	Retrieval-Augmented Generation
ReAct	Reasoning and Acting
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
RPA	Robotic Process Automation
UI	User Interface
UX	User Experience
VAB	VisualAgentBench
VLM	Visual Language Models
ViT	Vision Transformer
VQA	Visual Question Answering
SAM	Segment Anything Model
SoM	Set-of-Mark
STM	Short-Trem Memory

首字母缩写词	解释
人工智能 (AI)	人工智能 (Artificial Intelligence)
野生安卓 (AITW)	野生安卓环境 (Android in the Wild)
动物园安卓 (AITZ)	受控安卓环境 (Android in The Zoo)
应用程序编程接口 (API)	应用程序编程接口 (Application Programming Interface)
命令行界面 (CLI)	命令行界面 (Command-Line Interface)
对比语言 - 图像预训练 (CLIP)	对比语言 - 图像预训练 (Contrastive Language-Image Pre-Training)
思维链 (CoT)	思维链 (Chain-of-Thought)
层叠样式表 (CSS)	层叠样式表 (Cascading Style Sheets)
计算机使用代理 (CUA)	计算机使用代理 (Computer-Using Agent)
策略下完成 (CuP)	策略下完成 (Completion under Policy)
计算机视觉 (CV)	计算机视觉 (Computer Vision)
文档对象模型 (DOM)	文档对象模型 (Document Object Model)
直接偏好优化 (DPO)	直接偏好优化 (Direct Preference Optimization)
通用计算机控制 (GCC)	通用计算机控制 (General Computer Control)
生成式预训练变换器 (GPT)	生成式预训练变换器 (Generative Pre-trained Transformers)
图形用户界面 (GUI)	图形用户界面 (Graphical User Interface)
人机交互 (HCI)	人机交互 (Human-Computer Interaction)
超文本标记语言 (HTML)	超文本标记语言 (Hypertext Markup Language)
上下文学习 (ICL)	上下文学习 (In-Context Learning)
交并比 (IoU)	交并比 (Intersection over Union)
大型动作模型 (LAM)	大型动作模型 (Large Action Model)
大语言模型 (LLM)	大语言模型 (Large Language Model)
长短期记忆网络 (LSTM)	长短期记忆 (Long Short-Term Memory)
长期记忆 (LTM)	长期记忆 (Long-Term Memory)
蒙特卡罗树搜索 (MCTS)	蒙特卡罗树搜索 (Monte Carlo Tree Search)
专家混合模型 (MoE)	专家混合模型 (Mixture of Experts)
马尔可夫决策过程 (MDP)	马尔可夫决策过程 (Markov Decision Process)
多模态大语言模型 (MLLM)	多模态大语言模型 (Multimodal Large Language Model)
光学字符识别 (OCR)	光学字符识别 (Optical Character Recognition)
操作系统 (OS)	操作系统 (Operation System)
检索增强生成 (RAG)	检索增强生成 (Retrieval-Augmented Generation)
反应式行动 (ReAct)	推理与行动 (Reasoning and Acting)
强化学习 (RL)	强化学习 (Reinforcement Learning)
基于人类反馈的强化学习 (RLHF)	基于人类反馈的强化学习 (Reinforcement Learning from Human Feedback)
循环神经网络 (RNN)	循环神经网络 (Recurrent Neural Network)
机器人流程自动化 (RPA)	机器人流程自动化 (Robotic Process Automation)
用户界面 (UI)	用户界面 (User Interface)
用户体验 (UX)	用户体验 (User Experience)
视觉智能体基准测试 (VAB)	视觉智能体基准测试 (VisualAgentBench)
视觉语言模型 (VLM)	视觉语言模型 (Visual Language Models)
视觉变换器 (ViT)	视觉变换器 (Vision Transformer)
视觉问答 (VQA)	视觉问答 (Visual Question Answering)
任意分割模型 (SAM)	任意分割模型 (Segment Anything Model)
标记集 (SoM)	标记集 (Set-of-Mark)
短期记忆 (STM)	短期记忆 (Short-Trem Memory)

2.1 Survey on GUI Automation

2.1 GUI 自动化综述

GUI automation has a long history and wide applications in industry, especially in GUI testing [27]-[29] and RPA [6] for task automation [42].

GUI 自动化在工业界有着悠久的历史和广泛的应用，尤其是在 GUI 测试 [27]-[29] 和用于任务自动化的机器人流程自动化 (RPA)[6][42] 中。

Said et al., [30] provide an overview of GUI testing for mobile applications, covering objectives, approaches, and challenges within this domain. Focusing on Android applications, Li [31] narrows the scope further, while Oksanen et al., 32 explore automatic testing techniques for Windows GUI applications, a key platform for agent operations. Similarly, Moura et al., 72 review GUI testing for web applications, which involves diverse tools, inputs, and methodologies. Deshmukh et al., 33 discuss automated GUI testing for enhancing user experience, an area where LLMs also bring new capabilities. A cornerstone of modern GUI testing is computer vision (CV), which is used to interpret UI elements and identify actionable controls [34]. Yu et al., [35] survey the application of CV in mobile GUI testing, highlighting both its significance and associated challenges. In LLM-powered

Said 等人 [30] 提供了移动应用 GUI 测试的概述，涵盖了该领域的目标、方法和挑战。Li[31] 聚焦于 Android 应用，进一步缩小了研究范围，而 Oksanen 等人 [32] 探讨了 Windows GUI 应用的自动测试技术，Windows 是代理操作的重要平台。同样，Moura 等人 [72] 回顾了 Web 应用的 GUI 测试，涉及多样的工具、输入和方法。Deshmukh 等人 [33] 讨论了自动化 GUI 测试以提升用户体验的研究领域，LLM(大型语言模型) 在此也带来了新能力。现代 GUI 测试的基石是计算机视觉 (CV)，用于解析用户界面元素并识别可操作控件 [34]。Yu 等人 [35] 综述了 CV 在移动 GUI 测试中的应用，强调了其重要性及相关挑战。在 LLM 驱动的

On the other hand, RPA, which focuses on automating repetitive human tasks, also relies heavily on GUI automation for relevant processes. Syed et al., [36] review this field and highlight contemporary RPA themes, identifying key challenges for future research. Chakraborti et al., 37 emphasize the importance of shifting from traditional, script-based RPA toward more intelligent, adaptive paradigms, offering a systematic overview of advancements in this direction. Given RPA's extensive industrial applications, Enriquez et al., [38] and Ribeiro et al., 39 survey the field from an industrial perspective, underscoring its significance and providing a comprehensive overview of RPA methods, development trends, and practical challenges.

另一方面，RPA 专注于自动化重复的人类任务，也高度依赖 GUI 自动化来实现相关流程。Syed 等人 [36] 回顾了该领域，强调了当代 RPA 的主题，并指出未来研究的关键挑战。Chakraborti 等人 [37] 强调了从传统脚本驱动的 RPA 向更智能、适应性强的范式转变的重要性，系统性地概述了该方向的进展。鉴于 RPA 在工业中的广泛应用，Enriquez 等人 [38] 和 Ribeiro 等人 [39] 从工业视角对该领域进行了综述，强调其重要性并全面介绍了 RPA 的方法、发展趋势和实际挑战。

Both GUI testing [40] and RPA [41] continue to face significant challenges in achieving greater intelligence and robustness. LLM-powered GUI agents are poised to play a transformative role in these fields, providing enhanced capabilities and adding substantial value to address these persistent issues.

GUI 测试 [40] 和 RPA[41] 在实现更高智能化和鲁棒性方面仍面临重大挑战。LLM 驱动的 GUI 代理有望在这些领域发挥变革性作用，提供增强的能力并为解决这些持续存在的问题带来显著价值。

2.2 Surveys on LLM Agents

2.2 LLM 代理综述

The advent of LLMs has significantly enhanced the capabilities of intelligent agents [43], enabling them to tackle complex tasks previously out of reach, particularly those involving natural language understanding and code generation [44]. This advancement has spurred substantial research into LLM-based agents designed for a wide array of applications [45].

大型语言模型 (LLM) 的出现显著提升了智能代理的能力 [43]，使其能够处理此前难以企及的复杂任务，尤其是涉及自然语言理解和代码生成的任务 [44]。这一进展推动了针对广泛应用设计的基于 LLM 的代理的深入研究 [45]。

Both Xie et al., [46] and Wang et al., [47] offer comprehensive surveys on LLM-powered agents, covering essential background information, detailed component breakdowns, taxonomies, and various applications. These surveys serve as valuable references for a foundational understanding of LLM-driven agents, laying the groundwork for further exploration into LLM-based GUI agents. Xie et al., [59] provide an extensive overview of multimodal agents, which can process images, videos, and audio in addition to text. This multimodal capability significantly broadens the scope beyond traditional text-based agents [60]. Notably, most GUI agents fall under this category, as they rely on image inputs, such as screenshots, to interpret and interact with graphical interfaces effectively. Multi-agent frameworks are frequently employed in the design of GUI agents to enhance their capabilities and scalability. Surveys by Guo et al., [48] and Han et al., [49] provide comprehensive overviews of the current landscape, challenges, and future directions in this area. Sun et al., [50] provide an overview of recent methods that leverage reinforcement learning to strengthen multi-agent LLM systems, opening new pathways for enhancing their capabilities and adaptability. These surveys offer valuable insights and guidance for designing effective multi-agent systems within GUI agent frameworks.

Xie 等人 [46] 和 Wang 等人 [47] 均提供了关于 LLM 驱动代理的全面综述，涵盖了基础背景、详细组件解析、分类体系及多种应用。这些综述为理解 LLM 驱动代理奠定了基础，促进了对基于 LLM 的 GUI 代理的进一步探索。Xie 等人 [59] 还对多模态代理进行了广泛介绍，这类代理除了文本外还能处理图像、视频和音频。多模态能力显著拓宽了传统文本代理的应用范围 [60]。值得注意的是，大多数 GUI 代理属于此类，因为它们依赖图像输入 (如截图) 来有效解析和交互图形界面。多代理框架常被用于 GUI 代理的设计，以增强其能力和可扩展性。Guo 等人 [48] 和 Han 等人 [49] 的综述全面介绍了该领域的现状、挑战及未来方向。Sun 等人 [50] 则概述了利用强化学习强化多代理 LLM 系统的最新方法，为提升其能力和适应性开辟了新途径。这些综述为设计高效的多代理系统提供了宝贵的见解和指导。

In the realm of digital environments, Wu et al., [61] presents a survey on LLM agents operating in mobile environments, covering key aspects of mobile GUI agents. In a boarder scope, Wang et al., [62] present a survey on the integration of foundation models with GUI agents. Another survey by Gao et al., provides an overview of autonomous

在数字环境领域，Wu 等人 [61] 对移动环境中运行的 LLM 代理进行了综述，涵盖了移动 GUI 代理的关键方面。更广泛地，Wang 等人 [62] 综述了基础模型与 GUI 代理的整合。Gao 等人则提供了跨多种数字平台自主代理的概览

JOURNAL OF LATEX CLASS FILES, DECEMBER 2024 agents operating across various digital platforms [63], highlighting their capabilities, challenges, and applications. All these surveys highlighting emerging trends in this area.

《JOURNAL OF LATEX CLASS FILES, DECEMBER 2024》报道了这些代理的能力、挑战和应用。所有这些综述均强调了该领域的新兴趋势。

TABLE 2: Summary of representative surveys and books on GUI automation and LLM agents. A √symbol indicates that a publication explicitly addresses a given domain, while an ○ symbol signifies that the publication does not focus on the area but offers relevant insights. Publications covering both GUI automation and LLM agents are highlighted for emphasis.

表 2:GUI 自动化和 LLM 代理代表性综述及著作汇总。√ 符号表示该出版物明确涉及相关领域，○ 符号表示该出版物虽未专注于该领域但提供了相关见解。涵盖 GUI 自动化和 LLM 代理两者的出版物以加粗形式突出显示。

Survey	One Sentence Summary	Scope		
		GUI Automation	LLM Agent	LLM Agent + GUI Automation
Li et al., 27]	A book on how to develop an automated GUI testing tool.	✓		
Rodríguez et al., 28	A survey on automated GUI testing in 30 years.	✓		
Arnatovich et al., 29	A survey on automated techniques for mobile functional GUI testing.	✓		
Ivančić et al., [6]	A literature review on RPA.	✓		
Said et al., 30	An overview on mobile GUI testing.	✓		
Li 31	An survey on Android GUI testing.	✓		
Oksanen et al., 32	GUI testing on Windows OS.	✓		
Deshmukh et al., 33	A survey on GUI testing for improving user experience.	✓		
Bajammal et al., 34	A survey on the use of computer vision for software engineering.	✓		
Yu et al., 35	A survey on using computer for mobile app GUI testing.	✓		
Syed et al., 36	A review of contemporary themes and challenges in RPA.	✓		
Chakraborti et al., 37	A review of emerging trends of intelligent process automation.	✓		
Enriquez et al., 38	A scientific and industrial systematic mapping study of RPA.	✓		
Ribeiro et al., 39	A review of combining AI and RPA in industry 4.0.	✓		
Nass et al., 40	Discuss the chanllenges of GUI testing.	✓		
Agostinelli et al., 41	Discuss the research challenges of intelligent RPA.	✓		
Wali et al., 42]	A review on task automation with intelligent agents.	✓		
Zhao et al., 8	A comprehensive survey of LLMs.		✓	
Zhao et al., 44	A survey of LLM-based agents.		✓	
Cheng et al., 44	An overview of LLM-based AI agent.		✓	
Li et al., 45	A survey on personal LLM agents on their capability, efficiency and security.		✓	
Xie et al., [46]	A comprehensive survey of LLM-based agents.		✓	
Wang et al.,. 47	A survey on LLM-based autonomous agents.		✓	
Guo et al., 48	A survey of mult-agent LLM frameworks.		✓	
Han et al., 49	A survey on LLM multi-agent systems, with their challenges and open problems.		✓	
Sun et al., 50	A survey on LLM-based multi-agent reinforcement learning.		✓	
Huang et al., 51]	A survey on planning in LLM agents.		✓	
Aghzal et al., 52	A survey on automated planning in LLMs.		✓	
Zheng et al., 53	Discuss the roadmap of lifelong learning in LLM agents.		✓	
Zhang et al., 54	A survey on the memory of LLM-based agents.		✓	
Shen 13	A survey of the tool usage in LLM agents.		✓	
Chang et al., 55	A survey on evaluation of LLMs.		✓	
Li et al., 56	A survey on benchmarks multimodal applications.		✓	
Li et al., 57	A survey on benchmarking evaluations, applications, and challenges of visual LLMs.		✓	
Huang and Zhang 58	A survey on evaluation of multimodal LLMs.		✓	
Xie et al., 69	A survey on LLM based multimodal agent.		✓	□
Durante et al., 60	A survey of multimodal interaction with AI agents.		✓	0
Wu et al., 611	A survey of foundations and trend on multimodal mobile agents.		✓	✓
Wang et al., 62	A survey on the integration of foundation models with GUI agents.		✓	✓
Gao et al., 63	A survey on autonomous agents across digital platforms.		✓	✓
Dang et al., 64	A survey on GUI agents.		✓	✓
Liu et al., 65	A survey on GUI agent on phone automation.		✓	✓
Hu et al., 66	A survey on MLLM based agents for OS.		✓	✓
Shi et al., 167	A survey of building trustworthy GUI agents.		✓	✓
Ning et al., 68	A survey of agents for Web automation.		✓	✓
Tang et al., 69	A survey of GUI agents powered by (multimodal) LLMs.		✓	✓
Li and Huang et al., 70	A summary of GUI agents powered by foundation models and enhanced through reinforcement learning		✓	✓
Sager et al., 71	A review of AI agent for computer use.	O	✓	✓
Our work	A comprehensive survey on LLM-brained GUI agents, on their foundations, technologies, frameworks, data, models, applications, challenges and future roadmap.	✓	✓	

调查	一句话总结	范围		
		图形用户界面自动化 (GUI Automation)	大语言模型智能体 (LLM Agent)	大语言模型智能体 + 图形用户界面自动化 (LLM Agent + GUI Automation)
李等人，[27]	一本关于如何开发自动化图形用户界面测试工具的书。	✓		
罗德里格斯等人，[28]	一项关于 30 年自动化图形用户界面测试的调查。	✓		
阿尔纳托维奇等人，[29]	一项关于移动功能图形用户界面测试自动化技术的调查。	✓		
伊万契奇等人，[6]	一篇关于机器人流程自动化 (RPA) 的文献综述。	✓		
赛义德等人，[30]	一篇关于移动图形用户界面测试的概述。	✓		
李，[31]	一项关于安卓图形用户界面测试的调查。	✓		
奥克萨宁等人，[32]	Windows 操作系统上的图形用户界面测试。	✓		
德什穆克等人，[33]	一项关于为改善用户体验进行图形用户界面测试的调查。	✓		
巴贾马尔等人，[34]	一项关于计算机视觉在软件工程中应用的调查。	✓		
余等人，[35]	一项关于使用计算机进行移动应用图形用户界面测试的调查。	✓		
赛义德等人，[36]	一篇关于机器人流程自动化 (RPA) 当代主题和挑战的综述。	✓		
查克拉博蒂等人，[37]	一篇关于智能流程自动化新兴趋势的综述。	✓		
恩里克兹等人，[38]	一篇关于机器人流程自动化 (RPA) 的科学与工业系统映射研究。	✓		
里贝罗等人，[39]	一篇关于工业 4.0 中人工智能与机器人流程自动化 (RPA) 结合的综述。	✓		
纳斯等人，[40]	讨论图形用户界面测试的挑战。	✓		
阿戈斯蒂内利等人，[41]	讨论智能机器人流程自动化 (RPA) 的研究挑战。	✓		
瓦利等人，[42]	一篇关于使用智能体进行任务自动化的综述。	✓		
赵等人，[8]	一项关于大语言模型 (LLMs) 的全面调查。		✓	
赵等人，[44]	一项关于基于大语言模型 (LLM) 的智能体的调查。		✓	
程等人，[44]	一篇关于基于大语言模型 (LLM) 的人工智能智能体的概述。		✓	
李等人，[45]	一项关于个人大语言模型 (LLM) 智能体的能力、效率和安全性的调查。		✓	
谢等人，[46]	一项关于基于大语言模型 (LLM) 的智能体的全面调查。		✓	
王等人，[47]	一项关于基于大语言模型 (LLM) 的自主智能体的调查。		✓	
郭等人，[48]	一项关于多智能体大语言模型 (LLM) 框架的调查。		✓	
韩等人，[49]	一项关于大语言模型 (LLM) 多智能体系统及其挑战和开放性问题的调查。		✓	
孙等人，[50]	一项关于基于大语言模型 (LLM) 的多智能体强化学习的调查。		✓	
黄等人，[51]	一项关于大语言模型 (LLM) 智能体规划的调查。		✓	
阿格扎尔等人，[52]	一项关于大语言模型 (LLMs) 中自动化规划的调查。		✓	
郑等人，[53]	讨论大语言模型 (LLM) 智能体的终身学习路线图。		✓	
张等人，54	基于大语言模型的智能体的记忆研究综述。		✓	
沈 13	大语言模型智能体的工具使用研究综述。		✓	
常等人，55	大语言模型评估研究综述。		✓	
李等人，56	多模态应用基准测试研究综述。		✓	
李等人，57	视觉大语言模型的基准评估、应用和挑战研究综述。		✓	
黄和张 58	多模态大语言模型评估研究综述。		✓	
谢等人，69	基于大语言模型的多模态智能体研究综述。		✓	□
杜兰特等人，60	与人工智能智能体的多模态交互研究综述。		✓	0
吴等人，611	多模态移动智能体的基础和趋势研究综述。		✓	✓
王等人，62	基础模型与图形用户界面 (GUI) 智能体集成研究综述。		✓	✓
高等人，63	跨数字平台的自主智能体研究综述。		✓	✓
党等人，64	图形用户界面智能体研究综述。		✓	✓
刘等人，65	手机自动化中的图形用户界面智能体研究综述。		✓	✓
胡等人，66	基于多模态大语言模型的操作系统智能体研究综述。		✓	✓
施等人，167	构建可信图形用户界面智能体研究综述。		✓	✓
宁等人，68	网络自动化智能体研究综述。		✓	✓
唐等人，69	由 (多模态) 大语言模型驱动的图形用户界面智能体研究综述。		✓	✓
李和黄等人，70	由基础模型驱动并通过强化学习增强的图形用户界面智能体总结		✓	✓
萨格等人，71	计算机使用的人工智能智能体研究综述。	O	✓	✓
我们的工作	对具有大语言模型“大脑”的图形用户界面智能体进行全面综述，涵盖其基础、技术、			
框架、数据、模型、应用、挑战和未来路线图。	O	✓	✓	

Regarding individual components within LLM agents, several surveys provide detailed insights that are especially relevant for GUI agents. Huang et al., [51] examine planning mechanisms in LLM agents, which are essential for executing long-term tasks-a frequent requirement in GUI automation. Zhang et al., [54] explore memory mechanisms, which allow agents to store critical historical information, aiding in knowledge retention and decision-making. Additionally, Shen [13] surveys the use of tools by LLMs (such as APIs and code) to interact effectively with their environments, grounding actions in ways that produce tangible impacts. Further, Chang et al., [55] provide a comprehensive survey on evaluation methods for LLMs, which is crucial for ensuring the robustness and safety of GUI agents. Two additional surveys, [56] and [58], provide comprehensive overviews of benchmarks and evaluation methods specifically tailored to multimodal LLMs. The evaluation also facilitates a feedback loop, allowing agents to improve iteratively based on assessment results. Together, these surveys serve as valuable resources, offering guidance on essential components of LLM agents and forming a foundational basis for LLM-based GUI agents.

关于 LLM 代理中的各个组成部分，几项综述提供了详细见解，尤其适用于 GUI 代理。Huang 等人 [51] 研究了 LLM 代理中的规划机制，这对于执行长期任务至关重要——这是 GUI 自动化中的常见需求。Zhang 等人 [54] 探讨了记忆机制，使代理能够存储关键的历史信息，有助于知识保留和决策。此外，Shen[13] 综述了 LLM 使用工具 (如 API 和代码) 与环境有效交互的方法，使其行为具备实际影响力。Chang 等人 [55] 则提供了关于 LLM 评估方法的全面综述，这对于确保 GUI 代理的稳健性和安全性至关重要。另有两篇综述 [56] 和 [58] 专门针对多模态 LLM 的基准和评估方法进行了全面概述。评估还促进了反馈循环，使代理能够基于评估结果迭代改进。综上，这些综述作为宝贵资源，为 LLM 代理的关键组成部分提供指导，构成了基于 LLM 的 GUI 代理的基础。

Compared to existing surveys, our work offers a significantly more comprehensive and up-to-date overview of the LLM-powered GUI agent landscape. We curate and synthesize over 500 references, covering a wide range of topics including foundation models, data sources, system frameworks, benchmarks, evaluation methodologies, and practical deployments. While prior surveys often concentrate on narrower aspects on selected platform (e.g., web,

mobile), our survey takes a holistic perspective that spans the full development and deployment lifecycle. Beyond narrative summaries, we also provide consolidated reference tables for each subdomain, enabling readers to quickly categorize and locate relevant works across platforms and research themes—serving as a practical handbook for both researchers and practitioners. Furthermore, we incorporate foundational background material and propose evaluation taxonomies that make the survey accessible to newcomers, addressing gaps in prior work that often assume a high degree of prior familiarity.

与现有综述相比，我们的工作提供了更全面且最新的 LLM 驱动 GUI 代理领域概览。我们整理并综合了 500 多篇文献，涵盖基础模型、数据源、系统框架、基准、评估方法及实际部署等广泛主题。此前的综述多聚焦于特定平台 (如网页、移动端) 的狭窄方面，而我们的综述则采取全生命周期的整体视角。除了叙述性总结外，我们还为各子领域提供了整合的参考表，方便读者快速分类并定位跨平台及研究主题的相关工作——为研究者和从业者提供实用手册。此外，我们融入了基础背景材料并提出了评估分类法，使综述对新手友好，弥补了以往工作中常假设读者具备较高先验知识的不足。

3 BACKGROUND

3 背景

The development of LLM-brained GUI agents is grounded in three major advancements: (i) large language models (LLMs) [8], which bring advanced capabilities in natural language understanding and code generation, forming the core intelligence of these agents; (ii) accompanying agent architectures and tools [47] that extend LLM capabilities, bridging the gap between language models and physical environments to enable tangible impacts; and (iii) GUI automation [73], which has cultivated a robust set of tools, models, and methodologies essential for GUI agent functionality. Each of these components has played a critical role in the emergence of LLM-powered GUI agents. In the following subsections, we provide a brief overview of these areas to set the stage for our discussion.

基于 LLM 的 GUI 代理的发展依托于三大进展:(i) 大型语言模型 (LLMs)[8]，具备先进的自然语言理解和代码生成能力，构成这些代理的核心智能；(ii) 配套的代理架构和工具 [47]，扩展了 LLM 的能力，弥合语言模型与物理环境之间的鸿沟，实现实际影响；(iii)GUI 自动化 [73]，培养了一套完善的工具、模型和方法论，是 GUI 代理功能的关键支撑。这些组成部分共同推动了 LLM 驱动 GUI 代理的出现。以下小节将简要介绍这些领域，为后续讨论奠定基础。

3.1 Large Language Models: Foundations and Capabili- ties

3.1 大型语言模型: 基础与能力

The study of language models has a long and rich history [74], beginning with early statistical language models [75] and smaller neural network architectures [76]. Building on these foundational concepts, recent advancements have focused on transformer-based LLMs, such as the Generative Pre-trained Transformers (GPTs) [77]. These models are pretrained on extensive text corpora and feature significantly larger model sizes, validating scaling laws and demonstrating exceptional capabilities across a wide range of natural language tasks. Beyond their sheer size, these LLMs exhibit enhanced language understanding and generation abilities, as well as emergent properties that are absent in smaller-scale language models 78].

语言模型的研究历史悠久且丰富 [74]，始于早期的统计语言模型 [75] 和较小的神经网络架构 [76]。在这些基础概念上，近期进展聚焦于基于 Transformer 的大型语言模型，如生成式预训练变换器 (GPTs)[77]。这些模型在大规模文本语料上预训练，模型规模显著增大，验证了规模定律，并在广泛的自然语言任务中展现出卓越能力。除了规模庞大，这些 LLM 还表现出增强的语言理解与生成能力，以及小规模语言模型所不具备的涌现特性 [78]。

Early neural language models, based on architectures like recurrent neural networks (RNNs) [79] and long short-term memory networks (LSTMs) [80], were limited in both performance and generalization. The introduction of the Transformer model, built on the attention mechanism [81], marked a transformative milestone, establishing the foundational architecture now prevalent across almost all subsequent LLMs. This development led to variations in model structures, including encoder-only models (e.g., BERT [82], RoBERTa [83], ALBERT [84]), decoder-only models (e.g., GPT-1 [85], GPT-2 [86]), and encoder-decoder models (e.g., T5 [87], BART [88]). In 2022, ChatGPT [11] based on GPT-3.5 [89] launched as a groundbreaking LLM, fundamentally shifting perceptions of what language models can achieve. Since then, numerous advanced LLMs have emerged, including GPT-4 [90], LLaMA-3 [91], and Gemini [92], propelling the field into rapid growth. Today's LLMs are highly versatile, with many of them are capable of processing multimodal data and performing a range of tasks, from question answering to code generation, making them indispensable tools in various applications [93]-[96].

早期神经语言模型基于循环神经网络 (RNNs)[79] 和长短期记忆网络 (LSTMs)[80]，在性能和泛化能力上均有限。Transformer 模型的引入，基于注意力机制 [81]，标志着一个变革性里程碑，奠定了几乎所有后续 LLM 通用的基础架构。该发展催生了多种模型结构，包括仅编码器模型 (如 BERT[82]、RoBERTa[83]、ALBERT[84])、仅解码器模型 (如 GPT-1[85]、GPT-2[86]) 和编码器-解码器模型 (如 T5[87]、BART[88])。2022 年，基于 GPT-3.5[89] 的 ChatGPT[11] 作为突破性 LLM 发布，根本改变了人们对语言模型能力的认知。此后，众多先进 LLM 相继出现，包括 GPT-4[90]、LLaMA-3[91] 和 Gemini[92]，推动该领域快速发展。现今的 LLM 高度多才多艺，许多模型能够处理多模态数据，执行从问答到代码生成的多种任务，成为各类应用中不可或缺的工具 [93]-[96]。

The emergence of LLMs has also introduced significant advanced properties that invigorate their applications, making previously challenging tasks, such as natural language-driven GUI agents feasible. These advancements include:

LLM 的出现还带来了显著的高级特性，激发了其应用潜力，使得此前具有挑战性的任务，如基于自然语言的 GUI 代理成为可能。这些进展包括:

1) Few-Shot Learning [77]: Also referred to as in-context learning [97], LLMs can acquire new tasks from a small set of demonstrated examples presented in the prompt during inference, eliminating the need for retraining. This capability is crucial for enabling GUI agents to generalize across different environments with minimal effort.

1) 少样本学习 [77]: 也称为上下文学习 [97]，大型语言模型 (LLMs) 能够从推理时提示中提供的一小组示例中学习新任务，无需重新训练。这一能力对于使 GUI 代理能够以最小的努力在不同环境中实现泛化至关重要。

2) Instruction Following [98]: After undergoing instruction tuning, LLMs exhibit a remarkable ability to follow instructions for novel tasks, demonstrating strong generalization skills [89]. This allows LLMs to effectively comprehend user requests directed at GUI agents and to follow predefined objectives accurately.

2) 指令遵循 [98]: 经过指令微调后，LLMs 展现出对新任务指令的卓越遵循能力，表现出强大的泛化能力 [89]。这使得 LLMs 能够有效理解用户针对 GUI 代理的请求，并准确执行预设目标。

3) Long-Term Reasoning [99]: LLMs possess the ability to plan and solve complex tasks by breaking them down into manageable steps, often employing techniques like chain-of-thought (CoT) reasoning [100], [101]. This capability is essential for GUI agents, as many tasks require multiple steps and a robust planning framework.

3) 长期推理 [99]:LLMs 具备通过将复杂任务分解为可管理步骤来规划和解决问题的能力，常采用链式思维 (chain-of-thought, CoT) 推理技术 [100],[101]。这一能力对 GUI 代理尤为重要，因为许多任务需要多步骤和稳健的规划框架。

4) Code Generation and Tool Utilization [102]: LLMs excel in generating code and utilizing various tools, such as APIs [13]. This expertise is vital, as code and tools form the essential toolkit for GUI agents to interact with their environments.

4) 代码生成与工具使用 [102]:LLMs 擅长生成代码并利用各种工具，如 API[13]。这项专长至关重要，因为代码和工具构成了 GUI 代理与其环境交互的基本工具包。

5) Multimodal Comprehension [10]: Advanced LLMs can integrate additional data modalities, such as images, into their training processes, evolving into multimodal models. This ability is particularly important for GUI agents, which must interpret GUI screenshots presented as images in order to function effectively [103].

5) 多模态理解 [10]: 先进的 LLMs 能够将图像等额外数据模态整合进训练过程，发展为多模态模型。这一能力对 GUI 代理尤为重要，因为它们必须解读以图像形式呈现的 GUI 截图以实现有效功能 [103]。

To further enhance the specialization of LLMs for GUI agents, researchers often fine-tune these models with domain-specific data, such as user requests, GUI screenshots, and action sequences, thereby increasing their customization and effectiveness. In Section 8, we delve into these advanced, tailored models for GUI agents, discussing their unique adaptations and improved capabilities for interacting with graphical interfaces.

为了进一步提升 LLMs 在 GUI 代理领域的专业化，研究人员通常使用领域特定数据进行微调，如用户请求、GUI 截图和操作序列，从而增强其定制化和效能。在第 8 节中，我们将深入探讨这些面向 GUI 代理的高级定制模型，讨论其独特的适应性和改进的图形界面交互能力。

3.2 LLM Agents: From Language to Action

3.2 LLM 代理: 从语言到行动

Traditional AI agents have often focused on enhancing specific capabilities, such as symbolic reasoning or excelling in particular tasks like Go or Chess. In contrast, the emergence of LLMs has transformed AI agents by providing them with a natural language interface, enabling human-like decision-making capabilities, and equipping them to perform a wide variety of tasks and take tangible actions in diverse environments [12], [47], [104], [105]. In LLM agents, if LLMs form the ”brain” of a GUI agent, then its accompanying components serve as its ”eyes and hands”, enabling the LLM to perceive the environment’s status and translate its textual output into actionable

steps that generate tangible effects [46]. These components transform LLMs from passive information sources into interactive agents that execute tasks on behalf of users, which redefine the role of LLMs from purely text-generative models to systems capable of driving actions and achieving specific goals.

传统的 AI 代理通常专注于提升特定能力，如符号推理或在围棋、国际象棋等特定任务上的卓越表现。相比之下，LLMs 的出现通过提供自然语言接口，赋予 AI 代理类人决策能力，使其能够执行多样化任务并在多种环境中采取实际行动 [12],[47],[104],[105]。在 LLM 代理中，如果 LLMs 构成 GUI 代理的“大脑”，那么其配套组件则是“眼睛和手”，使 LLM 能够感知环境状态并将文本输出转化为可执行步骤，产生实际效果 [46]。这些组件将 LLMs 从被动的信息源转变为代表用户执行任务的交互式代理，重新定义了 LLMs 从纯文本生成模型到能够驱动行动并实现特定目标的系统的角色。

In the context of GUI agents, the agent typically perceives the GUI status through screenshots and widget trees [106], then performs actions to mimic user operations (e.g., mouse clicks, keyboard inputs, touch gestures on phones) within the environment. Since tasks can be long-term, effective planning and task decomposition are often required, posing unique challenges. Consequently, an LLM-powered GUI agent usually possess multimodal capabilities [59], a robust planning system [51], a memory mechanism to analyze historical interactions [54], and a specialized toolkit to interact with its environment [27]. We will discuss these tailored designs for GUI agents in detail in Section 5

在 GUI 代理的语境中，代理通常通过截图和控件树 [106] 感知 GUI 状态，然后执行操作以模拟用户行为 (如鼠标点击、键盘输入、手机触摸手势) 在环境中进行交互。由于任务可能是长期的，通常需要有效的规划和任务分解，这带来了独特挑战。因此，基于 LLM 的 GUI 代理通常具备多模态能力 [59]、稳健的规划系统 [51]、用于分析历史交互的记忆机制 [54] 以及与环境交互的专用工具包 [27]。我们将在第 5 节详细讨论这些针对 GUI 代理的定制设计。

3.3 GUI Automation: Tools, Techniques, and Challenges

3.3 GUI 自动化: 工具、技术与挑战

GUI automation has been a critical area of research and application since the early days of GUIs in computing. Initially developed to improve software testing efficiency, GUI automation focused on simulating user actions, such as clicks, text input, and navigation, across graphical applications to validate functionality [30]. Early GUI automation tools were designed to execute repetitive test cases on static interfaces [28]. These approaches streamlined quality assurance processes, ensuring consistency and reducing manual testing time. As the demand for digital solutions has grown, GUI automation has expanded beyond testing to other applications, including RPA [6] and Human-Computer Interaction (HCI) [107]. RPA leverages GUI automation to replicate human actions in business workflows, automating routine tasks to improve operational efficiency. Similarly, HCI research employs GUI automation to simulate user behaviors, enabling usability assessments and interaction studies. In both cases, automation has significantly enhanced productivity and user experience by minimizing repetitive tasks and enabling greater system adaptability [108], [109].

GUI 自动化自计算机图形界面诞生之初便是研究和应用的关键领域。最初为提升软件测试效率而开发，GUI 自动化侧重于模拟用户操作，如点击、文本输入和导航，以验证图形应用的功能 [30]。早期的 GUI 自动化工具设计用于在静态界面上执行重复测试用例 [28]。这些方法简化了质量保证流程，确保一致性并减少人工测试时间。随着数字解决方案需求的增长，GUI 自动化已扩展至测试之外的应用，包括机器人流程自动化 (RPA)[6] 和人机交互 (HCI)[107]。RPA 利用 GUI 自动化复制业务流程中的人工操作，自动化常规任务以提升运营效率。同样，HCI 研究采用 GUI 自动化模拟用户行为，支持可用性评估和交互研究。在这两种情况下，自动化显著提升了生产力和用户体验，减少重复任务并增强系统适应性 [108],[109]。

Traditional GUI automation methods have primarily depended on scripting and rule-based frameworks [4, [110]. Scripting-based automation utilizes languages such as Python, Java, and JavaScript to control GUI elements programmatically. These scripts simulate a user's actions on the interface, often using tools like Selenium [111] for web-based automation or Autolt [112] and SikuliX [113] for desktop applications. Rule-based approaches, meanwhile, operate based on predefined heuristics, using rules to detect and interact with specific GUI elements based on properties such as location, color, and text labels [4]. While effective for predictable, static workflows [114], these methods struggle to adapt to the variability of modern GUIs, where dynamic content, responsive layouts, and user-driven changes make it challenging to maintain rigid, rule-based automation [115].

传统的 GUI 自动化方法主要依赖于脚本和基于规则的框架 [4, [110]。基于脚本的自动化使用 Python、Java 和 JavaScript 等语言以编程方式控制 GUI 元素。这些脚本模拟用户在界面上的操作，通常使用 Selenium[111] 进行基于网页的自动化，或使用 Autolt[112] 和 SikuliX[113] 进行桌面应用程序的自动化。基于规则的方法则基于预定义的启发式规则，通过元素的位置、颜色和文本标签等属性检测并与特定 GUI 元素交互 [4]。虽然这些方法在可预测的静态工作流程中效果显著 [114]，但面对现代 GUI 中动态内容、响应式布局和用户驱动的变化时，难以维持刚性的基于规则的自动化 [115]。

CV has become essential for interpreting the visual aspects of GUIs [35], [116], [117], enabling automation tools to recognize and interact with on-screen elements even as layouts and designs change. CV techniques allow GUI automation systems to detect and classify on-screen elements, such as buttons, icons, and text fields, by analyzing screenshots and identifying regions of interest [118-120]. Optical Character Recognition (OCR) further enhances this capability by extracting text content from images, making it possible for automation systems to interpret labels, error messages, and form instructions accurately [121]. Object detection models add robustness, allowing automation agents to locate GUI elements even when the visual layout shifts [103]. By incorporating CV, GUI automation systems achieve greater resilience and adaptability in dynamic environments.

计算机视觉 (CV) 已成为解释 GUI 视觉元素的关键技术 [35]，[116]，[117]，使自动化工具能够识别并与屏幕上的元素交互，即使布局和设计发生变化。CV 技术通过分析截图并识别感兴趣区域，允许 GUI 自动化系统检测和分类屏幕元素，如按钮、图标和文本框 [118-120]。光学字符识别 (OCR) 进一步增强了这一能力，通过从图像中提取文本内容，使自动化系统能够准确理解标签、错误信息和表单指令 [121]。目标检测模型提升了鲁棒性，使自动化代理即使在视觉布局变化时也能定位 GUI 元素 [103]。通过引入计算机视觉，GUI 自动化系统在动态环境中实现了更强的适应性和弹性。

Despite advances, traditional GUI automation methods fall short in handling the complexity and variability of contemporary interfaces. Today's applications often feature dynamic, adaptive elements that cannot be reliably automated through rigid scripting or rule-based methods alone [122], [123]. Modern interfaces increasingly require contextual awareness [124], such as processing on-screen text, interpreting user intent, and recognizing visual cues.

These demands reveal the limitations of existing automation frameworks and the need for more flexible solutions capable of real-time adaptation and context-sensitive responses.

尽管取得了进展，传统 GUI 自动化方法在处理当代界面的复杂性和多样性方面仍显不足。如今的应用程序通常包含动态、自适应元素，单靠刚性的脚本或基于规则的方法难以实现可靠自动化 [122], [123]。现代界面越来越需要上下文感知能力 [124]，例如处理屏幕文本、理解用户意图和识别视觉线索。这些需求暴露了现有自动化框架的局限性，凸显了需要更灵活的解决方案，能够实现实时适应和上下文敏感的响应。

LLMs offer a promising solution to these challenges. With their capacity to comprehend natural language, interpret context, and generate adaptive scripts, LLMs can enable more intelligent, versatile GUI automation [125]. Their ability to process complex instructions and learn from context allows them to bridge the gap between static, rule-based methods and the dynamic needs of contemporary GUIs [126]. By integrating LLMs with GUI agents, these systems gain the ability to generate scripts on-the-fly based on the current state of the interface, providing a level of adaptability and sophistication that traditional methods cannot achieve. The combination of LLMs and GUI agents paves the way for an advanced, user-centered automation paradigm, capable of responding flexibly to user requests and interacting seamlessly with complex, evolving interfaces.

大型语言模型 (LLMs) 为这些挑战提供了有前景的解决方案。凭借理解自然语言、解析上下文和生成自适应脚本的能力，LLMs 能够实现更智能、多功能的 GUI 自动化 [125]。它们处理复杂指令并从上下文中学习的能力，使其能够弥合静态基于规则方法与现代 GUI 动态需求之间的差距 [126]。通过将 LLMs 与 GUI 代理集成，这些系统能够根据界面当前状态即时生成脚本，提供传统方法无法达到的适应性和复杂性。LLMs 与 GUI 代理的结合开辟了先进的以用户为中心的自动化范式，能够灵活响应用户请求，并与复杂且不断演变的界面无缝交互。

4 EVOLUTION AND PROGRESSION OF LLM- BRAINED GUI AGENTS

4 LLM 驱动 GUI 代理的演进与发展

”Rome wasn’t built in a day.” The development of LLM-brained GUI agents has been a gradual journey, grounded in decades of research and technical progress. Beginning with simple GUI testing scripts and rule-based automation frameworks, the field has evolved significantly through the integration of machine learning techniques, creating more intelligent and adaptive systems. The introduction of LLMs, especially multimodal models, has transformed GUI automation by enabling natural language interactions and fundamentally reshaping how users interact with software applications.

“罗马不是一天建成的。”LLM 驱动的 GUI 代理的发展是一个渐进的过程，基于数十年的研究和技术进步。从简单的 GUI 测试脚本和基于规则的自动化框架起步，该领域通过引入机器学习技术实现了显著演变，打造出更智能、更具适应性的系统。尤其是多模态大型语言模型 (LLMs) 的引入，通过支持自然语言交互，彻底改变了 GUI 自动化，并根本性地重塑了用户与软件应用的交互方式。

As illustrated in Figure 3 prior to 2023 and the emergence of LLMs, work on GUI agents was limited in both scope and capability. Since then, the proliferation of LLM-based approaches has fostered numerous notable developments across platforms including web, mobile, and desktop environments. This surge is ongoing and continues to

drive innovation in the field. This section takes you on a journey tracing the evolution of GUI agents, emphasizing key milestones that have brought the field to its present state.

如图 3 所示，在 2023 年及 LLMs 出现之前，GUI 代理的研究在范围和能力上都较为有限。此后，基于 LLM 的方法迅速普及，推动了包括网页、移动和桌面环境在内的多个平台上的诸多重要进展。这一浪潮仍在持续，持续推动该领域的创新。本节将带您回顾 GUI 代理的发展历程，重点介绍推动该领域达到现状的关键里程碑。

4.1 Early Automation Systems

4.1 早期自动化系统

In the initial stages of GUI automation, researchers relied on random-based, rule-based, and script-based strategies. While foundational, these methods had notable limitations in terms of flexibility and adaptability.

在 GUI 自动化的初期阶段，研究人员依赖随机、基于规则和基于脚本的策略。尽管这些方法奠定了基础，但在灵活性和适应性方面存在显著局限。

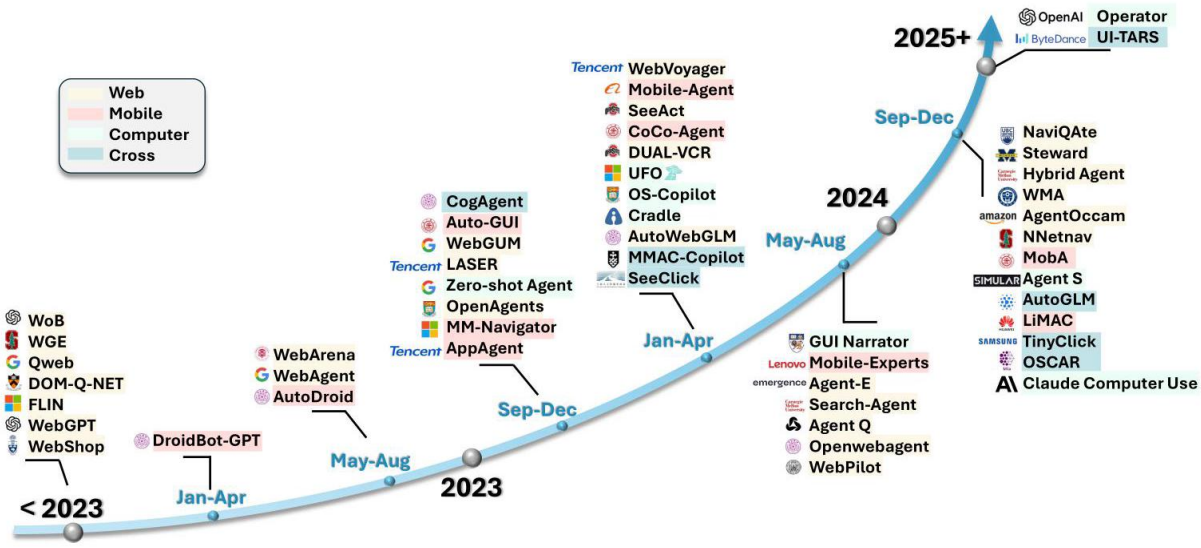


Fig. 3: An overview of GUI agents evolution over years.

图 3:GUI 代理多年来演进概览。

4.1.1 Random-Based Automation

4.1.1 基于随机的自动化

Random-based automation uses random sequences of actions within the GUI without relying on specific algorithms or structured models using monkey test [127]. This approach was widely used in GUI testing to uncover

potential issues by exploring unpredictable input sequences [128]. While effective at identifying edge cases and bugs, random-based methods were often inefficient due to a high number of redundant or irrelevant trials.

基于随机的自动化在 GUI 中执行随机动作序列，不依赖特定算法或结构化模型，常用猴子测试 (monkey test)[127]。该方法广泛应用于 GUI 测试, 通过探索不可预测的输入序列发现潜在问题 [128]。虽然在识别边缘情况和缺陷方面有效，但由于大量冗余或无关的尝试，随机方法效率较低。

4.1.2 Rule-Based Automation

4.1.2 基于规则的自动化

Rule-based automation applies predefined rules and logic to automate tasks. In 2001, Memon et al., 129 introduced a planning approach that generated GUI test cases by transforming initial states to goal states through a series of predefined operators. Hellmann et al., [4] (2011) demonstrated the potential of rule-based approaches in exploratory testing, enhancing bug detection. In the RPA domain, SmartRPA [130] (2020) used rule-based processing to automate routine tasks, illustrating the utility of rules for streamlining structured processes.

基于规则的自动化应用预定义的规则和逻辑来自动执行任务。2001 年，梅蒙 (Memon) 等人 [129] 引入了一种规划方法，该方法通过一系列预定义的操作符将初始状态转换为目标状态，从而生成图形用户界面 (GUI) 测试用例。赫尔曼 (Hellmann) 等人 [4](2011 年) 展示了基于规则的方法在探索性测试中的潜力，提高了错误检测能力。在机器人流程自动化 (RPA) 领域，SmartRPA [130](2020 年) 使用基于规则的处理来自动执行日常任务，说明了规则在简化结构化流程方面的实用性。

4.1.3 Script-Based Automation

4.1.3 基于脚本的自动化

Script-based automation relies on detailed scripts to manage GUI interactions. Tools like jRapture [5] (2000) record and replay Java-based GUI sequences using Java binaries and the JVM, enabling consistent execution by precisely reproducing input sequences. Similarly, DART [131] (2003) automated the GUI testing lifecycle, from structural analysis to test case generation and execution, offering a comprehensive framework for regression testing.

基于脚本的自动化依赖详细的脚本来管理图形用户界面交互。像 jRapture [5](2000 年) 这样的工具使用 Java 二进制文件和 Java 虚拟机 (JVM) 记录和重放基于 Java 的图形用户界面序列，通过精确重现输入序列实现一致的执行。同样，DART [131](2003 年) 自动化了图形用户界面测试的生命周期，从结构分析到测试用例的生成和执行，为回归测试提供了一个全面的框架。

4.1.4 Tools and Software

4.1.4 工具和软件

A range of software tools were developed for GUI testing and business process automation during this period. Mi-crosoft Power Automate [132] (2019) provides a low-code/no-code environment for creating automated work-

flows within Microsoft applications. Selenium [133] (2004) supports cross-browser web testing, while Appium [134 (2012) facilitates mobile UI automation. Commercial tools like TestComplete [135] (1999), Katalon Studio [136] (2015), and Ranorex [137] (2007) allow users to create automated tests with cross-platform capabilities.

在此期间，开发了一系列用于图形用户界面测试和业务流程自动化的软件工具。微软 Power Automate [132](2019 年) 为在微软应用程序中创建自动化工作流提供了一个低代码/无代码环境。Selenium [133](2004 年) 支持跨浏览器的网页测试，而 Appium [134](2012 年) 便于进行移动用户界面自动化。像 TestComplete [135](1999 年)、Katalon Studio [136](2015 年) 和 Ranorex [137](2007 年) 等商业工具允许用户创建具有跨平台功能的自动化测试。

Although these early systems were effective for automating specific, predefined workflows, they lacked flexibility and required manual scripting or rule-based logic. Nonetheless, they established the foundations of GUI automation, upon which more intelligent systems were built.

尽管这些早期系统在自动化特定的、预定义的工作流方面很有效，但它们缺乏灵活性，需要手动编写脚本或基于规则的逻辑。尽管如此，它们为图形用户界面自动化奠定了基础，更智能的系统在此基础上得以构建。

4.2 The Shift Towards Intelligent Agents

4.2 向智能代理的转变

The incorporation of machine learning marked a major shift towards more adaptable and capable GUI agents. Early milestones in this phase included advancements in machine learning, natural language processing, computer vision, and reinforcement learning applied to GUI tasks.

机器学习的融入标志着向更具适应性和能力的图形用户界面代理的重大转变。这一阶段的早期里程碑包括将机器学习、自然语言处理、计算机视觉和强化学习应用于图形用户界面任务方面的进展。

4.2.1 Machine Learning and Computer Vision

4.2.1 机器学习和计算机视觉

RoScript [110] (2020) was a pioneering system that introduced a non-intrusive robotic testing system for touchscreen applications, expanding GUI automation to diverse platforms. AppFlow [138] (2018) used machine learning to recognize common screens and UI components, enabling modular testing for broad categories of applications. Progress in computer vision also enabled significant advances in GUI testing, with frameworks [117] (2010) automating visual interaction tasks. Humanoid [139] (2019) uses a deep neural network model trained on human interaction traces within the Android system to learn how users select actions based on an app's GUI. This model is then utilized to guide test input generation, resulting in improved coverage and more human-like interaction patterns during testing. Similarly, Deep GUI 140 (2021) applies deep learning techniques to filter out irrelevant parts of the screen, thereby enhancing black-box testing effectiveness in GUI testing by focusing only on significant elements. These approaches demonstrate the potential of deep learning to make GUI testing more efficient and intuitive by aligning it closely with actual user behavior.

RoScript [110](2020 年) 是一个开创性的系统，它为触摸屏应用引入了一种非侵入式的机器人测试系统，将图形用户界面自动化扩展到了不同的平台。AppFlow [138](2018 年) 使用机器学习来识别常见的屏幕和用户界面组件，为广泛类别的应用程序实现模块化测试。计算机视觉的进展也使图形用户界面测试取得了重大进展，一些框架 [117](2010 年) 实现了视觉交互任务的自动化。Humanoid [139](2019 年) 使用在安卓系统内的人类交互轨迹上训练的深度神经网络模型，来学习用户如何根据应用程序的图形用户界面选择操作。然后利用该模型来指导测试输入的生成，从而在测试期间提高覆盖率并实现更类似人类的交互模式。同样，Deep GUI [140](2021 年) 应用深度学习技术过滤掉屏幕上无关的部分，从而通过仅关注重要元素来提高图形用户界面测试中黑盒测试的有效性。这些方法展示了深度学习通过紧密贴合实际用户行为，使图形用户界面测试更高效、更直观的潜力。

Widget detection, as demonstrated by White et al., 103 (2019), leverages computer vision to accurately identify UI elements, serving as a supporting technique that enables more intelligent and responsive UI automation. By detecting and categorizing interface components, this approach enhances the agent's ability to interact effectively with complex and dynamic GUIs [141].

正如怀特 (White) 等人 [103](2019 年) 所展示的，小部件检测利用计算机视觉准确识别用户界面元素，作为一种支持技术，使更智能、响应更灵敏的用户界面自动化成为可能。通过检测和分类界面组件，这种方法增强了代理与复杂动态图形用户界面有效交互的能力 [141]。

4.2.2 Natural Language Processing

4.2.2 自然语言处理

Natural language processing capabilities introduced a new dimension to GUI automation. Systems like RUSS [142] (2021) and FLIN [143] (2020) allowed users to control GUIs through natural language commands, bridging human language and machine actions. Datasets, such as those in [144] (2020), further advanced the field by mapping natural language instructions to mobile UI actions, opening up broader applications in GUI control. However, these approaches are limited to handling simple natural commands and are not equipped to manage long-term tasks.

自然语言处理能力为图形用户界面自动化引入了一个新维度。像 RUSS [142](2021 年) 和 FLIN [143](2020 年) 这样的系统允许用户通过自然语言命令控制图形用户界面，架起了人类语言和机器操作之间的桥梁。像 [144](2020 年) 中的数据集，通过将自然语言指令映射到移动用户界面操作，进一步推动了该领域的发展，为图形用户界面控制开辟了更广泛的应用。然而，这些方法仅限于处理简单的自然命令，无法处理长期任务。

4.2.3 Reinforcement Learning

4.2.3 强化学习

The development of environments like World of Bits (WoB) 145 (2017) enabled the training of web-based agents using reinforcement learning (RL). Workflow-guided exploration 146 (2018) improved RL efficiency and task performance. DQT [147] (2024) applied deep reinforcement learning to automate Android GUI testing by preserving widget structures and semantics, while AndroidEnv [148] (2021) offered realistic simulations for agent

training on Android. WebShop [149] (2022) illustrated the potential for large-scale web interaction, underscoring the growing sophistication of RL-driven GUI automation.

像比特世界 (World of Bits，WoB) [145](2017 年) 这样的环境的开发，使得能够使用强化学习 (RL) 训练基于网页的代理。工作流引导的探索 [146](2018 年) 提高了强化学习的效率和任务性能。DQT [147](2024 年) 应用深度强化学习来自动化安卓图形用户界面测试，同时保留小部件的结构和语义，而 AndroidEnv [148](2021 年) 为在安卓系统上进行代理训练提供了逼真的模拟。WebShop [149](2022 年) 展示了大规模网页交互的潜力，凸显了强化学习驱动的图形用户界面自动化日益提高的复杂性。

While these machine learning-based approaches were more adaptable than earlier rule-based systems [150], [151], they still struggled to generalize across diverse, unforeseen tasks. Their dependence on predefined workflows and limited adaptability required retraining or customization for new environments, and natural language control was still limited.

虽然这些基于机器学习的方法比早期基于规则的系统更具适应性 [150][151]，但它们仍然难以在各种不可预见的任务中实现泛化。它们依赖于预定义的工作流程，适应性有限，在新环境中需要重新训练或定制，并且自然语言控制仍然受限。

4.3 The Advent of LLM-Brained GUI Agents

4.3 大语言模型驱动的图形用户界面 (GUI) 代理的出现

The introduction of LLMs, particularly multimodal models like GPT-40 [93] (2023), has radically transformed GUI automation by allowing intuitive interactions through natural language. Unlike previous approaches that required integration of separate modules, LLMs provide an end-to-end solution for GUI automation, offering advanced capabilities in natural language understanding, visual recognition, and reasoning.

大语言模型 (LLM) 的引入，特别是像 GPT - 40 [93](2023 年) 这样的多模态模型，通过允许通过自然语言进行直观交互，从根本上改变了 GUI 自动化。与之前需要集成独立模块的方法不同，大语言模型为 GUI 自动化提供了端到端的解决方案，在自然语言理解、视觉识别和推理方面具有先进的能力。

LLMs present several unique advantages for GUI agents, including natural language understanding, multimodal processing, planning, and generalization. These features make LLMs and GUI agents a powerful combination. While there were earlier explorations, 2023 marked a pivotal year for LLM-powered GUI agents, with significant developments across various platforms such as web, mobile, and desktop applications.

大语言模型为 GUI 代理带来了几个独特的优势，包括自然语言理解、多模态处理、规划和泛化能力。这些特性使大语言模型和 GUI 代理成为强大的组合。虽然早期有相关探索，但 2023 年是大语言模型驱动的 GUI 代理的关键一年，在网页、移动和桌面应用等各种平台上都有显著的发展。

4.3.1 Web Domain

4.3.1 网页领域

The initial application of LLMs in GUI automation was within the web domain, with early studies establishing benchmark datasets and environments [145], [149]. A key milestone was WebAgent [152] (2023), which, alongside WebGUM [153] (2023), pioneered real-world web navigation using LLMs. These advancements paved the way for further developments [17], [154], [155], utilizing more specialized LLMs to enhance web-based interactions.

大语言模型在 GUI 自动化中的最初应用是在网页领域，早期的研究建立了基准数据集和环境 [145][149]。一个关键的里程碑是 WebAgent [152](2023 年)，它与 WebGUM [153](2023 年) 一起，开创了使用大语言模型进行现实世界网页导航的先河。这些进展为进一步的发展 [17][154][155] 铺平了道路，利用更专业的大语言模型来增强基于网页的交互。

4.3.2 Mobile Devices

4.3.2 移动设备

The integration of LLMs into mobile devices began with AutoDroid [156] (2023), which combined LLMs with domain-specific knowledge for smartphone automation. Additional contributions like MM-Navigator [157] (2023), AppAgent [18] (2023), and Mobile-Agent [158] (2023) enabled refined control over smartphone applications. Research has continued to improve accuracy for mobile GUI automation through model fine-tuning [159], [160] (2024).

大语言模型与移动设备的集成始于 AutoDroid [156](2023 年)，它将大语言模型与特定领域的知识相结合，用于智能手机自动化。其他贡献如 MM - Navigator [157](2023 年)、AppAgent [18](2023 年) 和 Mobile - Agent [158](2023 年) 实现了对智能手机应用的精细控制。通过模型微调 [159][160](2024 年)，移动 GUI 自动化的准确性研究仍在不断改进。

4.3.3 Computer Systems

4.3.3 计算机系统

For desktop applications, UFO [19] (2024) was one of the first systems to leverage GPT-4 with visual capabilities to fulfill user commands in Windows environments. Cradle [161] (2024) extended these capabilities to software applications and games, while Wu et al., [162] (2024) provided interaction across diverse desktop applications, including web browsers, code terminals, and multimedia tools.

对于桌面应用程序，UFO [19](2024 年) 是最早利用具有视觉能力的 GPT - 4 在 Windows 环境中执行用户命令的系统之一。Cradle [161](2024 年) 将这些能力扩展到软件应用程序和游戏，而 Wu 等人 [162](2024 年) 实现了跨多种桌面应用程序的交互，包括网页浏览器、代码终端和多媒体工具。

4.3.4 Industry Models

4.3.4 行业模型

In industry, the Claude 3.5 Sonnet model [163] (2024) introduced a ”computer use” feature capable of interacting with desktop environments through UI operations [164]. This signifies the growing recognition of LLM-powered GUI agents as a valuable application in industry, with stakeholders increasingly investing in this technology.

在行业中，Claude 3.5 Sonnet 模型 [163](2024 年) 引入了“计算机使用”功能，能够通过用户界面 (UI) 操作与桌面环境进行交互 [164]。这表明大语言模型驱动的 GUI 代理作为一种有价值的应用在行业中得到了越来越多的认可，利益相关者也在加大对这项技术的投资。

OpenAI quickly followed up by releasing Operator 165] in 2025, a Computer-Using Agent (CUA) similar to Claude, achieving state-of-the-art performance across various benchmarks. This development underscores the industry’s recognition of the value of GUI agents and its growing investment in the field. As interest continues to surge, GUI agent research and development are expected to become increasingly competitive, marking the beginning of a rapidly evolving landscape.

OpenAI 迅速跟进，于 2025 年发布了 Operator [165]，这是一个类似于 Claude 的计算机使用代理 (CUA)，在各种基准测试中取得了最先进的性能。这一发展凸显了行业对 GUI 代理价值的认可以及对该领域不断增加的投资。随着兴趣的持续高涨，GUI 代理的研发预计将变得越来越具有竞争力，标志着一个快速发展的局面的开始。

Undoubtedly, LLMs have introduced new paradigms and increased the intelligence of GUI agents in ways that were previously unattainable. As the field continues to evolve, we anticipate a wave of commercialization, leading to transformative changes in user interaction with GUI applications.

毫无疑问，大语言模型引入了新的范式，并以以前无法实现的方式提高了 GUI 代理的智能水平。随着该领域的不断发展，我们预计将迎来一波商业化浪潮，从而给用户与 GUI 应用程序的交互带来变革性的变化。

4.4 GUI Agent vs. API-Based Agent

4.4 GUI 代理与基于 API 的代理

In the field of LLM-powered agents operating within digital environments, the action space can be broadly categorized into two types:

在大语言模型驱动的、在数字环境中运行的代理领域，动作空间大致可分为两类:

1) GUI Agents, which primarily rely on GUI operations (e.g., clicks, keystrokes) to complete tasks.

1) GUI 代理，主要依赖 GUI 操作 (如点击、按键) 来完成任务。

2) API-Based Agents, which utilize system or application-native APIs to fulfill objectives. We show the principle of both agent types in Figure 4. Each type has distinct advantages, and a deeper understanding of these approaches is critical for designing effective agents.

2) 基于 API 的代理，利用系统或应用程序原生 API 来实现目标。我们在图 4 中展示了这两种代理类型的原理。每种类型都有其独特优势，深入理解这些方法对于设计高效代理至关重要。

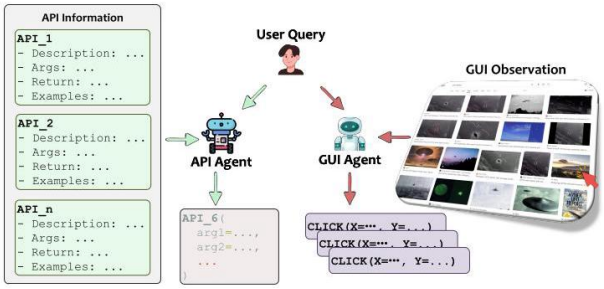


Fig. 4: The comparison between API agent vs. GUI agent.

图 4:API 代理与 GUI 代理的比较。

GUI operations provide a universal control interface that can operate across diverse applications using the same action primitives. This makes GUI agents highly generalizable, as they can interact with a wide range of software environments without requiring application-specific adaptations. However, GUI-based interactions are inherently more complex; even simple tasks may require multiple sequential steps, which can increase both the decision-making cost for the agent and the computational resources required for long-term, multi-step workflows. Another key aspect is the transparency of actions in GUI agents. Since GUI agents interact with applications in the same way a human would, by clicking, typing, and navigating through the interface, their actions are inherently more observable and interpretable to users. This transparency fosters better trust and comprehension in agent-computer interactions.

GUI 操作提供了一个通用的控制接口，可以使用相同的基本动作跨多种应用程序进行操作。这使得 GUI 代理具有高度的通用性，能够与广泛的软件环境交互，而无需针对特定应用进行适配。然而，基于 GUI 的交互本质上更为复杂；即使是简单任务也可能需要多个连续步骤，这会增加代理的决策成本和长期多步骤工作流所需的计算资源。另一个关键方面是 GUI 代理动作的透明性。由于 GUI 代理通过点击、输入和界面导航等方式与应用交互，类似于人类操作，其行为对用户来说更易观察和理解。这种透明性有助于增强用户对代理-计算机交互的信任和理解。

In contrast, API-based agents offer a more efficient and direct approach to task completion. By leveraging native APIs, tasks can often be fulfilled with a single, precise call, significantly reducing execution time and complexity. However, these native APIs are often private or restricted to specific applications, limiting accessibility and generalizability. This makes API-based agents less versatile in scenarios where API access is unavailable or insufficient. In addition, API-based agents operate behind the scenes, executing tasks through direct system calls, which, while often more efficient and reliable, can make their operations less visible and harder to debug for end users.

相比之下，基于 API 的代理提供了一种更高效、直接的任务完成方式。通过利用原生 API，任务通常可以通过一次精确调用完成，显著减少执行时间和复杂度。然而，这些原生 API 往往是私有的或仅限于特定应用，限制了其可访问性和通用性。这使得基于 API 的代理在 API 不可用或不足的场景中适用性较差。此外，基于 API 的代理在后台通过直接系统调用执行任务，虽然通常更高效且可靠，但其操作对终端用户来说较不透明，调试难度较大。

The most effective digital agents are likely to operate in a hybrid manner, combining the strengths of both approaches. Such agents can utilize GUI operations to achieve broad compatibility across software while exploiting native APIs where available to maximize efficiency and effectiveness. These hybrid agents strike a balance between generalization and task optimization, making them a critical focus area in this survey. For a more comprehensive comparison between GUI agents and API agents, please refer to [166].

最有效的数字代理很可能采用混合方式，结合两种方法的优势。这类代理可以利用 GUI 操作实现广泛的软件兼容性，同时在可用时利用原生 API 以最大化效率和效果。这些混合代理在通用性和任务优化之间取得平衡，是本综述的关键关注点。有关 GUI 代理与 API 代理的更全面比较，请参见文献 [166]。

5 LLM-BRAINED GUI AGENTS: FOUNDATIONS AND DESIGN

5 基于大型语言模型的 GUI 代理: 基础与设计

In essence, LLM-brained GUI agents are designed to process user instructions or requests given in natural language, interpret the current state of the GUI through screenshots or UI element trees, and execute actions that simulate human interaction across various software interfaces [19]. These agents harness the sophisticated natural language understanding, reasoning, and generative capabilities of LLMs to accurately comprehend user intent, assess the GUI context, and autonomously engage with applications across diverse environments, thereby enabling the completion of complex, multi-step tasks. This integration allows them to seamlessly interpret and respond to user requests, bringing adaptability and intelligence to GUI automation.

本质上，基于大型语言模型 (LLM) 的 GUI 代理旨在处理以自然语言给出的用户指令或请求，通过截图或 UI 元素树解读当前 GUI 状态，并执行模拟人类交互的操作，跨多种软件界面完成任务 [19]。这些代理利用 LLM 强大的自然语言理解、推理和生成能力，准确把握用户意图，评估 GUI 上下文，并自主与各种环境中的应用交互，从而实现复杂多步骤任务的完成。这种集成使其能够无缝解读和响应用户请求，为 GUI 自动化带来适应性和智能化。

As a specialized type of LLM agent, most current GUI agents adopt a similar foundational framework, integrating core components such as planning, memory, tool usage, and advanced enhancements like multi-agent collaboration, among others [47]. However, each component must be tailored to meet the specific objectives of GUI agents to ensure adaptability and functionality across various application environments.

作为 LLM 代理的专门类型，目前大多数 GUI 代理采用类似的基础框架，集成规划、记忆、工具使用及多代理协作等高级增强组件 [47]。然而，每个组件必须针对 GUI 代理的具体目标进行定制，以确保其在各种应用环境中的适应性和功能性。

In the following sections, we provide an in-depth overview of each component, offering a practical guide and tutorial on building an LLM-powered GUI agent from the ground up. This comprehensive breakdown serves as a cookbook for creating effective and intelligent GUI automation systems that leverage the capabilities of LLMs.

在接下来的章节中，我们将深入概述每个组件，提供构建基于 LLM 的 GUI 代理的实用指南和教程。此全面解析可视为创建高效智能 GUI 自动化系统的配方，充分利用 LLM 的能力。

5.1 Architecture and Workflow In a Nutshell

5.1 架构与工作流程概述

In Figure 5, we present the architecture of an LLM-brained GUI agent, showcasing the sequence of operations from user input to task completion. The architecture comprises several integrated components, each contributing to the agent's ability to interpret and execute tasks based on user-provided natural language instructions. Upon receiving a user request, the agent follows a systematic workflow that includes environment perception, prompt engineering, model inference, action execution, and continuous memory utilization until the task is fully completed.

在图 5 中，我们展示了基于 LLM 的 GUI 代理架构，呈现从用户输入到任务完成的操作序列。该架构包含多个集成组件，各自助力代理根据用户提供的自然语言指令解读并执行任务。代理在接收用户请求后，遵循环境感知、提示工程、模型推理、动作执行及持续记忆利用的系统化工作流程，直至任务完成。

In general, it consists of the following components:

总体而言，其组成包括以下部分:

1) Operating Environment: The environment defines the operational context for the agent, encompassing platforms such as mobile devices, web browsers, and desktop operating systems like Windows. To interact meaningfully, the agent perceives the environment's current state through screenshots, widget trees, or other methods of capturing UI structure [167]. It continuously monitors feedback on each action's impact, adjusting its strategy in real time to ensure effective task progression.

1) 操作环境: 环境定义了代理的运行上下文，涵盖移动设备、网页浏览器及 Windows 等桌面操作系统。为实现有效交互，代理通过截图、小部件树或其他 UI 结构捕获方法感知环境当前状态 [167]。它持续监控每个动作的反馈，实时调整策略，确保任务有效推进。

2) Prompt Engineering: Following environment perception, the agent constructs a detailed prompt to guide the LLM's inference [168]. This prompt incorporates user instructions, processed visual data (e.g., screenshots), UI element layouts, properties, and any additional context relevant to the task. This structured input maximizes the LLM's ability to generate coherent, context-aware responses aligned with the current GUI state.

2) 提示工程: 在环境感知之后，代理构建详细提示以引导 LLM 推理 [168]。该提示包含用户指令、处理后的视觉数据 (如截图)、UI 元素布局、属性及与任务相关的其他上下文。此结构化输入最大化 LLM 生成连贯且符合当前 GUI 状态的上下文响应的能力。

3) Model Inference: The constructed prompt is passed to a LLM, the agent's inference core, which produces a sequence of plans, actions and insights required to fulfill the user's request. This model may be a general-purpose LLM or a specialized model fine-tuned with GUI-specific data, enabling a more nuanced understanding of GUI interactions, user flows, and task requirements.

3) 模型推理: 构建的提示被传递给大型语言模型 (LLM)，即代理的推理核心，生成完成用户请求所需的一系列计划、动作和洞见。该模型可以是通用大型语言模型，也可以是经过 GUI 特定数据微调的专用模型，从而实现对 GUI 交互、用户流程和任务需求的更细致理解。

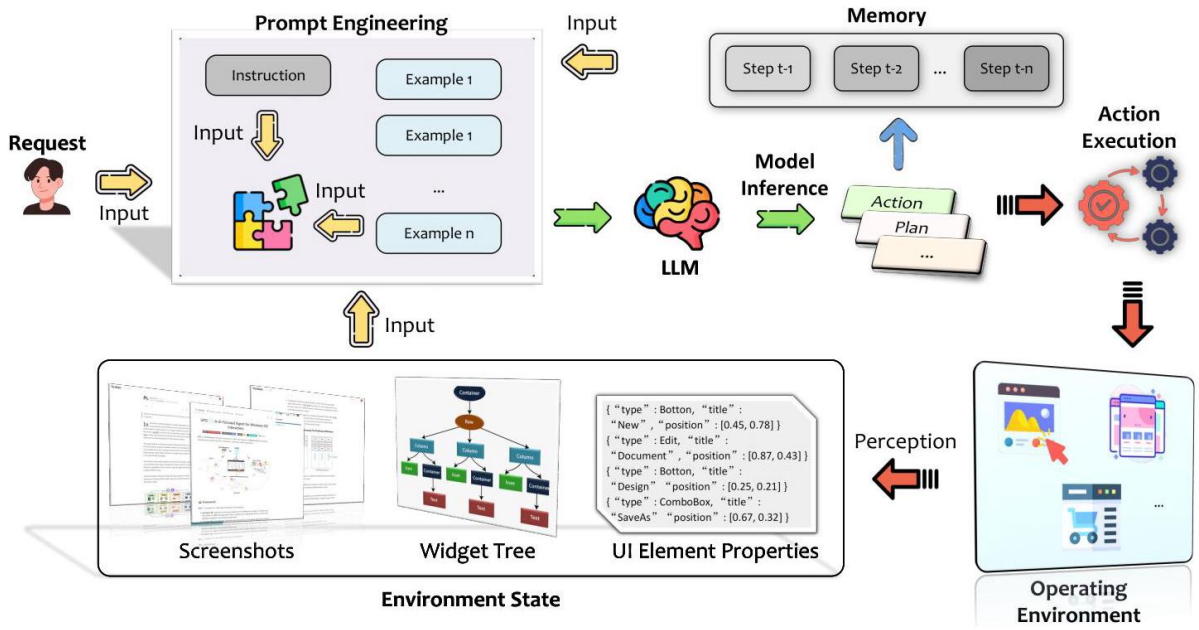


Fig. 5: An overview of the architecture and workflow of a basic LLM-powered GUI agent.

图 5: 基于大型语言模型的 GUI 代理的架构与工作流程概览。

4) Actions Execution: Based on the model's inference results, the agent identifies specific actions (such as mouse clicks, keyboard inputs, touchscreen gestures, or API calls) required for task execution [13]. An executor within the agent translates these high-level instructions into actionable commands that impact the GUI directly, effectively simulating human-like interactions across diverse applications and devices.

4) 动作执行: 基于模型的推理结果，代理识别执行任务所需的具体动作 (如鼠标点击、键盘输入、触摸手势或 API 调用)[13]。代理内的执行器将这些高级指令转化为直接影响 GUI 的可操作命令，有效模拟跨多种应用和设备的人类交互行为。

5) Memory: For multi-step tasks, the agent maintains an internal memory to track prior actions, task progress, and environment states [54]. This memory ensures coherence throughout complex workflows, as the agent can reference previous steps and adapt its actions accordingly. An external memory module may also be incorporated to enable continuous learning, access external knowledge, and enhance adaptation to new environments or requirements.

5) 记忆: 对于多步骤任务，代理维护内部记忆以跟踪先前动作、任务进展和环境状态 [54]。该记忆确保复杂工作流的连贯性，使代理能够参考之前的步骤并相应调整动作。还可引入外部记忆模块，实现持续学习、访问外部知识，并增强对新环境或需求的适应能力。

By iteratively traversing these stages and assembling the foundational components, the LLM-powered GUI agent operates intelligently, seamlessly adapting across various software interfaces and bridging the gap between language-based instruction and concrete action. Each component is critical to the agent's robustness, responsiveness, and capability to handle complex tasks in dynamic environments. In the following subsections, we detail the design and core techniques underlying each of these components, providing a comprehensive guide for constructing LLM-powered GUI agents from the ground up.

通过迭代遍历这些阶段并组装基础组件，基于大型语言模型的 GUI 代理能够智能运行，顺畅适应各种软件界面，弥合基于语言的指令与具体动作之间的鸿沟。每个组件对代理的稳健性、响应性及处理动态环境中复杂任务的能力至关重要。以下小节详细介绍这些组件的设计与核心技术，为从零构建基于大型语言模型的 GUI 代理提供全面指导。

5.2 Operating Environment

5.2 操作环境

The operating environment for LLM-powered GUI agents encompasses various platforms, such as mobile, web, and desktop operating systems, where these agents can interact with graphical interfaces. Each platform has distinct characteristics that impact the way GUI agents perceive, interpret, and act within it. Examples of GUIs from each platform are shown in Figure 6 This section details the nuances of each platform, the ways agents gather environmental information, and the challenges they face in adapting to diverse operating environments.

基于大型语言模型的 GUI 代理的操作环境涵盖多种平台，如移动端、网页端和桌面操作系统，这些代理可与图形界面交互。每个平台具有不同特性，影响 GUI 代理的感知、理解和行动方式。图 6 展示了各平台 GUI 示例。本节详述各平台的细微差别、代理收集环境信息的方式及其适应多样操作环境所面临的挑战。

Fig. 6: Examples of GUIs from web, mobile and computer platforms.

图 6: 来自网页、移动和计算机平台的 GUI 示例。

5.2.1 Platform

5.2.1 平台

The operating environment for LLM-powered GUI agents encompasses various platforms, such as mobile, web, and desktop operating systems, where these agents can interact with graphical interfaces. Each platform has distinct characteristics that impact the way GUI agents perceive, interpret, and act within it. Examples of GUIs from each platform are shown in Figure 6. This section details the nuances of each platform, the ways agents gather environmental information, and the challenges they face in adapting to diverse operating environments.

基于大型语言模型的 GUI 代理的操作环境涵盖多种平台，如移动端、网页端和桌面操作系统，这些代理可与图形界面交互。每个平台具有不同特性，影响 GUI 代理的感知、理解和行动方式。图 6 展示了各平台 GUI 示例。本节详述各平台的细微差别、代理收集环境信息的方式及其适应多样操作环境所面临的挑战。

1) Mobile Platforms: Mobile devices operate within constrained screen real estate, rely heavily on touch interactions [170], and offer varied app architectures (e.g., native vs. hybrid apps). Mobile platforms often use accessibility frameworks, such as Android's Accessibility AP | 171 and iOS's VoiceOver Accessibility Inspector [5] to expose structured information about UI elements. However, GUI agents must handle additional complexities in mobile environments, such as gesture recognition [169], app navigation [172], and platform-specific constraints (e.g., security and privacy permissions) [173], [174].

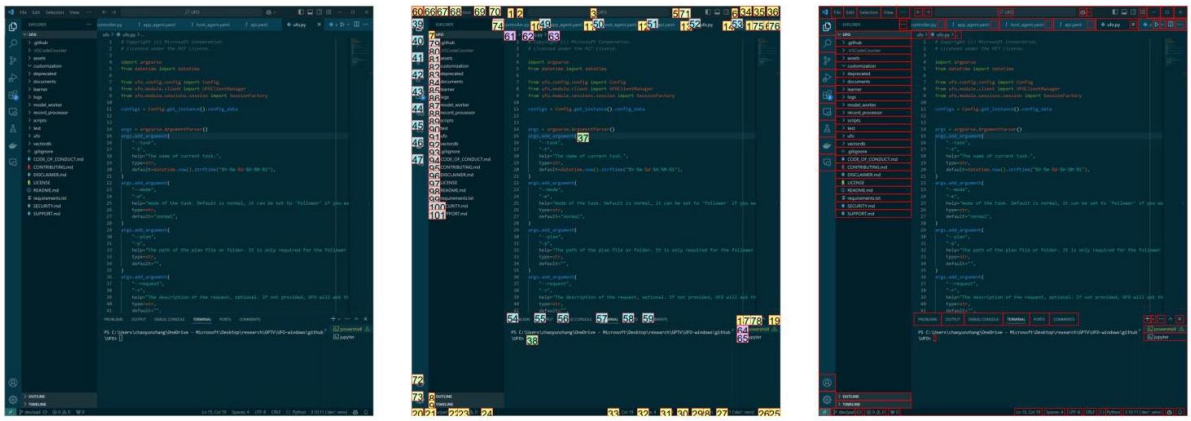
1) 移动平台: 移动设备屏幕空间有限，严重依赖触控交互 [170]，且应用架构多样 (如原生应用与混合应用)。移动平台通常使用辅助功能框架，如 Android 的 Accessibility API[171] 和 iOS 的 VoiceOver 辅助检查器 [5]，以暴露 UI 元素的结构化信息。然而，GUI 代理在移动环境中还需处理额外复杂性，如手势识别 [169]、应用导航 [172] 及平台特有限制 (如安全和隐私权限)[173]，[174]。

TABLE 3: Summary of platform-specific challenges, action spaces, and typical tasks for Web, Mobile, and Computer GUI environments.

表 3:Web、移动和计算机 GUI 环境中平台特定挑战、动作空间及典型任务的总结。

Platform	Typical GUI Challenges	Action Space	Representative Tasks
Mobile - Heavy reliance on touch and gesture recognition [169] - App architectures (native vs. hybrid) - Accessibility frameworks (e.g., Android's Accessibility API, iOS VoiceOver) - Platform-specific constraints (permissions, security, privacy)	- Constrained screen real estate - Tap, swipe, pinch, and other touch gestures - Virtual keyboard input - In-app navigation (menus, tabs) - Accessing hardware features (camera, GPS) - Messaging, social media posting - Location-based services and map interactions - Handling push notifications and permission dialogs	- App-based login and form filling	
Web - Asynchronous updates (AJAX, fetch APIs) - HTML/DOM-based structures - Cross-browser inconsistencies - DOM-based form filling - Link navigation and element inspection - JavaScript event triggering - Data extraction/web scraping - Searching and filtering (e.g., e-commerce) - Multi-step web navigation (redirects, pop-ups)	- Dynamic and responsive layouts - Click, hover, scroll - Form completion (registrations, checkouts)		
Computer - Multi-window operations system-level shortcuts - Automation APIs (e.g., Windows UI Automation [32] - Frequent software updates requiring adaptation - Complex, multi-layered software suites - Keyboard shortcuts and text input - Menu navigation, toolbars - Access to multiple application windows - Productivity software usage (office suites, IDEs) - Installing/uninstalling applications - Coordinating multi-application work-flows	- Full-fledged OS-level interfaces - Mouse click, drag-and-drop - File management and system settings		

平台	典型的图形用户界面挑战	操作空间	代表性任务
移动端 - 强烈依赖触摸和手势识别 [169] - 应用架构 (原生与混合) - 辅助功能框架 (如 Android 的 Accessibility API，iOS VoiceOver) - 平台特定限制 (权限、安全、隐私) - 虚拟键盘输入 - 应用内导航 (菜单、标签页) - 访问硬件功能 (摄像头、GPS) - 消息发送、社交媒体发布 - 基于位置的服务和地图交互 - 处理推送通知和权限对话框	- 屏幕空间受限 - 轻触、滑动、捏合等触摸手势 - 基于应用的登录和表单填写		
网页端 - 异步更新 (AJAX，fetch API) - 基于 HTML/DOM 的结构 - 跨浏览器不一致性 - 基于 DOM 的表单填写 - 链接导航和元素检查 - JavaScript 事件触发 - 数据提取/网页抓取 - 搜索和筛选 (如电子商务) - 多步骤网页导航 (重定向、弹窗)	- 动态响应式布局 - 点击、悬停、滚动 - 表单填写 (注册、结账)		
计算机端 - 多窗口操作及系统级快捷键 - 自动化 API(如 Windows UI Automation [32]) - 频繁的软件更新需适应 - 复杂多层的软件套件 - 键盘快捷键和文本输入 - 菜单导航、工具栏 - 访问多个应用窗口 - 办公软件使用 (办公套件、集成开发环境) - 安装/卸载应用程序 - 协调多应用工作流程	- 完整的操作系统级界面 - 鼠标点击、拖放 - 文件管理和系统设置		



(a) A clean GUI screenshot.

(b) A GUI screenshot with widgets highlighted by SoM.

(c) A GUI screenshot with widgets highlighted by bounding boxes.

Fig. 7: Examples of different variants of VS Code GUI screenshots.

图 7:VS Code 图形用户界面 (GUI) 截图不同变体示例。

2) Web Platforms: Web applications provide a relatively standardized interface, typically accessible through

Hy-pertext Markup Language (HTML) and Document Object Model (DOM) structures [175], [176]. GUI agents can leverage HTML attributes, such as element ID, class, and tag, to identify interactive components. Web environments also present dynamic content, responsive layouts, and asynchronous updates (e.g., AJAX requests) [177], requiring agents to continuously assess the DOM and adapt their actions to changing interface elements.

2) 网络平台: 网络应用程序提供了相对标准化的界面，通常可通过超文本标记语言 (HTML) 和文档对象模型 (DOM) 结构访问 [175]、[176]。GUI 智能体可以利用 HTML 属性 (如元素 ID、类和标签) 来识别交互组件。网络环境还存在动态内容、响应式布局和异步更新 (如 AJAX 请求) [177]，这要求智能体持续评估 DOM 并根据不断变化的界面元素调整其操作。

---

4. https://developer.android.com/reference/android/

4. https://developer.android.com/reference/android/

accessibilityservice/AccessibilityService

accessibilityservice/AccessibilityService

5. https://developer.apple.com/documentation/accessibility/

5. https://developer.apple.com/documentation/accessibility/

accessibility-inspector

accessibility-inspector

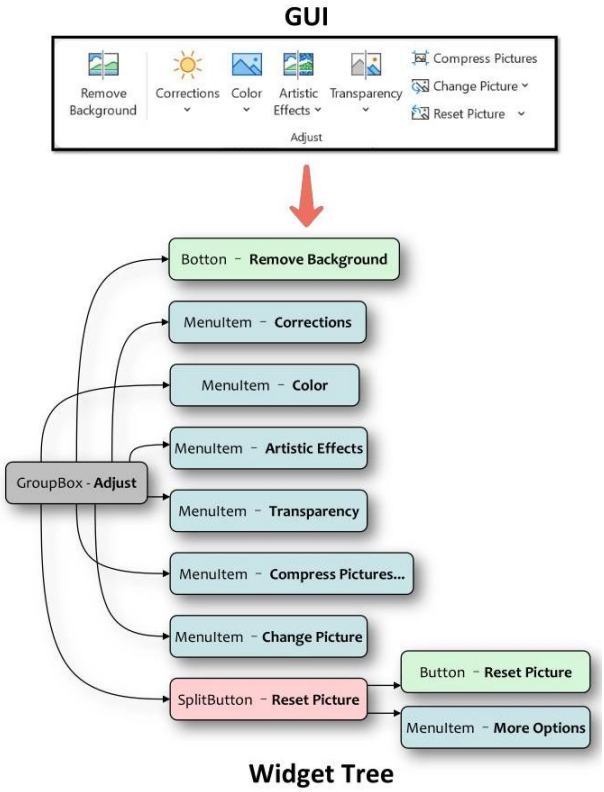


Fig. 8: An example of a GUI and its widget tree.

图 8: 一个 GUI 及其小部件树的示例。

3) Computer Platforms: Computer OS platforms, such as Windows, offer full control over GUI interactions. Agents can utilize system-level automation APIs, such as Windows UI Automation [32], to obtain comprehensive UI element data, including type, label, position, and bounding box. These platforms often support a broader set of interaction types, mouse, keyboard, and complex multi-window operations. These enable GUI agents to execute intricate workflows. However, these systems also require sophisticated adaptation for diverse applications, ranging from simple UIs to complex, multi-layered software suites.

3) 计算机平台: 计算机操作系统 (OS) 平台 (如 Windows) 可对 GUI 交互进行完全控制。智能体可以利用系统级自动化应用程序编程接口 (API)，如 Windows UI 自动化 [32]，来获取全面的 UI 元素数据，包括类型、标签、位置和边界框。这些平台通常支持更广泛的交互类型，如鼠标、键盘和复杂的多窗口操作。这使得 GUI 智能体能够执行复杂的工作流程。然而，这些系统还需要针对从简单 UI 到复杂的多层软件套件等各种应用进行复杂的适配。

In summary, the diversity of platforms, spanning mobile, web, and desktop environments, enable GUI agents to deliver broad automation capabilities, making them a generalized solution adaptable across a unified framework. However, each platform presents unique characteristics and constraints at both the system and application levels, necessitating a tailored approach for effective integration. By considering these platform-specific features, GUI agents can be optimized to address the distinctive requirements of each environment, thus enhancing their adaptability and reliability in varied automation scenarios.

综上所述，涵盖移动、网络和桌面环境的平台多样性使 GUI 智能体能够提供广泛的自动化能力，使其成为一种可在统一框架内适应不同场景的通用解决方案。然而，每个平台在系统和应用层面都呈现出独特的特征和限制，因此需要采用量身定制的方法进行有效集成。通过考虑这些特定于平台的特性，可以对 GUI 智能体进行优化，以满足每个环境的独特需求，从而提高它们在各种自动化场景中的适应性和可靠性。

5.2.2 Environment State Perception

5.2.2 环境状态感知

Accurately perceiving the current state of the environment is essential for LLM-powered GUI agents, as it directly informs their decision-making and action-planning processes. This perception is enabled by gathering a combination of structured data, such as widget trees, and unstructured data, like screenshots, to capture a complete representation of the interface and its components. In Table 4, we outline key toolkits available for collecting GUI environment data across various platforms, and below we discuss their roles in detail:

准确感知当前环境状态对于由大语言模型 (LLM) 驱动的 GUI 智能体至关重要，因为这直接影响它们的决策和行动计划制定过程。这种感知通过收集结构化数据 (如小部件树) 和非结构化数据 (如截图) 的组合来实现，以全面呈现界面及其组件。在表 4 中，我们列出了可用于在各种平台上收集 GUI 环境数据的关键工具包，下面我们将详细讨论它们的作用:

Widget	Widget Name	Position	Attributes
Remove Background	Button - 'Remove Background'	L-3810, T128, R-3708, B243	title='Remove Background'; auto_id='PictureBackgroundRemoval'; control_type='Button
Corrections	MenuItem - 'Corrections'	L-3689, T128, R-3592, B243	title='Corrections'; auto_id='PictureCorrectionsMenu'; control_type='MenuItem'
	MenuItem - 'Color'	L-3589, T128, R-3527, B243	title='Color'; auto_id='PictureColorMenu'; control type='MenuItem'
Artistic Effects -	MenuItem - 'Artistic Effects'	L-3524, T128, R-3448, B243	title='Artistic Effects'; auto id='PictureArtisticEffectsGallery'; control_type='MenuItem'
	MenuItem - 'Transparency'	L-3445, T128, R-3336, B243	title='Transparency'; auto_id='PictureTransparencyGallery'; control_type='MenuItem'
E. Compress Pictures	Button - 'Compress Pictures...'	L-3333, T128, R-3138, B164	title='Compress Pictures...'; auto id='PicturesCompress'; control type='Button'
<icon></icon>	MenuItem - 'Change Picture'	L-3333, T167, R-3149, B203	title='Change Picture'; auto id='PictureChangeMenu'; control_type='MenuItem'
<icon></icon>	SplitButton - 'Reset Picture'	L-3333, T206, R-3160, B242	title='Reset Picture'; control type='SplitButton'

控件	控件名称	位置	属性
删除背景	按钮 - “删除背景”	L-3810, T128, R-3708, B243	title=' 删除背景'; auto_id='PictureBackgroundRemoval'; control_type='Button
校正	菜单项 - “校正”	L-3689, T128, R-3592, B243	title=' 校正'; auto_id='PictureCorrectionsMenu'; control_type='MenuItem'
	菜单项 - “颜色”	L-3589, T128, R-3527, B243	title=' 颜色'; auto_id='PictureColorMenu'; control type='MenuItem'
艺术效果 -	菜单项 - “艺术效果”	L-3524, T128, R-3448, B243	title=' 艺术效果'; auto id='PictureArtisticEffectsGallery'; control_type='MenuItem'
	菜单项 - “透明度”	L-3445, T128, R-3336, B243	title=' 透明度'; auto_id='PictureTransparencyGallery'; control_type='MenuItem'
压缩图片	按钮 - “压缩图片...”	L-3333, T128, R-3138, B164	title=' 压缩图片...'; auto id='PicturesCompress'; control type='Button'
<icon></icon>	菜单项 - “更改图片”	L-3333, T167, R-3149, B203	title=' 更改图片'; auto id='PictureChangeMenu'; control_type='MenuItem'
<icon></icon>	分割按钮 - “重置图片”	L-3333, T206, R-3160, B242	title=' 重置图片'; control type='SplitButton'

Fig. 9: Examples of UI element properties in the PowerPoint application for GUI Agent interaction.

图 9:PowerPoint 应用中用于 GUI 代理交互的界面元素属性示例。

1) GUI Screenshots: Screenshots provide a visual snapshot of the application, capturing the entire state of the GUI at a given moment. They offer agents a reference for layout, design, and visual content, which is crucial when structural details about UI elements are either limited or unavailable. Visual elements like icons, images, and other graphical cues that may hold important context can be analyzed directly from screenshots. Many platforms have built-in tools to capture screenshots (e.g., Windows Snipping Too 7 macOS Screenshot Utility and Android's MediaProjection AP 9, and screenshots can be enhanced with additional annotations, such as Set-of-Mark (SoM) highlights [178] or bounding boxes 179 around key UI components, to streamline agent decisions. Figure 7 illustrates various screenshots of the VS Code GUI, including a clean version, as well as ones with SoM and bounding boxes that highlight actionable components, helping the agent focus on the most critical areas of the interface.

1) GUI 截图: 截图提供了应用程序的视觉快照，捕捉了特定时刻 GUI 的整体状态。它们为代理提供了布局、设计和视觉内容的参考，这在 UI 元素的结构细节有限或不可用时尤为重要。图标、图像及其他可能包含重要上下文的图形提示等视觉元素，可以直接从截图中进行分析。许多平台内置了截图工具 (例如 Windows 的截图工具、macOS 截图实用程序和 Android 的 MediaProjection API)，截图还可以通过附加注释进行增强，如关键 UI 组件周围的标记集 (Set-of-Mark，SoM) 高亮 [178] 或边界框 [179]，以简化代理的决策过程。图 7 展示了 VS Code GUI 的多种截图，包括干净版本，以及带有 SoM 和边界框突出显示可操作组件的版本，帮助代理聚焦界面中最关键的区域。

2) Widget Trees: Widget trees present a hierarchical view of interface elements, providing structured data about the layout and relationships between components [180]. We show an example of a GUI and its widget tree in Figure 8. By accessing the widget tree, agents can identify attributes such as element type, label, role, and relationships within the interface, all of which are essential for contextual understanding. Tools like Windows UI Automation and macOS's Accessibility AP [10] provide structured views for desktop applications, while Android's Accessibility API and HTML DOM structures serve mobile and web platforms, respectively. This hierarchical data is indispensable for agents to map out logical interactions and make informed choices based on the UI structure.

2) 小部件树: 小部件树展示了界面元素的层级视图，提供关于布局和组件之间关系的结构化数据 [180]。我们在图 8 中展示了一个 GUI 及其小部件树的示例。通过访问小部件树，代理可以识别元素类型、标签、角色及界面内的关系等属性,这些都是实现上下文理解的关键。Windows UI Automation 和 macOS 的 Accessibility API 提供了桌面应用的结构化视图，而 Android 的 Accessibility API 和 HTML DOM 结构则分别服务于移动和网页平台。这种层级数据对于代理绘制逻辑交互图谱并基于 UI 结构做出明智选择至关重要。

---

6. https://learn.microsoft.com/en-us/dotnet/framework/ui-automation/ ui-automation-overview

6. https://learn.microsoft.com/zh-cn/dotnet/framework/ui-automation/ui-automation-overview

7. https://support.microsoft.com/en-us/windows/

7. https://support.microsoft.com/en-us/windows/

use-snipping-tool-to-capture%2Dscreenshots%2D00246869%

使用截图工具捕获屏幕截图

2D1843%2D655f%2Df220%2D97299b865f6b

2D1843%2D655f%2Df220%2D97299b865f6b

8. https://support.apple.com/guide/mac-help/ take-a-screenshot-mh26782/mac

8. https://support.apple.com/guide/mac-help/ 截取屏幕截图-mh26782/mac

9. https://developer.android.com/reference/android/media/projection/ MediaProjection

9. https://developer.android.com/reference/android/media/projection/ MediaProjection(媒体投影)

TABLE 4: Key toolkits for collecting GUI environment data.

表 4: 收集图形用户界面环境数据的关键工具包。

Tool	Platform	Environment	Accessible Information	Highlight	Link
Selenium	Web	Browser platform) (Cross-	DOM elements, HTML structure, CSS properties	Extensive browser support and automation capabilities	https://www.selenium.dev/
Puppeteer	Web	Browser Firefox) (Chrome,	DOM elements, HTML/CSS. network requests	Headless browser automa tion with rich API	https://pptr.dev/
Playwright	Web	Browser platform) (Cross-	DOM elements, HTML/CSS. network interactions	Multi-browser support with automation and testing capabilities	https://playwright.dev/
TestCafe	Web	Browser platform) (Cross-	DOM elements, HTML structure, CSS properties	Easy setup with JavaScript/- TypeScript support	https://testcafe.io/
BeautifulSoup	Web	HTML Parsing	HTML content, DOM elements	Python library for parsing HTML and XML documents	https://www.crummy.com/software/ BeautifulSoup/
Protractor	Web	Browser (Angular)	DOM elements, Angular-specific attributes	Designed for Angular appli cations, integrates with Selenium	https://www.protractortest.org/
WebDriverIO	Web	Browser platform) (Cross-	DOM elements, HTML/CSS network interactions	Highly extensible with a vast plugin ecosystem	https://webdriver.io/
Ghost Inspector	Web	Browser platform) (Cross-	DOM elements, screenshots, test scripts	Cloud-based automated browser testing and monitoring	https://ghostinspector.com/
Cypress	Web	Browser platform) (Cross-	DOM elements, HTML/CSS. network requests	Real-time reloads and interactive debugging	https://www.cypress.io/
UIAutomator	Mobile	Android	UI hierarchy, widget properties, screen content	Native Android UI testing framework	https://developer.android.com/training/ testing/ui-automator
Espresso	Mobile	Android	UI components, view hierarchy, widget properties	Google's native Android UI testing framework	https://developer.android.com/training/ testing/espressc
Android View Hierarchy	Mobile	Android	UI hierarchy, widget properties, layout information	View hierarchy accessible via developer tools	https://developer.android.com/studio/ debua/layout-inspector
iOS Accessibility Inspector documentation/accessibility/ accessibility-inspector	Mobile	iOS	Accessibility tree, UI elements, properties	Tool for inspecting iOS app UI elements	https://developer.apple.com/
XCUITest documentation/xctest/user interface tests	Mobile	iOS	UI elements, accessibility properties, view hierarchy	Apple's iOS UI testing framework	https://developer.apple.com/
Flutter Driver	Mobile	Flutter apps	Widget tree, properties, interactions	Automation for Flutter applications	https://flutter.dev/docs/testing
Android's Me-diaProjection API	Mobile	Android	Screenshots, screen recording	Capturing device screen content programmatically	https://developer.android.com/reference/ android/media/projection/MediaProjection
Windows UI Automation	Computer	Windows	Control properties, widget trees, accessibility tree	Native Windows support with OS integration	https://docs.microsoft.com/windows/win32/ winauto/entry-uiauto-win32
Sikuli	Computer	Windows macOS Linux	Screenshots (image recognition), UI elements	Image-based automation using computer vision	http://sikulix.com/
AutoIt	Computer	Windows	Window titles, control properties, coordinates	Scripting language for Windows GUI automation	https://www.autoitscript.com/site/autoit/
Inspect.exe	Computer	Windows	UI elements, control properties, accessibility tree	Tool for inspecting Windows UI elements	https://docs.microsoft.com/windows/win32/ winauto/inspect-objects
macOS Accessibility API	Computer	macOS	Accessibility tree, UI elements, control properties	macOS support for accessibility and UI automation	https://developer.apple.com/accessibility/
Pywinauto	Computer	Windows	Control properties, UI hierarchy, window information	Python-based Windows GUI automation	https://pwww.electronis.org/docs/latest/
Electron Inspector	Computer	Electron apps	DOM elements, HTML/CSS JavaScript state	Tool for Electron applications	https://www.electronis.org/docs/latest/ tutorial/automated-testing
Windows Snipping Tool	Computer	Windows	Screenshots	Tool for capturing screen-shots in Windows	https://www.microsoft.com/en-us/windows/ tips/snipping-tool
macOS Screenshot Utility mac-help/take-a-screenshot-or% 2Dscreen-recording%2Dmh26782/mac	Computer	macOS	Screenshots, screen recording	Tool for capturing screen shots and recording screen	https://support.apple.com/guide/
AccessKit	Cross-Platform	Various OS	Accessibility tree, control properties, roles	Standardized APIs across platforms	https://github.com/AccessKit/accesskit
Appium	Cross-Platform	Android, iOS, Windows, macOS	UI elements, accessibility properties, gestures	Mobile automation framework	https://appium.io/
Robot Framework	Cross-Platform	Web, Mobile, Desktop	UI elements, DOM, screen-shots	Extensible with various libraries	https://robotframework.org/
Cucumber	Cross-Platform	Web, Mobile, Desktop	Step definitions, UI interactions	BDD framework supporting automation tools	https://scucumber.io/
TestComplete testcomplete/overview/	Cross-Platform	Web, Mobile, Desktop	UI elements, DOM, control properties	Tool with extensive feature set	https://smartbear.com/product/
Katalon Studio	Cross-Platform	Web, Mobile, Desktop	UI elements, DOM, screen-shots	All-in-one automation solution	https://www.katalon.com/
Ranorex	Cross-Platform	Web, Mobile, Desktop	UI elements, DOM, control properties	Tool with strong reporting fea tures	https://www.ranorex.com/
Applitools	Cross-Platform	Web, Mobile, Desktop	Screenshots, visual check points, DOM elements	AI-powered visual testing	https://applitools.com/

工具	平台	环境	可访问信息	高亮	链接
Selenium (Selenium)	网页	浏览器平台)(跨浏览器	DOM 元素，HTML 结构，CSS 属性	广泛的浏览器支持和自动化能力	https://www.selenium.dev/
Puppeteer (Puppeteer)	网页	浏览器 Firefox)(Chrome,	DOM 元素，HTML/CSS，网络请求	无头浏览器自动化，提供丰富的 API	https://pptr.dev/
Playwright (Playwright)	网页	浏览器平台)(跨浏览器	DOM 元素，HTML/CSS，网络交互	多浏览器支持，具备自动化和测试功能	https://playwright.dev/
TestCafe (TestCafe)	网页	浏览器平台)(跨浏览器	DOM 元素，HTML 结构，CSS 属性	易于设置，支持 JavaScript/TypeScript	https://testcafe.io/
BeautifulSoup (BeautifulSoup)	网页	HTML 解析	HTML 内容，DOM 元素	用于解析 HTML 和 XML 文档的 Python 库	https://www.crummy.com/software/BeautifulSoup/
Protractor (Protractor)	网页	浏览器 (Angular)	DOM 元素，Angular 特定属性	专为 Angular 应用设计，集成 Selenium	https://www.protractortest.org/
WebDriverIO (WebDriverIO)	网页	浏览器平台)(跨浏览器	DOM 元素，HTML/CSS，网络交互	高度可扩展，拥有庞大的插件生态	https://webdriver.io/
Ghost Inspector (Ghost Inspector)	网页	浏览器平台)(跨浏览器	DOM 元素，截图，测试脚本	基于云的自动化浏览器测试与监控	https://ghostinspector.com/
Cypress (Cypress)	网页	浏览器平台)(跨浏览器	DOM 元素，HTML/CSS，网络请求	实时重载与交互式调试	https://www.cypress.io/
UIAutomator (UIAutomator)	移动端	安卓	UI 层级，控件属性，屏幕内容	原生安卓 UI 测试框架	https://developer.android.com/training/testing/ui-automator
Espresso (Espresso)	移动端	安卓	UI 组件，视图层级，控件属性	谷歌原生安卓 UI 测试框架	https://developer.android.com/training/testing/espressc
安卓视图层级	移动端	安卓	UI 层级，控件属性，布局信息	通过开发者工具可访问的视图层级	https://developer.android.com/studio/debug/layout-inspector
iOS 辅助功能检查器 documentation/accessibility/ accessibility-inspector	移动端	iOS	辅助功能树，UI 元素，属性	用于检查 iOS 应用 UI 元素的工具	https://developer.apple.com/
XCUITest documentation/xctest/user interface tests	移动端	iOS	UI 元素，辅助功能属性，视图层级	苹果 iOS 界面测试框架	https://developer.apple.com/
Flutter 驱动	移动端	Flutter 应用	组件树，属性，交互	Flutter 应用的自动化	https://flutter.dev/docs/testing
安卓 MediaProjection API	移动端	安卓	截图，屏幕录制	以编程方式捕获设备屏幕内容	https://developer.android.com/reference/ android/media/projection/MediaProjection
Windows UI 自动化	计算机	Windows 系统	控件属性，组件树，辅助功能树	原生 Windows 支持及操作系统集成	https://docs.microsoft.com/en/windows/win32/ winauto/entry-uiauto-win32
Sikuli	计算机	Windows macOS Linux	截图 (图像识别)，UI 元素	基于图像的计算机视觉自动化	http://sikulix.com/
AutoIt	计算机	Windows 系统	窗口标题，控件属性，坐标	Windows 图形界面自动化脚本语言	https://www.autoitscript.com/site/autoit/
Inspect.exe	计算机	Windows 系统	UI 元素，控件属性，辅助功能树	Windows UI 元素检测工具	https://docs.microsoft.com/windows/win32/ winauto/inspect-objects
macOS 辅助功能 API	计算机	macOS 系统	辅助功能树，UI 元素，控件属性	macOS 对辅助功能和 UI 自动化的支持	https://developer.apple.com/accessibility/
Pywinauto	计算机	Windows 系统	控件属性，UI 层级，窗口信息	基于 Python 的 Windows 图形界面自动化	https://pwww.electronis.org/docs/latest/
Electron 检测器	计算机	Electron 应用	DOM 元素，HTML/CSS JavaScript 状态	Electron 应用工具	https://www.electronis.org/docs/latest/ tutorial/automated-testing
Windows 截图工具	计算机	Windows 系统	截图	Windows 系统截图工具	https://www.microsoft.com/en-us/windows/ tips/snipping-tool
macOS 截图工具 mac-help/take-a-screenshot-or%2Dscreen-recording%2Dmh26782/mac	计算机	macOS 系统	截图，屏幕录制	屏幕截图及录屏工具	https://support.apple.com/guide/
AccessKit	跨平台	多种操作系统	辅助功能树，控件属性，角色	跨平台标准化 API	https://github.com/AccessKit/accesskit
Appium	跨平台	安卓，iOS，Windows，macOS	UI 元素，辅助功能属性，手势	移动自动化框架	https://appium.io/
Robot Framework	跨平台	网页、移动端、桌面端	UI 元素、DOM、屏幕截图	可通过多种库进行扩展	https://robotframework.org/
Cucumber	跨平台	网页、移动端、桌面端	步骤定义、UI 交互	支持自动化工具的行为驱动开发 (BDD) 框架	https://scucumber.io/
TestComplete testcomplete/overview/	跨平台	网页、移动端、桌面端	UI 元素、DOM、控件属性	功能丰富的工具	https://smartbear.com/product/
Katalon Studio	跨平台	网页、移动端、桌面端	UI 元素、DOM、屏幕截图	一体化自动化解决方案	https://www.katalon.com/
Ranorex	跨平台	网页、移动端、桌面端	UI 元素、DOM、控件属性	具备强大报告功能的工具	https://www.ranorex.com/
Applitools	跨平台	网页、移动端、桌面端	屏幕截图、视觉检查点、DOM 元素	基于人工智能的视觉测试	https://applitools.com/

10. https://developer.apple.com/library/archive/documentation/

10. https://developer.apple.com/library/archive/documentation/

Accessibility/Conceptual/AccessibilityMacOSX/

Accessibility/Conceptual/AccessibilityMacOSX/

3) UI Element Properties: Each UI element in the interface contains specific properties, such as control type, label text, position, and bounding box dimensions, that help agents target the appropriate components. These properties are instrumental for agents to make decisions about spatial relationships (e.g., adjacent elements) and functional purposes (e.g., distinguishing between buttons and text fields). For instance, web applications reveal properties like DOM attributes (id, class, name) and CSS styles that provide context and control information. These attributes assist agents in pinpointing precise elements for interaction, enhancing their ability to navigate and operate within diverse UI environments. Figure 9 illustrates examples of selected UI element properties extracted by the Windows UI Automation API, which support GUI agents in decision-making.

3) UI 元素属性: 界面中的每个 UI 元素都包含特定属性，如控件类型、标签文本、位置和边界框尺寸，这些属性帮助代理定位合适的组件。这些属性对于代理判断空间关系 (例如相邻元素) 和功能用途 (例如区分按钮和文本框) 至关重要。例如，Web 应用程序会显示 DOM 属性 (id、class、name) 和 CSS 样式等属性，提供上下文和控件信息。这些属性帮助代理精确定位交互元素，增强其在多样 UI 环境中的导航和操作能力。图 9 展示了通过 Windows UI 自动化 API 提取的选定 UI 元素属性示例，支持 GUI 代理的决策过程。

4) Complementary CV Approaches: When structured information is incomplete or unavailable, computer vision techniques can provide additional insights [181]. For instance, OCR allows agents to extract text content

directly from screenshots, facilitating the reading of labels, error messages, and instructions [121]. Furthermore, advanced object detection [120] models like SAM (Segment Anything Model) [182], DINO [183] and OmniParser 184 can identify and classify UI components in various layouts, supporting the agent in dynamic environments where UI elements may frequently change. These vision-based methods ensure robustness, enabling agents to function effectively even in settings where standard UI APIs are insufficient. We illustrate an example of this complementary information in Figure 10 and further detail these advanced computer vision approaches in Section 5.7.1

4) 补充的计算机视觉方法: 当结构化信息不完整或不可用时，计算机视觉技术可以提供额外的洞见 [181]。例如，OCR(光学字符识别) 允许代理直接从截图中提取文本内容，便于读取标签、错误信息和指令 [121]。此外，先进的目标检测模型如 SAM(Segment Anything Model)[182]、DINO[183] 和 OmniParser[184] 能够识别和分类各种布局中的 UI 组件，支持代理在 UI 元素频繁变化的动态环境中工作。这些基于视觉的方法确保了鲁棒性，使代理即使在标准 UI API 不足的情况下也能有效运行。图 10 展示了此类补充信息的示例，5.7.1 节进一步详述了这些先进的计算机视觉方法。

Together, these elements create a comprehensive, multimodal representation of the GUI environment's current state, delivering both structured and visual data. By incorporating this information into prompt construction, agents are empowered to make well-informed, contextually aware decisions without missing critical environmental cues.

这些元素共同构建了 GUI 环境当前状态的全面多模态表示，提供结构化和视觉数据。通过将这些信息纳入提示构建，代理能够做出信息充分、具上下文感知的决策，不遗漏关键环境线索。

5.2.3 Environment Feedback

5.2.3 环境反馈

Effective feedback mechanisms are essential for GUI agents to assess the success of each action and make informed decisions for subsequent steps. Feedback can take several forms, depending on the platform and interaction type. Figure 11 presents examples of various types of feedback obtained from the environment.

有效的反馈机制对于 GUI 代理评估每个操作的成功与否并为后续步骤做出明智决策至关重要。反馈形式多样，取决于平台和交互类型。图 11 展示了从环境中获得的各种反馈示例。

1) Screenshot Update: By comparing before-and-after screenshots, agents can identify visual differences that signify state changes in the application. Screenshot analysis can reveal subtle variations in the interface, such as the appearance of a notification, visual cues, or confirmation messages, that may not be captured by structured data [185].

1) 截图更新: 通过比较操作前后的截图，代理可以识别应用状态变化的视觉差异。截图分析能揭示界面中的细微变化，如通知的出现、视觉提示或确认信息，这些可能无法通过结构化数据捕捉 [185]。

2) UI Structure Change: After executing an action, agents can detect modifications in the widget tree structure, such as the appearance or disappearance of elements, updates to element properties, or hierarchical shifts [186]. These changes indicate successful interactions (e.g., opening a dropdown or clicking a button) and help the agent determine the next steps based on the updated environment state.

2) UI 结构变化: 执行操作后，代理可检测控件树结构的修改，如元素的出现或消失、属性更新或层级变动 [186]。这些变化表明交互成功 (例如打开下拉菜单或点击按钮)，帮助代理基于更新的环境状态确定下一步操作。

3) Function Return Values and Exceptions: Certain platforms offer direct feedback on action outcomes through function return values or system-generated exceptions 187 . For example, API responses or JavaScript return values can confirm action success on web platforms, while exceptions or error codes can signal failed interactions, guiding the agent to retry or select an alternative approach.

3) 函数返回值和异常: 某些平台通过函数返回值或系统生成的异常提供操作结果的直接反馈 [187]。例如，API 响应或 JavaScript 返回值可确认 Web 平台上的操作成功，而异常或错误代码则指示交互失败，指导代理重试或选择替代方案。

These feedback provided by the environment is crucial for GUI agents to assess the outcomes of their previous actions. This real-time information enables agents to evaluate the effectiveness of their interventions and determine whether to adhere to their initial plans or pivot towards alternative strategies. Through this process of self-reflection, agents can adapt their decision-making, optimizing task execution and enhancing overall performance in dynamic and varied application environments.

环境提供的这些反馈对于 GUI 代理评估先前操作结果至关重要。实时信息使代理能够评估干预效果，决定是坚持初始计划还是转向替代策略。通过这种自我反思过程，代理可调整决策，优化任务执行，提升在动态多变应用环境中的整体表现。

5.3 Prompt Engineering

5.3 提示工程

In the operation of LLM-powered GUI agents, effective prompt construction is a crucial step that encapsulates all necessary information for the agent to generate appropriate responses and execute tasks successfully [168]. After gathering the relevant data from the environment, the agent formulates a comprehensive prompt that combines various components essential for inference by the LLM. Each component serves a specific purpose, and together they enable the agent to execute the user's request efficiently. Figure 12 illustrates a basic example of prompt construction in an LLM-brained GUI agent. The key elements of the prompt are summarized as follows:

在基于大型语言模型 (LLM) 的 GUI 代理操作中，有效的提示构建是关键步骤，涵盖代理生成适当响应和成功执行任务所需的所有信息 [168]。在收集环境相关数据后，代理制定综合提示，结合多种推理所需的组成部分。每个部分具有特定功能，共同支持代理高效执行用户请求。图 12 展示了 LLM 驱动 GUI 代理中提示构建的基本示例。提示的关键要素总结如下:

1) User Request: This is the original task description provided by the user, outlining the objective and desired outcome. It serves as the foundation for the agent's actions and is critical for ensuring that the LLM understands the context and scope of the task.

1) 用户请求: 这是用户提供的原始任务描述，概述目标和期望结果。它是代理行动的基础，对于确保 LLM 理解任务的上下文和范围至关重要。

2) Agent Instruction: This section provides guidance for the agent's operation, detailing its role, rules to follow, and specific objectives. Instructions clarify what inputs the agent will receive and outline the expected outputs from the LLM, establishing a framework for the inference process. The core agent instructions are usually embedded within the base system prompt of the LLM, with supplementary instructions dynamically injected or updated based on environmental feedback and contextual adaptation.

2) 代理指令: 本部分为代理操作提供指导，详细说明其角色、遵循规则和具体目标。指令明确代理将接收的输入和 LLM 预期输出，建立推理过程的框架。核心代理指令通常嵌入 LLM 的基础系统提示中，辅以根据环境反馈和上下文适应动态注入或更新的补充指令。

3) Environment States: The agent includes perceived GUI screenshots and UI information, as introduced in Section 5.2.2. This multimodal data may consist of various versions of screenshots (e.g., a clean version and a SoM annotated version) to ensure clarity and mitigate the risk of UI controls being obscured by annotations. This comprehensive representation of the environment is vital for accurate decision-making.

3) 环境状态: 代理包括感知到的 GUI 截图和 UI 信息，如第 5.2.2 节所述。这些多模态数据可能包含多个版本的截图 (例如，干净版本和带 SoM 注释的版本)，以确保清晰度并减少 UI 控件被注释遮挡的风险。这种对环境的全面表示对于准确决策至关重要。

4) Action Documents: This component outlines the available actions the agent can take, detailing relevant documentation, function names, arguments, return values, and any other necessary parameters. Providing this 5.4 information equips the LLM with the context needed to select and generate appropriate actions for the task at hand.

4) 操作文档: 该组件概述了代理可执行的操作，详细说明相关文档、函数名称、参数、返回值及其他必要参数。提供这些 5.4 信息使大型语言模型 (LLM) 具备选择和生成适当操作以完成任务的上下文。

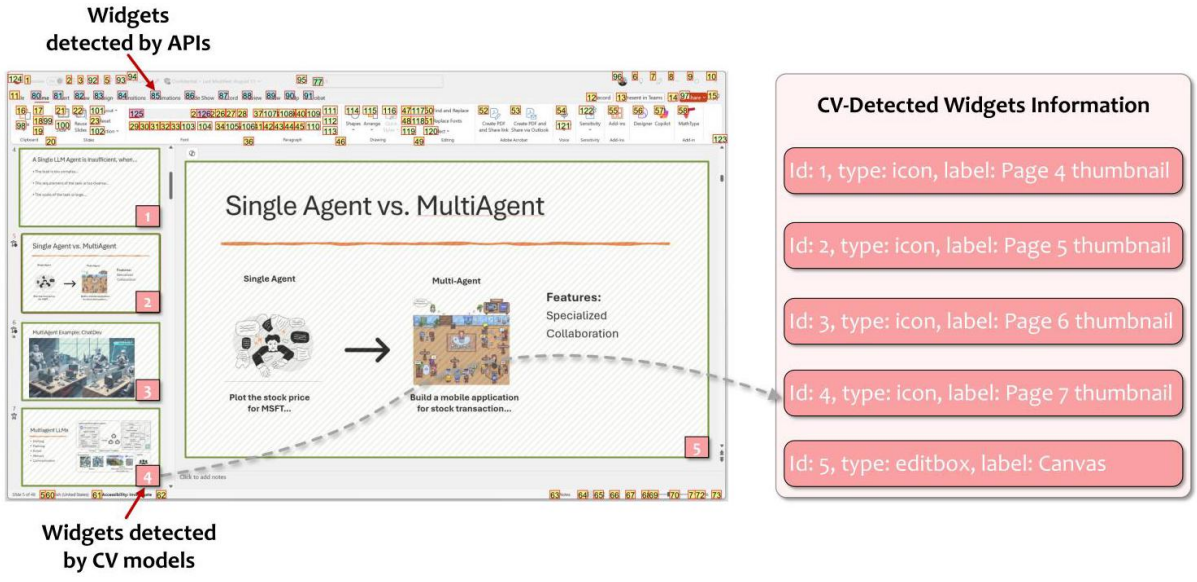


Fig. 10: An example illustrating the use of a CV approach to parse a PowerPoint GUI and detect non-standard widgets, inferring their types and labels.

图 10: 一个示例，展示如何使用计算机视觉 (CV) 方法解析 PowerPoint GUI 并检测非标准控件，推断其类型和标签。

5) Demonstrated Examples: Including example input/output pairs is essential to activate the in-context learning 97 capability of the LLM. These examples help the model comprehend and generalize the task requirements, enhancing its performance in executing the GUI agent's responsibilities.

5) 演示示例: 包含输入/输出示例对对于激活 LLM 的上下文学习能力 97 至关重要。这些示例帮助模型理解并泛化任务需求，提升其执行 GUI 代理职责的表现。

6) Complementary Information: Additional context that aids in planning and inference may also be included. This can consist of historical data retrieved from the agent's memory (as detailed in Section 5.6) and external knowledge sources, such as documents obtained through retrieval-augmented generation (RAG) methods 188], [189]. This supplemental information can provide valuable insights that further refine the agent's decision-making processes.

6) 补充信息: 还可包含辅助规划和推理的额外上下文。这可能包括从代理记忆中检索的历史数据 (详见第 5.6 节) 以及通过检索增强生成 (RAG) 方法获得的外部知识源 [188]，[189]。这些补充信息能提供有价值的见解，进一步优化代理的决策过程。

The construction of an effective prompt is foundational for the performance of LLM-powered GUI agents. By systematically incorporating aforementioned information, the agent ensures that the LLM is equipped with the necessary context and guidance to execute tasks accurately and efficiently.

构建有效的提示词是 LLM 驱动 GUI 代理性能的基础。通过系统地整合上述信息，代理确保 LLM 具备执行任务所需的上下文和指导，从而实现准确高效的操作。

5.4 Model Inference

5.4 模型推理

The constructed prompt is submitted to the LLM for inference, where the LLM is tasked with generating both a plan and the specific actions required to execute the user's request. This inference process is critical as it dictates how effectively the GUI agent will perform in dynamic environments. It typically involves two main components: planning and action inference, as well as the generation of complementary outputs. Figure 13 shows an example of the LLM's inference output.

构建好的提示词提交给 LLM 进行推理，LLM 负责生成执行用户请求所需的计划和具体操作。该推理过程至关重要，决定了 GUI 代理在动态环境中的表现。通常包括两个主要部分: 规划与动作推理，以及补充输出的生成。图 13 展示了 LLM 推理输出的示例。

5.4.1 Planning

5.4.1 规划

Successful execution of GUI tasks often necessitates a series of sequential actions, requiring the agent to engage in effective planning [52], [190]. Analogous to human cognitive processes, thoughtful planning is essential to organize tasks, schedule actions, and ensure successful completion [51], 191]. The LLM must initially conceptualize a long-term goal while simultaneously focusing on short-term actions to initiate progress toward that goal 192.

成功执行 GUI 任务通常需要一系列连续动作，要求代理进行有效规划 [52]，[190]。类似于人类认知过程，周密的规划对于组织任务、安排操作并确保顺利完成至关重要 [51]，[191]。LLM 必须首先构思长期目标，同时关注短期动作以推动目标实现 [192]。

To effectively navigate the complexity of multi-step tasks, the agent should decompose the overarching task into manageable subtasks and establish a timeline for their execution 193]. Techniques such as CoT reasoning [100] can be employed, enabling the LLM to develop a structured plan that guides the execution of actions. This plan, which can be stored for reference during future inference steps, enhances the organization and focus of the agent's activities.

为有效应对多步骤任务的复杂性，代理应将整体任务分解为可管理的子任务，并制定执行时间表 [193]。可采用链式推理 (CoT)[100] 等技术，使 LLM 制定结构化计划，指导动作执行。该计划可存储以供后续推理步骤参考，提升代理活动的组织性和专注度。

The granularity of planning may vary based on the nature of the task and the role of the agent [51]. For complex tasks, a hierarchical approach that combines global planning (identifying broad subgoals) with local planning (defining detailed steps for those subgoals) can significantly improve the agent's ability to manage long-term objectives effectively 194].

规划的粒度可能因任务性质和代理角色而异 [51]。对于复杂任务，结合全局规划 (识别广泛子目标) 与局部规划 (定义子目标的详细步骤) 的分层方法，能显著提升代理有效管理长期目标的能力 [194]。

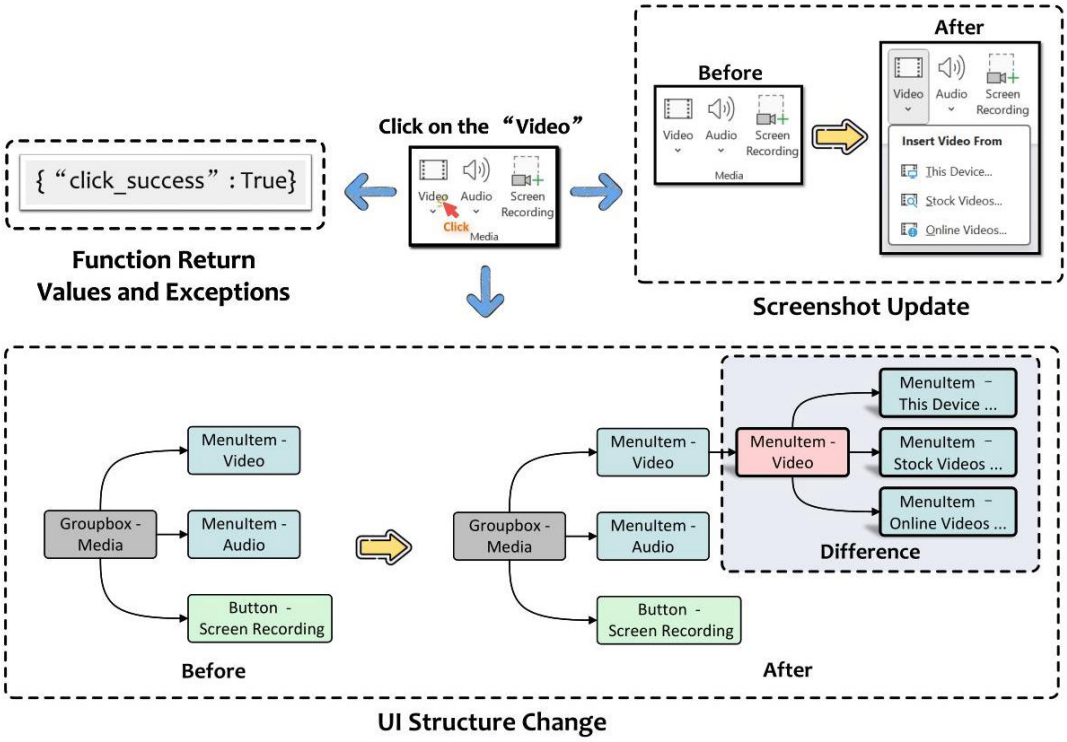


Fig. 11: Examples of various types of feedback obtained from a PowerPoint application environment.

图 11: 从 PowerPoint 应用环境获得的各种反馈类型示例。

5.4.2 Action Inference

5.4.2 动作推理

Action inference is the core objective of the inference stage, as it translates the planning into executable tasks. The inferred actions are typically expressed as function call strings, encompassing the function name and relevant parameters. These strings can be readily converted into real-world interactions with the environment, such as clicks, keyboard inputs, mobile gestures, or API calls. A detailed discussion of these action types is presented in Section 5.5

动作推理是推理阶段的核心目标，将规划转化为可执行任务。推断出的动作通常以函数调用字符串形式表达，包含函数名及相关参数。这些字符串可直接转换为与环境的实际交互，如点击、键盘输入、移动手势或 API 调用。第 5.5 节对这些动作类型进行了详细讨论。

The input prompt must include a predefined set of actions available for the agent to select from. The agent can choose an action from this set or, if allowed, generate custom code or API calls to interact with the environment [161]. This flexibility can enhance the agent’s adaptability to unforeseen circumstances; however, it may introduce reliability concerns, as the generated code may be prone to errors.

输入提示必须包含代理可选的预定义动作集。代理可从中选择动作，或在允许的情况下生成自定义代码或 API 调用以与环境交互 [161]。这种灵活性增强了代理对突发情况的适应能力，但可能带来可靠性问题，因为生成的代码可能存在错误。

10. https://developer.apple.com/library/archive/documentation/ AppleScript/Conceptual/AppleScriptLangGuide/introduction/ASLR_ intro.html

10. https://developer.apple.com/library/archive/documentation/ AppleScript/Conceptual/AppleScriptLangGuide/introduction/ASLR_ intro.html

---

11. https://www.macosxautomation.com/automator/

11. https://www.macosxautomation.com/automator/

12. https://docs.blender.org/manual/en/latest/sculpt_paint/sculpting/

12. https://docs.blender.org/manual/en/latest/sculpt_paint/sculpting/

introduction/gesture tools.html

introduction/gesture tools.html

13. https://developer.android.com/reference/android/speech/

13. https://developer.android.com/reference/android/speech/

SpeechRecognizer

SpeechRecognizer

14. https://developer.apple.com/documentation/sirikit/

14. https://developer.apple.com/documentation/sirikit/

15. https://pypi.org/project/pyperclip/

15. https://pypi.org/project/pyperclip/

16. https://clipboardjs.com/

16. https://clipboardjs.com/

17. https://developer.android.com/develop/sensors-and-location/

17. https://developer.android.com/develop/sensors-and-location/

sensors/sensors overview

sensors/sensors overview

18. https://learn.microsoft.com/en-us/previous-versions/office/

18. https://learn.microsoft.com/en-us/previous-versions/office/

office-365-api/

office-365-api/

19. https://developer.android.com/reference

19. https://developer.android.com/reference

20. https://developer.apple.com/ios/

20. https://developer.apple.com/ios/

21. https://learn.microsoft.com/en-us/windows/win32/api/

21. https://learn.microsoft.com/en-us/windows/win32/api/

22. https://developer.apple.com/library/archive/documentation/

22. https://developer.apple.com/library/archive/documentation/

Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.

Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.

23. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

23. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

24. https://axios-http.com/docs/api_intro

24. https://axios-http.com/docs/api_intro

25. https://platform.openai.com/docs/overview

25. https://platform.openai.com/docs/overview

5.4.3 Complementary Outputs

5.4.3 补充输出

In addition to planning and action inference, the LLM can also generate complementary outputs that enhance the agent's capabilities. These outputs may include reasoning processes that clarify the agent's decision-making (e.g., CoT reasoning), messages for user interaction, or communication with other agents or systems, or the status of the task (e.g., continue or finished). The design of these functionalities can be tailored to meet specific needs, thereby enriching the overall performance of the GUI agent.

除了规划和动作推断外，大型语言模型 (LLM) 还可以生成补充输出，以增强代理的能力。这些输出可能包括阐明代理决策过程的推理过程 (例如，链式思维 (CoT) 推理)、用于用户交互的消息、与其他代理或系统的通信，或任务状态 (例如，继续或完成)。这些功能的设计可以根据具体需求进行定制，从而丰富 GUI 代理的整体性能。

By effectively balancing planning and action inference while incorporating complementary outputs, agents can navigate complex tasks with a higher degree of organization and adaptability.

通过有效平衡规划与动作推断，同时结合补充输出，代理能够以更高的组织性和适应性应对复杂任务。

5.5 Actions Execution

5.5 动作执行

Following the inference process, a crucial next step is for the GUI agent to execute the actions derived from the inferred commands within the GUI environment and subsequently gather feedback. Although the term "GUI agent" might suggest a focus solely on user interface actions, the action space can be greatly expanded by incorporating various toolboxes that enhance the agent's versatility. Broadly, the actions available to GUI agents fall into three main categories: (i) UI operations [144], (ii) native API calls [196], and (iii) AI tools [197]. Each category offers unique advantages and challenges, enabling the agent to tackle a diverse range of

在推断过程之后，关键的下一步是 GUI 代理在 GUI 环境中执行从推断命令中得出的动作，并随后收集反馈。尽管“GUI 代理”一词可能暗示仅关注用户界面动作，但通过整合各种工具箱，可以大幅扩展动作空间，提升代理的多样性。总体而言，GUI 代理可用的动作主要分为三类:(i) 用户界面操作 [144]，(ii) 本地 API 调用 [196]，以及 (iii) 人工智能工具 [197]。每类动作各有优势和挑战，使代理能够更有效地处理多样化的

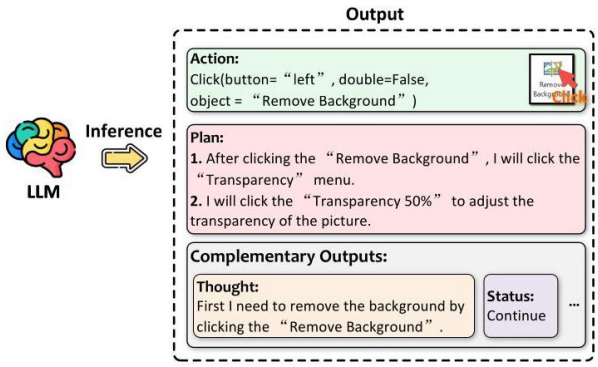


Fig. 13: An example of the LLM’s inference output in a GUI agent.

图 13:LLM 在 GUI 代理中的推断输出示例。

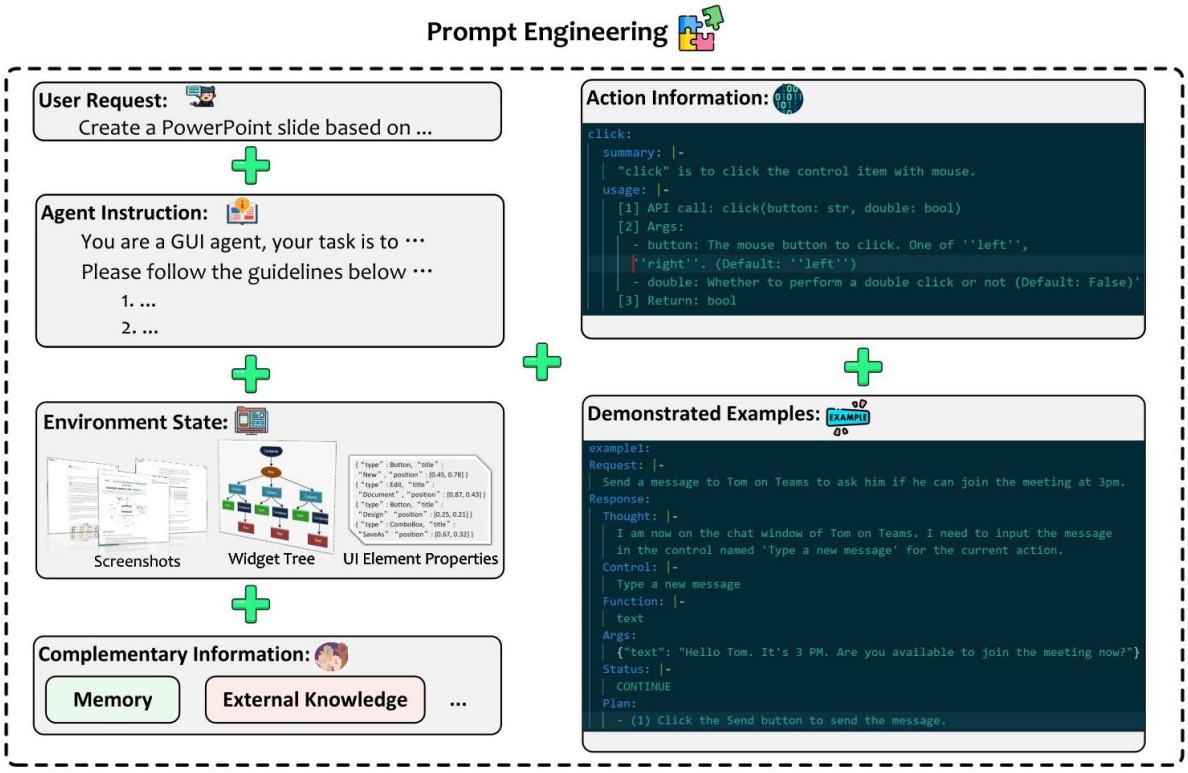


Fig. 12: A basic example of prompt construction in a LLM-brained GUI agent.

图 12: 基于 LLM 的 GUI 代理中提示构建的基本示例。

tasks more effectively. We summarize the various actions commonly used in GUI agents, categorized into distinct types, in Table 5 and provide detailed explanations of each category below.

任务。我们在表 5 中总结了 GUI 代理中常用的各种动作，按类别区分，并在下文详细说明每个类别。

5.5.1 UI Operations

5.5.1 用户界面操作

UI operations encompass the fundamental interactions that users typically perform with GUIs in software applications. These operations include various forms of input, such as mouse actions (clicks, drags, hovers), keyboard actions (key presses, combinations), touch actions (taps, swipes), and gestures (pinching, rotating). The specifics of these actions may differ across platforms and applications, necessitating a tailored approach for each environment.

用户界面操作涵盖用户在软件应用的 GUI 中通常执行的基本交互。这些操作包括各种输入形式，如鼠标动作 (点击、拖拽、悬停)、键盘动作 (按键、组合键)、触控动作 (轻触、滑动) 和手势 (捏合、旋转)。这些动作的具体细节可能因平台和应用而异，因此需要针对每个环境进行定制。

While UI operations form the foundation of agent interactions with the GUI, they can be relatively slow due to the sequential nature of these tasks. Each operation must be executed step by step, which can lead to increased latency, especially for complex workflows that involve numerous interactions. Despite this drawback, UI operations are crucial for maintaining a broad compatibility across various applications, as they leverage standard user interface elements and interactions.

尽管用户界面操作构成了代理与 GUI 交互的基础，但由于这些任务的顺序执行特性，操作速度可能较慢。每个操作必须逐步执行，这在涉及大量交互的复杂工作流程中可能导致延迟增加。尽管如此，用户界面操作对于保持跨各种应用的广泛兼容性至关重要，因为它们利用了标准的用户界面元素和交互方式。

5.5.2 Native API Calls

5.5.2 本地 API 调用

In contrast to UI operations, some applications provide native APIs that allow GUI agents to perform actions more efficiently. These APIs offer direct access to specific functionalities within the application, enabling the agent to execute complex tasks with a single command 198]. For instance, calling the Outlook API allows an agent to send an email in one operation, whereas using UI operations would require a series of steps, such as navigating through menus and filling out forms [199].

与 UI 操作相比，一些应用程序提供本地 API，允许 GUI 代理更高效地执行操作。这些 API 提供对应用程序中特定功能的直接访问，使代理能够通过单一命令执行复杂任务 [198]。例如，调用 Outlook API 可以让代理一次性发送电子邮件，而使用 UI 操作则需要一系列步骤，如导航菜单和填写表单 [199]。

While native APIs can significantly enhance the speed and reliability of action execution, their availability is limited. Not all applications or platforms expose APIs for external use, and developing these interfaces can require substantial effort and expertise. Consequently, while native APIs present a powerful means for efficient task completion, they may not be as generalized across different applications as UI operations.

虽然本地 API 能显著提升操作执行的速度和可靠性，但其可用性有限。并非所有应用程序或平台都对外开放 API，且开发这些接口可能需要大量的努力和专业知识。因此，尽管本地 API 是高效完成任务的强大手段，但它们在不同应用中的通用性不及 UI 操作。

TABLE 5: Overview of actions for GUI agents.

表 5:GUI 代理操作概览。

Action	Category	Original Executor	Examples	Platform	Environment	Toolkit
Mouse actions	UI Operations	Mouse	Click, scroll, hover, drag	Computer	Windows	UI Automation 6, Pywinauto
Mouse actions	UI Operations	Mouse	Click, scroll, hover, drag	Computer	macOS	AppleScript 10, Automator 11
Mouse actions	UI Operations	Mouse	Click, scroll, hover, drag	Web	Browser	Selenium, Puppeteer
Keyboard actions	UI Operations	Keyboard	Typing, key presses, shortcuts	Computer	Windows	UI Automation 6. Pywinauto
Keyboard actions	UI Operations	Keyboard	Typing, key presses, shortcuts	Computer	macOS	AppleScript 10, Automator 1
Keyboard actions	UI Operations	Keyboard	Typing, key presses, shortcuts	Web	Browser	Selenium. Puppeteer
Touch actions	UI Operations	Touchscreen	Tap, swipe, pinch, zoom	Mobile	Android	Appium, UIAu-tomator
Touch actions	UI Operations	Touchscreen	Tap, swipe, pinch, zoom	Mobile	iOS	Appium, XCUITest
Gesture actions	UI Operations	User hand	Rotate, multi-finger gestures	Mobile	Android, iOS	Appium, G tureTools 12
Voice commands	UI Operations	User voice	Speech input, voice commands	Mobile	Android	SpeechRecognize 13
Voice commands	UI Operations	User voice	Speech input, voice commands	Mobile	iOS	SiriKit 14
Clipboard operations	UI Operations	System clipboard	Copy, paste	Cross-platform	Cross-OS	Pyperclip Clipboard.js 16
Screen interactions	UI Operations	User	Screen rotation, shake	Mobile	Android, iOS	Device sensors APIs 17
Shell Commands	Native API Calls	Command Line Interface	File manipulation, system operations, script execution	Computer	Unix/Linux, macOS	Bash, Terminal
Application APIs	Native API Calls	Application APIs	Send email, create document. fetch data	Computer	Windows	Microsoft Office COM APIs 18
Application APIs	Native API Calls	Application APIs	Access calendar. send messages	Mobile	Android	Android SDK APIs 19
Application APIs	Native API Calls	Application APIs	Access calendar, send messages	Mobile	iOS	iOS SDK APIs 20
System APIs	Native API Calls	System APIs	File operations, network requests	Computer	Windows	Win32 API 21
System APIs	Native API Calls	System APIs	File operations, network requests	Computer	macOS	Cocoa APIs 22
Web APIs	Native API Calls	Web Services	Fetch data, submit forms	Web	Browser	Fetch API [23] . Axios 24
AI Models	AI Tools	AI Models	Screen understanding, summarization, image generation	Cross-platform	Cross-OS	DALL·E 1951 OpenAI APIs 26

操作	类别	原始执行者	示例	平台	环境	工具包
鼠标操作	用户界面操作	鼠标	点击，滚动，悬停，拖拽	计算机	Windows	UI Automation 6，Pywinauto
鼠标操作	用户界面操作	鼠标	点击，滚动，悬停，拖拽	计算机	macOS	AppleScript 10，Automator 11
鼠标操作	用户界面操作	鼠标	点击，滚动，悬停，拖拽	网页	浏览器	Selenium，Puppeteer
键盘操作	用户界面操作	键盘	输入，按键，快捷键	计算机	Windows	UI Automation 6，Pywinauto
键盘操作	用户界面操作	键盘	输入，按键，快捷键	计算机	macOS	AppleScript 10，Automator 1
键盘操作	用户界面操作	键盘	输入，按键，快捷键	网页	浏览器	Selenium，Puppeteer
触控操作	用户界面操作	触摸屏	点击，滑动，捏合，缩放	移动端	Android	Appium，UIAutomator
触控操作	用户界面操作	触摸屏	点击，滑动，捏合，缩放	移动端	iOS	Appium，XCUITest
手势操作	用户界面操作	用户手势	旋转，多指手势	移动端	Android，iOS	Appium，GestureTools 12
语音命令	用户界面操作	用户语音	语音输入，语音命令	移动端	Android	SpeechRecognize 13
语音命令	用户界面操作	用户语音	语音输入，语音命令	移动端	iOS	SiriKit 14
剪贴板操作	用户界面操作	系统剪贴板	复制，粘贴	跨平台	跨操作系统	Pyperclip，Clipboard.js 16
屏幕交互	用户界面操作	用户	屏幕旋转，摇晃	移动端	Android，iOS	设备传感器 API 17
Shell 命令	本地 API 调用	命令行界面	文件操作，系统操作，脚本执行	计算机	Unix/Linux，macOS	Bash，终端
应用程序 API	本地 API 调用	应用程序 API	发送邮件，创建文档，获取数据	计算机	Windows	Microsoft Office COM API 18
应用程序 API	本地 API 调用	应用程序 API	访问日历，发送消息	移动端	Android	Android SDK API 19
应用程序 API	本地 API 调用	应用程序 API	访问日历，发送消息	移动端	iOS	iOS SDK 应用程序接口 20
系统应用程序接口	本地 API 调用	系统应用程序接口	文件操作，网络请求	计算机	Windows	Win32 应用程序接口 21
系统应用程序接口	本地 API 调用	系统应用程序接口	文件操作，网络请求	计算机	macOS	Cocoa 应用程序接口 22
Web 应用程序接口	本地 API 调用	Web 服务	获取数据，提交表单	网页	浏览器	Fetch API [23] 。Axios 24
人工智能模型	人工智能工具	人工智能模型	屏幕理解，摘要，图像生成	跨平台	跨操作系统	DALL·E 1951 OpenAI 应用程序接口 26

5.5.3 AI Tools

5.5.3 AI 工具

The integration of AI tools into GUI agents represents a transformative advancement in their capabilities. These tools can assist with a wide range of tasks, including content summarization from screenshots or text, document enhancement, image or video generation (e.g., calling ChatGPT [11], DALL-E 195]), and even invoking other agents or Copilot tools for collaborative assistance. The rapid development of generative AI technologies enables GUI agents to tackle complex challenges that were previously beyond their capabilities.

将 AI 工具集成到 GUI 代理中代表了其能力的变革性进步。这些工具可以协助完成广泛的任务, 包括从截图或文本中提取内容摘要、文档增强、图像或视频生成 (例如调用 ChatGPT[11]、DALL-E[195]), 甚至调用其他代理或 Copilot 工具进行协作辅助。生成式 AI 技术的快速发展使 GUI 代理能够应对此前超出其能力范围的复杂挑战。

By incorporating AI tools, agents can extend their functionality and enhance their performance in diverse contexts. For example, a GUI agent could use an AI summarization tool to quickly extract key information from a lengthy document or leverage an image generation tool to create custom visuals for user presentations. This integration not only streamlines workflows but also empowers agents to deliver high-quality outcomes in a fraction of the time traditionally required.

通过整合 AI 工具，代理可以扩展其功能并提升在多种场景下的表现。例如，GUI 代理可以使用 AI 摘要工具快速提取冗长文档中的关键信息，或利用图像生成工具为用户演示创建定制视觉内容。这种集成不仅简化了工作流程，还使代理能够在传统所需时间的一小部分内交付高质量成果。

5.5.4 Summary

5.5.4 总结

An advanced GUI agent should adeptly leverage all three categories of actions: UI operations for broad compatibility, native APIs for efficient execution, and AI tools for enhanced capabilities. This multifaceted approach enables the agent to operate reliably across various applications while maximizing efficiency and effectiveness. By skillfully navigating these action types, GUI agents can fulfill user requests more proficiently, ultimately leading to a more seamless and productive user experience.

一个先进的 GUI 代理应熟练利用三类操作: 广泛兼容的 UI 操作、高效执行的本地 API 以及增强能力的 AI 工具。这种多维度方法使代理能够在各种应用中可靠运行，同时最大化效率和效果。通过巧妙驾驭这些操作类型，GUI 代理能够更高效地满足用户需求，最终带来更流畅且富有成效的用户体验。

5.6 Memory

5.6 记忆

For a GUI agent to achieve robust performance in complex, multi-step tasks, it must retain memory, enabling it to manage states in otherwise stateless environments. Memory allows the agent to track its prior actions, their outcomes, and the task's overall status, all of which are crucial for informed decision-making in subsequent steps [200]. By establishing continuity, memory transforms the agent from a reactive system into a proactive, stateful one, capable of self-adjustment based on accumulated knowledge. The agent's memory is generally divided into two main types: Short-Term Memory [201] and Long-Term Memory [202]. We show an overview of different types of memory in GUI agents in Table 6

为了使 GUI 代理在复杂的多步骤任务中表现稳健，必须具备记忆能力，从而在本质上无状态的环境中管理状态。记忆使代理能够追踪先前的操作、其结果及任务的整体状态，这些对于后续步骤的明智决策至关重要 [200]。通过建立连续性，记忆将代理从反应式系统转变为主动的、有状态系统，能够基于积累的知识自我调整。代理的记忆通常分为两大类: 短期记忆 [201] 和长期记忆 [202]。我们在表 6 中展示了 GUI 代理中不同类型记忆的概览。

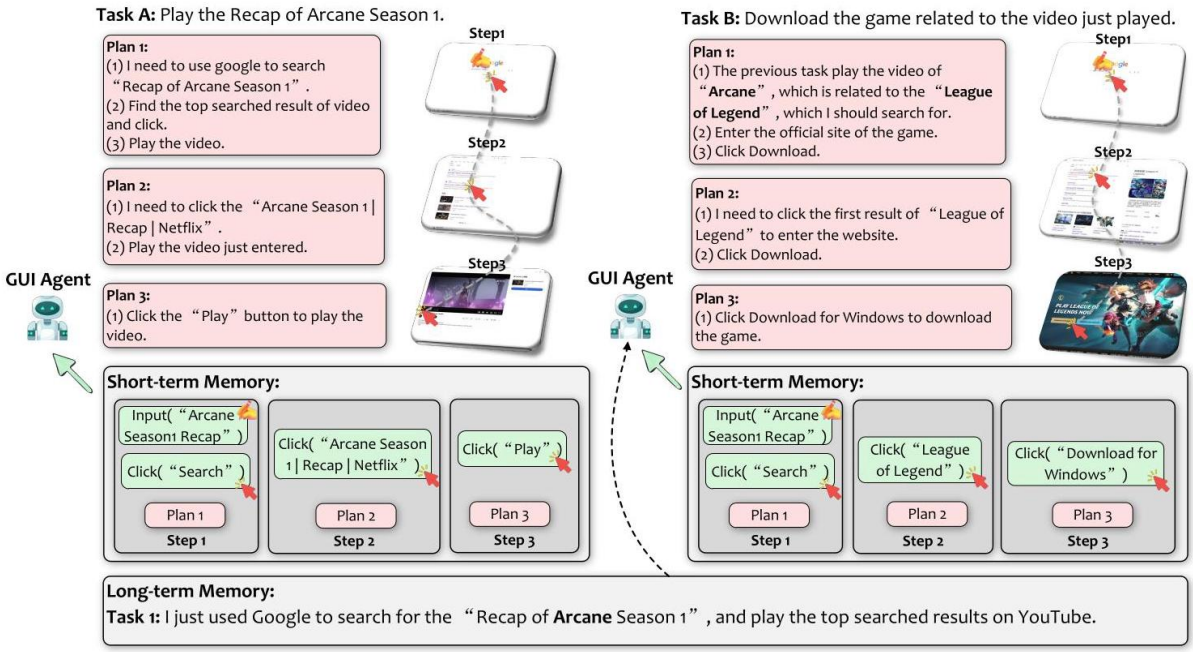


Fig. 14: Illustration of short-term memory and long-term memory in an LLM-brained GUI agent.

图 14: 具备大型语言模型 (LLM) 核心的 GUI 代理中短期记忆与长期记忆的示意图。

TABLE 6: Summary of memory in GUI agents.

表 6:GUI 代理中记忆的总结。

Memory Element	Memory Type	Description	Storage Medium/Method
Action	Short-term	Historical actions trajectory taken in the environment	In-memory, Context window
Plan	Short-term	Plan passed from previous step	In-memory, Context window
Execution Results	Short-term	Return values, error traces, and other environmental feedback	In-memory, Context window
Environment State	Short-term	Important environment state data, e.g., UI elements	In-memory, Context window
Self-experience	Long-term	Task completion trajectories from historical tasks	Database, Disk
Self-guidance	Long-term	Guidance and rules summarized from historical trajectories	Database, Disk
External Knowledge	Long-term	Other external knowledge sources aiding task completion	External Knowledge Base
Task Success Metrics	Long-term	Metrics from task success or failure rates across sessions	Database, Disk

记忆元素	记忆类型	描述	存储介质/方法
动作	短期	环境中采取的历史动作轨迹	内存中，上下文窗口
计划	短期	来自上一步的计划	内存中，上下文窗口
执行结果	短期	返回值、错误追踪及其他环境反馈	内存中，上下文窗口
环境状态	短期	重要的环境状态数据，例如用户界面元素	内存中，上下文窗口
自我经验	长期	历史任务的完成轨迹	数据库，磁盘
自我指导	长期	从历史轨迹总结的指导和规则	数据库，磁盘
外部知识	长期	辅助任务完成的其他外部知识来源	外部知识库
任务成功指标	长期	跨会话任务成功或失败率的指标	数据库，磁盘

5.6.1 Short-Term Memory

5.6.1 短期记忆

Short-Term Memory (STM) provides the primary, ephemeral context used by the LLM during runtime [203]. STM stores information pertinent to the current task, such as recent plans, actions, results, and environmental states, and continuously updates to reflect the task's ongoing status. This memory is particularly valuable in multi-step tasks, where each decision builds on the previous one, requiring the agent to maintain a clear understanding of the task's trajectory. As illustrated in Figure 14 during the completion of independent tasks, the task trajectory, comprising actions and plans-is stored in the STM. This allows the agent to track task progress effectively and make more informed decisions.

短期记忆 (STM) 为大型语言模型 (LLM) 在运行时提供主要的、短暂的上下文 [203]。STM 存储与当前任务相关的信息，如最近的计划、动作、结果和环境状态，并持续更新以反映任务的进行状态。这种记忆在多步骤任务中尤为重要，因为每个决策都建立在前一个决策之上，要求代理保持对任务轨迹的清晰理解。如图 14 所示，在完成独立任务时，包含动作和计划的任务轨迹被存储在 STM 中。这使得代理能够有效跟踪任务进展并做出更明智的决策。

However, STM is constrained by the LLM's context window, limiting the amount of information it can carry forward. To manage this limitation, agents can employ selective memory management strategies, such as selectively discarding or summarizing less relevant details to prioritize the most impactful information. Despite its limited size, STM is essential for ensuring coherent, contextually aware interactions and supporting the agent's capacity to execute complex workflows with immediate, relevant feedback.

然而，STM 受限于 LLM 的上下文窗口，限制了其能够携带的信息量。为应对这一限制，代理可以采用选择性记忆管理策略，例如有选择地丢弃或总结不太相关的细节，以优先保留最重要的信息。尽管容量有限，STM 对于确保连贯且具上下文感知的交互至关重要，并支持代理在执行复杂工作流程时获得即时且相关的反馈。

5.6.2 Long-Term Memory

5.6.2 长期记忆

Long-Term Memory (LTM) serves as an external storage repository for contextual information that extends beyond the immediate runtime 204. Unlike STM, which is transient, LTM retains historical task data, including previously completed tasks, successful action sequences, contextual tips, and learned insights. LTM can be stored on disk or in a database, enabling it to retain larger volumes of information than what is feasible within the LLM's immediate context window. In the example shown in Figure 14 when the second task requests downloading a game related to the previous task, the agent retrieves relevant information from its LTM. This enables the agent to accurately identify the correct game, facilitating efficient task completion.

长期记忆 (LTM) 作为一个外部存储库，用于保存超出即时运行时的上下文信息 [204]。与短暂的 STM 不同，LTM 保留历史任务数据，包括先前完成的任务、成功的动作序列、上下文提示和学习到的见解。LTM 可以存储在磁盘或数据库中，使其能够保存比 LLM 即时上下文窗口更大量的信息。如图 14 所示，当第二个任务请求下载与前一个任务相关的游戏时，代理从其 LTM 中检索相关信息，从而准确识别正确的游戏，促进高效完成任务。

LTM contributes to the agent's self-improvement over time by preserving examples of successful task trajectories, operational guidelines, and common interaction patterns. When approaching a new task, the agent can leverage RAG techniques to retrieve relevant historical data, which enhances its ability to adapt strategies based on prior success. This is similar to the lifelong learning [53], which makes LTM instrumental in fostering an agent's capacity to "learn" from experience, enabling it to perform tasks with greater accuracy and efficiency as it accumulates insights across sessions. For instance, [205] provides an illustrative example of using past task trajectories stored in memory to guide and enhance future decision-making, a technique that is highly adaptable for GUI agents. It also enables better personalization by retaining information about previous tasks.

LTM 通过保存成功任务轨迹的示例、操作指南和常见交互模式，促进代理随时间的自我提升。在处理新任务时，代理可以利用检索增强生成 (RAG) 技术检索相关历史数据，增强其基于先前成功调整策略的能力。这类似于终身学习 [53]，使 LTM 成为促进代理“从经验中学习”能力的关键，使其随着跨会话积累的见解，能够更准确高效地执行任务。例如，[205] 提供了一个利用存储在记忆中的过去任务轨迹指导和提升未来决策的示例，这一技术对图形用户界面 (GUI) 代理尤为适用。它还通过保留先前任务的信息，实现更好的个性化。

5.7 Advanced Enhancements

5.7 高级增强

While most LLM-brained GUI agents incorporate fundamental components such as perception, planning, action execution, and memory, several advanced techniques have been developed to significantly improve the reasoning and overall capabilities of these agents. Here, we outline shared advancements widely adopted in research to guide the development of more specialized and capable LLM-brained GUI agents.

虽然大多数基于 LLM 的 GUI 代理包含感知、规划、动作执行和记忆等基本组件，但已经开发出多种高级技术，显著提升这些代理的推理能力和整体性能。这里我们概述了研究中广泛采用的共享进展，以指导更专业、更强大的基于 LLM 的 GUI 代理的开发。

5.7.1 Computer Vision-Based GUI Grounding

5.7.1 基于计算机视觉的 GUI 定位

Although various tools (Section 4) enable GUI agents to access information like widget location, captions, and properties, certain non-standard GUIs or widgets may not adhere to these tools' protocols [243], rendering their information inaccessible. Additionally, due to permission management, these tools are not always usable. Such incomplete information can present significant challenges for GUI agents, as the LLM may need to independently locate and interact with required widgets by estimating their coordinates to perform actions like clicking—a task that is inherently difficult without precise GUI data.

尽管各种工具 (第 4 节) 使 GUI 代理能够访问控件位置、标题和属性等信息，但某些非标准 GUI 或控件可能不遵循这些工具的协议 [243]，导致其信息无法获取。此外，由于权限管理，这些工具并非总是可用。这种信息不完整对 GUI 代理构成重大挑战，因为 LLM 可能需要通过估算控件坐标独立定位并与所需控件交互以执行点击等操作——在缺乏精确 GUI 数据的情况下，这是一项本质上困难的任务。

CV models offer a non-intrusive solution for GUI grounding directly from screenshots, enabling the detection, localization, segmentation, and even functional estimation of widgets [103], 244-246]. This approach allows agents to interpret the visual structure and elements of the GUI without relying on system-level tools or internal metadata, which may be unavailable or incomplete. CV-based GUI parsing provides agents with valuable insights into interactive components, screen layout, and widget functionalities based solely on visual cues, enhancing their ability to recognize and act upon elements on the screen. Figure 10 provides an illustrative example of how a CV-based GUI parser works. While standard API-based detection captures predefined widgets, the CV model can identify additional elements, such as thumbnails and canvases, which may not have explicit API representations in the PowerPoint interface. This enhances widget recognition, allowing the agent to detect components beyond the scope of API detection. We show an overview of related GUI grounding models and benchmarks in Table 7 8 and 9

计算机视觉 (CV) 模型提供了一种非侵入式的 GUI 定位解决方案，直接从截图中实现控件的检测、定位、分割甚至功能估计 [103], 244-246]。该方法使代理能够基于视觉线索解读 GUI 的视觉结构和元素，而无需依赖可能不可用或不完整的系统级工具或内部元数据。基于 CV 的 GUI 解析为代理提供了关于交互组件、屏幕布局和控件功能的宝贵见解，提升了其识别和操作屏幕元素的能力。图 10 展示了基于 CV 的 GUI 解析器的示例。标准的基于 API 的检测捕获预定义控件，而 CV 模型能够识别额外元素，如缩略图和画布，这些在 PowerPoint 界面中可能没有明确的 API 表示。这增强了控件识别能力，使代理能够检测超出 API 检测范围的组件。相关 GUI 定位模型和基准的概览见表 7、8 和 9。

A notable example is OmniParser [184], which implements a multi-stage parsing technique involving a fine-tuned model for detecting interactable icons, an OCR module for extracting text, and an icon description model that generates localized semantic descriptions for each UI element. By integrating these components, OmniParser constructs a structured representation of the GUI, enhancing an agent's understanding of interactive regions and functional elements. This comprehensive parsing strategy has shown to significantly improve GPT-4V's screen comprehension and interaction accuracy.

一个显著的例子是 OmniParser[184]，它实现了多阶段解析技术，包括用于检测可交互图标的微调模型、用于提取文本的 OCR 模块以及生成每个 UI 元素本地语义描述的图标描述模型。通过整合这些组件，OmniParser 构建了 GUI 的结构化表示，增强了代理对交互区域和功能元素的理解。这种综合解析策略显著提升了 GPT-4V 对屏幕的理解和交互准确性。

Such CV-based GUI grounding layers provide critical grounding information that significantly enhances an agent's ability to interact accurately and intuitively with diverse GUIs. This is particularly beneficial for handling custom or nonstandard elements that deviate from typical accessibility protocols. Additionally, prompting methods like iterative narrowing have shown promise in improving the widget grounding capabilities of VLMs [208]. Together, these approaches pave the way for more adaptable and resilient GUI agents, capable of operating effectively across a broader range of screen environments and application contexts.

基于计算机视觉 (CV) 的 GUI 定位层提供了关键的定位信息，显著增强了智能体准确且直观地与多样化 GUI 交互的能力。这对于处理偏离典型无障碍协议的自定义或非标准元素尤为有益。此外，诸如迭代缩小 (iterative narrowing) 等提示方法在提升视觉语言模型 (VLMs) 的小部件定位能力方面展现出潜力 [208]。这些方法共同为更具适应性和鲁棒性的 GUI 智能体铺平了道路，使其能够在更广泛的屏幕环境和应用场景中高效运行。

Several works have introduced benchmarks to evaluate the GUI grounding capabilities of models and agents. For instance, ScreenSpot [25] serves as a pioneering benchmark designed to assess the GUI grounding performance of LLM-powered agents across diverse platforms, including iOS, Android, macOS, Windows, and web environments. It features a dataset with over 600 screenshots and 1,200 instructions, focusing on complex GUI components such as widgets and icons. This benchmark emphasizes the importance of GUI grounding in enhancing downstream tasks like web automation and mobile UI interaction. Building upon this, ScreenSpot-Pro [241] extends the scope to more professional, high-resolution environments. This evolved version includes 1,581 tasks with high-quality annotations, encompassing domains such as software development, creative tools, CAD, scientific applications, and office productivity. Key features of ScreenSpot-Pro include authentic high-resolution screenshots and meticulous annotations provided by domain experts.

多项研究提出了用于评估模型和智能体 GUI 定位能力的基准测试。例如，ScreenSpot [25] 作为开创性基准，旨在评估基于大型语言模型 (LLM) 的智能体在 iOS、Android、macOS、Windows 及网页环境等多平台上的 GUI 定位表现。其数据集包含 600 多张截图和 1200 条指令，重点关注复杂的 GUI 组件，如小部件和图标。该基准强调 GUI 定位在提升下游任务 (如网页自动化和移动 UI 交互) 中的重要性。在此基础上，ScreenSpot-Pro [241] 将范围扩展至更专业的高分辨率环境。该升级版本包含 1581 个任务，配备高质量注释，涵盖软件开发、创意工具、计算机辅助设计 (CAD)、科学应用及办公效率等领域。ScreenSpot-Pro 的关键特征包括真实的高分辨率截图和由领域专家提供的细致注释。

These benchmarks provide critical evaluation criteria for assessing GUI grounding capabilities, thereby advancing the development of GUI agents for improved GUI understanding and interaction.

这些基准为评估 GUI 定位能力提供了关键的评价标准，推动了 GUI 智能体在 GUI 理解与交互方面的发展。