庭宇科技 | 铸基计划 高质量数字化转型推进行动 | 甲子光年 JAZZYEAR

跨 OS GUI 智能体基础设施白皮书

—— 重新定义人机交互自动化

执行摘要

进入2026年，OpenClaw的横空出世，宣告全球人工智能正式从Chat时代走到了Act时代，各主要厂商不再只做Agent，而是推动Agent成为AI时代新的超级入口，GUI Agent也成为这一探索中最重要的路径。GUI Agent正在引发继“命令行”到“图形界面”之后的第三次人机交互革命，其核心是通过UI直接进行操作，从“人操作机器”转向“机器理解并执行人的意图”，使操作系统回归“用户意图执行者”的本质。这一方面意味着当前以应用为核心的“数据孤岛”将彻底被打破，另一方面意味着Agentic OS将成为新的超级入口。这些改变将彻底颠覆现有的产业生态格局，带来万亿级别的市场机遇。

庭宇科技是全球领先的边缘智算基础设施服务商，也是近年国内垂直领域增长最快的边缘云厂商之一。2024年，庭宇科技就开始了在GUI Agent领域的探索，并且在2025年10月发布了GUI Agent产品Lybic，成为国内发布的首个基于边缘智算架构的GUI Agent产品。Lybic补足了庭宇科技在Agentic OS领域的拼图，使庭宇科技形成了以“边缘云-GUI Agent-云手机/电脑”为主的产品体系，初步构建了以边缘智算为核心的AI全栈基础设施生态。Lybic以庭宇科技的边缘智算为底座，以沙箱为保障的运行环境，配合“全模型+全工具链”的开放生态，形成了“毫秒级延时、高安全保障、全场景适应”的产品优势，有效解决了GUI Agent运行过程中面临的延迟长、精度差、安全保障低的痛点。

《跨OS GUI智能体基础设施白皮书》由庭宇科技和铸基计划联合发布，提出了对GUI Agent的发展现状、技术路径、落地场景及未来前景提出的研判与思考。希望通过本白皮书，与全球从业者共同探讨这一变革性技术的未来，并致力于通过深耕GUI Agent市场，为推动全球通用人工智能的发展贡献庭宇科技力量，实现庭宇科技以“让算力无处不在，让智能触手可及”的使命。

从“连接信息”到“执行意图”：GUI Agent重新定义人机交互自动化

上世纪五十年代，达特茅斯会议的召开标志着人工智能概念正式诞生。从此以后，通过语言、视觉等自然方式对计算机进行操作，实现真正的智能化和自动化人机互动，就成了人工智能行业从业者跨越七十年的共同目标。2022年ChatGPT的横空出世，让人类通过自然语言与机器进行交互并且执行任务成为可能，这也直接催生了最近三年的AI Agent创业热潮。

随着产业探索的深入，从业者逐渐意识到，当前以API为核心路径的Agent路线存在难以逾越的障碍，一是覆盖率瓶颈，API Agent高度依赖API接口，初步估计全球当前仅有不到5%的软件开放了完整的API接口，这使得Agent在数量巨大的“黑盒”软件面前寸步难行；二是认知维度确缺失，多数API Agent仅能在数据层面进行交互，丢失了界面布局、图标隐喻等关键的视觉上下文信息；三是跨生态协作割裂，受限于接口壁垒，传统Agent往往沦为单一软件内的“半自动”工具，难以处理跨平台、跨应用的长链路复杂任务。这就造成了当前Agent多数是“半自动”智能体，重复、繁琐的跨软件操作还需要人工完成。总的来说，以API为主要路径的Agent仍然没办法克服当前计算机行业发展留下来的弊病，即无法对碎片化的数据进行多模态、大规模、跨平台、自动化的调用和整理，并且最终实现智能化的输出。

在短期无法重构全球软件生态的前提下，直接通过图形界面完成感知与操作，成为绕开接口壁垒、实现规模化自动化的重要实现路径。GUI Agent是一种基于多模态大模型，能够模拟人类用户，通过视觉感知和模拟操作，直接与多端图形用户界面进行交互的智能体。它的优势在于摆脱了对API接口和RPA脚本的依赖，通过强大的视觉语义理解，打破了应用间的数据围墙，实现了真正的跨App、跨平台、跨生态操作。GUI Agent的出现，将智能设备从刻板的“代码执行者”进化为灵活的“意图代理人”，从根本上重塑了人机交互范式，是人类通往通用人工智能道路上的里程碑式跨越。

进入2025年，全球GUI Agent赛道全面爆发，Anthropic、微软、OpenAI、字节跳动、阿里巴巴等国际国内巨头都推出了GUI Agent产品，作为全球领先的边缘智算基础设施服务商，庭宇科技也于2025年推出了基于边缘算力架构的GUI Agent基础设施平台产品Lybic，且收获了不俗的市场反响。

本白皮书基于庭宇科技近年在GUI Agent领域的思考、探索和实践，深度剖析了GUI Agent的市场格局、技术演进路径及关键落地场景。我们希望通过本白皮书，为全球从业者提供一份可参考的“落地指南”，并致力于通过持续深耕边缘AI基础设施，推动通用人工智能早日普惠千行百业。

CONTENTS

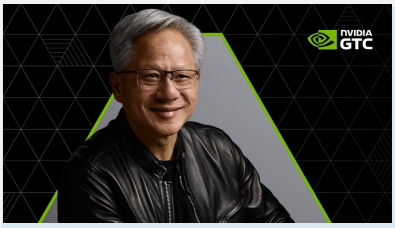
目 录

第一章

GUI Agent，重新定义人机交互自动化

1.1 Agent，人工智能走向应用的关键阶段

进入2026年，全球科技领袖和科技公司已经达成了一致：AI的下半场不再是卷模型参数，而是卷Agent的落地与执行。在刚刚过去的2026年NVIDIA GTC大会上，黄仁勋将Agentic AI推向了绝对的C位，并将其定义为“下一代计算平台”。从2025年开始，吴恩达也将他的工作重点全面转向了“Agentic Workflows”。OpenAI CEO山姆·奥特曼、百度CEO李彦宏也认为，人工智能的未来是Agent带来的智能体落地。Agent给世界带来的改变，绝不是发明了一种“更聪明的聊天软件”，而是掀起了一场自PC和互联网诞生以来，最彻底的“人机交互范式与商业软件形态”的底层革命。



“每一家IT公司，每一家企业，每一家SaaS公司，最终都将演变成一家AaaS（Agentic-as-a-Service）公司。”

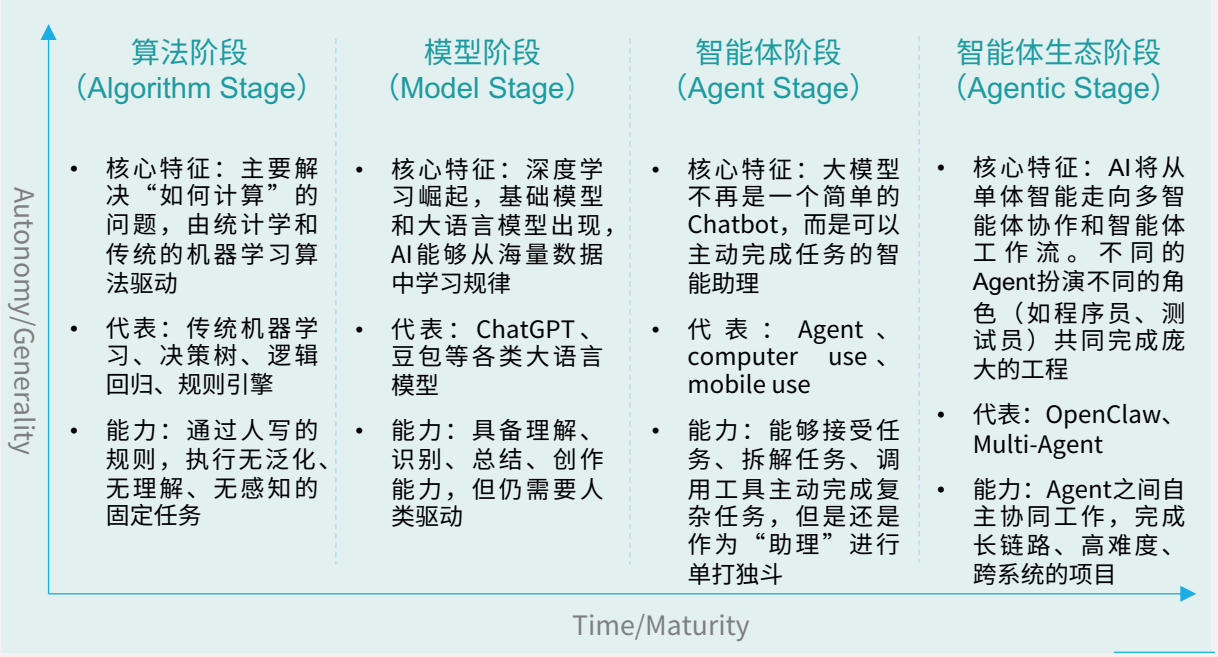
——黄仁勋
2026年3月，GTC大会

“我认为AI智能体工作流将推动人工智能的巨大进步——甚至可能超过下一代基础模型。”

——吴恩达
2024年3月，《The Batch》专栏

随着各种Agent和OpenClaw的横空出世，人工智能的发展和演变路径逐步变得清晰。以自主能力和泛化能力为评价标准，人工智能的发展经过算法阶段、模型阶段、智能体阶段、智能体生态阶段，当前正在经历从模型阶段向智能体阶段的快速发展和跨越。随着未来以OpenClaw为代表的各类Multi-Agent产品的出现，人工智能最终将进入多智能体组成的智能体生态阶段。智能体将根据需求，主动完成长链路、跨系统的高难度项目，改变整个人类社会的协作形态。

图：人工智能的演进阶段



资料来源：甲子光年整理

1.2 Agent，具备感知-决策-执行闭环的智能系统

“Agent”起源于拉丁语“agens”，词根“ag-”的核心涵义是“to do” “to act”，是拉丁语中极具行动力的一个词汇，英语中“act、action、active、agenda”等词均源于此。“Agent”这个词首次出现在英语中可追溯至14晚期，最初作为法律和哲学的术语被引用，原意为“行动者”、“代理人”。20世纪中期，“Agent”一词开始被应用于计算机科学领域，用于描述“能自主执行特定任务的软件程序”，如“network agent” “intelligent agent”等。

在AI Agent中，“Agent”一词体现了AI Agent的特点，即既为“自主行动者”，也是“代表用户行事的代理人”，是“具备感知-决策-执行闭环的智能系统”。Agent的核心在于自主性的增强，这种增强的核心要义是可以去独立完成一个工作节点，在某个工作节点几乎可以减少人类的审核，让整个事件的流程在此刻完成闭环，同时时间和金钱成本降到最低。评价一个AI Agent的重要标准是，在流程上的节点上完成了什么程度的自动化。

本白皮书对AI Agent采用谷歌的定义，即AI Agent可被定义为一种应用：它通过观察世界并利用手头可用的工具采取行动，以期达成既定目标。智能体具有自主性，可在无需人工干预的情况下独立运行，尤其当被赋予明确目标或任务时。智能体还可主动推进目标实现进程：即便未获得人类的明确指令集，也能自行推理下一步应采取何种行动以最终达成目标。

图：AI Agent涵义解释

图：Agent Runtime逻辑架构图

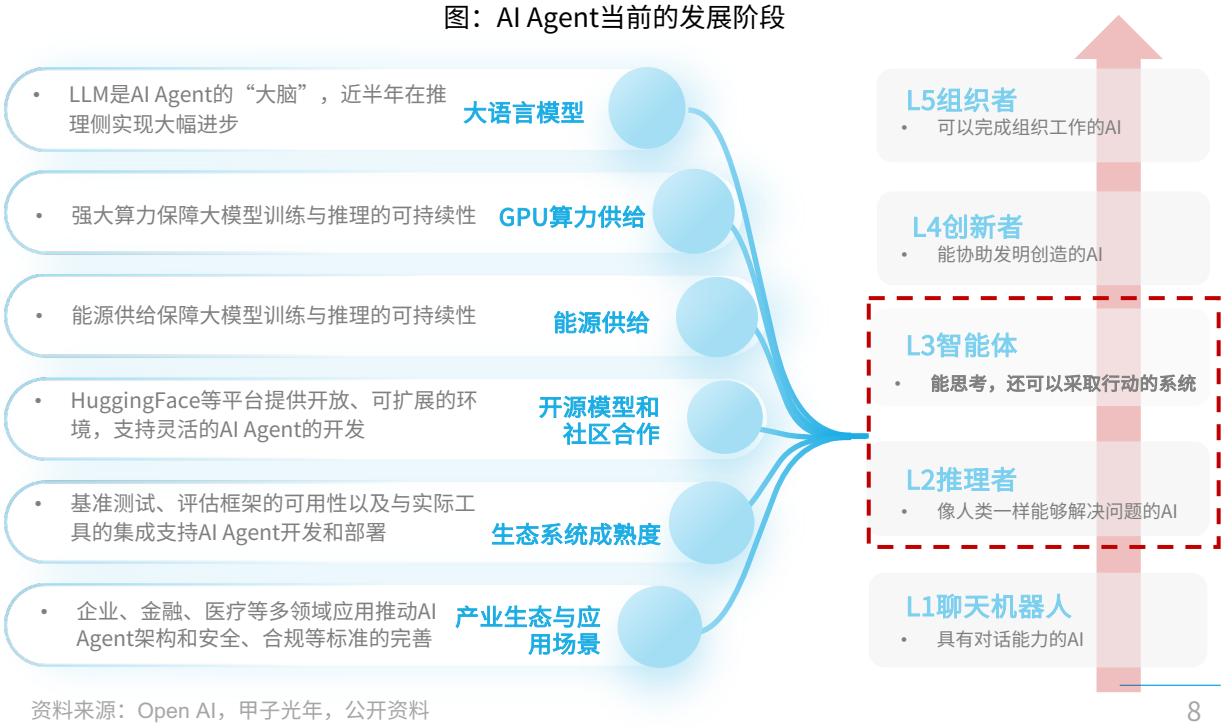
资料来源：Google，甲子光年，公开资料

随着大模型的进化，工具和核心架构技术的进步，AI Agent实现了快速的发展，核心路径一是通过多智能体分工和工作流编排，降低对模型发展能力的依赖，二是从文本向多模态进化，推动实现“全域感知”，增强对现实世界的“感知精准度”，三是标准化协议和多模态融合突破，共同降低集成成本。从行业落地情况看，当前可稳定交付的Agent能力主要集中在L2–L3阶段。跨系统执行能力不足是阻碍Agent进一步向组织级与创新级演进的关键因素。

表：AI Agent核心变迁路径

项目	2023年：基础工具调用	2024年：工具增强阶段	2025年：自主智能阶段
核心架构	• 单体(Monolithic)	• 链式(Chain)	• 图/网络 (Graph/MAS)
核心特征	• LLM与外部工具简单连接，实现"能做事"的基础能力	• 具备多步骤任务规划和动态调整能力，向"类人决策"迈进	• "感知-决策-执行-反思"闭环形成，具备类人级任务执行能力
基础模型能力	• LLM仅能处理文本，理解能力有限，推理深度不足	• 多模态模型突破，能解析图像、视频、音频 • 树状思考支持并行探索多条解决方案，推理计策能力提升	• "慢思考"能力，能自主决定何时搜索、分析、总结 • 模态实现融合：文本、图像、声音的统一表征，跨模态理解准确率提升
技术特征	• LLM从"孤岛智能"到"连接物理世界"的转变，能调用API、搜索、文件操作等工具 • 通过"思维链"(CoT)技术增强推理能力，使Agent能展示思考过程	• 长上下文支持(10万+token)，支持复杂任务链的记忆与推理 • 环境反馈驱动的动态调整，遇错能重试或切换策略 • 多模态感知：模型能理解图像、视频，实现"看屏幕操作应用"	• MCP(模型上下文协议)统一工具接口，实现"一处集成，处处可用“ • 多智能体协作：形成"数字团队"，将复杂任务拆解给不同专长Agent • Computer Use能力：直接操作计算机界面，无需API

图：AI Agent当前的发展阶段



资料来源：Open AI，甲子光年，公开资料

1.3 GUI Agent，重新定义人机交互自动化

1.3.1 GUI Agent简介

GUI Agent全称为Graphical User Interface Agent，中文为“图形用户界面智能体”，是指一种基于大模型，能够模拟人类的视觉感知和操作行为，直接通过识别屏幕上的图形用户界面来自主规划并执行任务的智能体，也被称为“UI Agent”“VLA Agent”。GUI Agent正在引发继“命令行”到“图形界面”之后的第三次人机交互革命，其核心是从“人操作机器”转向“机器理解并执行人的意图”，最终将消灭“应用孤岛”，使操作系统回归‘用户意图执行者“的本质。

图：交互发展史的三次浪潮

	CLI阶段	GUI阶段	GUI Agent阶段
交互对象	• 指令	• 控件	• 意图
交互方式	• 输入指令码或命令符	• 鼠标点击、触屏操作、图标 、按钮 、菜单	• 多模态输入（语音 、文字、图像）
核心差异	• 人要懂机器的“语言”	• 人要懂机器的“操作规则”	• 机器懂人的“自然需求”
代表系统/技术	• DOS、UNIX、早期Linux	• 苹果系统、Windows系统、手机触屏界面	• 各类GUI Agent工具
应用场景	• 仅专业程序员、科研人员使用	• 普通用户办公、娱乐	• 普通用户的跨平台任务

1.3.2 GUI Agent：人机交互自动化的范式革命

GUI Agent通过感知-推理-执行-反思的完整闭环，将人机交互自动化从传统的“脆弱脚本执行”推向“智能自主操作”的新高度，让人机交互从“人适配机器”向“机器适配人”转变，彻底重塑了自动化的核心逻辑与应用边界。

交互逻辑革命:从“规则驱动”到“理解驱动”，终结“脚本依赖”

传统人机交互自动化，如RPA、脚本工具等，核心都是“硬编码规则”，人工定义每一步操作的固定路径，本质是“机器执行预设指令”。而GUI Agent以多模态大模型为核心，实现“理解式交互”，本质是“机器理解意图后自主决策”。

- 从“像素或坐标依赖”到“语义理解”：传统方案绑定界面元素的固定位置或 ID，界面微调即失效，GUI Agent通过VLM解析界面语义，即使元素位置变化，仍能通过视觉特征精准定位。
- 从“单步指令”到“意图拆解”：传统自动化需用户拆分任务为“打开”、“点击”等单步指令；GUI Agent可直接理解自然语言意图（如“处理本周销售数据”），通过LLM自动拆解为“提取Excel数据→跨应用填入ERP→调用图表工具”的完整流程，甚至自主处理中间异常。
- 从“无反馈闭环”到“动态反思”：传统方案执行失败后需人工排查，GUI Agent通过“内省独白”实现自我纠错，如执行“提交报销单”失败时，自动分析是“未填必填项”还是“弹窗遮挡按钮”，并调整操作路径。

用户角色革命：从“操作执行者”到“意图决策者”，重构人机分工

- 传统人机交互中，用户是“自动化的奴隶”，需先拆解任务、学习工具操作、手动排查错误，GUI Agent将用户从“机械操作”中解放，升级为“意图输出者+结果审核者”。
- 从“步骤主导”到“意图主导”：传统场景中，用户需明确“每一步点哪里、输入什么”，GUI Agent中，用户仅需输出核心意图（如“将Excel中的本周销售数据填入ERP并提交”），Agent自动拆解步骤、处理异常、生成结果。
- 从“工具学习者”到“需求定义者”：传统自动化工具需用户学习复杂的流程配置、函数调用，非技术用户难以上手，GUI Agent 支持“自然语言交互+可视化反馈”，用户无需学习工具规则，仅需用日常语言描述需求，Agent实时反馈执行进度，用户仅需在关键节点确认。

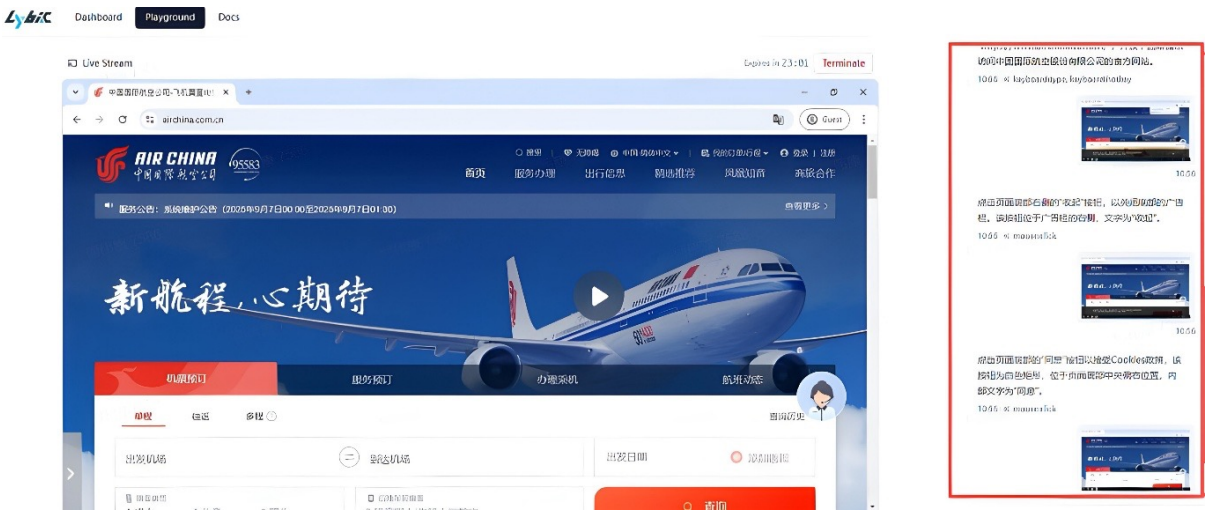
能力边界革命：从“单点固定流程”到“跨域动态适配”，打破“场景牢笼”

- 传统自动化受限于“平台锁定、流程固定、无泛化能力”，仅能处理“单一软件、固定步骤、无干扰”的简单场景；GUI Agent 凭借“多模态感知+长程推理”，实现“跨平台、跨应用、动态场景”的全链路自动化，能力边界呈指数级拓展。
- 跨平台兼容，从“单系统专属”到“全场景覆盖”：传统RPA需为Windows、IOS、Android、Web分别编写脚本，跨平台适配成本极高，GUI Agent通过统一视觉感知或混合输入，实现一套逻辑适配PC端、移动端、Web甚至工业控制界面。
- 长程任务突破，从“3步上限”到“百步闭环”：传统自动化仅能处理5步以内的简单任务，GUI Agent通过短期记忆、长期记忆、分层规划，可完成“行程规划”等复杂跨应用任务。

生态模式革命：从“单点工具”到“Agent协作生态”，构建智能交互网络

- 传统人机交互自动化是“孤立工具”，每个工具仅能处理单一场景（如Excel自动化工具），无法跨工具协同，GUI Agent基于“统一交互层+Agent协作协议”，形成“多Agent分工、跨场景协同、端云联动”的生态模式，成为AI时代人机交互的基础设施。
- 从“封闭工具”到“开源生态”：传统自动化工具多为商业闭源，二次开发难度大；GUI Agent已形成开源生态，开发者可基于开源框架快速定制行业Agent。
- 从“单点自动化”到“多Agent协同”：传统方案无法实现“跨工具协作”，GUI Agent能够支持“Agent-to-Agent”交互，通过统一协议实现多Agent分工。

图：庭宇科技GUI Agent产品Lybic工作画面



中国边缘云计算头部企业推出的GUI Agent产品Lybic（灵臂），该产品通过纯视觉识别理解图形界面，模拟人类“看屏幕→理解→操作”的认知逻辑，在示例操作中，仅需在浏览器页面中输入订购机票的指令，Lybic便能直接跳转到国航页面，并且完成机票购买的整个操作流程。

1.3.3 GUI Agent与其他自动化路线的对比

1.3.3.1 GUI Agent VS API Agent：AI自主执行的两条路线

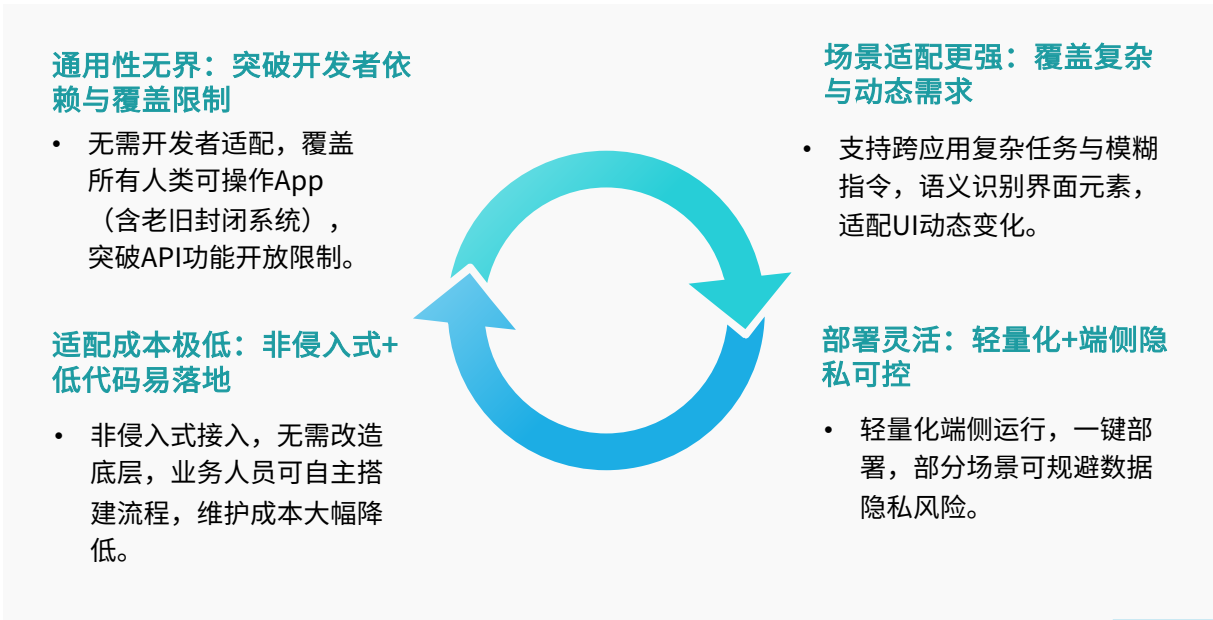
GUI Agent推动交互从“用户手动操作”转向“AI 自主执行”，业界形成两条技术路线，即API路线和GUI路线，API路线依托标准化语义接口，需开发者适配，功能受接口开放度限制（如Apple Intelligence+App Intents）；GUI路线靠多模态大模型识别界面、模拟触控，无需开发者配合，可覆盖应用全功能（如AutoGLM等智能体）。

表：GUI Agent与API Agent特点对比

维度	API Agent特点	GUI Agent特点
通用性	• 低，依赖应用开发者适配，功能局限于现有API	• 高，无需开发者适配，理论上人类可用的App均能覆盖，适配新功能或未暴露功能
可靠性	• 高，应用核心API版本稳定时，意图匹配API即可保障任务完成	• 低，存在三方面问题： • 复杂界面识别精度不足 • 多步骤操作失败率高 • UI改版易失效
性能	• 高效，单次调用完成复杂任务，仅需后端程序驱动	• 低效，耗时/资源消耗大： • 简单任务需多次截图分析，视觉推理算力成本高 • 截图上传数据量/延迟高于API
隐私风险	• 低，应用端可精细管理数据/权限，操作基于明确API，可控性强	• 高，需读取屏幕权限，易暴露聊天记录、密码等敏感信息，依赖用户高度信任
商业阻力	• 一般，由应用开发者决定开放能力	• 极大，未经厂商授权，易遭应用厂商通过技术/法律手段抵制

相比传统API集成，GUI Agent在成本、周期、覆盖度上优势显著：开发维护成本更低，业务人员可自主搭建流程，无需专业团队适配接口；能覆盖 API未开放的长尾功能，实现全量应用的自动化。

图：GUI Agent相比API Agent的优势



数据来源：微软

1.3.3.2 GUI Agent VS 传统RPA：企业自动化的成本-效率范式转移

GUI Agent与传统 RPA的核心分野，是 “规则固化方案”向“智能自适应方案”的技术跃迁，其特性直接决定落地成本与价值覆盖的代际差异：

- 开发端：传统RPA需技术人员为单一任务拆解步骤、绑定界面元素，不同任务需单独构建模板，开发依赖专业技能；GUI Agent依托LLM与自然语言交互，业务人员无需编程，可直接描述需求自主搭建流程，大幅降低落地门槛。
- 维护端：传统RPA与界面元素强绑定，系统版本迭代易导致流程失效，需人工逐一修改；GUI Agent凭借多模态感知与自适应能力，可自主识别界面变动，无需人工干预适配多数场景，减少长期维护投入。
- 价值覆盖：传统RPA聚焦高重复、单系统线性流程；GUI Agent依托LLM推理能力，可拆解跨应用复杂任务，将自动化从基础流程拓展至中高价值业务场景。

图：GUI Agent相比传统RPA的优势

1

智能化优势：从“被动执行”等到“主动决策”

- GUI Agent拥有RPA所不具备的智能化能力，如自然语言对话、流程推理、风险提示、自主学习等，交互体验大幅优于RPA。

2

开发成本优势：从“工程师绑定”到“业务端自主”

- RPA通过流程图或脚本编写流程，每一项任务都需要单独模板，灵活性较差，开发成本较高、周期较长。GUI Agent支持无编排自主建流程，用自然语言一键配置，流程上线周期从“周级”缩至“分钟级”，释放技术资源。

3

运维成本优势：从“被动调试”到“主动自适应”

- 传统RPA遇界面变更或异常需人工频繁调试，运维成本高；GUI Agent靠视觉+大模型精准识别界面，异常场景自动调路径，近零人工干预，大幅降低长期持有成本。

图：GUI Agent与传统RPA的差异

特性	GUI Agent	传统 RPA
核心技术	大型语言模型、多模态感知	规则驱动、脚本编程
用户交互方式	自然语言描述任务	固定规则配置
灵活性	高（可适应界面变化）	低（界面变化需要重新配置）
推理能力	强（支持复杂任务推理）	弱（仅执行固定流程）
学习与适应	有自适应与记忆能力	无学习能力
应用场景	跨应用协作、复杂任务	重复性高、流程固定的简单任务
维护成本	低（动态适应减少维护需求）	高（界面变化需频繁调整）

数据来源：微软、北京大学、上海AILab

1.3.3.3 GUI VS VUI：智能语音控制硬件，但GUI Agent能做到更多

VUI Agent（语音智能体）以语音为核心交互模态的智能执行主体，通过语音识别（ASR）接收指令、语音合成（TTS）反馈结果，依托自然语言理解模型解析意图并完成任务，核心是“语音驱动的自动化执行”。VUI与GUI Agent是“场景互补、技术可融合”的智能体形态。

- 互补：VUI主打“免手和免视” 快速交互，GUI Agent聚焦“复杂界面”深度操作，二者可形成“语音指令-视觉执行”闭环。
- 场景细分：短期VUI主导轻交互场景，GUI Agent主导复杂场景；长期向“多模态智能体”演进。

图：GUI Agent与VUI Agent的差异

对比维度	VUI Agent（语音智能体）	GUI Agent（图形界面智能体）
核心交互模态	• 听觉主导（语音输入+语音反馈）	• 视觉主导（界面识别+自动操作）
技术核心依赖	• 语音识别、语音合成、NLU模型	• VLM模型、CV/OCR、系统操作权限
操作方式	• 无实体界面操作，通过调用系统和应用接口完成任务	• 模拟人类手动操作，直接作用于图形界面
核心能力侧重	• 快速响应简单指令，适配 “免手/免视” 场景	• 处理多步骤、跨APP复杂任务，适配“无接口系统”操作
典型应用场景	• 车载驾驶、智能家居、智能音箱	• 自动化办公、跨平台服务、无接口系统操作
跨平台适配性	• 依赖平台语音接口，需定制适配（如不同车机、音箱系统），泛化能力弱	• 纯视觉建模（如AGUVIS、UI-TARS），支持Windows/macOS/移动端，无需接口开放，适配性强
信息处理密度	• 信息输出有限，难以传递复杂数据	• 信息密度高，可处理界面化复杂数据
核心优势	• 解放双手/视觉，适配双手忙碌、视觉注意力受限场景，响应迅速，交互自然	• 无需开发者适配，覆盖全量图形界面应用；支持多步骤、跨APP自动化；可处理复杂界面任务
核心局限	• 易受环境噪音、方言影响；复杂任务难以完成；跨平台适配成本高	• 易受界面改版、弹窗干扰导致操作失效；存在隐私风险；依赖本地算力，操作延迟高于语音交互

1.4 GUI Agent兴起的驱动因素：多模态大模型崛起和市场的自动化升级需求

随着多模态大模型取得突破和市场需求涌现，GUI Agent成为Agent的新方向。2025年，Anthropic、阿里、字节都推出了基于自身业务的GUI Agent，MultiOn、庭宇科技等国内外知名创业公司也推出了GUI Agent产品。其中字节跳动也发布了通用GUI Agent产品UI-TARS，并且推出了基于GUI Agent的手机；国内边缘云头部企业庭宇科技于2025年11月发布了GUI Agent基础设施平台产品Lybic，并且也于2025年末完成了数亿元的C轮融资。

表：突破性GUI Agent和基础设施相关产品

产品名称	企业名称	发布时间	产品特点
Computer Use	Anthropic	2024.10	• 官方提供API，让模型原生支持鼠标移动、点击、输入。无需外挂，视觉识别精度极高，主要面向PC和Linux环境。
OmniParser	Microsoft	2024.10	• Windows 的接管者。OmniParser解决了屏幕截图转结构化数据的难题（纯视觉）。
Project Jarvis	Google	2025	• Chrome的接管者。深度集成于Chrome浏览器，旨在通过Gemini模型接管所有网页操作，比如购物、订票等。
Mobile-Agent-v3	阿里巴巴	2025.8	• 跨平台GUI虚拟层，支持iOS/Android，应用识别准确率89%。
Lybic	庭宇科技	2025.11	• 定位为新一代GUI Agent基础设施平台，通过纯视觉识别理解图形界面，模拟人类“看屏幕→理解→操作”的认知逻辑，支持Windows/macOS/Linux/Android多个系统，核心组件识别准确率达86%，响应速度快，低于200ms，能够深度理解界面语义，支持复杂决策，并且实现大规模部署。
UI-TARS Desktop	字节跳动	2026.1	• 首个超越GPT-4o的国产纯视觉GUI Agent，登顶OSWorld与GitHub热榜；支持PC/手机跨端统一执行，提供2B/7B/72B多规模模型，视觉感知+动作推理+记忆机制完整开源。

驱动因素一：Agent面临的智能化困境

在AI Agent发展初期，以传统的API路线为主，即软件向Agent开放接口，这就造成AI Agent只能在内置于应用之中，只能实现部分自动化。GUI Agent通过多模态大模型的引入和工作流的重塑，解决了AI Agent面临的问题。

- “API覆盖率不足”造成的“长尾软件孤岛”。全球软件生态中，拥有完善API的软件有大约不到20%，绝大多数软件缺乏API接口，包括企业老旧的ERP系统、本地桌面软件、以及为了商业利益封闭接口的超级APP。GUI Agent通过非侵入式连接，直接像人一样操作前端界面，打通了数据孤岛，将那些“沉默”的系统接入自动化流程。

- 传统RPA的“脆弱性”与“维护成本”问题。传统RPA基于规则的脚本，依赖固定的屏幕坐标或底层代码，一旦软件升级、UI改版、分辨率变化，或者突然弹出广告窗，RPA脚本就会立刻报错或点错位置，智能体工作流就无法实现。

- 跨应用、跨生态造成的“碎片化工作流”。现代工作是碎片化的，需要在不同的应用、窗口、浏览器之间切换，比如财务报销，需要先上传发票，之后填写说明，之后走OA审批，这中间需要切换大量窗口，做很多无用的工作，API Agent也无法解决，GUI Agent可以直接操作系统，在不同窗口之前切换，高效完成工作。

- 复杂软件使用和学习门槛较高。复杂的软件，如修图软件、剪辑软件、工业软件等，有成百上千项功能，初学者使用门槛极高，GUI Agent可以通过简单的指令，降低复杂软件的使用门槛。

驱动因素二：大模型底层技术的突破性进展

VLM视觉理解能力飞跃，从“鉴赏”到“操作”的进化

大模型是AI Agent发展的核心和基础，VLM是GUI Agent能够诞生并且能够实现快速发展的基础。从2023到2025年，VLM大模型在精准度和分辨率方面实现了大幅度跃升，实现了从“鉴赏”到“操作”的进化。

表：VLM的进化过程

项目	内容	2023年 (视觉觉醒期)	2024年 (精准定位期)	2025年 (原生操作期)
特征	总体特征	• 模型开始具备通用的视觉推理能力，但主要用于对话和描述，操作能力极弱	• Agent能点击鼠标，技术重心从“描述”转向了“定位(Grounding)”	• VLM开始专门为GUI Agent定制，强调速度和原生控制，控制速度和丝滑程度与人类相似
指标1	ScreenSpot准确率	<20%-60%	>90%	95%
解释	GUI元素定位基准	• “脸盲”阶段，只能依靠文本匹配，无法匹配图标	• OmniParser的出现是分水岭，专门针对UI界面训练，不需要文字标签，仅凭图标形状就能识别功能	• 解决了“密集排布”的难题。即使是Photoshop这种甚至有上百个按钮的复杂界面，或者股票软件的密集数据屏，模型也能精准区分每一个元素
指标2	DocVQA准确率	~70%	~90%	95%
解释	复杂文档问答基准	• 擅长读自然段落，但遇到多栏排版（如报纸）、跨页表格或者手写体时，经常乱码或串行	• 能读懂复杂报表，在图表理解（ChartQA）上超越了普通人类，并且能理解数据之间的对齐关系	• 具备了“视觉逻辑推理”能理解逻辑与隐含信息。例如，能通过发票上的印章颜色判断真伪，或者通过签字的笔迹判断审批人
指标3	分辨率支持	固定低分(224px/512px)	动态高分(Any-Res/4K)	自适应高效高分
解释	视觉清晰度	• 早期版本会将大图压缩。这导致GUI Agent在操作时经常出现“幻觉”，看不清小字	• 引入了切片技术。把一张1080P的截图切成9张小图喂给模型。看得清了，但推理速度变慢，清晰但浪费Token	• 引入了Visual Token Selection技术，既保留了4K清晰度，又把推理速度提了上来，实现了“实时操作”

LLM实现从“直觉式反应”到“深思熟虑”的进化

VLM解决了Agent的“识别”问题，LLM的决策规划规划能力决定了Agent工作的理解深度和工作深度的问题。2023-2025年，LLM通过支持结构的优化，决策和输出能力进展迅速，决策深度和复杂程度都有了质的飞跃。

表：人工智能大语言模型的进化过程

项目	2023年	2024年	2025年
思维特征	直觉反应	思考并且修正	深思熟虑
规划深度	单步规划	短期规划	长链路全局规划
认知结构	• ReAct范式 • CoT(Chain of Thought)	• 自我反思(Reflexion) • ToT (Tree of Thoughts)	• 推理时计算(Inference-Time Compute) • 蒙特卡洛树搜索(MCTS)的内化 • 多智能体博弈与协作(Multi-Agent Orchestration)
GUI表现	• 只能做“打开网页”等简单动作	• 能处理弹窗、验证码等异常	• 能完成“跨系统数据搬运”等复杂流

第二章

技术架构和核心实现

2.1 设计思路和挑战

2.1.1 GUI Agent的工作流程

与传统API Agent相比，GUI Agent的特点是绕过API接口的“后门”，而是走“前门”，对图形界面进行理解并且操作。GUI Agent设计的难点，是在人类模糊、跳跃的“意图世界”与计算机严谨、死板的“执行世界”之间，架设一座桥梁，并且最大化地减少从感知、推理、定位、执行每一步的误差，让GUI Agent在能够精准理解人类的意图，并且实现精准的操作和执行，减少长链条工作流程中产生的误差累计。

图：GUI Agent工作流程

2.1.2 GUI Agent的面临的主要挑战

GUI Agent的落地是一项复杂的系统性工程，真实的跨OS物理界面充满了非结构化的视觉噪音与高频的动态突变。在这个复杂的交互场域中，智能体试图实现人类级别的“眼手脑协调”时，不可避免地遭遇了底层技术范式与复杂物理场景的剧烈碰撞。当前GUI Agent在迈向工业级可用的进程中，必须跨越感知、定位、决策与环境四大维度的严峻挑战。

挑战一：感知，高分辨率与细粒度的冲突

人的眼睛是具备“变焦”功能呢的，既能看清4K屏幕的全貌，也能看清12px的小字。但对模型来说，很难兼顾不同清晰度信息的变化同时做到准确识别。

- 分辨率悖论：如果把1080P/4K的屏幕截图直接喂给模型，Token消耗极高且推理极慢。如果把截图压缩到512x512，很多关键的小图标，如Excel的筛选箭头、关闭按钮，那么就会变成噪点，模型根本看不见。
- 信息密度过载：GUI界面不仅有图像，还有密集的文字。OCR和VLM很难在复杂的背景下精准提取信息。

挑战二：定位，概率语义与精确坐标的错位

定位是将大模型生成的模糊“语义意图”精确映射到屏幕上确定的“物理坐标”是的过程，是连接大模型和行动之间的“神经系统”，核心的作用是帮助GUI Agent实现“眼手协调”。但在这一过程中，大模型经常被误导，产生各类幻觉。

- 偏见幻觉：模型受训练数据中高频元素的“路径依赖”影响，即使目标元素位置变化或未出现，仍倾向于点击训练中常见的位置和图标。
- 误导幻觉：高分辨率、元素密集的界面中（如4K屏幕的工业控制界面），模型注意力被背景噪声（如广告弹窗、相似图标）吸引，误将干扰元素当作目标。
- 混淆幻觉：对小尺寸元素（如1080P屏幕中小于10×10像素的“设置”图标）或相似元素（如“保存”图标），模型无法精准区分，预测坐标在多个元素间“漂移”，导致“点空气”。

挑战三：决策，长链条任务的误差累积

GUI Agent任务操作通常需要几步到几十步，但是由于上下文遗忘、推理能力欠缺、幻觉、缺乏全局规划能力等原因，造成误差累计，导致工作结果的南辕北辙。

- 长程记忆与状态跟踪失效：大模型的上下文窗口有限，长链条中会“忘记”前序关键信息，后续决策基于不完整信息而造成理解和决策偏差。
- 任务拆解与依赖关系误判：长链条任务的子步骤存在强先后依赖，如“必须先填发票信息，才能提交报销单”，模型误判依赖关系，如“未填发票就提交”，前序错误直接传导至全流程。

挑战四：环境，动态性和稳定性的对抗

相比于文本Agent和API Agent，GUI Agent面对的是一个跨平台、强动态、资源密集且多变的环境，这就造成GUI Agent在应用过程中出现跨平台适配成本高、动态环境泛化能力差、资源消耗大、推理效率低等多种问题。

- 异构性强，跨平台适配成本高：GUI Agent面对的是一个多OS、多系统、多架构的环境，使用方式和交互逻辑不统一，需要为不同平台适配不同的逻辑，成本较高。
- 动态性强，界面和场景不可预测：GUI Agent面对的是一个高度动态的环境，随时出现弹窗广告、权限需求、登录环境的转变，需要GUI Agent具备超强的动态泛化能力。
- 资源消耗高，拖慢推理效率：GUI环境是“有状态且资源消耗高”的复杂系统，处理高分辨率图像时，视觉模型的消耗量呈现几何数增长，同时，在执行长链条任务时，内存占用在GB级别，推理延迟增加至5-8秒，资源消耗量远超传统的Agent。

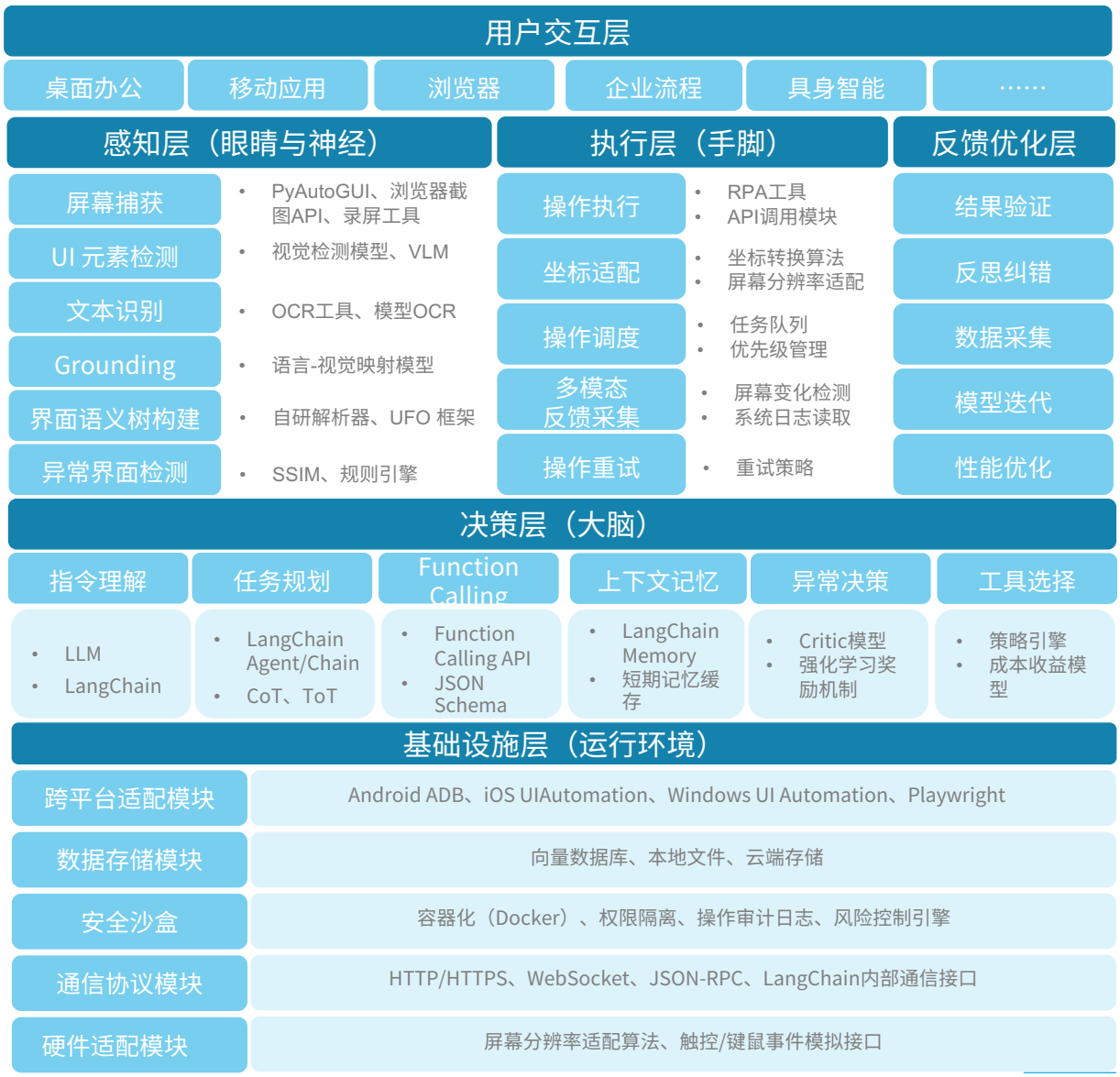
2.2核心模块解析

2.2.1 GUI Agent核心模块

GUI Agent的技术架构围绕“视觉理解-自主决策-精准执行-持续进化”核心目标，采用“分层解耦+模块协同”设计，形成了六个层次组成的结构闭环。

- 用户交互层：对接具体业务场景，将通用架构适配到不同需求。
- 感知层：将图像转化为AI可理解的结构化信息，解决“看不懂界面”的问题
- 执行层：将决策层的指令转化为实际界面操作，进行命令的执行操作。
- 反馈优化层：基于执行结果验证任务完成度，收集数据优化模型和策略。
- 决策层：理解用户指令、拆解任务、生成操作策略，是GUI Agent的核心中枢。
- 基础设施层：提供跨平台运行环境、数据存储、安全防护和通信能力。

图：GUI Agent核心模块和组成



2.2.2 感知模块：构建高精度的数字视网膜

2.2.2.1 核心内容

感知模块是连接大模型“抽象语义空间”与操作系统“物理像素空间”的唯一桥梁，主要任务是将非结构化的屏幕像素流，转化为结构化的环境信息。它不仅要“看见”屏幕，还要“看懂”屏幕上有什么、在哪里、是什么状态。

感知模块是GUI Agent的基础能力和最核心入口，如果Agent看不准屏幕，后续的决策和执行都无从谈起，同时，由于GUI Agent存在误差传导的效应，感知模块的的精度、广度、深度直接决定了后续决策、执行模块的执行能力。

环境信息采集模块：获取界面原始数据

- 视觉数据：全屏幕/目标区域截图、视频流，捕捉界面布局、颜色、位置等非结构化信息。
- 结构化元数据：界面部件树（Widget Tree）、DOM结构（Web端）、无障碍树（Accessibility Tree），提供元素层级关系、属性等结构化信息。
- 辅助传感器数据：设备状态（如屏幕分辨率、系统版本）、操作上下文（如当前活跃窗口、键盘/鼠标状态），用于适配跨平台交互（如移动端触摸区域校准、桌面端键鼠坐标转换）。

UI元素检测与分类模块：基础解析环节识别界面“可交互组件”

- 元素定位：通过边界框标注按钮、输入框、弹窗、图标等可交互元素的像素坐标，如“登录按钮在（x:120，y:350，w:80，h:40）”。
- 元素分类：区分元素类型（如“文本框”“复选框”）、功能语义（如“提交”“删除”“关闭”），甚至状态（如“可点击”“禁用”）。
- 噪声过滤：排除非交互元素（如广告横幅、静态背景图、加载动画），减少冗余信息干扰。

视觉-文本对齐：解决“用户指令→界面元素”的精准绑定问题

- 语义-Grounding：将自然语言指令中的描述（如“红色确认按钮”）与界面中具体元素匹配，输出唯一对应的元素坐标。
- 视觉-Grounding：对模糊指令（如“关掉那个弹窗”），通过界面视觉特征（弹窗的“悬浮层级”“关闭图标样式”）定位目标，无需依赖文本标签。
- 不存在元素识别：当指令中的元素不存在，输出“元素缺失”信号，避免决策层误操作。

界面语义解析模块：理解“整体界面逻辑”

- 界面结构树：将元素按层级关系组织，体现元素间的从属、依赖关系。
- 动态状态感知：识别界面临时状态，如“加载中动画”“验证码弹窗”“网络错误提示”，标注“不可操作时段”或“需优先处理的干扰项”。
- 场景语义映射：关联界面与任务场景，如看到“订单号输入框+支付按钮”即判断为“支付场景”，为决策层提供“场景化认知”。

2.2.2.2 面临的困难和挑战

感知模块的技术困难本质是“通用AI模型”与“专用GUI场景”的适配鸿沟：AI模型擅长处理结构化数据，而GUI界面是人类设计的“非结构化视觉符号系统”，且具有动态性、异构性、隐性逻辑性。解决这些困难的关键，在于构建GUI专用的视觉-语义理解模型。

挑战1：细粒度元素识别的精度难题

- 分辨率不匹配：屏幕截图的像素规模远大于VLM的输入分辨率（通常224×224或448×448），缩放过程中小尺寸交互元素导致细节丢失，造成识别率低于可用水平。
- 相似元素状态歧义：视觉特征高度相似但是功能和状态不同的元素，比如广告的“确定”按钮和真实的“确定”按钮，现有的模型难以区分，造成误判率较高。

挑战2：动态界面的实时感知滞后

- 动态元素的干扰：弹窗、轮播广告、加载动画等临时元素会遮挡或混淆目标元素，模型难以区分“常驻交互元素”与“临时干扰元素”，导致任务中断。
- 实时数据刷新的感知失效：股票行情、物流轨迹等实时刷新的界面区域，模型的感知结果存在“时间差”，无法同步捕捉数据变化。

挑战3：空间定位的“语义鸿沟”

- 视觉-文本对齐是感知模块对接决策层的核心桥梁，其困难在于自然语言指令的模糊性与界面元素空间位置的精确性之间的矛盾，这是当前最致命的技术瓶颈。主流的大模型本质上是概率预测模型，是在海量的图文对上训练的，大模型能够完美地理解画面，但却无法精确地指出它在哪里，具体表现就是GUI Agent会出现坐标幻觉和对位置的混淆，在识别和点击的时候出现误操作。

挑战4：界面结构与场景认知的深层障碍

- 界面层级结构的认知不足：模型难以构建元素间的从属、依赖关系，如“弹窗→关闭按钮”，无法理解“输入框未填写则提交按钮禁用”这类隐性规则，导致后续操作决策失误。
- 场景语义映射的泛化性差：界面元素组合与任务场景的映射，如“订单号输入框+支付按钮”即“支付场景”，目前依赖人工定义规则或少量样本训练，无法泛化到未见过的非标界面，缺乏异常场景识别能力，导致有可能被钓鱼网站利用。

2.2.2.3 技术考量维度

在技术考量维度方面，感知模块面临着一个“不可能三角”，即极端的通用性、极高的精准度、极低的成本与延迟，感知模块的技术选择就是主要在以上几个主要维度之间进行博弈和寻找平衡。

考量维度1：数据源的获取路径——代码流VS像素流

- 数据源获取路径是感知模块最底层的基建路线之争，一方面决定了Agent要采用的技术路径，另一方面决定了Agent的通用性、精确性的上限。当前形成了基于底层代码树和纯视觉多模态大模型两种主要方式，代码树的优势是速度快、成本低、精准度高，但劣势就是脆弱和泛化性差，像素流的优势是泛化性强，但劣势就是精准度低而且成本消耗高。随着技术探索的深入，以代码流为优先同时衔接纯视觉路线的混合路径成为Agent的首选。

图：代码流和像素流技术栈对比

项目	代码流	像素流
核心数据对象	• DOM Tree/UIA Tree/XML	• Bitmap /H.264 Stream
数据量级	• 极小(KB级)	• 巨大(MB级，需压缩)
Web端接口	• Chrome DevTools Protocol(CDP)	• CDP
Windows端接口	• UI Automation API	• DXGI Desktop Duplication
Android端接口	• AccessibilityService	• MediaProjection API
处理引擎	• LLM	• VLM/CNN

考量维度2：空间分辨率与颗粒度处理——全局语境vs局部微观

- 这是针对高分辨率屏幕与细粒度UI控件之间的矛盾所设立的考量维度。
- 全局语境是将整张屏幕截图喂给模型，旨在让模型看懂整体排版、页面逻辑，代价就是图像压缩会导致关键小图标模糊，导致模型漏检。
- 局部微观是采用区域切片技术，只把重点区域的高分辨率局部图发送给模型，代价是牺牲了全局视野，导致模型缺乏上下文和全局的判断能力。

考量维度3：时态动态性捕获——静态单帧vs连续数据流

- GUI界面不是静止的画面，而是充满了悬停菜单、广告、加载动画等一系列提示的动态画面。
- 静态单帧通过截图进行分析，优势是逻辑和架构简单，Token消耗能够一次性结清；缺点就是极易产生状态滞后或者漏掉关键的瞬态提示，导致后续决策全部基于过期信息。
- 连续数据流通过引入高频帧差进行分析或者通过流媒体进行实时监测，能够捕捉UI的每一帧变化，劣势是对端侧设备、网络带宽等基础设施要求极高，同时Token消耗是单帧模式的数十倍。

考量维度4：算力分布拓扑——云端重推理vs边缘轻计算

- 算力分布决定了GUI Agent 在真是商业环境中的商业可行性和合规性，同时决定了Agent的泛化认知上限、数据延迟和安全性。
- 云端布局是将所感知到的复杂画面上传给视觉大模型，利用其庞大的世界知识进行识别，缺点是单步延迟长，及时交互体验差，且面临敏感的数据隐私和合规问题。
- 边缘布局是在用户的PC或者沙箱内布置参数极小的边缘模型，复杂高频、简单的识别任务，劣势是边缘模型能力有限，而且无法做到实时更新，面对新的复杂组件容易卡壳。

图：云端大脑和边缘小脑的对比

项目	云端大脑	边缘小脑
代表模型	• Claude Sonnet、OpenAI GPT-4o、Qwen-Max、Google Gemini	• Gemini Nano专用轻量CV模型、SLM
算力位置	• 中心化GPU集群	• 本地PC、轻量沙箱
交互频率	• 低频，遇到复杂任务、未知报错调用	• 高频，实时监听、滑动、等待
优势和专长	• 深度逻辑推理、常识泛化、任务全局规划	• 视觉坐标锚定、瞬态变化检测、简单规则反射
成本	• 极高，按Token计费	• 边际成本接近于零
延迟	• 长，延迟约2-5秒	• 短，小于50毫秒

2.2.2.4 技术解决方案

根据感知模块面对的技术挑战和考量维度，当前形成了四条主流的技术解决方案，分别是纯底层代码萃取方案、纯视觉大模型直出方案、基于标记集的视觉锚定方案、端云协同多模态动态切片方案。四种方案根据使用目的和场景不同，各有优劣势。

SoM方案已经逐步成为了当前的主流路径。因为在当前大模型缺乏原生精准坐标输出能力的情况下，开发者只需要用YOLO或者GroundingDINO画框，再外接一个大模型就能实现跑通，SoM用极低的开发门槛暂时解决了“看不到”和“点不准”的问题，但仍然没有解决复杂界面密集遮挡和响应延迟的商业化痛点。在未来一段时期内，技术路径将从SoM向原生坐标直出（即Native Grounding）路径演进，同时，Agent的搭建门槛将进一步提升，边缘算力将成为解决当前延迟和算力成本膨胀的重要路径。

- 纯底层代码萃取方案：该方案传统桌面自动化在Agent时代的自然延伸，追求极致的确定性。底层逻辑是绕过视觉渲染层，直接调用操作系统无障碍接口（如Windows UIA）或浏览器内核调试协议（如Chrome CDP），将UI界面逆向解析为机器可读的树状结构文本。该方案的优势是绝对的准确率和极低的成本，劣势是极度依赖白盒环境，在高渲染环境下会失效。
- 纯视觉大模型直出方案：该方案100%依赖像素流，逻辑是抛弃底层系统接口，完全模拟人类视觉。将全局的高清屏幕截图直接“喂”给多模态大模型，依靠模型内部庞大的世界知识去理解界面语义并直接回归目标元素的坐标。优势是通用性极强，能够实现“所见即所得”，劣势是Token消耗高、成本高、延迟时间长，并且当前存在比较严重的坐标幻觉。
- 基于标记集的视觉锚定方案（SoM）：这是目前为了弥补大模型“点不准”而广泛采用的折中优化路线。该路线采用“先检测画框，后大模型推理”的接力模式。首先利用轻量级目标检测模型在截图的所有可交互元素上画出带数字编号的边框，随后大模型只需阅读这些边框，输出目标元素的“数字编号”。优势是兼顾了视觉方案的黑盒穿透率，同时大幅度降低了坐标漂移的概率，劣势是在复杂业务系统中，标记框依然存在相互遮挡，导致严重误判。
- 端云协同多模态动态切片方案：底层逻辑是建立一套动态路由的混合引擎。在白盒环境下优先萃取代码流保证极速，遭遇黑盒时无缝切换视觉兜底。同时，利用端侧小模型实时监听屏幕连续帧，仅在发生变化时，截取鼠标周围的高清局部画面上传云端。优势是能够极大的降低使用成本，同时能够保证精准度和泛化性，劣势工程架构复杂，基础设施建设的门槛较高。

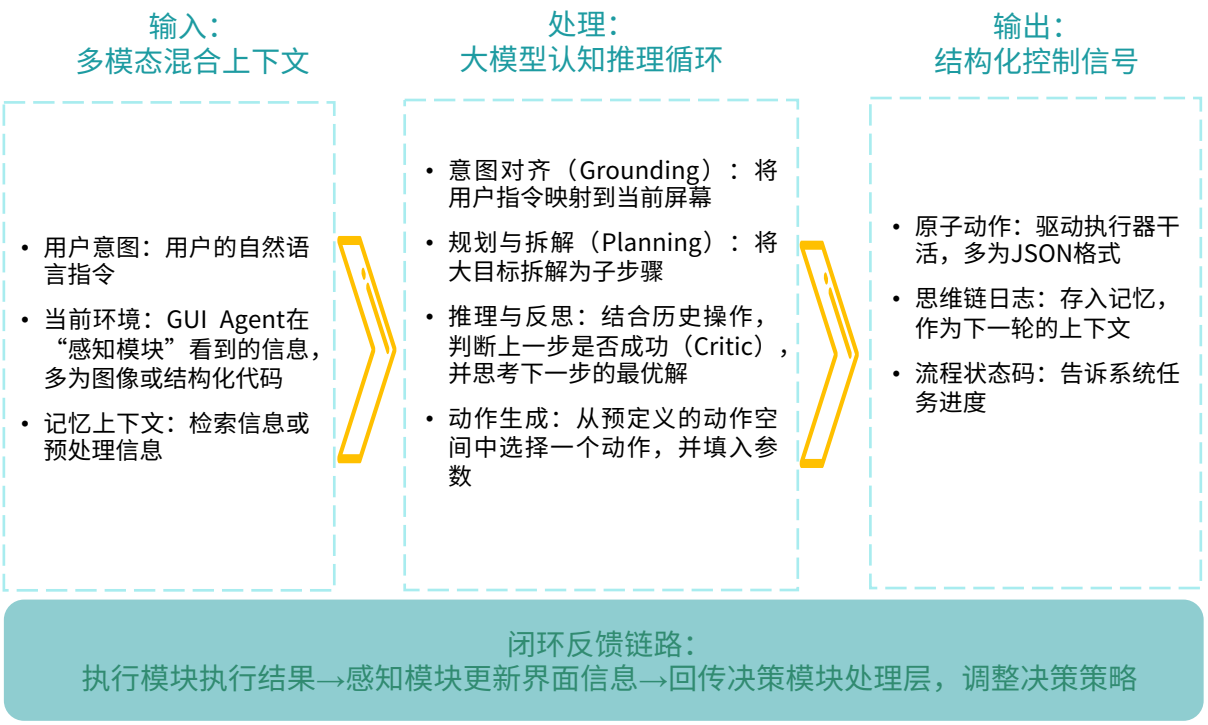
图：四种主流解决方案的技术特点对比

项目	纯底层代码萃取方案	纯视觉大模型直出方案	基于标记集的视觉锚定方案	端云协同多模态动态切片方案
数据源	• 100%代码流	• 100%依赖像素流	• 像素流	• 代码流为主，像素流兜底
空间颗粒度	• 既有全局树结构，又能精确定位到局部元素的绝对坐标	• 偏向全局语境，牺牲了局部微观精度	• 结合了全局（大模型看图）与局部（小模型画框）	• 全局保留低分低辨率语义，局部鼠标位置截取高清切片
时态性	• 依赖静态单帧或简单的事件监听	• 静态单帧，通过“截图-上传-等待”等几个步骤	• 静态单帧，但单帧的处理时间被拉长（增加了一道画框的工序）	• 引入边缘端的连续流监听，只有当界面发生真实变化时才触发感知
算力分布	• 边缘轻计算为主，仅使用极低的CPU	• 极端的云端重推理	• 云+端混合，端侧/小云做检测画框，大云做逻辑推理	• 边缘小脑负责高频反馈和ROI裁剪，云端大脑负责低频的复杂认知
缺点	• 面对“黑盒维度”（如Citrix、全屏游戏），直接致盲	• 坐标幻觉导致的“点歪”现象频发 • 高昂的Token成本和数秒的延迟阻碍了商业化落地	• 在高度密集的业务系统中，满屏的标记框会互相遮挡，引发密集恐惧，干扰大模型判断	• 要求高并发、低延迟、高可靠性的企业级全场景复杂业务
适用场景	• 内部白盒系统、标准化的Web表单填报	• 跨平台的通用问答、粗颗粒度的网页导航	• 复杂的非标准界面、缺乏底层代码的黑盒应用	• 对基础设施要求极高，需要重构底层抓帧协议与端云流媒体通信链路

2.2.3 决策与规划模块：感知与执行的核心桥梁

如果说感知模块是GUI Agent的“眼睛”，那么决策模块就是它的“前额叶皮层”。决策模块是连接感知模块与执行模块的核心中枢，也是GUI Agent实现“自主完成任务”的关键核心，它承接感知模块输出的结构化界面认知，结合用户的自然语言任务意图，通过任务拆解、操作规划、异常判断、动态调整，最终输出标准化、可执行的操作指令，同时实现对任务全流程的动态管控。

图：决策模块流程



2.2.3.1 核心职能

GUI Agent的决策模块采用“分层递进+记忆支撑+闭环反馈”的架构设计，从底层到上层实现“意图→规划→操作→纠错”的全流程决策，同时记忆层为各层级提供上下文与规则支撑。

- 意图解析与对齐：将用户模糊的自然语言指令（如“整理本月财务数据”）与感知模块的场景认知进行精准对齐，消解指令歧义、明确任务目标，并且输出结构化的任务目标。
- 任务规划与拆解层（核心决策层1）：将结构化任务目标拆解为可执行的层级化任务序列，即把复杂任务拆分为多步子任务，再将子任务拆分为原子操作（如点击、输入），同时明确任务的执行顺序、依赖关系、优先级。
- 操作选择与决策层（核心决策层2）：针对任务规划层输出的单个原子操作，结合感知模块的实时界面信息，选择最优的操作对象、操作方式、执行参数，是“从规划到具体操作”的最后一步决策，同时输出标准化可执行指令。
- 异常检测与纠错层（闭环管控层1）：核心职责是全程监控感知模块的输出和执行模块的反馈结果，实时识别任务执行中的异常情况，并快速生成纠错策略，避免任务中断，这是决策模块“鲁棒性”的核心体现，也是区别于传统RPA“固定流程执行”的关键。
- 记忆与上下文关联层（支撑层）：核心职责是为决策模块的所有层级提供上下文记忆、规则记忆、用户习惯记忆，解决GUI Agent长程任务上下文遗忘的问题，实现任务的连贯性和个性化决策与感知模块的记忆系统联动，构成GUI Agent的全局记忆体系。

2.2.3.2 面临的困难和挑战

挑战1：长链路任务的“记忆遗忘”

- 对多步骤、跨场景、跨应用的长程任务，当前决策模块的层级化拆解能力和上下文记忆能力存在明显短板：比如出现步骤遗漏、逻辑混乱、发生上下文遗忘。主要原因是LM的注意力机制在长上下文中会出现衰减，且缺乏对“任务进度条”的显性感知能力。

挑战2：陷入死循环与重复操作

- 这是GUI Agent最常见的“愚蠢”行为。Agent点击了“下一步”，但页面因为网络原因没跳转，Agent以为自己没点到，于是又点了一次，还是没跳转，最终在一个页面死循环，直到Token耗尽。根本原因是Agent缺乏对“状态变化”的敏感度和反思机制，无法判定自己是否卡住了。

挑战3：泛化环境下的“决策幻觉”

- 幻觉难题贯穿GUI Agent的各个模块，在决策层主要表现为对象幻觉、逻辑幻觉和规则幻觉，即臆测不存在的按钮、弄错操作逻辑和操作规则
- 主要原因是是大模型的概率预测特性和GUI Agent的确定性的矛盾，一是模型会脑补和生成原本不存在的动作和按钮；二是视觉和语义存在断层，大模型很难将视觉和操作完美对应，导致生成无效指令；三是对大模型对长尾环境缺乏专业知识和理解逻辑。

挑战4：企业级合规的“创造力诅咒”与“脱轨风险”

- 大模型的最大优势是涌现的创造力，但在企业级GUI自动化中，这恰恰是最大的风险。一方面在企业级业务线中，需要严格按照SOP执行，但大模型在遇到未知情况中，会出现“盲猜”和“试探”的情况；另一方面，大模型更倾向于输出自然语言，如果不能将其意图稳定地收束为底层执行模块能够理解的的JSON代码契约，可能会引发解析的困难甚至崩溃。

2.2.3.3 技术考量维度

在感知模块，Agent面临的是“如何正确认识并且理解信息”的挑战，在决策与规划模块，Agent面临的是“系统智商与绝对理性的博弈”，即让大模型具备处理复杂未知情况的能力，同时又能够确保执行的确定性、安全性和精确性。

考量维度1：规划与推理范式——单步反应式vs全局推演式

- 规划与推理范式是决定GUI Agent智商上限与行为模式的底层认知算法底座。目前形成了单步反应式（如ReAct模式）和全局式推演（如ToT思维树、MCTS蒙特卡洛树搜索）两种主要范式。
- 单步反应式：基于当前看到的屏幕状态，立即决定下一步的动作。优势是响应快、Token消耗低，更适合标准化、短链路的任务，劣势是Agent很容易忘记最初的整体目标。
- 全局式推演：在下达真实物理指令前，先在模型的思维沙盘中进行多分支的预演。优势是能够应对逻辑复杂的网状跨系统任务，劣势是延迟高，算力成本高。

考量维度2：上下文与长程记忆管理——滑动窗口vs外挂结构化记忆

- 滑动窗口记忆(Sliding Window Context)：较为简单的把过去N步的历史截图和文本全部放进大模型的提示词。优势是架构简单，无需复杂的中间件，劣势是极易触发Token溢出灾难，同时历史噪音过多会导致大模型注意力偏移。
- 外挂结构化记忆与SOP注入：为Agent建立独立的短期与长期记忆库。只在工作记忆中保留当前界面的关键元素，同时利用RAG技术，实时调取企业的《标准操作手册(SOP)》作为硬性约束注入当前决策流。优势是Token消耗低，同时能够大概率确保Agent沿着主线进行工作，缺点是需要构建复杂的向量数据库与记忆摘要。

考量维度3：动作意图的输出约束——开放式生成vs强类型契约

- Agent在做好计划后，要通过开放式代码或者JSON库将命令进行对外传输，这是Agent操作灵活性和风控安全之间的博弈。
- 开放式代码生成：允许大模型直接输出一段Python或Shell脚本，让底层系统运行。优势是自由度极高，几乎能让模型调用系统的一切能力，劣势是风险大，如果出现幻觉会造成不可逆的破坏。
- 强类型函数契约：为Agent定义了一个极其严苛的JSON动作库，只允许Agent执行特定动作。优势是构建了护栏，保证了输出意图的结构化和可解析性，劣势是牺牲了操作灵活性。

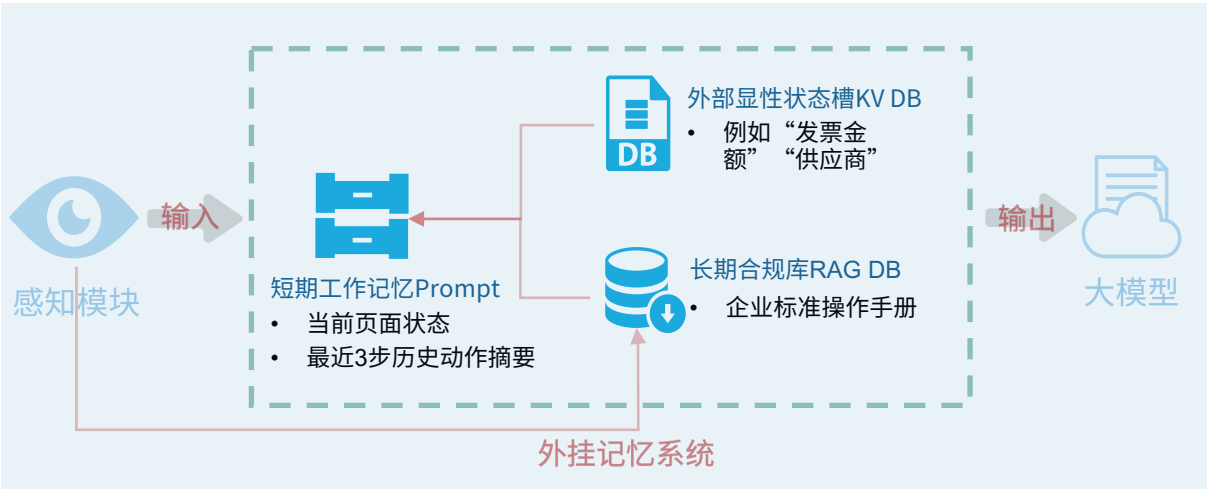
2.2.3.4 技术解决方案

解决方案 1：基于RAG的长短记忆双轨分离架构，解决长链条任务的“记忆遗忘”难题

针对长链条任务的记忆遗忘与上下文溢出问题，当前形成了两种解决方案，最初是采用滑动窗口的方案，即简单粗暴的保留最近3～5步的历史截图，丢弃更早的记忆，这种方式虽然能够避免系统崩溃，但是没办法解决“记忆遗忘”的问题，无法应对复杂的多步骤工作。当前主流方案是“结构化外挂记忆”方案，即对Agent的大脑内存进行动静分离。

- 极致压缩的“短期工作记忆”：决策中枢不再向大模型输入过去的所有历史截图。Prompt窗口中只保留当前屏幕的高精度状态以及最近3步的文本动作摘要。这保证了模型拥有极简的当下语境，算力消耗降至最低。
- 基于Key-Value的“显性状态槽”：对于任务中提取的关键变量（如账号、发票号），决策模块不再指望大模型自己记住，而是强制要求大模型将这些值写入一个外部的、结构化的KV数据库中。当后续需要填表时，直接从数据库中读取，实现了变量记忆的100%绝对留存。
- SOP知识的RAG动态注入：我们将企业几十页的《标准操作手册》切片，存入本地向量数据库。当感知模块识别到当前页面是“发票上传页”时，决策模块会瞬间触发RAG，只把关于“发票上传”的那三条规则提取出来，强行作为System Prompt注入给大模型。Agent就像一个带着动态外挂说明书的数字员工，走到哪一页，就调取哪一页的记忆。

图：外挂记忆系统流程图



解决方案 2：强化自我修正和安全兜底，解决死循环与重复操作难题

死循环和重复操作是Agent在决策与规划模块面临的重要挑战，当前主流技术领域形成了三种机制来解决该难题。其中，第一类核心机制是第一道防线，也是当前最主流的处理方案。第二类增强机制是第二道逻辑防线，主要是通过引入外部反馈机制对操作结果进行判断，是第一类机制的增强机制。第三类兜底机制是兜底方案，主要是防止前两个机制失效后构建的安全底线。

这五种方案不是互斥关系，在技术实现中，企业根据自身需求对五种方案进行不同的配置，主要核心还是第一类，第二类和第三类根据企业需求进行配置；在任务执行过程中，五种方案根据任务需求交替协同来完成任务。

图：死循环和重复操作的部分解决机制

机制	方案	解决的痛点	具体内容
第一类：核心机制	显式操作历史与结构化状态记忆	• Agent不知道自己刚才点了什么	• 不在Prompt堆砌过多的历史截图，这会引起注意力色散和Token溢出，而是维护一个精简的、文本化的最近N步动作队列，比如如果Agent在最近3步操作中出现重复，基础推理能力就会停止Agent的操作。
	基于ReAct框架的“反思与自我修正”	• Agent缺乏对自己动作后果的评估能力	• 目前VLM/LLM领域的绝对主流推理范式。它强制模型遵循Thought->Action->Observation的循环。 • 比如，Thought进行规划，Action进行执行，Obeservation观察操作之后的屏幕变化，如此循环实现Prompt结构中的自我反思。
第二类：增强机制	基于底层状态差异的即时反馈	• 仅靠VLM看截图，无法分辨是否生效	• 在端侧（边缘计算）利用轻量级代码，在点击后瞬间比对DOM树属性（如是否从false变为true）。如果没有变化，直接向云端大脑返回硬性的负反馈信号：“状态未改变”。这种“端侧状态比对+云端逻辑反思”的组合，是高效率基建的主流选择
	层级化规划与长期目标锚定	• 低层Actor在细节中打转，忘记了最终目标	• 对于跨系统、长链路的任务，这是主流的架构。它模仿人类，设计“CEO（Planner）”和“员工（Actor）”。 • CEO制定全局目标，员工负责点击。如果员工报告连续5次点击提交按钮未果，CEO会介入，判断“该报销路径阻塞”，从而修改全局规划为“撤销该单据，重新填报”，彻底跳出微观死循环。这对于提升基建在极端复杂场景下的通过率至关重要。
第三类：兜底机制	兜底安全机制	• 大脑彻底发疯，上面的机制全失效	• 在System Prompt中写入硬规则：“如果你连续3次执行相同动作且Observation report无变化，你必须被强制强制尝试不同的操作路径。”同时，在代码层面设置一个hard-coded的阈值（如N=10），一旦达到，强制Agent报错或复位。这是风险控制的主流手段。

解决方案 3：交叉验证和置信性评分，降低泛化环境下的“决策幻觉”

- 造成决策与规划模块出现“决策幻觉”的主要原因有三个，分别是大模型的“看不准”“不懂装懂”和“异想天开”，因此针对以上的三个原因，分别形成了三种针对性的机制，即多模态语义锚定与交叉验证、置信度评分和动作可行性沙盒。

- 多模态语义锚定与交叉验证：决策中枢在下达物理坐标指令前，会强制进行“双源对齐”。一是视觉与底层代码的交叉比对，当VLM基于视觉截图输出“基于xx坐标的提交按钮”时，决策引擎不会立刻放行，而是去查阅感知模块上报的底层UI树。如果发现相应坐标在底层代码中只是一个不可交互的背景板，引擎会立刻判定VLM产生了“视觉幻觉”，并在内部打回重做。二是基于SoM的实体锚定，对于彻底没有底层代码的黑盒环境，决策模块会调用端侧的轻量级目标检测模型，在所有疑似可点击的元素上画上数字编号。强制大模型“看编号做决策”，将极其容易出错的“绝对坐标心算”，降维成准确度更高的“多选题”，从根源上消灭了空间位置幻觉。

- 置信度评分：针对面对极其模糊或罕见的泛化界面，大模型“强行瞎猜”导致的灾难性业务后果。决策模型被赋予了“承认自己不会”的能力。一是置信度评分阈值，在生成每一步规划时，决策模型必须同步输出其对当前界面理解的“置信度评分”。二是动态退避与求助，如果Agent遇到完全陌生的复杂泛化环境，导致推演的置信度跌破安全阈值。系统级状态机会立刻挂起当前任务，触发人机协同机制。它会将当前的疑问通过弹窗发给人类员工，并且寻求决策指导。

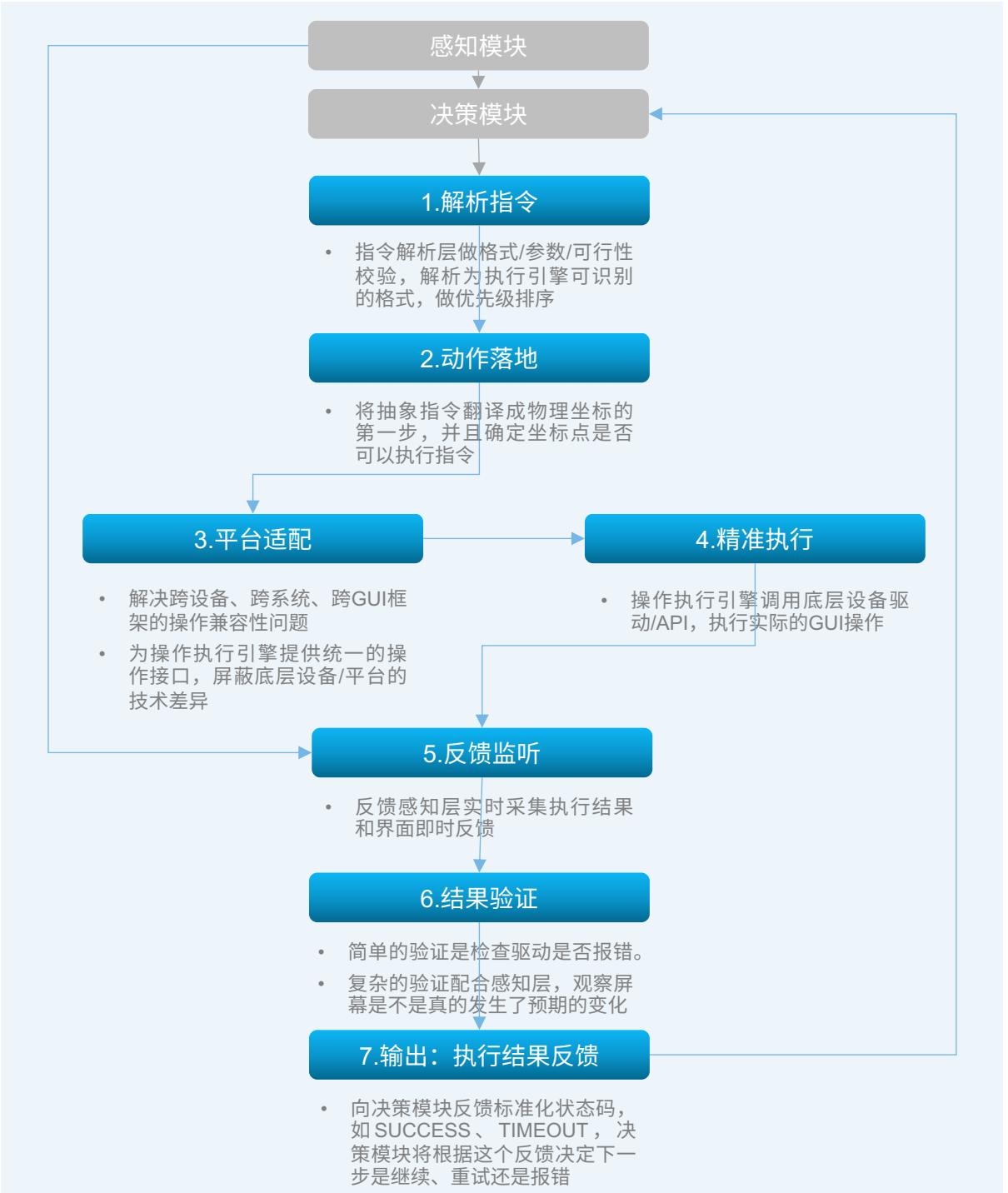
- 动作可行性沙盒：大模型在泛化环境中容易生成逻辑上听起来合理，但在当前物理状态下根本不可能执行的幻觉指令，因此在技术上，在决策指令流向“执行模块”之前，必须穿过一层“逻辑沙盒”。一是校验当前的物理状态，当大模型企图执行某个指令时，沙盒会反向校验环境状态是否处于可见和可执行的状态。二是强制状态预期声明，Agent强制要求大模型在输出动作的同时，必须附带其“预期结果”。通过这两层机制为大模型的发散性套上了严谨的“物理枷锁”，尽量确保每一次动作输出“语法正确”，而且“逻辑自洽”，降低泛化环境下连环错误的可能性。

2.2.4 执行模块：从决策指令到实际操作的最后一环

在GUI Agent的技术架构中，如果说感知模块是“眼睛”，决策模块是“大脑”，那么执行模块就是Agent的“双手”，它是连接“数字大脑”与“数字世界”的最后一道桥梁。它的核心职责是将决策模块生成的抽象指令转化为操作系统能够识别的具体底层信号，同时将执行结果、界面反馈、操作异常实时回传至感知和决策模块，形成 “感知→决策→执行→反馈”的闭环。

2.2.4.1 核心职能

图：执行模块流程



职能1：动作空间，定义 Agent能做什么动作

- 这是Agent的能力边界。执行模块首先需要定义一套标准的“动作词汇表”。主要包括原子操作、复合操作、API调用。
- 原子操作：最基础的硬件模拟信号，包括鼠标类（Move、Click等）、键盘类（Type、Press等）、触控类（Tap、Swipe等）。
- 复合操作：为了提高效率，封装好的一连串动作，比如“全选并复制”“自动填充并回车”等。
- API调用：直接调用系统API，比如直接调用Python脚本读取文件。

职能2：驱动适配层，适配具体干活的中间件

- 由于操作系统和应用环境千差万别，执行模块必须包含一个强大的适配层，用来屏蔽底层差异。
- Web端驱动：主要技术栈包括Selenium,、Playwright、Puppeteer，特点是能够直接操作DOM元素，稳定性最高。
- 桌面端驱动：主要技术栈包括PyAutoGUI,Windows UIAutomation,Accessibility API等，特点是需要处理复杂的窗口句柄和屏幕坐标转换。
- 移动端驱动：主要技术栈包括Appium,Android ADB(Android Debug Bridge)等。

职能3：动作落地，解决‘点哪里’的问题

- 这是执行模块最难的技术点。决策层只说“点击提交按钮”，执行层必须将其翻译为精确的坐标。
- 坐标映射:将DOM元素的ID或SoM标记的数字，反查出其在当前屏幕分辨率下的中心点坐标。
- 校准:处理不同分辨率、DPI缩放带来的坐标偏移问题。
- 可视性检查:在点击前，检查该元素是否被其他弹窗遮挡。如果被遮挡，执行模块需要报错或尝试滚动屏幕。

职能4：安全与反馈，确保动作有效且无害

- 执行模块在做完动作之后，对动作进行检查。
- 执行反馈：检查动作是否做完，并且返回状态码，比如Success 、Fail、Timeout。
- 安全沙箱：一是对敏感操作进行拦截并且请求人工确认，二是对操作频率进行限制。

2.2.4.2 面临挑战

当前执行模块面临的所有技术困难，本质都是标准化的执行逻辑和技术方案与异构、动态、复杂、低算力的实际落地场景的核心矛盾，同时需解决执行反馈失真、高风险场景操作管控等落地痛点。

挑战1：跨平台和异构GUI环境的通用适配难题，适配成本高、覆盖度低

这是执行模块最核心的落地瓶颈，执行模块的核心价值是“一次指令多平台执行”，但实际GUI场景的设备、系统高度异构，导致适配成为最突出的难题之一。

- 底层操作接口碎片化：无统一标准，不同的设备、系统的GUI操作底层驱动独立，缺乏统一执行接口，执行模块需要为每个平台开发专属适配插件，维护成本随着平台数量指数级上升，无法真正实现跨平台统一执行。
- 跨平台和跨应用操作：不同平台和应用在操作逻辑、执行环境不同，造成衔接困难或者无法执行。

挑战2：执行的精准度和稳定性问题，坐标漂移和失控错误

执行模块要求的是在准确的时间实现像素级的精准操作，但GUI Agent在执行过程中，面对的是空间、时间和多层级的结构，因此执行中经常出现的就是空间错位、时序错位和层级错位。

- 空间错位：最主要的表现就是“坐标漂移”，一是由于Agent在云端和设备之间迁移造成成比例的扩大或者缩小，造成执行过程中的偏移；二是Agent面对紧凑排列的按钮点击不准确；三是在多显示器或者窗口飞最大化场景下，Agent操作错误。

- 时序错位：这是Agent在动态网页和远程操作中经常出现的现象，一是未等页面元素加载完毕，Agent已经执行完动作，出现“空挥”现象；二是因为有横幅广告、异常跳动导致的误触；三是在拖拽或者长按的过程中，由于延迟和卡顿造成的操作失败。
- 层级错位：Agent在操作中物理坐标识别正确，但是目标按钮被挡住，导致操作错了层级，比如页面上肉眼无法发现的透明广告、弹窗和不可交互的状态栏。

挑战3：高等级安全风控与“反作弊”拦截

- 部分终端、高涉密系统、反外挂引擎会屏蔽软件级的虚拟按键，Agent有时候无法跨过这道安全拦截，无法进行任务的操作。

2.2.4.3 技术考量维度

如果说决策与规划模块的考量维度是关于“智商与逻辑”的算力博弈，那么执行模块的技术考量维度，则完全是关于“物理渗透与系统底座”的工程化博弈。

考量维度1：跨OS架构的“原生调用与统一抽象”博弈

- 面对Windows、macOS、Linux和移动端极其碎片的底层代码，执行引擎怎么写才能兼顾性能和可维护性，博弈的两端分别为纯原生绑定和构建统一的抽象隔离层。
- 纯原生绑定的路径：为Windows写一套C++的Win32 API调用，为macOS写一套Swift的XPC调用。优势是执行效率最高，延迟最低；劣势是成本极高，版本更新困难。
- 构建统一的抽象隔离层：定义一套统一的“动作协议”。无论上层AI输出什么，都先翻译成统一协议，再由各平台极简的Runner去执行。优势是将“AI大脑”与“OS四肢”彻底解耦，极具扩展性。劣势是增加了一层协议转换的开销。

考量维度2：物理映射的“绝对指令与自适应校准”博弈

- 博弈的核心是，大模型吐出的逻辑坐标，在不同用户的真实物理屏幕上，该如何精确对齐。
- 绝对坐标：大模型要求点击哪个坐标，鼠标就直接飞到目标位置。优势是成本较低，劣势是泛化性差，如果用户电脑有DPI缩放或者有多屏拼接，会导致“点错位置”。
- 动态空间几何转换：执行模块内部维持一套实时的“屏幕空间转换矩阵。优势是引擎能够通过矩阵运算，将“逻辑坐标”映射为”物理所表“，劣势是需要实时侦听显示，增加计算负担。

考量维度3：控制深度的“侵入性与反屏蔽”博弈

- 核心内容是Agent应该在操作系统的哪一层去模拟人类的鼠标和键盘操作。
- 应用层API（浅层）：使用Selenium、PyAutoGUI或基于Accessibility的接口。优势是开发极快，跨平台成本低；劣势是较为脆弱，遇到网银客户端、政务加密系统容易被拦截。
- 内核态驱动/硬件级模拟（深层）：在操作系统的Ring 0层注册虚拟HID驱动。优势是能够避免软件级的风控拦截；劣势是开发门槛较高，且容易引发系统级崩溃。

2.2.4.4 技术解决方案

解决方案1：跨平台/异构GUI环境的通用适配难题

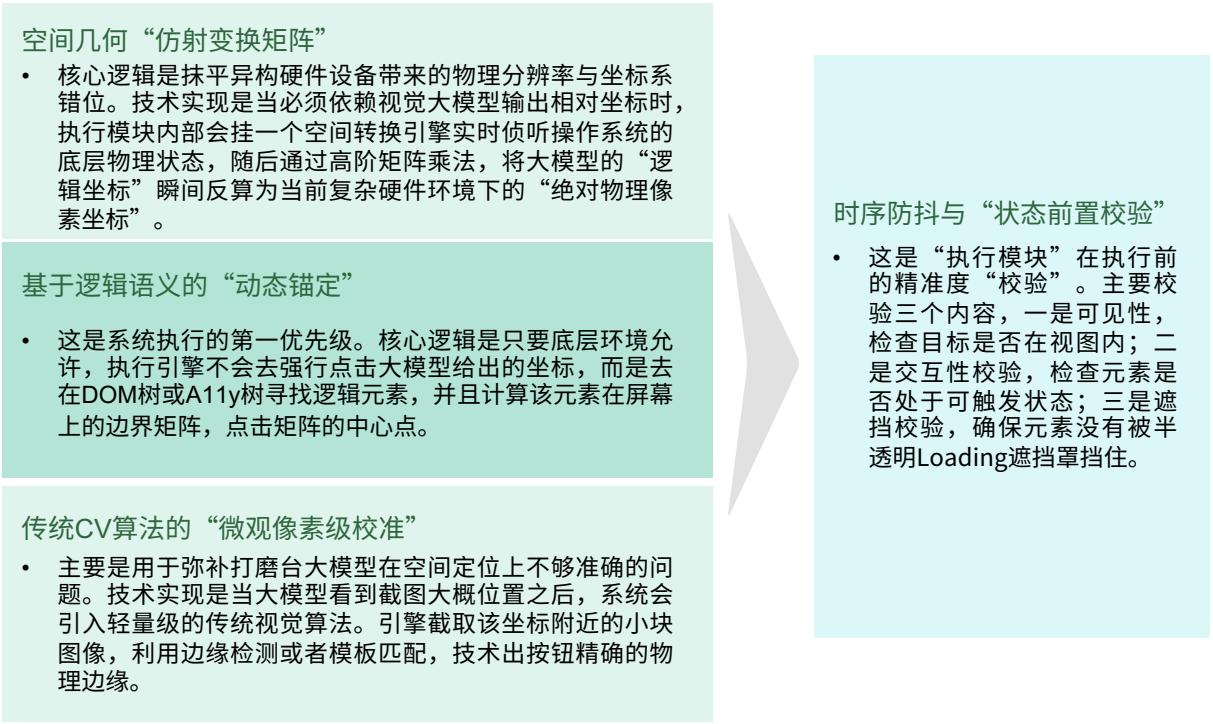
- 当前业界有以下三个主流解决方案，来解决Agent的通用适配难题。
- 基于“无障碍标准与DOM”的语义树方案：系统直接全盘读取Windows的UI Automation树或macOS的Accessibility API树，将极其复杂的异构界面，降维统摄成了一棵“逻辑语义树”。Agent只要遍历这棵树，就能瞬间知道哪里是按钮、哪里是输入框，实现了跨OS的无缝寻址。
- JSON指令协议+端侧原生执行器：核心逻辑是在云端大模型和本地操作系统之间，拉起一道强隔离的“动作抽象层”。在云端，大模型只输出统一格式的JSON意图；在端侧，目标设备上部署轻量级中间件，Web端主流选用Playwright或Puppeteer，桌面端选用封装好的Python库。它们接收到JSON后，在本地静默翻译成系统原生的底层调用并执行。优点是未来接入任何新系统，只需要开发一个轻量级的端侧Runner，适配成本较低。

- 纯视觉多模态方案：该方案一般用于兜底，即在其他方案失效的情况下，通过SoM的方式直接读取标签，根据标签计算绝对的物理坐标，发送物理鼠标点击。

解决方案2：执行的精准度和稳定性问题

- 针对执行的精确度和稳定性方面，当前形成了由逻辑语义动态动态锚定、空间几何仿射变换矩阵、传统CV算法接入的像素级校准和状态前置校验组成的“四位一体”的解决方案。

图：精准度和稳定性的解决方案



解决方案3：高等级安全风控与“反作弊”拦截

针对安全防控和“反作弊”拦截，当前形成了两种主要方案，一种是“环境隐匿+仿生轨迹”的路线，该路线配置更为简单，权限要求也更低；另一种是内核级虚拟硬件驱动，该方案穿透性更强，但研发门槛更高。

“环境隐匿+仿生轨迹”的路线

- 该方案的的基础原理是在行为层采用仿生学物理运动引擎，应对风控系统的“行为审计”；在环境层通过底层指纹抹除应对Web端的防御。该方案的优点是布置相对简单，边际成本低，较易大规模并发部署。劣势是权限较低，在某些场景下会失效。
- 环境隐匿——底层指纹抹除：技术方案是通过端侧Runner启动浏览器或应用的瞬间，进行极深度的环境隔离与伪装，一是通过底层Hook，强行抹除所有暴露自动化框架存在的探针标记；二是动态伪造Canvas绘图指纹、AudioContext音频指纹，以及合法的User-Agent和真实的屏幕分辨率信息，进行硬件指纹欺骗；三是直接修改Chromium等底层引擎的C++源码重新编译，从根源上摘除所有能被JavaScript探测到的自动化接口。
- 行为层伪装——仿生学物理运动引擎：用复杂的数学物理模型，逆向生成带有“人类生理学瑕疵”的运动轨迹，影响安全风控系统的判断，达到操作的目的。

内核级虚拟硬件驱动

- 该方案属于系统级方案，主要是应对高涉密客户端软件对API操作的直接屏蔽。
- 技术实现是在系统中注册一个带有合法企业数字签名的虚拟USB驱动，当大模型下发点击指令时，引擎不经过任何应用层API，直接向操作系统的底层总线发送原始的硬件中断电信号，操作系统和安全软件会无条件信任执行操作，最终实现对高加密软件的降维穿透。

2.2.5 反馈优化模块：从“新手”进化为“专家”的关键组件

如果说决策模块是“大脑”，执行模块是“手”，那么反馈优化模块就是Agent的“免疫系统”和“进化引擎”。它的存在是为了解决Agent在不可预测的动态环境中的鲁棒性和自适应性问题。反馈优化模块是决定Agent能否从“新手”进化为“专家”的关键组件。

2.2.5.1 核心职能

反馈优化模块的职责归纳为“查、析、改、存”，即结果校验、归因诊断、动态修正和经验进化。

- 结果校验：判断上一步动作是否生效，以及是否达到了预期的子目标。比如决策层下令“点击登录”，执行层点完了。反馈层通过阅读屏幕，发现还在登录页且弹出了红色报错，于是判定：“动作执行物理成功，但业务失败”。
- 归因诊断：当校验不通过时，分析错误产生的根本原因。比如反馈层需要阅读屏幕上的报错信息，将其转化为机器可理解的错误原因。
- 动态修正：根据诊断结果，生成实时的补救策略，指导决策模块调整下一步计划。比如如果错误原因是“密码错误”，策略是“尝试另一个密码”。
- 经验进化：将任务的成功路径或失败教训写入长期记忆，下次出现类似情况直接调用。

2.2.5.2 面临的困难和挑战

反馈优化模块最重要的作用是对Agent的操作进行校验和修正，校验和修正的前提是能够对Agent的操作准确性和标准进行正确的判断，因此，反馈优化模块面临的三个主要挑战就是对动作验证的“状态误判”、判官模型的“阿谀幻觉”以及归因和追溯困难。

挑战1：动作验证的“状态误判”

- 在传统的监督学习中，数据都有标签。但在Agent运行的真实GUI环境中，缺乏明确的“成功或失败”信号。具体表现为假阳性和缺乏量化标准。
- 假阳性：比如Agent以为自己订票成功了，比如看到了“恭喜”两个字，但其实那是广告弹窗上的“恭喜中奖”，真正的订单并没有生成。
- 缺乏量化标准：对于美化图片和PPT这种模糊任务，缺乏量化标准，导致反馈失败。

挑战2：判官模型的“阿谀与幻觉”

- 通常情况下，GUI Agent是用一个大模型（Critic）去检查另一个大模型（Actor）的工作，但是缺乏一个绝对客观的验证层。
- 阿谀效应：大模型倾向于讨好人类或确认自己的前序行为。如果Agent自己觉得自己做得对，Critic模型往往也会顺着说“是的，你做得对”，从而漏过错误。
- 能力倒挂：如果执行任务的是GPT-4，而为了省钱用 GPT-3.5做Critic，那么裁判员根本看不懂运动员的高级操作，会导致错误的反馈。

挑战3：归因和溯源困难

- 在长链路任务中，Agent可能没有将奖励或者错误进行正确的归因，导致后续任务执行的失败。
- 错误的归因溯源：比如Agent可能会因为一次网络延迟导致的任务失败，错误的归因为“我不应该点那个按钮”，导致下次正常情况也无法使用该按钮。
- “奖励设计”的时效：如何在Agent执行过程中，每一步都给予适当的反馈，一直是强化学习面临的难题。比如在长链路任务中，从第1步到第49步，Agent做的都非常成功，但第50步失败导致任务失败，这就造成Agent可能出现步骤困惑。
- 经验库失效：反馈优化的终局是更新Agent的记忆参数和模型，但是随着任务使用场景的增多，造成任务执行中噪音增大，比如模型学会了“京东”界面的操作，却错误的用在“淘宝”界面。

2.2.5.3 技术考量维度

反馈优化模块的技术考量维度主要集中于如何判定出现的错误，并且在判定出现错误之后，要通过何种方式去处理错误。

考量维度1：对反馈信号的判定深度

- 对反馈信号的判定深度需要Agent做到反应速度快，并且不出现自欺欺人的“幻觉”。Agent设计需要在“底层确定性”和“视觉语义深度”之间追求平衡。
- 依赖“底层确定性”：直接监听操作系统底层的物理状态，如拦截网络层的AJAX请求状态码或者监听前端DOM树的节点变化。优势是几乎无延迟、无成本，但容易出现不了解业务流程出现的“虚假成功”。
- 依赖“视觉语义深度”：操作完成后，截取当前的屏幕，并且发给多模态大模型对状态进行判断，优势是大模型有强大的认知能力，缺点是延迟时间长、算力成本高，而且大模型会出现幻觉。

考量维度2：系统出现错误的兜底能力

- Agent在执行过程中，通常环境是充满变数，当发生预期之外的错误时，Agent通常会采取自主执行或者熔断退出两种方式。
- 弹性自主推演自愈：信任和释放大模型的泛化推理能力，当Agent遇到的错误反馈给大模型时，大模型会自主规划和拟定一条补救路线，优势是能够提升Agent的执行韧性和流畅度，劣势是大模型极易在补救过程中出现连环错误和越权操作。
- 熔断退出：当模块发现当前状态出现偏离时，立刻触发系统异常，冻结后续所有任务，Agent原地宕机并且生成错误日志。优势是极大降低了错误概率，劣势是扼杀了大模型的泛化能力。

考量维度3：长期进化模式的博弈

- Agent的长期进化模式决定了产品的天花板，也是产品拉开代差的重要手段。主要方式有模型参数微调和动态经验池与RAG外挂两种方式。
- 模型参数微调：将错误案例打包进行重新训练，优势是反应快，但算力成本高，无法及时生效，而且存在遗忘风险。
- 动态经验池与RAG外挂：不改动模型参数，将错误教训转化为文本规则，存入本地数据库，下次出现同类情况时动态注入System Prompt。

2.2.5.4 技术解决方案

解决方案1：引入双塔多模态校验，解决动作验证的状态误判

- 针对Agent的状态误判问题，当前形成了以验证标准前置、“环境静息”侦测、双塔多模态校验等三种方案于一体，前、中、后结合，按时间顺序组成的动作校验组合。
- 基于契约的“预期状态断言”注入：发生于决策和规划模块生成动作的瞬间，动作还未发给执行模块。实现过程中，决策与规划模块输出的指令不是简单的开放性指令，而是必须强制绑定一个断言字段，比如“页面出现绿色的支付成功对勾，或者跳转回主页”，并且大模型对预期产生的结果校验。
- 防抖延迟与“环境静息”侦测：产生于动作发生的瞬间，主要是用于确认系统已经处于真正的稳定状态，解决动态UI加载过程中的时序错位问题。一方面通过网络静息侦测，执行模块点击之后，引擎会监听底层网卡流量，只有连续500毫秒内没有新的网络请求发出或者接收，才认为前端数据加载完毕。另一方面监听前端UI树的重绘事件，只有当页面停止“闪烁”、转圈画面消失，反馈模块才被允许截图校验屏幕。
- 双塔多模态校验引擎：目前业界解决状态误判较为主流的架构方案。它将反馈动作拆分为“物理级确认”与“业务级确认”两座校验塔。左塔通过底层确定性嗅探，技术原理是动作发出的瞬间，端侧探针以毫秒级速度监听操作系统的底层反馈。例如，网络层的AJAX请求是否返回或者鼠标是否真实发生了位移，如果左塔探测到物理阻断，直接向大脑报错，无需唤醒VLM。右塔主要负责视觉语义对齐，当左塔确认“物理动作已执行”后，系统截取当前屏幕，将其送入右侧的多模态大模型通道。右塔不看代码，只看“画面语义”。它能像人类一样读懂弹出的红色警告框，如“余额不足”，从而推翻左塔的物理成功判定，输出真实的业务级失败信号。

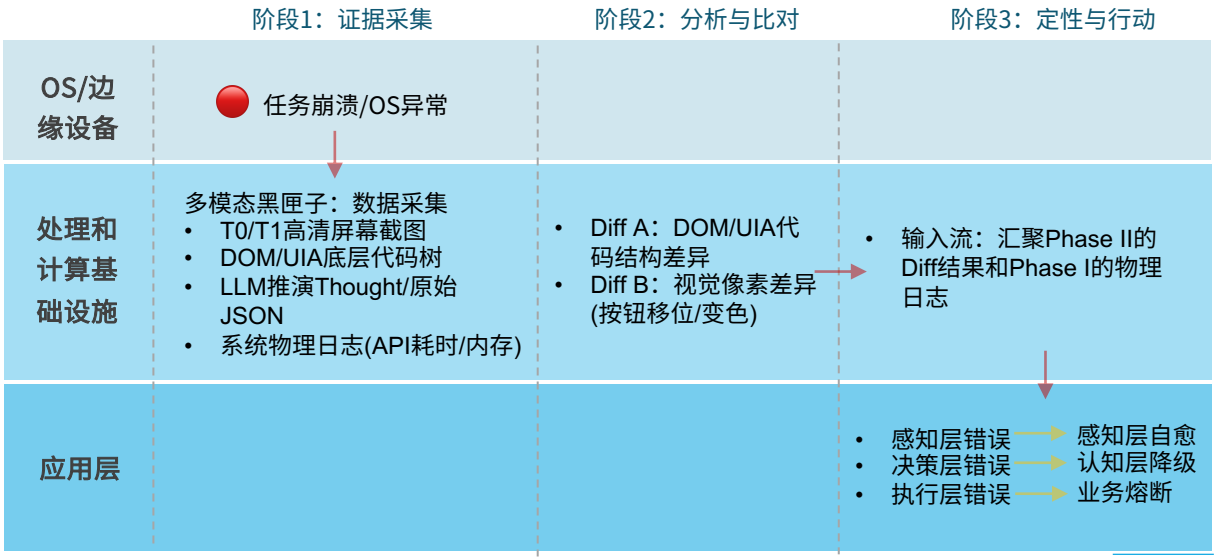
解决方案2：引入工程制衡策略，解决大模型的阿谀奉承

- 在反馈优化模块，因为Agent要对从感知到执行的模块进行检查、反馈和优化，是Agent的重要兜底，因此大模型的“讨好型人格”会对Agent的工作产生更为严重的影响。因此，在技术实现上，在输入侧、算力侧、任务侧引入了上下文剥夺、模型异构化、时序差分对比三种方式，杜绝大模型出现阿谀奉承，影响Agent的反馈和校验。
- 上下文剥夺（输入侧）：在技术实现上，在“执行模块”和“右塔判官模型”之间设立一道严格的信息防火墙。当右塔被唤醒时，系统禁止将决策大脑之前的思考过程或执行动作透传给判官。判官收到的Prompt被强制剥夺了所有前置上下文，变成了一场极其冷酷的“盲测”。判官模型只需要判断执行前的“预期状态”和执行后的截屏，判断执行是否成功并且给出理由。
- Actor-Critic模型的异构化（算力侧）：实现逻辑是避免Actor和Critic使用的是同一个大模型，防止同一个模型产生同样的幻觉。因此，在技术实现上，Actor追求泛化与创造力，采用参数量极大的多模态模型，Critic模型追求追求刻板、精准与低延迟，采用专门针对UI界面微调过的、参数量较小的端侧专有模型，两种模型相互校验，降低同一种幻觉出现的概率。
- 时序差分视觉对比（执行侧）：主要逻辑是赋予Critic模型“找不同”的能力，通过执行前和执行后的截屏对比，让Critic模型强制找到操作前后的差异，并且找到逻辑。

解决方案3：建立多模态诊断网络，解决归因和溯源困难

- 针对“反馈优化模块”的归因和溯源困难，根据成熟度当前形成了“大模型反思”“RPA+大模型”和“多模态原生基建流”三种路径，三种路径从易到难，对架构设计的要求也逐步提高。
- 纯大模型反思流：绝大多数早期的开源项目和极客写的轻量级自动化脚本，架构设计较为简单，当操作报错时，系统只是简单地捕获一段Python的Exception文本，然后把这段纯文本丢给大模型，问：“报错了，你分析一下为什么？”劣势就是极易出现严重的归因幻觉。
- “RPA+大模型”缝合流：高度依赖前端代码的DOM树层级和XPath绝对路径。它们的“归因”极其机械，如找不到设定的Xpath，就直接报UI_Selector_Not_Found。劣势就是泛化性极差并且由于没有“基准环境Diff引擎”，无法对错误做到争取的归因和区分。
- 多模态原生基建流：这是一个步骤最为复杂、对基础设施要求最高和可靠性最高的方案。该方案由三个主要部分组成。一是端侧多模态“黑盒”快照留存。在Agent运行底层，挂载一个极度轻量级的“时序记录仪”，几率每一次执行操作前后的对照，包括高清截图、DOM树、Prompt输入、JSON输出和思考过程。二是基础环境漂移Diff引擎。主要用于解决动态环境下的非预期问题。逻辑是当Agent在某一次任务成功之后，Diff引擎会把成功案例保存并且存入本地数据库，业务再次崩溃之后，Diff引擎会对前后两个案例进行交叉对比，并且进行计算。三是将以上内容按照步骤进行分析并且反馈给系统或者工程师。

图：多模态原生基建流的工作流程



2.2.6 沙盒：GUI Agent的安全可控性保障

在GUI Agent的技术架构中，沙盒是GUI Agent工业化落地的核心基础设施。沙盒是核心的“安全与可控性保障组件”，其本质是一个隔离的虚拟环境，即Agent的所有操作，比如界面交互、数据处理、外部调用，均在沙盒内执行，而非直接作用于真实系统。沙盒的存在，确保了Agent执行环境的绝对安全，同时在必要的时候能够通过虚拟技术提升Agent性能，推动实现商业化和规模化。

图：GUI Agent中沙盒的核心作用

模块	作用
感知模块	• 沙盒为感知模块提供纯净的环境。感知模块需要看屏幕。沙盒通过其内部的“虚拟显存”，为感知模块源源不断地输出高保真的像素流和底层DOM和A11y树。 • 更重要的是，沙盒排除了外部物理环境的干扰，比如不会有人突然晃动鼠标，不会有微信突然弹窗，确保感知模块“看”到的数据是100%纯净且与任务相关。
决策与规划模块	• 沙盒是决策模块的“防火墙”。决策模块负责高维度的战略思考，沙盒处理了所有底层的物理与环境伪装细节，让决策大脑可以全神贯注地规划任务路径。
执行模块	• 沙盒是执行模块的“物理受体”。沙盒为执行模块提供了真正的“着力点”，让Agent的虚拟鼠标有地方滑、虚拟键盘有地方敲。同时，沙盒底层的动态指纹伪装，也极大地保护了执行动作不被目标系统的风控引擎拦截。
反馈优化模块	• 沙盒为反馈优化模块提供了安全环境。当执行失败、目标软件崩溃、甚至Agent遭遇了网页病毒时，沙盒能在不危及宿主机的前提下，安全地截取“崩溃瞬间的屏幕快照”和“底层系统报错日志”。

2.2.6.1 核心职能

整个沙盒架构构建了一个从指令下达到状态感知的精密物理隔离闭环。在输入阶段，外部云端大脑首先向系统注入“初始化配置参数”，随后持续下发“动态执行指令流”来驱动具体的任务运转。

指令进入沙盒后，所有的操作和“不受信任的应用程序”均被严格禁锢在受限资源池中。沙盒通过构建虚拟文件系统、限制网络访问并控制CPU/内存分配，将其与底层的宿主系统切断。所有的底层交互都必须经过系统调用过滤，确保操作的爆炸半径被锁在容器内部。

动作执行完毕后，沙盒在输出阶段向外回传“多模态视觉与语义快照”供大模型进行视觉感知，同时精准输出“状态反馈与底层遥测”作为任务的评估依据，从而完美实现安全、闭环的跨OS智能体交互。

图：沙盒的内部机制以及输入和输出

2.2.6.2 面临的困难和挑战

沙盒是GUI Agent技术最深、成本最高的模块之一。沙盒解决的最主要问题就是异构环境统一、精准性和稳定性的问题，主要面临的挑战也在于隔离与性能矛盾、精准度和稳定性、开发和维护成本。

挑战1：隔离级别与性能和安全的矛盾

- 隔离与性能的矛盾表现在，隔离越强，安全性越高，宿主设备的消耗也越大、延迟也就越高；轻量沙盒隔离较弱，但是又容易出现逃逸、污染等现象，GUI 实际使用场景中既要求强隔离，又要求低延迟，在平衡方面存在巨大困难。

挑战2：精准性和稳定性问题

- 沙盒让大模型与目标软件隔了一层“虚拟玻璃”，这层玻璃会导致严重的感知和执行偏差。主要原因包括三个方面，一是视频压缩损耗与OCR致盲，高压缩率会导致画面模糊；二是网络延迟会造成时空错位；三是虚拟操作的屏幕分辨率与DPI缩放比例造成物理了物理漂移。

挑战3：环境一致性造成的操作失败

- 沙盒为了追求极致的隔离和并发密度，制造了一个苛刻的运行环境，这些环境的维度影响了操作的稳定性。一是算力的节流与内存溢出，当Agent在沙盒里打开一个机器消耗内存的商业软件是，容易造成内存溢出机制，导致目标软件被强制关闭；二是沙盒主打“阅后即焚”，这种过度纯净的操作逻辑会触发目标软件和网站的重复操作；三是虚拟驱动可能会遭遇底层的物理排斥，造成软件界面“卡死”。

挑战4：开发和运营成本

- 开发成本：一是异构底层的深度虚拟化与物理穿透成本，研发团队要对键盘、鼠标、显卡进行虚拟化，并且要跨越多个生态进行底层适配，时间和资金成本极高；二是专供视觉大模型的流媒体基建成本，GUI Agent沙盒需要打造基于WebRTC的像素推送技术。
- 运营和维护成本：一是GUI沙盒为了渲染真实的图形界面，资源消耗远高于API；另一个是GUI环境极其脆弱，运营团队需要耗费极大精力去修补环境漏洞和漂移。

2.2.6.3 技术考量维度

考量维度1：隔离级别和虚拟化选型

- 隔离级别和虚拟化选型影响启动速度、高并发密度和安全级别，决定了“用什么技术把Agent关起来”。当前主要有传统全虚拟化、操作系统级虚拟化、微虚拟机、安全沙盒容器等几种方案。

考量维度2：环境一致性与防漂移机制

- 环境一致与防漂移机制主要是需要保证每次启动都是标准、干净而且可以复现的环境，包括镜像是否只读、是否使用写时复制、如何禁止系统更新、弹窗、主题、DPI变化等，以及为了保证以上细节的完美呈现采取哪些机制。

考量维度3：性能开销与资源效率

- 优秀的沙盒架构要避免成为资源的无底洞，而是要成为算力的放大器。GUI Agent对内存和CPU消耗极大，同时要求低延迟和低消耗。因此，沙盒的设计要在保证隔离、画质和延迟度前提下，最大化的降低CPU、GPU、内存和算力的消耗。

考量维度4：GUI渲染与画面采集能力

- GUI渲染与画面采集能力是感知模块的基础，决定了Agent是否能够完成“感知”的第一步动作。考量的维度主要包括屏幕的采集方式、是否支持硬件加速、透明窗口、分层窗口以及渲染的方式和画面传输的方式和效率。

考量维度5：输入模拟精度与兼容性

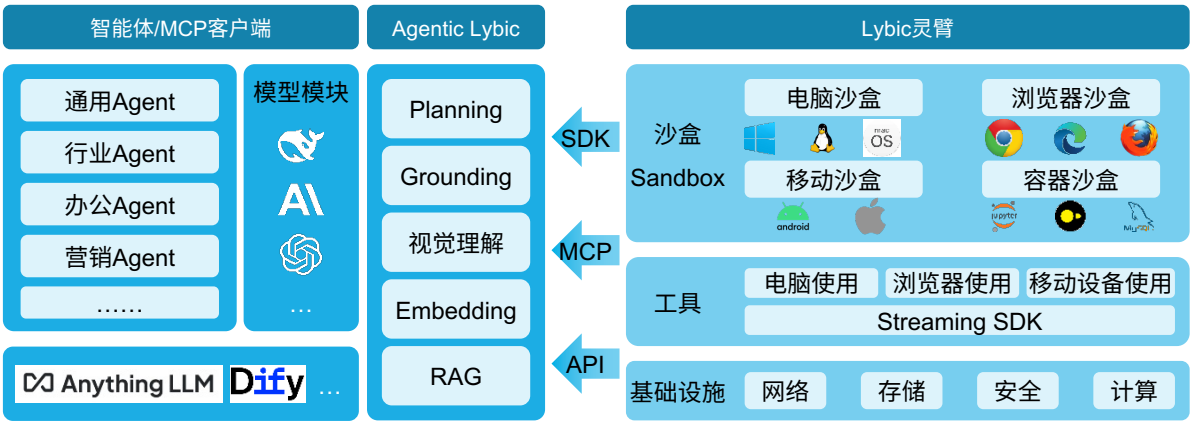
- 输入模拟精度与兼容性是执行模块的基础，决定了执行模块是否能够将感知和规划的行动进行准确的执行。主要考虑的因素包括虚拟鼠标和键盘的精准程度、输入驱动级别、操作稳定性以及能否或者是否需要绕过安全和反拦截软件。

2.2.6.4 技术解决方案：庭宇科技云端智能沙盒解决方案

沙盒是GUI Agent落地的核心基础设施，沙盒的搭建需要在隔离强度、兼容性、经济性之间取得平衡，同时需要监督操作精度与各类平台框架，可以沙盒是GUI Agent真正走向落地中最具挑战的模块。庭宇科技依托自身在GUI Agent和边缘云领域的深耕，成功打造了“沙盒+GUI Agent”的一体化解决方案Lybic。Lybic的使命是“为你的智能体提供电脑和手机”，即开即用，能够在几分钟内为智能体构建电脑和手机操作能力。

为你的智能体提供
电脑和手机

图：Lybic灵臂产品技术架构图



图：Lybic灵臂产品优势

视觉引擎·像素级操作

- Lybic结合了无损高保真流媒体推流与多模态探针技术，让大模型既能获取底层的UI逻辑树，又能像人类一样直接“看”屏幕像素。

视觉引擎·像素级操作

- Lybic通过云原生架构，将极其沉重的底层IaaS/PaaS资源进行了高度封装。通过开放标准API，零基础设施负担，支持模型直接调用，无额外依赖和使用限制。

跨语言兼容·全栈适配

- Lybic在中间件层做了大量的接口适配和协议转换工作，其底层控制指令能够与当前主流的AI编排框架无缝对接。确保适配LangChain、Vercel等技术栈，即插即用，无需改动既有代码。

优势

全球覆盖·秒级部署

- 依赖庭宇科技强大的分布式边缘计算网络，以及极其硬核的容器快照热恢复技术，消灭了传统虚拟机长达数十秒的冷启动开机过程。实现在任意区域毫秒级启动安全沙盒，快速联通使用。

智能调度·大规模并发

- Lybic通过底层的Kubernetes容器编排、vGPU细粒度切分以及内存去重技术，打破了单台物理服务器的并发天花板。支持千级任务并发，系统自动扩缩容，保障任务高效稳定运行。

实时可控·自动化执行

- 系统在执行自动化脚本的同时，保留了人类通过WebRTC或WebSocket直接下发键鼠中断的最高权限通道。任务实时流式呈现，支持暂停、编辑、续行，可实时监控，随时介入操作。

2.3 主流技术路径

2.3.1 三大技术路径

当前GUI Agent形成了三个较为主流的技术路径，分别是端到端视觉大模型路径、代码生成路径和多智能体协作路径。从系统级设计的高度来看，主流的技术路径并非仅仅是决策算法的区别，而是“感知、决策、执行、反馈”这四个模块如何组装和协作的根本差异。

路径1：端到端视觉大模型路径

端到端视觉大模型路径是目前最前沿、也是OpenAI(GPT-4o)和Anthropic(Claude Computer Use)主推的路径。核心设计逻辑是“Pixel-to-Action”，即整个Agent就是一个巨大的多模态模型。它不需要复杂的DOM解析，直接看屏幕截图，直接输出点击坐标。

端到端视觉大模型路径的优势是通用性极强，任何软件都可以使用。核心痛点是“通用模型的GUI场景效率低、精度不足”，因此设计思路是“让模型专而精”，即通过在GUI界面数据集（如Web UI、APP界面截图）上对小模型进行微调，使其天生擅长“控件识别+坐标定位”，无需依赖SoM标注等辅助环节。流程设计上追求“紧凑性”，减少中间数据流转成本，让“截图→指令”的延迟最短，从而适配高频、高精度的GUI自动化场景，实现“效率+精度”的最优解。

图：垂直端到端模型GUI Agent工作流程

GUIAgent 工作流程	调用工具	与上下环节的协调逻辑
用户请求		• 触发流程，向“界面截图”环节传递需求，明确操作目标（如“批量输入手机号”）
界面截图	• GUI Agent截图组件（优先调用底层渲染数据）	• 接收用户需求触发，捕获当前界面视觉数据（截图或渲染帧），输出给“SoM视觉标注”环节作为处理素材
调用专用VLM	• GUI垂直小模型（Fuyu-8B、SeeClick、CoAgent、MiniGPT-4-GUI）	• 接收界面截图，通过专项训练的模型能力，同时完成“控件识别+精准定位”，输出控件坐标+操作类型数据
输出归一化坐标	• GUI Agent坐标解析模块（适配不同屏幕分辨率的坐标转换器）	• 接收专用模型输出的原始坐标，转换为标准化归一化坐标，传递给执行器
输出操作	• 轻量化操作执行器（如专用RPA插件、系统原生操作API）	• 接收归一化坐标指令，直接定位控件位置并执行操作，响应速度比通用执行器快

由于端到端大模型的原理是通过模型直接看原图，并且需要对坐标进行猜测，在精确性上存在缺陷。在此基础上，各大模型和Agent厂商开发出了“通用VLM+视觉提示流”(General VLM+Visual Prompting/Set-of-Mark)这一路径。

相比传统的端到端路径，通用VLM+视觉提示流的优势是通过对模型看到的界面进行检测、分类、画框、映射等预处理，从而使模型由直接看原图，变为从原图中选择一个正确的答案，极大提升了模型的操作精度，同时抑制了大模型产生的坐标幻觉。

图：通用VLM+视觉提示流路径工作流程

图：通用VLM+视觉提示流GUI Agent工作流程

特性维度	端到端视觉大模型	通用VLM+视觉提示流
思维模式	直觉式	符号式
输出形式	坐标	标签索引ID，比如#15
准确率	低，存在漂移	极高，依赖检测器
可解释性	黑盒，不知道为什么点那里	白盒，知道它选了哪个ID
系统复杂度	低，单模型架构	中，需引入检测/OCR模型
适用场景	闲聊、创意生成、粗略浏览	RPA、复杂软件操作、表单填写

路径2：代码生成路径

代码生成路径是开源社区早期的主流路径。设计逻辑是“Text-to-Code”，即Agent不直接去“点”按钮，而是现场编写一段Python/Selenium代码，然后在一个沙箱环境里运行这段代码来操作电脑。优先调用界面原生结构化数据，结合通用大模型推理，实现精准操作的技术路径，核心是 “结构化数据+通用模型”的组合。

代码生成路径优势是结构化数据比截图更精准，能获取控件状态和逻辑关系，适合复杂场景的自动化需求，比如批量任务和精确数据处理。缺点是面对非标UI会彻底失效，比如无DOM的游戏界面。

图：代码生成路径工作流程

GUI Agent工作流程	调用工具	与上下环节的协调逻辑
用户请求		• 触发流程，向“调用DOM/Accessibility API”环节传递需求，明确操作目标
调用DOM/Accessibility API	• GUI Agent API 调用组件（适配Web的DOM解析器、移动端的Accessibility Service）	• 接收需求后，自动调用当前界面的原生API，获取控件的结构化数据，输出给“获取控件信息”环节
获取控件文本 / 界面信息	• 结构化数据解析工具（如 Cheerio、JSON Schema 验证器）	• 过滤API返回的冗余数据，提取核心信息（如表单必填项列表、提交按钮状态、控件ID），整理为标准化数据格式（如JSON）
通用模型推理	• 通用大模型（GPT-4、Claude 3 Opus）+ 代码理解工具（Code Llama、Code GeeX）	• 接收“用户需求+结构化数据”，推理操作逻辑（如“是否已填完必填项”“提交按钮是否可用”），决策操作方式（优先API调用，还是界面模拟）
输出操作	• GUI Agent操作执行器（API调用模块 + 界面模拟模块）	• 接收模型推理结果，优先调用功能API（如表单提交接口）完成操作；若API不可用，生成界面操作指令，通过执行器完成操作

路径3：多智能体协作路径

多智能体协作路径是这是Agentic OS和企业级复杂场景的首选路径。设计逻辑是系统不再是一个全能的Agent，而是拆分为“规划官”、“执行官”等多个独立的Agent实例，它们共享同一个工作流，通过专业分工，解决了单体大模型无法克服的认知负荷瓶颈。

多智能体协作路径的优势一是延展了任务的上下文记忆能力，极大地提升了长任务的完成率，二是多智能体的内部对抗机制带来了强大的自我纠错能力和质检能力，三是模块化的架构能够更方面的进行拆解和升级。核心劣势一是多智能体协作造成的成本爆炸，另一个就是多智能体相互交流造成严重的响应延迟，三是有可能陷入沟通死循环，不仅没有完成任务，还浪费了大量的Token，对编排逻辑要求极高。

图：多智能体协作路径工作流程

2.3.1.4 三大技术路径的主要区别

三条主流技术路径都依赖大模型进行语义理解和逻辑推理，但在感知、决策、执行、反馈四个模块呈现了不同的处理方式和判断标准。

在输入侧，即感知与决策层面，三者分别代表了机器、人类与组织的思维范式。代码生成路径如同“程序员”，感知依赖DOM树等结构化数据，决策是通过逻辑推演生成可执行的代码脚本，追求结构化的精准；端到端视觉路径则像“仿生人”，感知上直接摄入高分辨率屏幕截图，决策依靠大模型将图像语义直接映射为像素坐标，具备极强的跨应用泛化能力；而多智能体协作路径采用“项目组”模式，感知上由观察员Agent过滤关键信息，决策上则通过规划师Agent制定SOP，通过多角色的分层协商来解决长链路复杂任务的规划难题。

在输出侧，即执行与反馈层面，三者对“成功”的定义与纠错方式截然不同。代码生成路径的执行依赖代码解释器在沙箱中运行，反馈机制源于捕捉程序的报错日志，侧重于逻辑语法的毫秒级纠错；端到端视觉路径通过模拟底层键鼠信号进行物理操作，反馈依赖于前后截图的视觉差异对比，更贴近人类的直觉判断；多智能体协作路径则在执行层封装了标准化的工具调用，反馈上引入专门的“批评家Agent”进行逻辑质检，一旦发现结果偏离预期，会直接驳回任务要求重做，通过这种内部对抗机制确保了企业级业务的极高稳定性。

表：三条主要技术路径在不同模块的对比

项目	代码生成路径	端到端视觉路径	多智能体协作路径
感知模块	文本/结构化数据主导 • 核心依赖：HTML源码、DOM树、Accessibility Tree • 辅助：OCR，仅用于无法获取DOM时 • 逻辑：将界面解析为代码变量	纯像素/视觉主导 • 核心依赖：高分辨率屏幕截图 • 逻辑：像人眼一样直接理解图像语义，不依赖底层代码结构	混合/过滤型感知 • 核心依赖：由Observer Agent负责的信息摘要 • 逻辑：既看图也看代码，但只提取与当前任务相关的“上下文片段”传给决策层
决策模块	编程逻辑生成 • 输出物：Python或JS脚本 • 思考方式：逻辑推演。将任务转化为if-else或循环语句 • 特点：一次生成，批量执行	直觉/符号映射 • 输出物：坐标或标签ID • 思考方式：空间几何推理。意图直接映射到屏幕位置 • 特点：单步推理，看一步做一步	分层规划与协商 • 输出物：SOP • 思考方式：Planner定战略，Actor定战术 • 特点：存在内部对话，决策是“商量”出来的
执行模块	代码解释器 • 运行环境：沙箱 • 动作：API调用、Selenium/Playwright驱动 • 优势：速度极快，可后台运行	鼠标/键盘模拟器 • 运行环境：宿主操作系统 • 动作：模拟点击、拖拽 • 优势：通用性强，所见即所得	工具调用 • 运行环境：根据角色而定 • 动作：Actor Agent调用预封装好的Tools • 优势：动作被标准化封装，不易出错
反馈模块	程序级反馈 • 信号：stderr、return code、程序崩溃 • 纠错：重新Debug代码逻辑	视觉级反馈 • 信号：截图对比、视觉变化检测 • 纠错：重新识别坐标，或微调点击位置	角色级反馈 • 信号：Critic的评审意见 • 纠错：驳回任务，要求Actor重新规划或执行

图：三条主要技术路径的核心不同点

感知“颗粒度”不同

- 代码生成路径看到的是“代码”。通过阅读网页源码来理解世界，后果是精度极高，但遇到无源码界面就无法操作。
- 视觉路径看到的是“像素”。它像一个普通用户，只看屏幕表面，后果是泛化性极强，但容易产生幻觉。
- 多智能体路径看到的是“语境”。它像一个管理团队，只关心关键信息。

决策的“时空观”不同

- 代码生成路径是“批处理思维”。倾向于一次性把多步的代码全写完。优点是效率高，缺点是中间不允许出错。
- 视觉路径是“流式思维”，看一帧，点一下，再看一帧，再点一下。优点是灵活，缺点是慢，容易迷失长远目标。
- 多智能体路径是“层级思维”。Planner看长远，Actor看当下，Critic看过去。优点是稳健，缺点是成本高。

反馈“来源”不同

- 代码生成路径信任“编译器”，坚持不报错就是成功。
- 视觉路径信任“眼睛”，只要画面变化就认为是成功。
- 多智能体路径信任“Agent队友”，质检员说过了才是成功。

2.3.2 技术路径决策框架

在技术路径选择决策方面，GUI Agent需要考虑的主要维度包括精确度、泛化能力、开发成本、运行成本、响应速度、安全性、维护成本和可解释性等多个维度。

表：三条技术路径在不同纬度的表现

项目	代码生成	端到端视觉	多智能体协作
精准度	高 • 依赖底层API或精确的DOM选择器，操作是确定性的，不存在点击偏差	一般 • 存在像素级预测误差，虽然引入SoM可缓解，但本质上仍是概率性预测	较高 • 通过引入Critic角色进行自我质检和反思，能有效拦截并修正单体模型的执行错误
泛化能力	较弱 • 对DOM结构极度敏感。一旦网站改版或遇到Canvas或游戏等无DOM界面，脚本即失效	强 • 基于RGB像素理解世界。像人眼一样，无论底层是HTML还是Flash，只要看得到就能操作	较强 • 具备策略调整能力。遇到未知情况时，Planner智能体能尝试换一种思路或工具来解决问题
开发成本	中 • 需要构建沙箱环境，且需为不同应用适配特定的解析器或API定义	低 • 主要工作是提示词工程，无需复杂的环境适配	高 • 架构复杂，需要编排多个SOP，并调试角色间的配合逻辑
运行成本	低 • 一次生成，无限复用。生成的代码可在本地CPU运行，无需消耗昂贵的GPU推理算力	一般 • 每一帧操作都需要上传高分辨率截图给大模型，算力消耗量大且带宽要求高	高 • 成本最高。智能体之间的每一轮对话都在消耗算力，通常是单体成本的3到10倍
响应速度	快 • 代码运行是毫秒级的。一旦逻辑生成完毕，执行过程几乎无延迟	慢 • 受限于视觉大模型的推理速度及图片上传时间，响应速度在秒级	极慢 • 存在“内部开会”时间。多个模型串行思考和交互，导致最终响应用户缓慢，响应速度通常为数十秒
安全性	较高 • 生成的代码逻辑可见，且无需将屏幕数据传出内网，适合高密场景	低 • 需要把数据上传云端且运行过程中缺乏中间环节进行有效阻拦	一般 • 数据需要上传云端 • 本质是逻辑驱动，而且有Critic对中间过程进行二次确认
维护难度	较高 • UI微调可能导致选择器失效，需要频繁重新生成代码或人工修补	较低 • 鲁棒性强。只要按钮的视觉特征没有发生剧烈变化，就不影响具体操作	低 • 可以对GUI Agent进行模块化拆解并且修复，无需进行整体推倒重来
可解释性	强 • 白盒操作，能够通过代码找到错误源头	较弱 • 黑盒操作，难理解具体的操作流程和操作逻辑	一般 • 过程可见，能通过不同智能体的沟通过程复盘决策流程

图：GUI Agent路径选择决策筛选框架

图：GUI Agent适用场景

	标签	场景	
代码生成路径	• 专用的 • 高频的 • 后台的	• 企业内部RPA升级：财务自动对账、税务申报、HR薪资录入 • 自动化运维：服务器巡检、批量日志抓取、自动化测试脚本执行 • 高频数据采集：每日从特定网站抓取百万级数据。	
端到端视觉路径	• 通用的 • C端的 • 探索的	• 个人AI手机/电脑助手：帮用户点外卖、修图、在设置里开关深色模式 • 跨应用信息搜集：如“帮我看看这三家电商网站哪个价格最低” • 非标应用操作：操作远程桌面、Flash网页、游戏界面	
多智能体协作路径	• 复杂的 • 决策型的 • 高价值的	• 复杂供应链/业务流程：跨多个系统，流程长，需要Planner统筹的业务 • 法律/审计辅助：自动审查合同并高亮风险，需要Critic角色进行多轮质检和自我反思 • 智能客服/专家系统：处理极其复杂的用户客诉，需要调用查询、赔付、安抚等多个工具，需要灵活的策略调整能力	

第三章

产品形态和场景落地

3.1 手机：触屏之后的又一次交互革命

2007年，iPhone诞生之后，手机第一次成了人类生活中最重要的智能硬件。过去十几年，以触摸为核心的GUI交互始终是主导范式。尽管语音助手，如Siri、Google Assistant，作为重要的补充模态出现，但它们在绝大多数场景下仍被设计为GUI的附属和补充，通常用于发起一个简单任务，最终结果仍需在GUI上确认或查看，未能成为独立的、完成复杂任务的核心交互方式。所以，问题不在于技术本身是“变体”，而在于其“附属地位”未能撼动GUI的主导格局。GUI Agent的出现，有望打通人与手机智能交互的“最后一公里”，从根本上改变人与手机的交互方式，将手机从“图形交互时代”带入“多模态交互时代”。

图：不同交互模式下购物对比

图：GUI Agent对手机造成的影响

1 打破硬件创新瓶颈

- GUI Agent将手机从“图形交互”代入“多模态交互”的新时代，操作性和引用性大幅提升，让“AI手机”时代真正来临，同时将带动硬件、算力需求大幅提升

2 推动技术平权，降低操作门槛

- 手机从以图形交互为主转为“多模态交互”，操作门槛进一步降低，对老人、视障人士和其他残障人士较为友好，进一步推动实现了技术平权

打破数据孤岛的“物理外挂” 3

- 现有的手机生态依然是以APP等手机应用建立的“数据孤岛”，GUI Agent有网作为“粘合剂”，逐步打破现有的孤立状态，营造新的手机生态和操作可能性

APP隐身，“以人为中心”的交互时代 4

- 通过GUI Agent，用户不需要跟图形界面和APP进行交互，APP有望走入“隐身”和“后台”，手机将真正成为一个整体，同时将改变现有APP的商业模式，如APP广告植入模式改变

3.2 电脑：从通用终端走向“个人智能工作站”时代

GUI Agent带来的交互方式的改变，在软件方面，GUI Agent打破了软件之间的壁垒，改变了用户对计算机的输入和操作方式，提升了软件之间的协作能力，将计算机从由单个应用组成的硬件转变为数个应用协同工作且能够无缝衔接的“超级工作站”。在硬件方面，计算机配置从“满足应用”向“满足应用之间的协同”转变，多模态输入设备、算力硬件将成为标配，同时计算机将逐步改变传统形态，走向多元化和异形化。

1 交互范式：从键盘操作到多模态操作

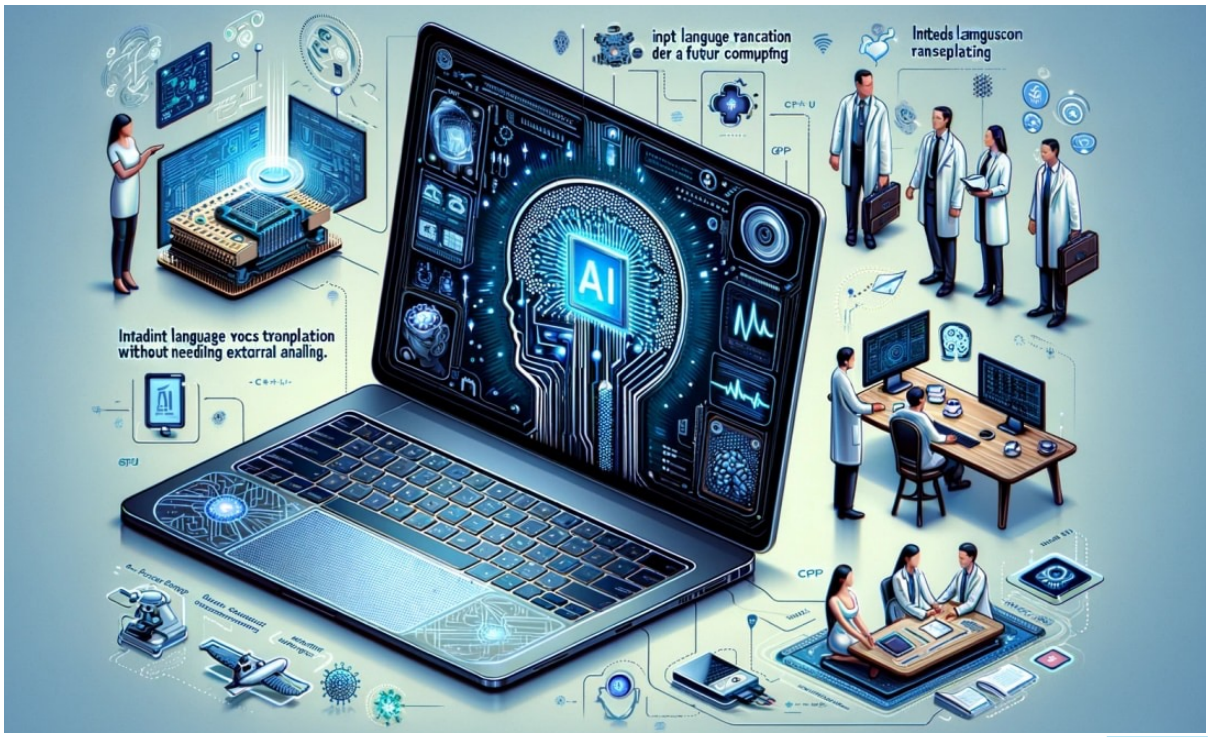
- 交互从单一模态转为多模态：除键盘和鼠标之外，语音、手势、视觉理解将成为重要的输入设备，多模态交互时代到来。
- 无视软件底层接口的万能驱动：无论是前沿的云端SaaS，还是缺乏API的本地专业软件（如CAD、老旧本地财务软件、甚至单机游戏），自然语言都可以通过纯视觉与GUI物理模拟，实现跨软件的无缝接管。

2 生产效率迎来质变：提升百倍的生产率机器

- 办公自动化水平提升：彻底打通PC端处于数据孤岛的各个客户端。GUI Agent 能够像人类一样盯着微信客户端里的非结构化聊天记录，自动提取信息并填入本地Excel宏表格中，一台电脑的多窗口并行协作不再需要人类双眼作为中介，一台电脑同时操作多个窗口和多个任务将成为常态。

3 AI PC硬件标准迎来大幅度提升

- 端侧视觉计算的崛起：NPU的标配不仅是为了跑大语言模型，更是为了支撑毫秒级的本地计算机视觉与高精度OCR渲染。PC屏幕不再仅仅是给人看的，而是实时生成数字语义快照供GUI引擎进行物理寻址的底座。
- 形态的变革：显示器从追求更高清晰度转变为更精准的UI元素识别；无屏、多屏PC将改变现有PC的传统形态；眼球追踪、语音输入等全模态感知系统出现。



数据来源：IDC、IC修真院；

3.3 智能穿戴：进化成为独立的智能终端

智能眼镜、智能手表、智能手环等智能穿戴当前面对的重要问题就是触屏时代的交互问题，即智能穿戴有了部分手机的功能，却由于没有屏幕或者屏幕过小，导致实现功能单一，当前智能作为手机的“辅助”产品，GUI Agent可能对当前智能穿戴产生救赎性影响，由GUI Agent产生的交互方式变化，将极大提升智能穿戴的交互水平，智能穿戴有望成为独立的智能终端。

1. **智能穿戴具备真正的执行能力和独立性：**当前智能穿戴多作为数据和图像的感应和采集角色出现，GUI Agent将改变以屏幕为主的交互方式，智能穿戴将能够被语音等方式输入执行，通过云端或者同步手机等其他方式，完成全部工作流程。
2. **流量入口从手机扩散到其他智能设备：**当前，除手机之外的智能产品都受困于没有屏幕带来的交互难题，GUI Agent让智能眼镜、智能穿戴和智能座舱突破了交互难题，未来都有可能成为与手机相当体量的流量入口，从当前的“辅助产品”变为“主导产品”。
3. **智能穿戴App生态改变：**当前智能穿戴使用的App需要单独订制开发，但功能也未能较为单一，造成智能穿戴App生态的贫瘠和匮乏，GUI Agent有望实现对App的“原生复用”，极大改变了当前可用App贫瘠的情况。
4. **传感器的地位获得进一步提升：**当前包括摄像头在内的智能传感设备主要是为了数据采集和拍照，之后传感器将成为“服务GUI Agent”的传感设备，传感器的定位和功能发生了显著改变，同时数量将大幅度提升。

智能眼镜

实现“视觉即交互”

- GUI Agent解决了智能眼镜的输入困难，同时能让智能眼镜直接接入手机APP生态，实现注视即操作，加上智能眼镜的感知优势和轻量化优势，使智能眼镜有望取代手机，成为最重要的智能设备。

智能手表

从附属到独立工作

- GUI Agent将解决智能手表一直以来面临的“屏幕尺寸与交互复杂度的矛盾”。
- 从输入方式上，一方面，智能手表将去触控化，语音将成为主要的输入方式，另一方面，屏幕形态使命发生变化，手表也将“异形化”。

智能手环

交互能力“越级”

- 智能手环一直受限于“无屏幕”或者屏幕太小，GUI Agent让智能手环首次具备了交互能力。智能手环不仅能够独立操作，同时能够通过上云和跨OS操作，实现对智能家居和其他硬件设备的连接和控制。

智能耳机

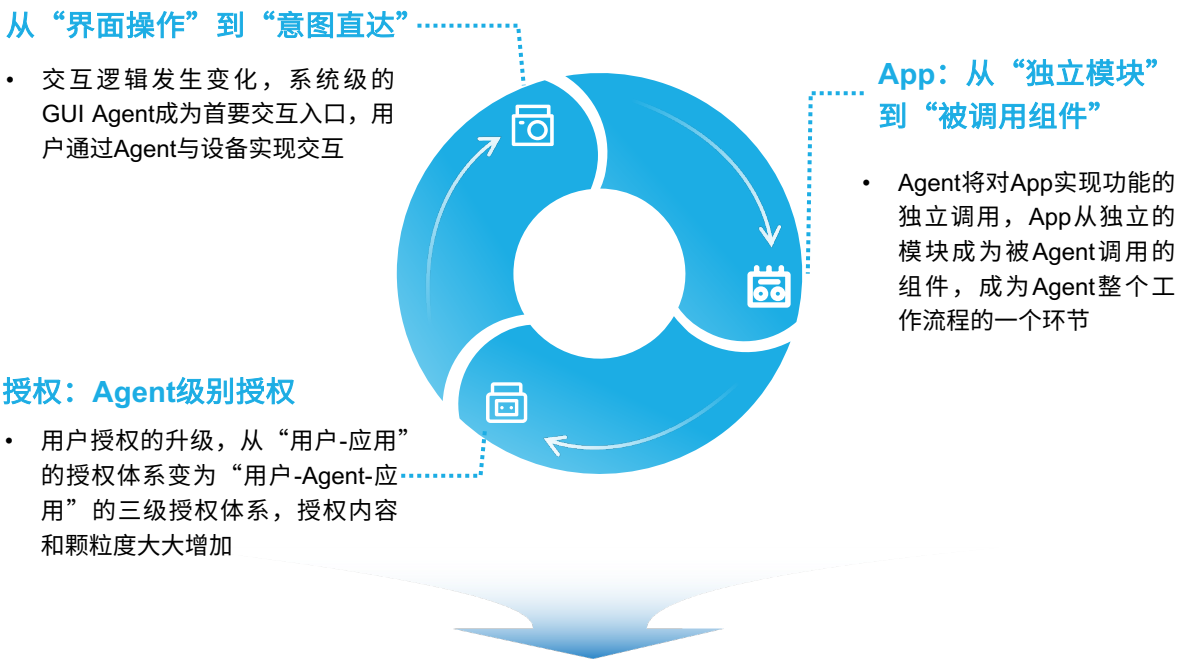
到智能终端的物种级跃迁

- 当前智能耳机只具备收听和“切歌”等基本功能，GUI Agent能够让用户通过智能耳机输入指令，实现完全脱离屏幕的“App”级操作。

3.4 OS生态：去App化来临，OS成为最大的流量入口

当前，OS的作用主要是作为平台出现，为App创造运行环境，而大的流量入口被主要的App分割。GUI Agent出现以后，有望取代OS传统的交互层，同时按照任务调用App，可能会取代App，成为新的“超级入口”，同时影响现有的App盈利生态。

图：GUI Agent带来的交互方式变化

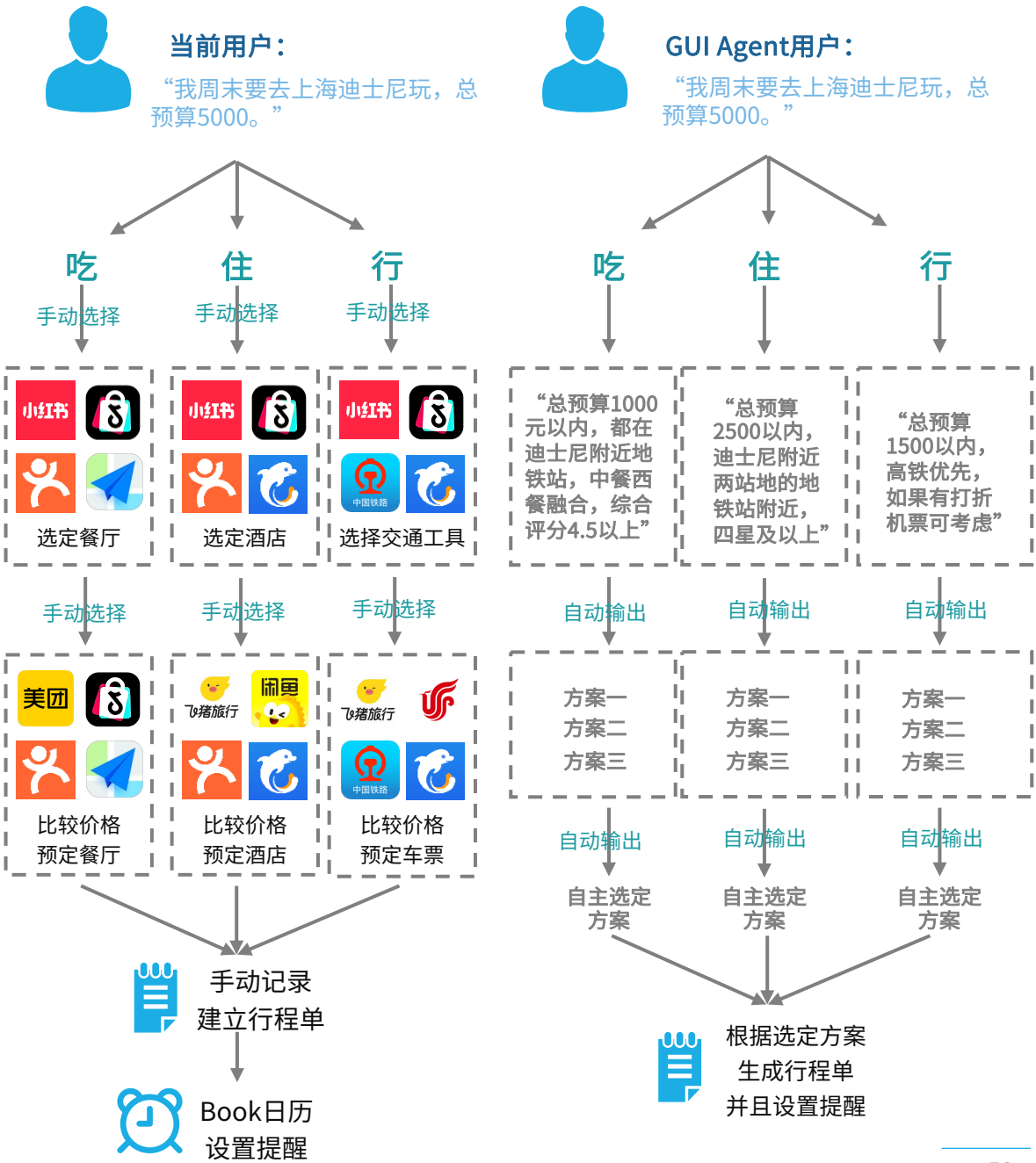


项目	当前	未来
核心定位	• App启动器，负责管理硬件和启动软件	• 意图代理，负责理解用户并分发任务
交互单元	• 被分割的独立应用	• 技能，App被拆解为可调用的原子能力
流量入口	• 超级App垄断：微信、抖音、Chrome等	• 系统级Agent或者computer/mobile use
文件管理	• 树状目录，靠文件名和路径索引	• 向量数据库，靠语义和视觉记忆索引
开发思路	• 碎片化适配	• Agent标准，围绕Agent建立开发标准，并且会推出“Agent to Agent”互联标准
盈利模式	• 平台化盈利模式，多为应用商店抽成、广告抽成、订阅服务	• 以服务为主的盈利模式，多为服务佣金、广告植入佣金
生态博弈	• 不同App对流量入口的争夺	• OS厂商、超级App、第三方GUI Agent对流量入口的争夺

3.5 行程规划：多行程合并，“跨平台噩梦”的终结

差旅或者旅行行程规划是典型的复杂路线规划，而是很多人的“噩梦”，因为要规划吃、住、行的行程，一方面每一环节都不可或缺，另一方面每个环节都需要打开不同的APP，旅行规划通常要打开携程（订票）、大众点评（查餐厅）、手机日历（看时间）、高德地图（看路线）、笔记软件（记录笔记）等多个APP才能完成整个行程规划，这中间繁琐的信息比对和搬运工作经常让人崩溃，多数时候是“满怀信心开始，崩溃草草收尾”。GUI Agent出现能极大简化复杂的行程规划，通过一段详细描述的语音或者文字完成整个工作，在此过程中用户的工作从“操作”变为“检查和修正”，极大降低工作难度，提升效率。

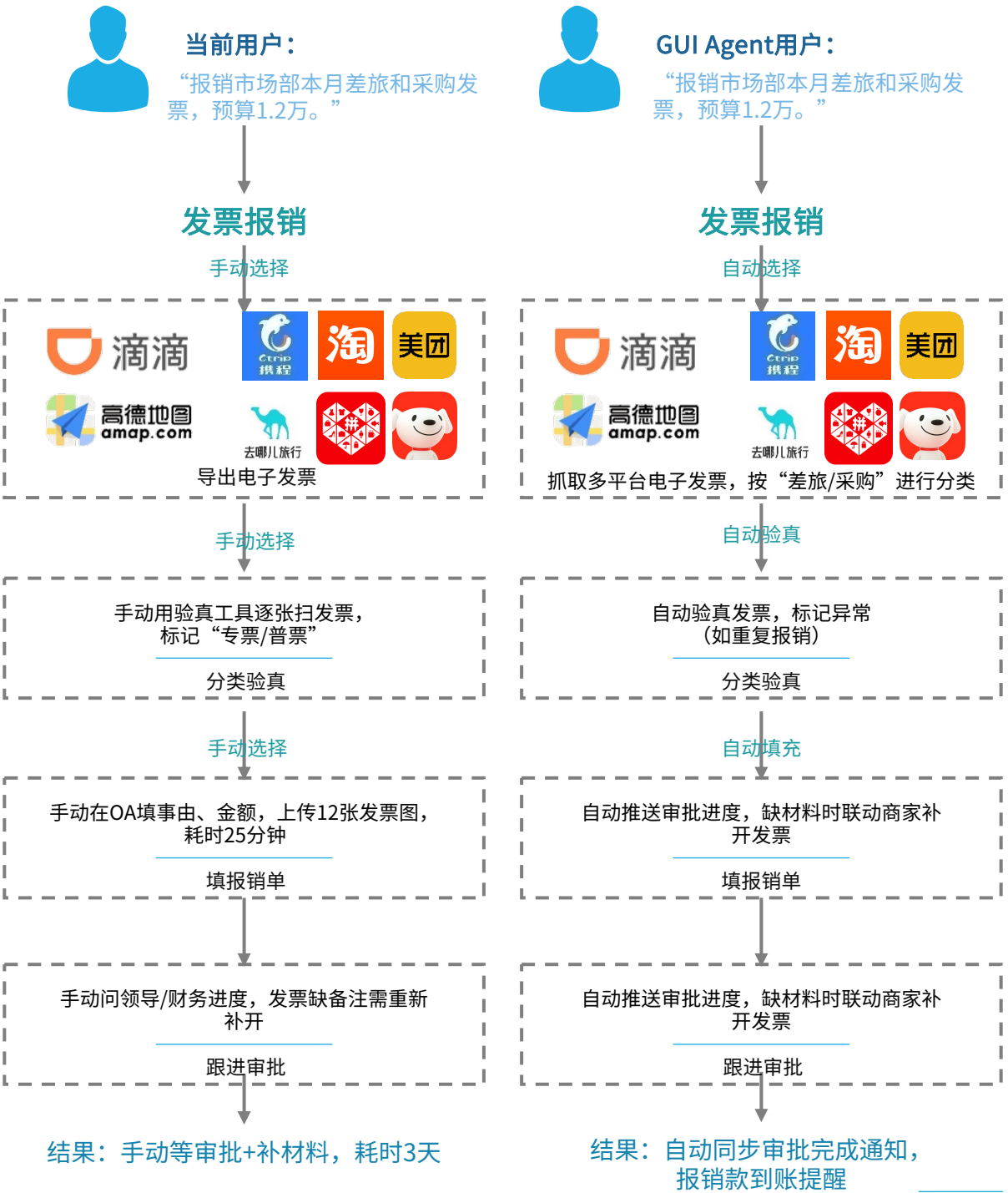
图：GUI Agent带来的流程变化



3.6 发票报销：减少重复劳动，实现全流程自动化

企业发票报销是职场“折腾痛点”，首先得打开滴滴、京东、高德、美团、携程、去哪儿网讲发票开出，之后通过企业OA、微信、钉钉、飞书、企业微信来跟进审批进度，手动整理和反复补材料常耽误3-5天，最后“发票堆了半周才报完”。GUI Agent能自动抓取多平台发票，完成验真、填单、跟进，用户仅需确认信息。

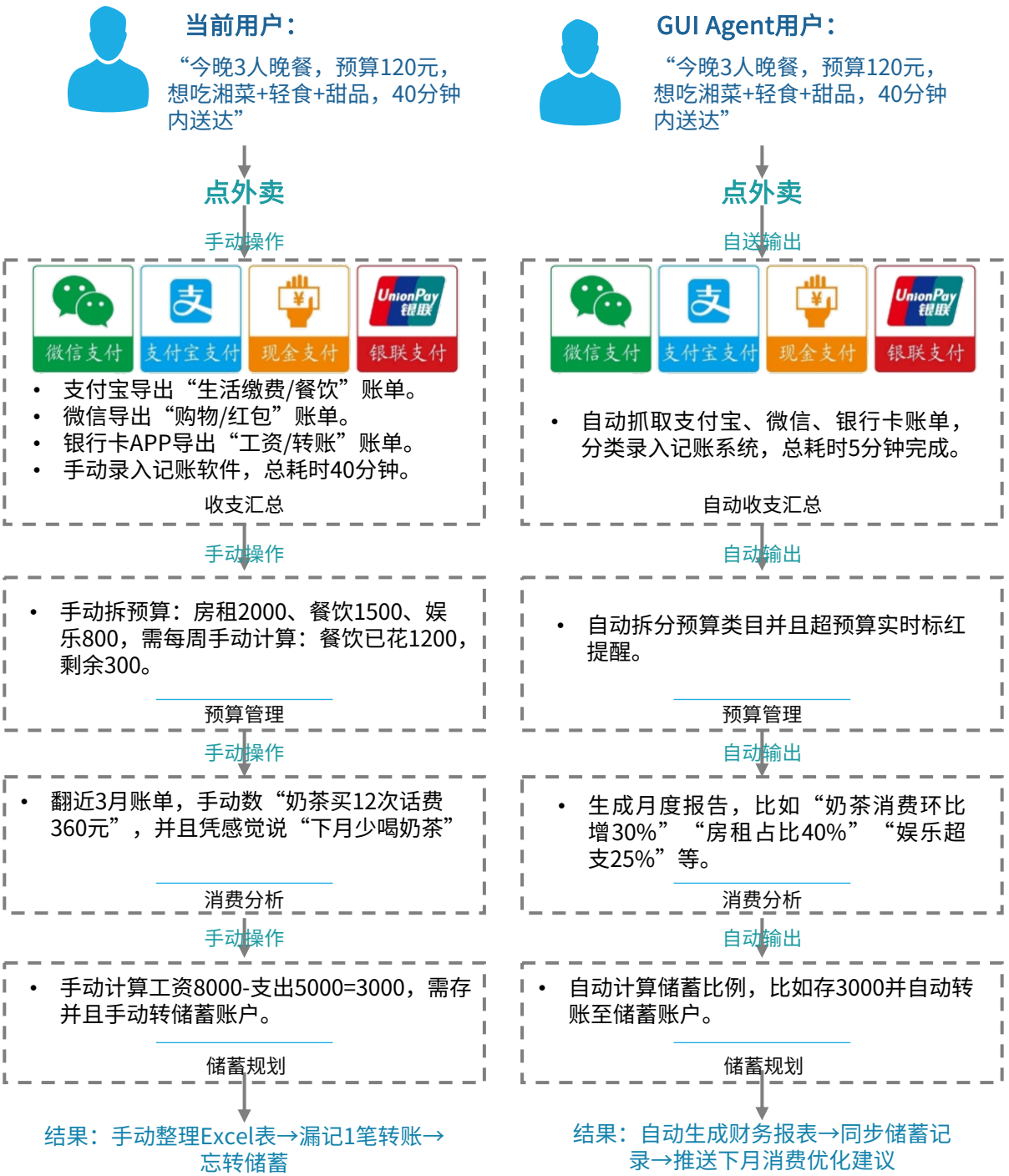
图：GUI Agent带来的流程变化



3.7 家庭财务管理：实现跨平台、自动化数据整合

个人和家庭财务管理是典型的“数据整合噩梦”，要统计收支、控制预算、分析消费、规划储蓄，得打开支付宝、微信、银行卡APP、记账软件、Excel等多个工具，手动导出、录入、计算常出现漏记和算错，最后“账单堆了半月才整理完，还不知道钱花哪了”。GUI Agent能自动跨平台整合财务数据，用户只需说明预算需求，全流程自动完成，用户工作从“手动折腾”变为“检查确认”。

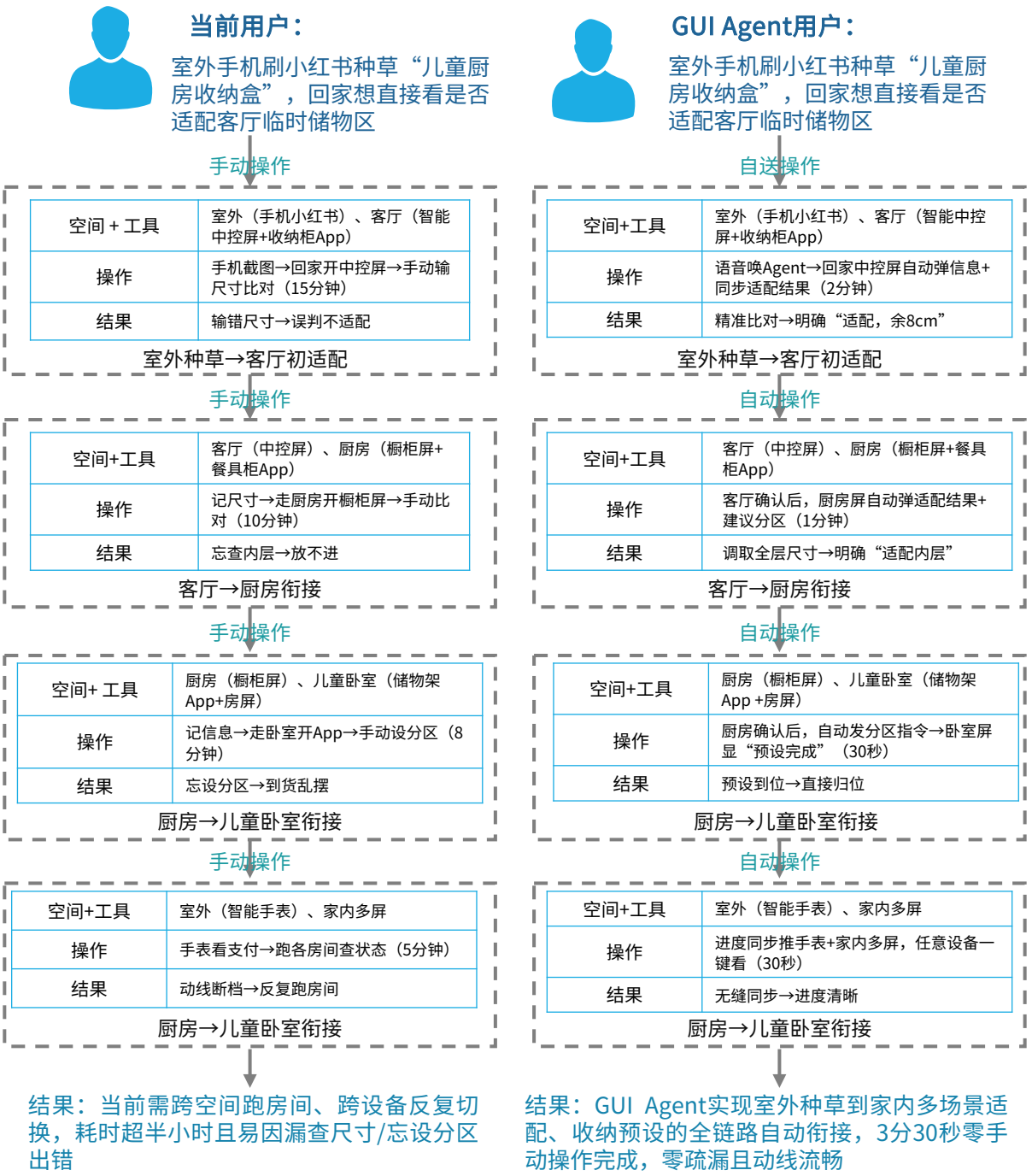
图：GUI Agent带来的流程变化



3.8 购物流程：从种草到购买的全自动衔接

种草-家居适配常遇“跨空间和跨设备割裂”，室外用手机种草家居品，回家后要在客厅、厨房、儿童卧室等智能家具场景里，手动切换手机、客厅智能中控屏、厨房橱柜屏、儿童卧室储物架App等工具，跨空间传信息、查尺寸，动线衔接全断档，最后“跑遍每个房间，还漏了适配细节”。GUI Agent 可打通室外和家内多智能家具场景的跨空间和跨设备链路，用户只需说明需求，全自动衔接。

图：GUI Agent带来的流程变化



第四章
未来与展望

4.1 Agentic OS时代来临

在机器和计算机诞生的数百年以来，一直是“人”要去适应“机器”的规则和逻辑，而Agentic OS有望首次让机器来使用人的操作规则，机器和硬件将变成一个“全能助理”，完全按照人的操作意图进行工作，这不仅将极大拉低人机交互门槛，而且完全颠覆了数百年以来的人机交互逻辑。

图：GUI Agent的发展阶段

阶段一：辅助发展期

- 应用为中心，GUI Agent是重要的辅助工具。
- OS整体架构不变，GUI Agent以辅助工具的形式出现，垂直领域的Agentic OS出现。
- 具备初步的跨应用处理能力，但处理步骤较少、等待时间长、跨应用需要手动处理。

阶段二：融合发展期

- OS-Level Agent出现，与GUI Agent融合并行，并且逐步向Agentic OS迈进。
- OS级别的智能体出现，GUI Agent有调度系统原生的能力。
- 多Agent框架成熟，并且植入Agentic OS应用。

阶段三：原生智能期

- Agentic OS成为人机交互的默认形态。
- Agentic OS重构OS内核，设备成为用户的“超级分身”和“超级助理”，能够帮助用户主动完成任务。

图：Agentic OS的形态

1 人通过意图直接操作设备

- 人机交互彻底告别鼠标、键盘、触屏，转向自然语言和意图驱动，用户只需描述目标，Agent自主规划、执行、反馈。
- 系统内置反思与记忆机制，能持续学习用户偏好，主动预判需求、提前准备方案，实现“主动服务”而非“被动响应”。

2 从“App为中心”到“Agent为中心”

- 底层：整个操作系统抽象为可被 Agent直接调用的原子能力池，不再依赖GUI或API封装。
- 调度层：原生支持多Agent协同调度，自动完成任务拆解、资源分配、冲突协调与故障自愈，实现“千人千面”的个性化算力供给。

3 从“单体执行”到“群体智能”

- 支持跨设备、跨系统的Agent组网，形成智能体网络，手机、PC、IoT等设备上的Agent可无缝协作，完成超复杂长链路任务。
- Agent可自主优化策略、学习新技能、甚至生成新的子Agent，系统整体智能随使用不断提升。

4 设备成为人类的“超级助理”

- 设备的角色从工具成为“超级助理”，设备不仅能够完成工作，还是个人的超级知识库、全天候工作伙伴、信息搜集和翻译资料库，全天候、长线程、不间断、前置性的帮助个人处理复杂的工作和任务，成为按照个人需求构建的超级助理。

4.2 阻碍与挑战

GUI Agent当前的发展尚处于初级阶段，在从“研发”到“能用”到“通用”到“大规模商用”过程中，尚存在技术、成本、安全三个最明显的阻碍。

技术瓶颈：技术的鲁棒性和延迟

在技术层面，GUI Agent表现出的最大问题是“短、慢、脆、准确率差”，造成这些问题的原因主要是多模态对齐精度、长程推理、泛化性不足和训练数据等多种原因。

- 长程推理与状态管理：跨应用、多步骤任务中，上下文记忆易丢失，任务逻辑易断裂，当前GUI Agent最多执行任务的步骤在20步以下，20步以上的复杂任务崩溃率几乎100%。
- 端到端延迟：由于GUI Agent工作流程复杂，视觉延迟、模型推理延迟、网络传输延迟一系列远处，造成每次执行任务推理过程至少需要2至4秒，这一速度无法适应高效率的工作，当前部分工作甚至低于人类工作速度。
- 对动态环境的脆弱性：当前电脑、手机UI环境复杂，广告弹窗、快速跳转、软件更新造成Agent对环境适应的脆弱性，经常出现“卡死”或者报错。
- 准确率的流失漏斗：由于坐标预测幻觉、逻辑链断裂和动态噪声一系列问题，造成GUI Agent在处理长程任务时成功率极低。

成本瓶颈：高昂的推理成本和基础设施开销

在成本层面，一方面，在推理层面，由于视觉大模型的使用，GUI Agent有惊人的Token消耗量，另一方面，在基础设施方面，GUI Agent对算力和部署设备有极高的要求，这两项开支造成了GUI Agent在商业化上存在困境。

- 推理成本：一是由于GUI界面复杂、细节多、理解复杂，造成了视觉信息的“高维膨胀”，另一方面超长上下文和图漂流造成的Token的几何级消耗，还有推理过程中的Token消耗，这都让GUI Agent使用过程中成本极高。比如，一个简单的“订票”流程就有可能产程数十元的Token费用。
- 基础设施成本：运行一个API Agent只需要几MB内存，但运行一个GUI Agent需要4GB+内存和数个CPU核心。一方面原因是GUI Agent需要一个高精、完整的图形操作系统，同时必须有显卡、窗口管理器和大量应用介入；另一方面视频和图像对设备CPU/GPU要求极高，同时对网络速度要求极高，当前多数设备和网络都无法满足GUI Agent的流畅运行需求。

安全和隐私：风险防控和商业挑战

- 安全风险防控：安全方面主要包括误操作风险造成的数据和财产流失、权限滥用造成的操作风险和隐私泄露风险。
- 合规性风险：不同国家对数据跨境、隐私保护法规差异巨大，同时缺乏统一标准，造成GUI Agent全球化落地困难。
- 伦理阻碍：主要包括GUI Agent造成的安全和风险责任归属缺乏明确规范。

庭宇科技

铸基计划
高质量数字化转型推进行动

甲子光年
JAZZYEAR

北京庭宇科技有限公司是全球领先的边缘智算基础设施服务商。庭宇成立于2019年，始终致力于构建以边缘智算为核心的AI全栈基础设施生态，为智能时代提供核心算力支撑。庭宇运营着国内最大的分布式GPU算力池，构建了覆盖全国超1500个县市及东南亚的高性能边缘节点网络，基础设施已深度赋能云桌面、AI Agent等多元场景，技术实力与市场占有率持续领先。当前，庭宇正通过Lybic灵臂等前沿产品，积极推动AI智能体的普及与应用，构建连接物理与数字世界的智能桥梁。

“铸基计划”是中国信通院积极响应高质量发展战略而发起的专项行动，专注于企业数字化转型中面临的痛点、难点问题，发挥中国信通院的生态位优势，通过链接数字化转型供给侧和需求侧，助推数字化转型高质量发展。该计划通过构建测评体系、制定评估标准、搭建供需对接平台等方式开展工作，工作覆盖标准制定、产品技术评测、解决方案创新等领域。

TBD

扫码关注
庭宇科技

扫码关注
甲子光年

甲子光年
JAZZYEAR

法律声明

版权声明

本报告由甲子光年智库制作完成，报告内容的版权及相关知识产权均归北京甲子光年科技服务有限公司所有。任何单位或个人在引用本报告内容时，须保持内容的原始性，不得进行歪曲、删改或误导性引用，并须注明报告出处为“甲子光年智库”。否则，由此引发的一切后果由引用方自行承担，甲子光年保留追责权利。

免责条款

本报告中的行业数据、市场预测和相关分析主要来源于甲子光年研究团队通过桌面研究、专家访谈、问卷调查、公开数据整理及甲子光年产品数据等方式获得，部分数据通过甲子光年自主统计预测模型进行估算。我们已尽合理努力确保数据的准确性、完整性与可靠性，但甲子光年不对其作出任何明示或暗示的保证。在任何情况下，本报告中包含的内容、数据或观点均不构成对任何单位或个人的投资建议、法律建议或其他形式的专业意见，相关决策应由读者自行判断并承担风险。 由于调研方法和样本范围的限制，报告中发布的数据结果仅反映特定时间段、特定对象的调研情况，具有一定的局限性，仅供读者作参考用途。甲子光年对因使用本报告内容而产生的任何直接或间接损失不承担任何法律责任。

甲子光年
JAZZYEAR

THANKS

谢谢

北京甲子光年科技服务有限公司是一家科技智库，包含智库、媒体、社群、企业服务版块，立足于中国科技创新前沿阵地，动态跟踪头部科技企业发展和传统产业技术升级案例，致力于推动人工智能、大数据、物联网、云计算、AR/VR交互技术、信息安全、金融科技、大健康等科技创新在产业之中的应用与落地

扫码关注甲子光年公众号

分析师

何伟康：微信（18910027586）

智库院长

宋涛：微信（stgg_6406）

商务合作

郑爽： 18600502376（微信同号）