arXiv:2603.08425v1 [cs.AI] 9 Mar 2026

IronEngine：迈向通用人工智能助手

面向通用人工智能助手的系统设计、多模型编排与工程实践

系统设计概览

作者：Xi Mo 所属机构：
NiusRobotLab 频道：YouTube 频道
@NiusRobotLab日期：2026年3月

本技术报告及相关源代码，包括实验和结果分析，均由人工智能智能体根据作者提示全自动设计完成。

目录

1 引言 **1**

2 相关工作 **3**

2.1 推理与行动耦合 .32.2 工具使用与 API 集成 .32.3 记忆与知识管理 .42.4 多智能体协作 .42.5 网页和图形用户界面智能体 .42.6 产品化人工智能助手与代码智能体 .52.7 OpenClaw 生态系统：OpenClaw、NanoClaw 和 IronClaw .5

3 设计目标 6

4 总体架构 7

5 实现证据与工程特性 7

6 模块分解 7

7 三阶段流水线 8

7.1 阶段 1：讨论 .97.2 阶段 2：模型切换 .97.3 阶段 3：执行 .10

8 工具系统　　架构1 **1**

8.1 工具类别 .118.2 智能路由 .118.3 网络搜索架构.118.4 网页内容提取 .118.5 多媒体分析 .128.6 GUI 自动化 .128.7 执行安全 .12

9 记忆与技能系统 **12**

9.1 分层记忆架构 .129.2 向量化技能仓库 .13

10 自适应模型管理 13

10.1 模型目录 .1310.2 VRAM感知上下文预算.1310.3 分层提示系统 .1410.4 思考深度控制 .1410.5 动态能力检测 .14

11 MCP 兼容性与开放生态战略 **15**

12 实验评估 1512.1 实验设置 .1512.2 文件操作基准测试 .1512.3 流水线阶段时间分布 .15

12.4 工具路由准确率.1512.5 多模型协作分析 .1612.6 显存和资源使用 .1612.7 搜索去重与缓存 .1712.8 错误模式与恢复 .1712.9 与代表性系统的比较 .1712.10 跨模型工具感知 .1812.11 全流程集成测试 .1812.12 工具模型可靠性 .1912.13 多场景诊断测试 .19

13 接口、部署与产品定位 20

13.1 部署模式 .2013.2 产品定位 .2013.3 桌面工作台设计理念 .20

14 安全与隐私 21

14.1 权限管理 .2114.2 执行沙箱 .2114.3 URL 安全 .2114.4 干预机制 .2214.5 SOUL 编辑控制.2214.6 本地优先隐私 .2214.7 内部英语处理 .2214.8 纵深防御设计 .22

15 讨论 22

15.1 架构权衡 .2215.2 可扩展性考量 .2215.3 与 OpenClaw 生态系统的架构比较 .2315.4 设计原则与工程洞见 .2315.5 泛化能力.2315.6 局限性 .23

16 未来工作 24

17 结论 24

图目录

1 与 IronEngine 相关的 AI 智能体研究和产品演进简化轨迹。蓝色节点表示研究里程碑；橙色节点表示面向产品的系统。 . . . . . 72 IronEngine 总体架构，展示四层设计：交互界面（顶部）、统一编排器（中部）、能力模块（中下部）和环境接口（底部）。 83 IronEngine 桌面版用户界面。左侧面板提供模型和执行配置，以及规划器/审查器/工具模型选择器。右侧区域支持对话，并显示内联思考块、工具执行徽章和质量评分指标。.94 IronEngine 三阶段流水线。阶段 1（讨论）：规划器和审查器协作提升计划质量，不执行工具。阶段 2（模型切换）：从讨论模型切换到执行模型，并感知显存状态。阶段 3（执行）：执行器反复运行工具，直至任务完成。 .106 网络搜索的四策略回退链。按顺序尝试各策略；首次成功后即终止回退链。 .115 IronEngine 中的工具分发流程。解析模型输出，通过 130+别名规范化类型，若不匹配则自动纠正，并通过多层回退链进行调度。 . . . . . 127 IronEngine 中的记忆层级与生命周期。会话和流水线条目通过两种合并策略整合为每日摘要和精炼后的长期知识。上下文检索会为每个新请求选出最相关的条目（2 个正面 + 1 和负面示例）。 148 各基准任务在不同流水线阶段的执行时间分布。模型推理（规划器 + 审查器 + 执行器）占总时间的大部分，而实际工具执行耗时不到 2 秒。模型切换阶段的耗时固定为 ∼27 秒。 .169 规划器—审查器质量分数变化。红色柱表示第1轮的初始分数；绿色柱表示审阅者反馈后的最终分数。审查器提供的结构化反馈（问题 + 建议）使 14B规划器能够将计划质量从 0.10–0.20 提升至 0.80–0.85，而较大型的 27B规划器通常在第一轮就能达到可接受的质量。 .19

表格目录

1 项目中主要系统主张与工程证据之间的映射.8 2 按功能域组织的 IronEngine 工具类别.12 3 基于模型规模的分层提示系统.14 5 工具路由自动纠正统计.15 4 文件操作基准测试结果。所有任务均达到 100% 正确性（4/4 通过）。. . . . . 16 6 按流水线阶段统计的显存使用情况.16 7 与代表性人工智能助手系统的详细能力对比。评级：S=强，M=中型，W=弱，–=不适用.18 8 跨模型工具感知测试结果。所有模型借助 IronEngine 的分层提示系统均实现了完美的工具类型识别.18 9 跨模型配置与场景的完整流水线集成测试结果。质量评分展示了第1轮（初始） $\rightarrow$ 第2轮（审阅者反馈之后）的进展.19 10 工具模型可靠性对比。翻译成功 $=$ 有效JSON输出；执行成功=正确工具结果.19 11 多场景诊断测试结果。六个场景中有五个通过（83.3%）。旅行规划失败是由于测试环境缺乏网络访问。对于评分为 ≥ 7 的任务会自动触发技能学习.20 12 与代表性人工智能智能体框架及产品的系统定位比较 21

IronEngine：迈向通用人工智能助手

面向通用 AI 助手的系统设计、多模型编排与工程实践

Xi Mo NiusRobotLab 2026年3月

摘要： 随着大语言模型从单轮对话系统演进为能够调用工具、使用记忆并与环境交互的长期运行智能体，AI 助手的实际价值越来越取决于系统架构，而非仅仅取决于模型能力。本文介绍了 IronEngine，这是一款通用人工智能助手平台，其架构围绕统一的编排核心构建，连接桌面用户界面、REST 和 WebSocket API、Python 客户端、本地和云端模型后端、持久化记忆、任务调度、可复用技能、24 类工具执行、MCP 兼容扩展性以及面向硬件的集成。IronEngine 引入了一个三阶段流水线——讨论（规划器—审查器协作）、模型切换（显存感知切换）和执行（工具增强型行动循环）——将规划质量与执行能力分离。该系统采用分层记忆架构，支持多级整合；基于 ChromaDB 构建向量化技能仓库；并配备自适应模型管理层，支持 92 种模型配置文件和显存感知上下文预算，以及具备 130+ 别名规范化和自动错误纠正功能的智能工具路由系统。我们展示了文件操作基准测试的实验结果：在四项异构任务中，任务完成率达到 100 %，平均总耗时为 1541 秒；并与具有代表性的 AI 助手系统进行了详细比较，包括 ChatGPT、Claude Desktop、Cursor、Windsurf 以及开源智能体框架。在不披露专有提示词或核心算法的前提下，本文分析了该平台的架构分解、子系统设计、实验性能、安全边界和比较工程优势。本研究由此将 IronEngine 定位为通用个人助理、自动化框架以及未来以人为中心的智能体平台的系统导向型基础。

关键词： 通用人工智能助手；人工智能智能体系统；多模型编排；本地大语言模型集成；持久记忆；技能学习；工具路由；任务调度；MCP 兼容性；显存管理；安全性

1 引言

AI 助手的发展格局经历了根本性转变。从 Transformer 架构开始，[1] 再到 GPT-3 展示少样本上下文学习，[2],大语言模型（LLM）已从文本补全引擎发展为具备推理、工具使用和多步骤任务执行能力的系统。由 LLaMA 等模型催化的开源运动 [3]，使高性能模型能够进行本地部署，而 GPT-4 [4] 和 Gemini [5] 则推动了多模态推理的前沿。这种能力与可访问性的融合创造了独特机遇：构建不只是围绕单一模型端点的封装器，而是能够将模型能力组织成稳定、可控且可复用行为的综合系统。

近期研究为人工智能智能体确立了若干基础范式。ReAct 展示了在单一模型循环中交替进行推理与行动的价值 [6]。思维链提示表明，明确的推理轨迹有助于改进任务分解 [7]。Toolformer 证明，模型能够学会自主调用外部工具 [8]。检索增强生成（RAG）确立了外部非参数化记忆的重要性 [9]。生成式智能体展示了记忆流、反思和规划如何在模拟环境中维持连贯的长期行为 [10]。Reflexion 引入了言语自我批评，作为迭代改进的一种机制 [11]。这些贡献共同构成了现代智能体系统的基础模块。

在框架层面，Auto-Gen [12], CAMEL [13], MetaGPT [14], ChatDev [15],OpenManus [16], 和 OpenClaw [17] 等系统加速了从孤立的模型调用向智能体生态系统的转变，这些生态系统具备基于角色的协作、工作流和持久化执行界面。与此同时，产品化人工智能助手——包括搭载 GPT-4o 的 ChatGPT [18], 搭载 MCP 的 Claude Desktop [19, 20], Cursor [21], Windsurf [22], 和 GitHub Copilot [23]——已将 AI 智能体能力带给数百万终端用户。

然而，经过仔细审视，当前系统仍普遍存在五个系统性问题，每个问题都代表着一个独特的工程挑战，而现有解决方案只能部分应对。

碎片化问题。 如今的 AI 助手彼此孤立：ChatGPT 是网络界面，Claude Code 是命令行工具，Cursor 是集成开发环境插件，Open Interpreter 则是 Python 库。需要自动化桌面应用程序、搜索网络、处理文件并通过即时通讯平台发送消息的用户，必须在多个互不相通的工具之间切换，而每个工具都有自己的模型后端、上下文窗口和交互范式。没有任何单一系统能提供一个统一的编排核心，将所有这些功能整合到同一个架构体系之下，迫使用户在便利性（精致但功能狭窄的产品）与通用性（灵活但彼此割裂的框架）之间做出选择。

单模型瓶颈。 当前大多数系统依赖单一模型，在一次推理过程中处理所有认知功能——规划、自我评估、工具调用和输出生成。这种设计造成了内在的矛盾：大型模型（30B+参数）擅长复杂的多步骤推理，但在处理简单的翻译或格式处理任务时会浪费计算资源；而小型模型（3–8B 参数）能够在消费级硬件上高效运行，却无法维持连贯的多轮规划。很少有系统提供异构模型分配架构，能够在单个任务中为不同的认知角色分配不同规模的模型，更不用说管理在单个 GPU 上切换模型所需的 GPU 内存生命周期了。

短暂助手问题。 当前的 AI 助手在不同会话之间大多是无状态的。每次对话都从空白状态开始，不具备对过往交互、习得技能或累积领域知识的记忆。尽管一些系统提供对话历史记录，但没有任何系统提供综合性解决方案，将分层记忆与生命周期策略（创建、整合、衰减）、从成功执行中进行质量筛选的技能学习、优先检索经过验证知识的评分驱动检索，以及取代过时信息的矛盾检测结合起来。缺乏结构化持久化，迫使用户在每次新会话中重新解释自己的偏好、重新教授工作流，并重新提供上下文。

本地部署挑战。 涉及隐私的工作负载——个人通信、专有文档、医疗记录、金融数据——要求完全本地推理，确保数据不会离开用户的设备。然而，在消费级硬件上进行本地部署会带来云端系统无需面对的挑战：模型异构性（不同架构需要采用不同的量化策略）、上下文窗口限制（显存限制了模型可处理的词元数量），以及智能资源调度（在单块 24GB GPU 上依次加载 14B规划器和 8B 审查器）。很少有系统能提供感知显存的模型生命周期管理，根据可用硬件资源自动选择、加载和卸载模型。

工具集成问题。 每种 AI 助手都将工具实现为定制插件，缺少统一调度层。系统不支持别名规范化（无法识别“web_search”、“search_web”、“google”和“browse”指的是同一种能力），不支持自动错误纠正（无法将指定了错误工具类型的模型请求重定向到正确工具——这是小型模型常见的故障模式），不支持回退链（主要方法失败时尝试其他执行策略），也不支持跨类别路由（无法将单个用户请求调度到 24 个类别中合适的工具）。模型指定了错误的工具类型时，系统会直接失败，而不会智能地重定向请求。

除了这五个问题，同时解决它们所需的工程规模本身也是一项重大挑战。IronEngine 是一个大型系统工程项目：由**97 个源文件**组成，共有 **46,690 行** Python 代码。核心引擎层（`iron_engine/`）包含 **35,157 行**代码，其中工具路由子系统（`tool_router.py`，8,202 行）和流水线编排器（`pipeline.py`，5,289 行）是最复杂的组件。用户界面层（`ui/`）共 7,348 行，API 服务层（`api/`）则有 298 行。27 个关键核心模块共**26,932 行** 代码（占核心引擎的 76.6%），涵盖流水线编排、工具路由、记忆管理、技能学习、模型会话、权限控制、浏览器自动化、桌面图形用户界面控制、网络工具、URL 安全分析和 MCP 兼容性。这一规模体现了 IronEngine 作为完整平台的特性，而非单一用途组件。

IronEngine 从这一系统视角出发进行设计。它并非针对单一交互模态或模型后端进行优化，而是提供了一个统一的引擎，将用户界面交互、应用程序接口、本地和远程模型、分层记忆、任务执行、技能获取、工具路由、兼容 MCP 的可扩展性以及面向设备的集成，整合到一个连贯的平台中。该平台引入了由讨论、模型切换和执行组成的三阶段流水线，将规划质量（由审查器评估）与执行能力（由具备工具访问权限的专用执行器负责）分离。这种架构分离支持异构模型分配，可根据任务需求和可用硬件资源，将不同规模和能力的模型分配给不同角色。

本文研究 IronEngine，并非将其视为模型基准测试，而是将其视为对一个问题的架构性应对：如何构建一款实用的通用 AI 助手。我们介绍该系统的

设计原则、架构分解、工具、记忆、技能和模型管理的子系统设计、文件操作基准测试上的实验评估，以及与具有代表性的人工智能助手系统进行的比较分析。全文聚焦于工程设计决策及其依据，不披露专有提示词、敏感算法或核心实现细节。

2 相关工作

人工智能智能体系统的发展涵盖多种研究传统。我们将相关工作归纳为六个类别，并讨论 IronEngine 与各研究方向的关联及其拓展。

2.1 推理与行动耦合

将推理与环境中的行动相结合，是智能体系统的一项基础范式。ReAct [6] 确立了思维轨迹与行动步骤交替进行的方式，使模型能够依据观察到的结果来进行推理。思维链提示 [7] 表明，明确的中间推理能够提升多步骤问题求解能力，这一原则也自然适用于智能体规划。Reflexion [11] 引入了以语言表达的自我反思作为学习信号，使智能体无需更新权重便能通过多次尝试不断改进。语言智能体认知架构 [24] 为这些实证发现与既有认知科学模型之间的联系提供了理论框架。IronEngine以此为基础，将推理（规划器）、评估（审查器）和行动（执行器）划分为独立的流水线阶段，每个阶段都可以由针对相应角色进行优化的不同模型负责。

后续研究将这些范式扩展到了更复杂的架构。Inner Mono- logue [25] 表明，将环境反馈输入推理循环，能让具身智能体从失败中恢复。行动链方法将高层目标分解为有依据的动作序列，但通常依赖单一模型同时进行推理和行动；当推理质量与动作执行具有不同的计算需求时，这便会形成瓶颈。这类耦合架构的一个根本局限在于，同一模型必须同时保持规划连贯性并生成语法正确的工具调用，而这两项任务适合不同的模型规模和训练目标。IronEngine 在架构层面将推理与行动解耦，以解决这一问题：规划器生成文本计划，由审查器评估其质量；只有当计划达到质量阈值后，独立的执行器模型才会将其转换为工具调用。这种分离使规划可以使用更大型、能力更强的模型，而执行则使用更小型、专门处理工具的模型，从而兼顾质量与资源效率。

2.2 工具使用与 API 集成

工具增强型语言模型是实用型 AI 助手的一项关键能力。Toolformer [8]展示了自监督工具使用，使模型能够学习何时以及如何调用计算器、搜索引擎和其他实用工具。ToolLLM [26] 借助基于决策树的推理框架，将这一方法扩展至 16,000+个真实世界 API。Gorilla [27] 专注于提高 API 调用生成的准确性，并通过检索增强微调减少幻觉。Hug-gingGPT [28] 展示了如何将大语言模型用作控制器，协调专用 AI 模型处理复杂的多模态任务。WebGPT [29] 开创了结合人类反馈的浏览器辅助问答。这些系统通常专注于单一的工具使用范式或 API 接口。IronEngine 则在此基础上扩展，提供统一的工具路由层，通过智能调度、自动类型纠正和多层回退链来管理 24 个工具类别，使单个助手能够在同一执行框架内处理文件操作、网页浏览、GUI 自动化、网络管理和多媒体分析。

工具集成中一个关键但尚未得到充分探索的维度是广度与深度。大多数研究系统都专注于 API 级工具调用（向网络服务发送 HTTP 请求），而现实世界中的桌面自动化需要涵盖截然不同交互方式的工具：CLI 命令、通过无障碍 API 检查 GUI 元素、通过 CDP 操作浏览器 DOM、键盘和鼠标模拟、网络套接字操作，以及多媒体处理流水线。此外，小型模型的工具使用可靠性会显著下降：一个拥有 3.8B 个参数的工具模型可能生成语法正确的 JSON，却指定了错误的工具类型（例如，“web_read”而不是“web_search”），而大多数系统对此类故障的处理方式只是报告错误。IronEngine 引入了以往工作中没有的两种机制：(1) 别名规范化，通过前缀匹配、后缀剥离和语义等价表，将130+ 变体工具类型字符串映射到 24 个规范类别；(2) 自动错误纠正，根据指令内容检测工具类型不匹配（例如，当“browse”类型搭配搜索查询时，会将其重定向至“web_search”），并应用基于内容的启发式规则（例如，指令中提及“.exe”文件时，不会被错误地重定向至二进制读取）。与直接采用模型指定的

模型指定的路由。

2.3 记忆与知识管理

持久化记忆对于跨会话运行的智能体至关重要。检索增强生成 [9]将对外部语料库的检索确立为处理知识密集型任务的一种机制。生成式智能体 [10] 引入了记忆流，并将重要性评分、反思和规划整合为连贯的认知架构。增强型语言模型 [30] 全面综述了外部记忆、检索和工具使用如何扩展模型能力，使其超越参数化知识。ChromaDB [31] 及类似的向量数据库为大规模基于嵌入的检索提供了基础设施。
IronEngine 的记忆系统通过实现分层整合架构，超越了简单的检索增强生成。该架构包含四种条目类型（会话、流水线、每日摘要、精炼内容）、两种合并策略（快速去重和基于模型的整合）、用户评分集成以及空闲时后台处理。这种设计将记忆视为受生命周期策略管理的资源，而非原始对话记录日志。

近期的专用记忆系统在特定方向上推动了技术发展。MemGPT [32] （后来发展为 Letta 框架）引入了受操作系统启发的记忆层级结构，包含主上下文和外部存储，并通过函数调用管理记忆分页。Mem0 为大语言模型应用提供记忆层，能够自动提取和检索用户偏好。这些系统展现了结构化记忆的重要性，但通常只处理记忆生命周期的某个方面。MemGPT 专注于会话内上下文管理（通过虚拟分页扩展有效上下文长度），而 Mem0 则着重于跨会话偏好提取。
IronEngine 的记忆架构有三点不同：（1）它实现了双重合并生命周期——当条目数量超过阈值时，合并 A 会自动触发并快速进行基于哈希的去重；而合并 B 则由审查器监督，采用基于模型的每日整合来生成精炼摘要；（2）它将用户评分（来自用户界面的 1–10 分制评分）整合到记忆检索中，因此会优先回忆评分较高的会话，并降低评分较低会话的优先级；（3）它会检测矛盾，将新检索到的工具结果中的数值（价格、版本、规格）与已存储的记忆条目进行比较，自动用新信息取代过时信息，避免模型依据过时记忆产生幻觉。

2.4 多智能体协作

多智能体系统探索多个大语言模型实例如何有效协作。CAMEL [13] 引入了具有基于角色交互机制的交际智能体，用于开放式探索。MetaGPT [14]将软件生产组织为由标准操作流程协调的多角色流程。ChatDev [15] 进一步探索了面向软件工程工作流的交际智能体。综合综述 [33–35] 指出，智能体画像、通信拓扑、共享记忆和评估方法是核心主题。AutoGen [12] 和 crewAI [36] 提供了灵活的框架，用于定义交互模式可定制的多智能体对话。
IronEngine 采用结构化的多角色方法，使用固定的流水线阶段，而非自由形式的多智能体对话，优先考虑可预测性和可控性，而不是对话灵活性。规划器—审查器讨论循环无需承担管理任意智能体拓扑的开销，即可实现质量保证。

多智能体系统所选择的通信拓扑，从根本上决定了其可靠性特征。CAMEL 的角色扮演方法允许两个智能体进行开放式对话，但没有正式的质量门控——任一智能体都可能生成低质量输出，并在未经检查的情况下传播。MetaGPT 基于 SOP 的协调机制施加了结构约束，但与软件工程工作流紧密耦合。
ChatDev 的基于阶段的方法（设计、编码、测试、编写文档）提供了顺序式质量检查，但仅适用于特定领域。AutoGen 的可对话智能体框架在定义交互模式方面提供了最大的灵活性，但将质量保证留作应用层面问题。IronEngine 明确作出了一项架构权衡：牺牲自由形式多智能体对话的拓扑灵活性，换取包含正式质量门控的固定三阶段流水线。审查器给出的数值质量分数（0.0–1.0）充当客观阈值，必须达到该阈值才能继续执行；审查器提供的结构化反馈（问题和建议部分）则给出了可付诸行动的改进指引。这种设计确保每项执行的计划都经过明确的质量审查，而灵活的多智能体拓扑本身并不能提供这一保证。其代价是，IronEngine 无法在运行时动态创建新的智能体角色或重新配置通信拓扑——这是为了换取可预测的质量表现而接受的一项限制。

2.5 网页和图形用户界面智能体

与网页浏览器和桌面应用程序交互的面向环境的智能体代表了一个重要的前沿方向。Mind2Web [37] 推出了一个基准测

试和数据集，面向能够跨多个不同网站运行的通用型网页智能体。WebArena [38] 提供了一个逼真的网络环境，用于评估自主浏览智能体。SWE-agent [39] 和 SWE-bench [40] 通过代码仓库级代码修改，建立了自动化软件工程基准测试。Open Interpreter [41] 展示了将本地代码执行作为自然语言接口的方式。IronEngine 将网页和桌面 GUI 自动化整合到其工具路由框架中，包括带有反检测措施的基于 CDP 的浏览器控制 [42], 、具备自动验证码处理功能的多引擎网页搜索，以及用于桌面应用程序控制的 Windows UI 自动化（UIA）。

桌面图形用户界面自动化面临的挑战与网页浏览截然不同。OS- World [43] 建立了一套全面的基准，用于评估自主智能体在真实桌面操作系统上的表现；结果显示，即使是最先进的模型，在复杂桌面任务上的成功率也低于 12%。其难点源于桌面用户界面框架的多样性：Win32 控件、WPF 元素、Qt 小部件和基于 Electron 的应用程序，以不同方式提供无障碍信息，有些则完全不提供。例如，微信（基于 Qt 5.15 构建）在深度为 3 时仅公开约 16 个 UIA 元素，且无法通过无障碍树获取联系人姓名，因此需要采用一种根本不同的方法，即依靠屏幕截图和基于坐标的交互。IronEngine 通过分层自动化策略应对这一问题：对于无障碍支持完善的应用程序，优先使用 UIA；对于无障碍信息极少的应用程序，则改用基于屏幕截图的视觉分析（借助本地视觉模型），并结合基于坐标的鼠标和键盘模拟。系统将应用程序专属知识（例如微信托盘图标坐标和表情符号面板布局）作为先天技能，同时将成功的交互序列记录为习得技能，以便在不同会话间重复使用。

2.6 产品化人工智能助手与代码智能体

人工智能助手的产品化进程显著加快。ChatGPT 与 GPT-4o [18] 通过托管于云端的界面提供多模态对话和工具使用功能。支持 MCP 的 Claude Desktop [19, 20] 着重强调协议层面的可扩展性，以便连接外部工具和数据源。Cursor [21] 和 Windsurf [22] 专注于以代码为中心的人工智能辅助，并与集成开发环境深度集成。GitHub Copilot [23] 提供行内代码建议和基于聊天的编程辅助。AutoGPT [44] 开创了自主目标导向型智能体行为。

这些产品可以从两个维度进行分类：云端依赖 和领域覆盖广度。云端产品（ChatGPT、Claude、Gemini）具备最强的模型能力，但所有用户数据都必须经过外部服务器，因此不适用于隐私敏感型工作负载。代码中心助手（Cursor、Windsurf、Copilot、Claude Code）与其目标领域（软件开发）深度集成，但其范围有意限定得较窄——它们不尝试提供桌面自动化、即时消息、多媒体分析或网络管理功能。自主智能体框架（AutoGPT、OpenManus、crewAI）覆盖的任务范围最广，但通常缺乏正式的质量保证机制、显存感知型模型管理，或超越对话日志的结构化记忆。IronEngine 在这一领域中占据独特位置：它结合了本地优先部署与异构模型后端（Ollama [45], LM Studio [46]），工具类别覆盖范围广泛，涵盖代码编辑以外的 24 个类别；通过审查者角色实现正式的质量保证；并提供统一编排层，通过同一套流水线逻辑支持桌面用户界面、REST API 和 Python 客户端。

2.7 OpenClaw 生态系统：OpenClaw、NanoClaw 和 IronClaw

OpenClaw 项目 [17] 及其衍生版本构成了一个快速发展的个人 AI 助手生态系统。OpenClaw 本身被定位为一款始终在线的个人 AI 助手，采用以网关为中心的架构。其设计着重于多通道消息路由（Telegram、WhatsApp、短信、电子邮件）、持久化技能、设备节点管理，以及覆盖用户自有设备的控制平面。该系统擅长在各种消息平台上保持持续可用，并将助手视为长期运行的服务模式，而非基于会话的工具。

NanoClaw [47] 将 OpenClaw 架构改造为适用于资源受限设备（Raspberry Pi、移动 SoC）的边缘部署方案。它通过激进的模型量化、精简技能集，以及轻量级网关来实现这一点；当本地推理能力不足时，该网关会代理请求至能力更强的上游节点。NanoClaw 的主要贡献在于部署灵活性：它表明，个人助手范式可以扩展到物联网和嵌入式环境，不过自主性和工具覆盖范围会有所降低。

IronClaw [48] 专为面向硬件的 AI 智能体场景定制 OpenClaw 框架，目标应用包括嵌入式系统、机器人控制和工业自动化。它引入了传感器数据摄取管道、执行器命令抽象，以及用于物理世界交互的安全联锁。IronClaw 的主要贡献在于弥合对话式人工智能智能体与实时硬件控制回路之间的差距，

将二者连接起来。

PicoClaw [49]将 OpenClaw 范式推向了极致的小型化：面向 Arduino、ESP32 和 STM32 等微控制器。PicoClaw 使用预编译技能集，不进行运行时模型推理，而是依赖上游的 NanoClaw 或 OpenClaw 实例进行规划和决策。微控制器仅执行从上游节点接收的确定性技能流程。PicoClaw 展示了在硬件条件极其有限的情况下部署 AI 智能体的探索方向，此时“智能体”被简化为不具备本地智能的技能执行器。它的贡献主要在概念层面——展示了智能体范式可以如何沿设备层级逐步分解——不过，在传感器网络和简单的执行器控制方面也存在实际应用的可能性。

总体而言，OpenClaw 生态系统涵盖范围广泛，从云规模网关（OpenClaw）到边缘设备（NanoClaw）、工业硬件（IronClaw），再到裸机微控制器（PicoClaw）。尽管覆盖面令人印象深刻，但该生态系统中的各个变体也存在一些共同的局限性。没有任何变体实现正式的计划质量审查（相当于 IronEngine 的审查者角色）。它们不支持在单个任务中进行多模型协作——每个变体针对每个请求仅使用一个模型。工具调度依赖基于关键词的技能匹配，且不具备别名规范化或自动纠错功能。记忆系统采用扁平结构（仅在技能级别持久化），不支持分层整合或矛盾检测。

IronEngine 与 OpenClaw 生态系统在几个根本方面有所不同。

编排深度。OpenClaw 采用消息路由网关模型，将收到的消息调度给技能处理器。IronEngine 则实现了结构化的三阶段流水线，明确划分规划、质量审查和执行阶段，并通过 OpenClaw 架构中所没有的审查者角色提供正式的质量保证。

多模型协作。OpenClaw 生态系统通常为每个请求使用单一模型。IronEngine 则在单个任务中为不同的认知角色分配最多四个不同的模型（规划器、审查器、执行器、工具），并通过感知显存的生命周期管理，按顺序加载模型，从而能够运行超出可用 GPU 内存容量的大型模型。

工具系统复杂度。IronEngine 的 24 类工具路由器，具备智能调度、130+ 别名规范化、自动错误纠正和多层回退链，因此其工具集成能力远超基于技能的 OpenClaw 调度方式。仅自动纠错机制——无需用户干预，即可检测并重新路由输入错误的工具类型——就是一项独特的可靠性功能。

记忆架构。IronEngine 的分层记忆包含四种条目类型、双重合并策略、用户评分集成和空闲时间整合功能，相比 OpenClaw 生态系统中的扁平技能记忆，提供了更结构化的状态管理。

桌面工作台范式。OpenClaw 针对无头多渠道消息传递进行了优化，而 IronEngine 提供了完整的桌面工作台，可实时呈现编排过程（思考区块、工具执行徽章、质量评分），使 AI 的决策过程清晰可见且便于调试。

图1 概述了与本文最相关的研究和产品发展历程，重点展现了从基础技术到系统级集成的演进过程。

3 设计目标

IronEngine 围绕六项目标进行设计，旨在解决引言中指出的工程差距。

G1：统一的交互界面。所有面向用户的入口——桌面用户界面（PySide6 [50]）、REST/WebSocket API 和 Python 客户端——均采用相同的编排逻辑。通过用户界面提交的任务，与通过应用程序接口提交的任务遵循完全相同的规划、审查和执行流程。

G2：基于角色的模型协作。IronEngine 不依赖单一模型承担所有认知功能，而是为不同模型分配不同角色（规划器、审查者、执行器、工具模型）。这样便能以经济高效的方式分配任务：由大型模型处理复杂规划，小型模型则执行工具转换。

G3：本地优先与混合部署。该平台通过集成 Ollama [45] 和 LM Studio [46]，，可完全在消费级硬件上运行，同时也支持云端 API 后端。这种双重能力确保对隐私敏感的工作负载能够完全本地运行。

G4：持久且主动的行为。通过分层记忆、习得技能、定时任务以及空闲时段 Pulse 系统，IronEngine 可在各会话之间保持连续性，并能在无需用户提示的情况下执行后台整合。

G5：协议级可扩展性。MCP（模型上下文协议）[19] 兼容性使系统能够在运行时发现并集成外部工具，从而在不修改核心代码的情况下扩展其能力范围。

G6：可控的安全性。权限-

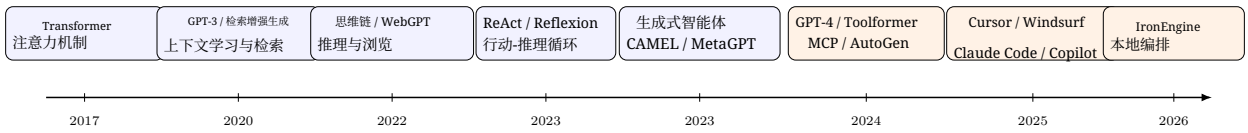

图1：简化与 IronEngine 相关的 AI 智能体研究与产品演进轨迹。蓝色节点表示研究里程碑；橙色节点表示面向产品的系统。

管理、执行沙箱、干预点和角色分离确保助手在用户定义的边界内运行。系统从不直接执行 Shell 命令，并通过结构化调度层验证所有工具调用。

4 总体架构

IronEngine 采用由交互层、编排层、能力层和环境层组成的四层架构，如图 2所示。

交互层 提供三个入口：具备模型选择、执行监控和主题自定义功能的 PySide6 桌面工作台；用于服务集成的基于 FastAPI 的 REST/WebSocket 服务器；以及用于嵌入和脚本编写的纯 Python 客户端。三者共享同一流水线实例。

该**编排层**实现三阶段流水线（第7节），并管理每个请求的生命周期：上下文组装、角色专属模型加载、规划、质量审查、工具执行、干预处理、记忆存储以及技能学习。

该**能力层**由模块化子系统组成：提供方注册表（Ollama、LM Studio、兼容 OpenAI 的云 API）、记忆系统（采用分层整合的 MemoMap）、技能仓库（基于 ChromaDB 的向量化存储）、工具路由器（24 个类别，具备智能分发）以及任务调度器（按日/周/月重复执行的任务）。

该**环境层**将系统连接到外部资源：兼容 MCP 的工具服务器、本地文件系统与存储、网页浏览器（通过 CDP 和 Playwright）、桌面应用程序（通过 UIA）、网络实用工具以及硬件设备。

这一架构的一大优势在于，所有入口点都共用相同的编排逻辑，同时提供方、工具和存储保持模块化。新增模型后端、工具类别或交互界面时，只需在相应的层级边界进行修改，而不必改动整个系统。

5 实现证据与工程特性

IronEngine 的若干核心能力可以通过可观察到的项目结构和文档得到印证。表1 将主要系统主张与代码库中的工程产物对应起来。我们的目的并非披露敏感的实现细节，而是证明系统描述对应着真实的工程产物，而非抽象的提案。

这一证据映射表明，IronEngine 的独特之处并不在于单一组件，而在于多个组件如何被整合进同一个以编排为中心的运行时。

6 模块分解

该系统由八个主要模块组成，这些模块共同构成一个可长期运行、可扩展的平台，而非单一的聊天前端。

桌面用户界面模块。基于 PySide6 的界面充当操作控制台，而非简单的聊天外壳。图3展示了当前界面。左侧面板整合了模型选择（规划器、审查器、工具、视觉）、执行配置和权限设置。右侧面板将对话界面与运行时监控相结合，包括思考区块（可折叠的推理轨迹）、工具执行徽章、质量评分和干预提示。该用户界面支持 10 种程序生成的科幻主题，带有玻璃拟态效果，并在各个面板中持续渲染背景。

API与客户端层。FastAPI 服务器通过 REST端点提供同步请求，并通过 WebSocket通道传输流式事件。Python 客户端（`IronEngineClient`）提供基于回调的集成方式，而 `IronEngineQtClient` 则为图形用户界面嵌入提供 Qt 信号转发。两种客户端类型均支持完整的 17 项回调接口，包括思考、结论、工具执行、权限提示、质量评分、模型选择和阶段转换通知。

多模型协作模块。 流水线为可能不同的模型实例分配不同角色——规划器、审查器、执行器和工具模型——

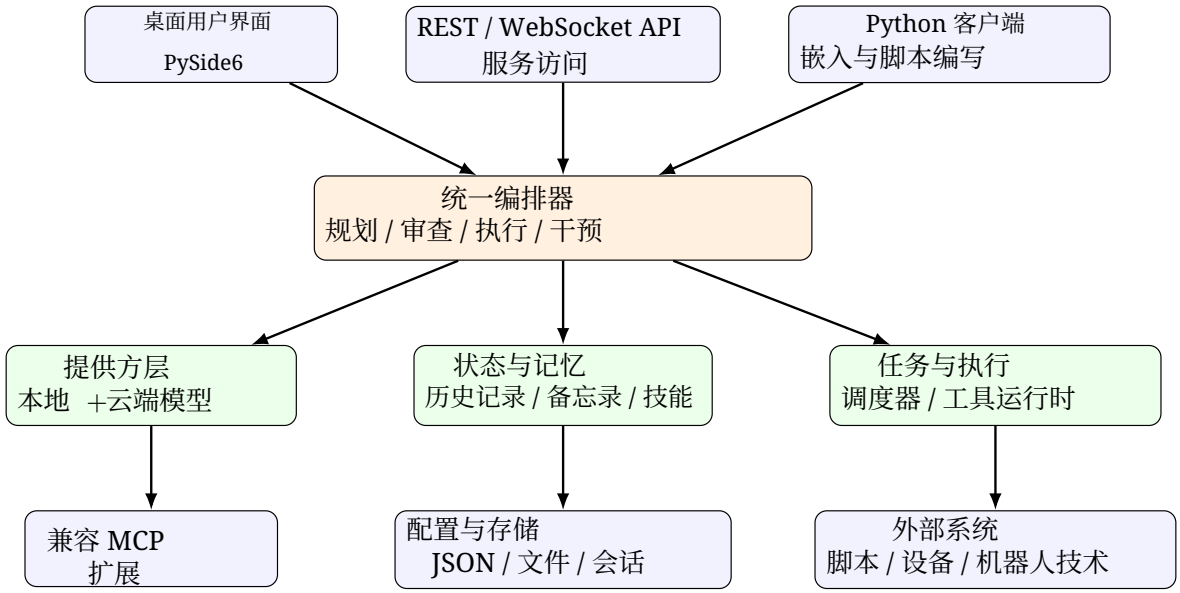

图 2：IronEngine 的总体架构，展示四层设计：交互界面（顶部）、统一编排器（中部）、能力模块（中下部）以及环境接口（底部）。

表1：项目中主要系统主张与工程证据之间的映射

证据位置	能力	解释
`main.py`	多入口运行时	`ui`, `api`, `cli` 种模式共用一个引擎
`docs/API.md`	服务接口	REST、WebSocket、Python 客户端界面
`docs/ARCHITECTURE.md`	流水线设计	规划/审查/执行、技能、记忆、MCP
`pipeline.py`	三阶段编排	讨论、切换、执行；17 个回调
`tool_router.py`	24 类调度	路由、自动纠错、 130+ 别名
`memomap.py`	分层记忆	会话/流水线/摘要/精炼条目
`skill_store.py`	技能仓库	ChromaDB、嵌入搜索、学习
`mcp_client.py`	MCP 客户端桥接	服务器连接、工具发现、调度
`model_session.py`	VRAM 管理	模型加载/卸载、GPU 监控
`main_window.py`	桌面工作台	模型选择器、工具监控、主题

每个角色仅加载其相关的 SOUL（受限运行系统）部分，从而减少输入令牌消耗。提供方注册表通过统一接口支持 Ollama、LM Studio 和兼容 OpenAI 的云 API。

记忆与技能层。 详见第 9节。

任务调度器。 The `PlannedTaskStore` man-ages recurring tasks (daily/weekly/monthly) with JSON-based persistence and thread-safe operations. Pulse 系统在空闲时间提供后台处理，当系统检测到用户连续 5 分钟没有活动时，便会触发记忆整合和任务执行。

本地大语言模型集成层。 详见第 10节。

兼容 MCP 的扩展层。 详见第 11节。

面向硬件的兼容层。 系统可以通过结构化执行接口，将工具调用调度到外部脚本、设备控制器和机器人系统，从而支持与基于 Arduino 的平台及其他硬件环境集成。

7 三阶段流水线

IronEngine 编排的核心是一个三阶段流水线，将规划质量与执行能力分离。这一设计基于这样的观察：任务分解（需要广泛的世界知识和推理）与工具执行（需要精确的语法和格式遵循）在认知需求上存在根本差异。图 4 展示了这三个阶段。

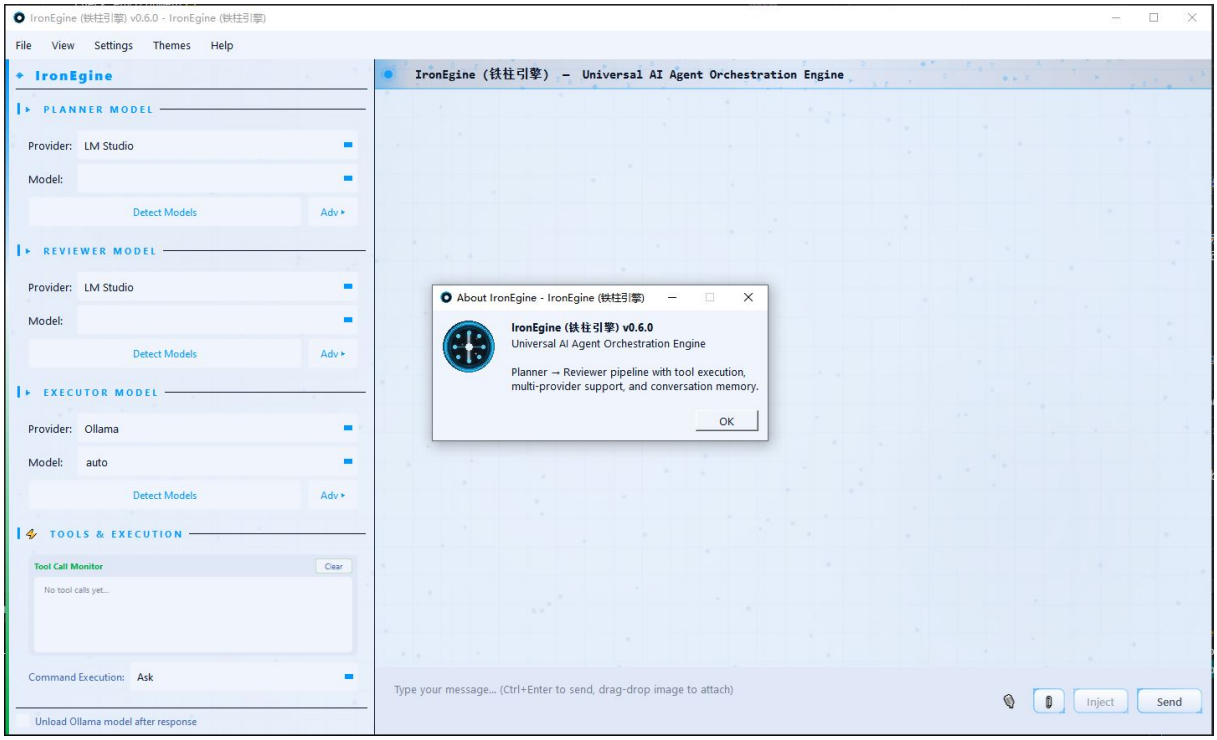

图3：IronEngine 桌面用户界面。左侧面板展示模型和执行配置，并提供规划器/审查器/工具模型选择器。右侧区域支持对话，并显示内联思考块、工具执行徽章和质量评分指标。

7.1 阶段 1：讨论

在讨论阶段，规划器模型会根据用户请求、汇集的上下文（对话历史、已加载的记忆、相关技能、工具清单）以及 SOUL 行为准则，生成任务分解计划。该计划会指定要调用哪些工具以及调用顺序，但此阶段不会实际执行任何工具。

随后，审查器模型会评估计划的质量，检查是否存在幻觉（声称取得结果却未调用工具的计划）、记忆复用（重复使用过时的记忆数据，而非获取最新信息）、完整性和可行性。审查器会给出质量分数，并提供结构化反馈。如果分数低于可配置的阈值，计划就会返回给规划器进行修订，并附上审查器的反馈。

这一讨论循环最多迭代`max_rounds`次（通常为 3–5 次），或直至达到质量阈值。此阶段会采用四种反幻觉机制：

1.**记忆重复检测**：从已加载的记忆中提取信息性词元，并检查模型是否在未调用任何工具的情况下引用这些词元（阈值：词元重叠率达到 40% 且 $\geq$3命中）。数值（价格、版本）会受到特殊处理： $\geq$2个相同的非平凡数值会触发警告。

2.**禁用短语拒绝机制**：六个禁用短语（“我无法 browse”、“无法访问”、“作为 AI”等）会进行模式匹配，并自动拒绝，分数为 0.00。

3. **评分与文本矛盾检测**：如果审查器使用 $\geq$2 拒绝短语（“不符合”“不充分”“缺少”），但给出的分数为 $\geq$0.60，则分数会被限制为 0.40。这可以防止小型模型口头上否决计划，却在数值上予以认可。

4.**记忆矛盾检测**：将工具执行结果中的数值与已加载记忆中的数值进行比较。发现差异时，系统会自动标记过时记忆条目，并在审查器进行评估前插入警告。

7.2 阶段 2：模型切换

一旦满意的计划获批，系统便会从讨论转入执行。这需要从 GPU 显存中卸载规划器和审查器模型，并加载执行器模型。显存管理系统通过特定提供商的 API监控 GPU 内存，并确保模型安全切换。

模型切换遵循四步流程：(1) 完成已批准的计划文本；(2) 通过特定提供商的API从GPU显存中卸载规划器和审查器；(3) 选择并加载执行器模型（由用户配置，或从具备工具能力的3–14B

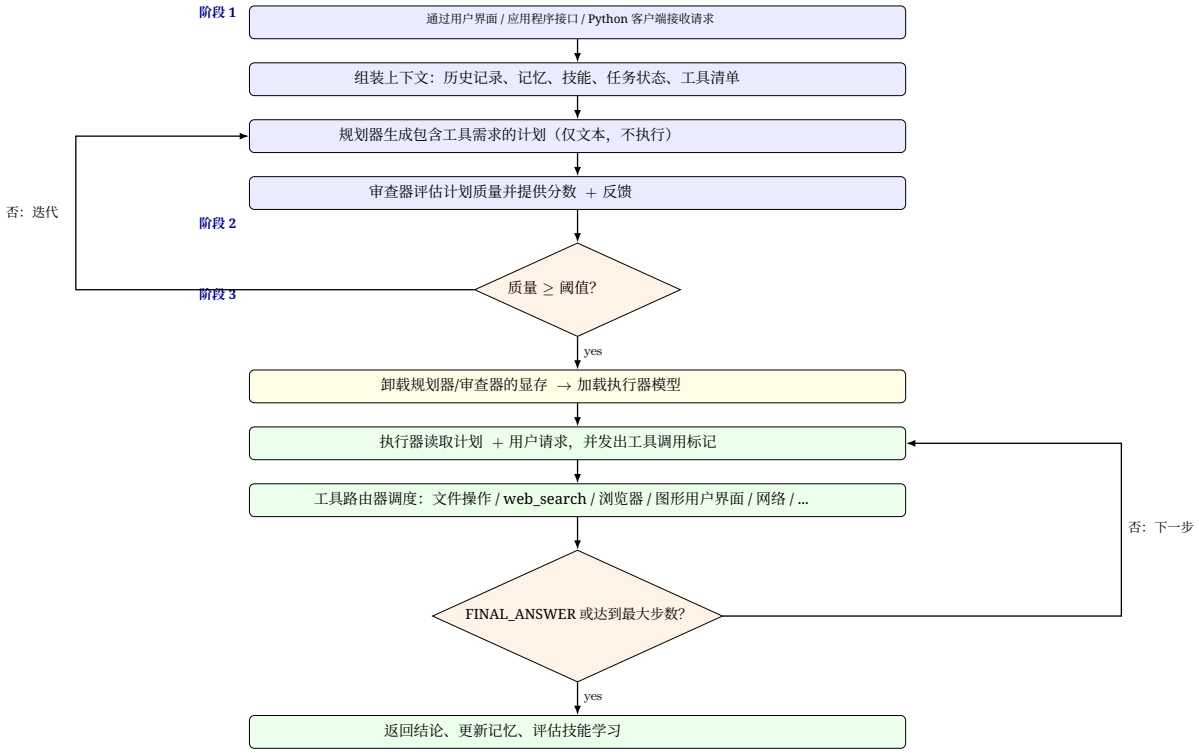

图 4：IronEngine 三阶段流水线。阶段 1（讨论）：规划器和审查器协作提升计划质量，不执行工具。阶段 2（模型切换）：从讨论模型切换到执行模型，并感知显存状态。阶段 3（执行）：执行器反复运行工具，直至任务完成。

模型中自动选择）；(4) 准备执行上下文，其中包括计划、用户请求和相关技能流程。

生成式任务检测。系统使用基于正则表达式的检测来识别创意／生成式请求（撰写故事、代码、诗歌、表格）。另有一个检测器用于检查计划是否包含工具指示项（文件保存、运行、调试、网络搜索、应用控制）。如果任务纯粹是生成式任务，且计划中没有工具指示项，系统就允许直接输出，而无需执行工具。混合任务（编写代码 + 保存／调试）仍会强制执行工具。

简单计划优化。如果获批的计划包含四个或更少的工具调用，或任务完全属于生成式任务，则会重复使用规划器模型作为执行器，从而跳过模型切换开销（根据模型规模不同，约为 27–90 秒）。这项优化可降低简单任务的延迟，同时为复杂任务保留完整的流水线。

7.3 阶段 3：执行

执行器模型接收获批的计划和原始用户请求，然后以结构化格式迭代生成工具调用标记：`[TOOL_CALL: instruction | type: cli/file_ops/... | context: ...]`。每个标记都指定一条指令（要执行什么操作）、工具类型（由哪个处理程序处理）以及可选的上下文。工具路由器（第8节）会调度这些调用并返回带时间戳的结果，供执行器用于下一步操作。

技能集成。在执行第一步之前，系统会使用用户消息和计划摘要作为查询，在技能仓库中进行搜索。匹配的技能会展开其操作流程（递归解析嵌套的技能引用，最多深入 3 层），并作为上下文指导提供给执行器。执行器也可以通过 `[SKILL: name | param: value]`标记显式调用技能。

执行器最多运行 10 步。它会发出 `FINAL_ANSWER` 标记来终止，这表示要向用户呈现结论。如果执行器在同一步中同时发出工具调用和 `FINAL ANSWER`，则会先执行工具，以便结论能够纳入实际结果。对于非生成式任务，如果执行器在没有进行任何工具调用的情况下于第 1 步就试图结束，过早结束`FINAL_ANSWER` 防护机制会检测到这一情况，并注入一条更正提示以重试。

结论来源逻辑。呈现给用户的最终结论通过优先级链进行选择：（1）规划器的 `FINAL_ANSWER`文本，若其包含实质性内容（而非“为了提供……”或“步骤 1：”这类规划式语言）；（2）已去除元审查评论的审查器输出；（3）最后一个执行器步骤的输出。该逻辑用于处理模型将答案放在思考标签中，或

审查器自行执行的情况。

执行完成后，系统会将此次交互存入记忆（如果用户提供了评分，也会一并保存），评估这段经验是否构成可学习的技能（依据评分 ≥ 7 和与最近现有技能的余弦距离 >0.5），并在适用时更新任务状态。

8 工具系统架构

工具系统是 IronEngine 的核心差异化优势，在统一路由框架中提供 24 种不同的工具类别。与依赖函数调用 API 或插件架构的系统不同，IronEngine 的工具路由器可通过别名规范化、自动错误纠正和多层回退链实现智能调度。

8.1 工具类别

表2 列出了 IronEngine 支持的 24 种工具类别，并按功能域进行了分类。

8.2 智能路由

工具路由系统采用了多种机制来提高可靠性：

别名规范化。 超过 130 个别名被映射到规范工具类型。例如，“shell”、“terminal”、“cmd”和“powershell”都规范化为`cli`；“download”、“fetch”和“scrape”都规范化为`web read`。这减轻了规划器必须使用准确工具类型名称的负担。

双向自动纠错。 当模型指定了错误的工具类型时，路由器会检测到不匹配并重新定向。例如，包含文件路径模式（复制、移动、删除操作）的 `cli`调用会自动重定向到 `file_ops`，并剥离命令行专用标志（例如， `/b`、 `-l`）。反过来， `file_ops`调用中的网页 URL 会重定向到 `web read` 或 `browser`。

多层回退链。 每种工具类型都定义了回退策略。例如，网络搜索采用四步策略链：(1) 通过持久化 Chrome 实例进行基于 CDP 的 Google 搜索；(2) DuckDuckGo HTTP POST；(3) Bing HTTP GET；(4) 通过可见浏览器进行 Google+Bing 搜索。如果任何策略成功，则跳过后续策略。

图 5 展示了从模型输出到执行的工具分发流程。

8.3 网页搜索架构

网页搜索是最复杂的工具类别，需要采用多种策略来应对搜索引擎速率限制、验证码挑战和 JavaScript 渲染内容。主要策略是使用持久运行的 Chrome 实例，通过端口 19222 上的 CDP（Chrome 开发者工具协议）访问 Google 搜索，并加入类似人类行为的随机延迟（每次访问间隔 3–8 秒），.....）。结果直接从 DOM 中解析，无需屏幕截图，并会自动解码 Google重定向URL。

为增强结果，系统会自动获取最多 10 个排名靠前的 URL，并进行安全过滤（采用 10 分制启发式评分的钓鱼网站拦截列表、电商URL检测和垃圾URL过滤）。如果 HTTP 获取 JavaScript 密集型网站或反机器人网站时失败，CDP 回退机制会在新的浏览器标签页中打开该 URL，读取渲染后的 DOM，然后关闭该标签页。

反检测措施包括 patchright [42]（Playwright 的反机器人分支）、基于 EasyList 的广告拦截（含 9 万条规则，并启用搜索引擎域名白名单），以及通过 CDP 鼠标事件模拟和随机偏移自动解决 Cloudflare Turnstile 验证码。

图 6 展示了四策略网页搜索回退链。

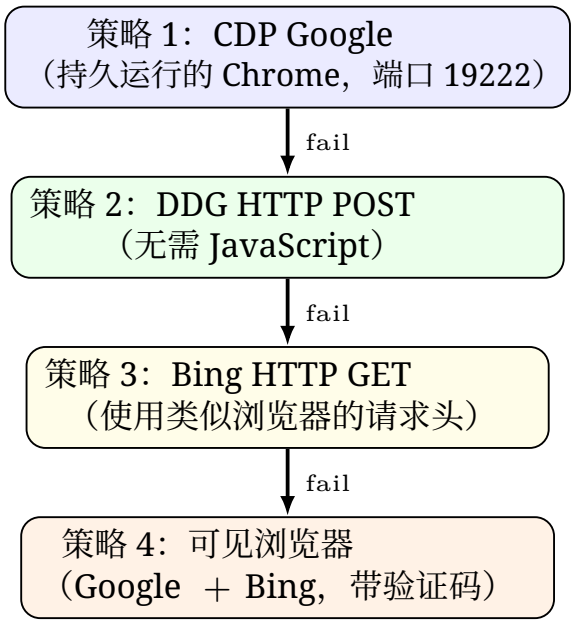

图 6：网页搜索四策略回退链。按顺序尝试各策略；首次成功后即终止回退链。

8.4 网页内容提取

获取网页内容时，系统采用多阶段处理方式：(1) 使用类似浏览器的标头执行 HTTP GET；(2) 如果文本提取结果少于 200 个字符，则通过基于OCR的视觉回退将页面分割为多个屏幕截图，并使用视觉模型逐一分析；(3) 如果两种方法都失败，则通过 CDP 浏览器回退在新标签页中打开该网址，等待页面渲染——

表 2：按功能域组织的 IronEngine 工具类别

领域	工具类型	描述
文件系统	文件_操作、文件_写入、归档、二进制_读取	列出、复制、移动、重命名、删除文件；创建/覆盖文件；压缩/解压归档；读取二进制文件元数据
Web	网络_搜索、网络_阅读、浏览器	支持验证码处理的多引擎搜索；支持光学字符识别的 HTTP 内容提取备用方案；通过 CDP 实现完整的浏览器自动化
通信	微信	用于自动消息发送、文件传输、发送贴图和消息读取的功能。微信
桌面图形用户界面	图形用户界面_自动，应用_控制，自动_输入	鼠标/键盘自动化；窗口管理和 UIA 元素交互；自动表单输入
媒体	图像_生成，图像_读取，语音_文本转语音，语音_语音转文本，视频_分析，音频_分析	图像生成与视觉分析；文本转语音和语音转文本；视频帧分析与音频转录
系统	命令行界面、分类、网络	沙箱命令执行；文本分类；WiFi、FTP、ping、DNS、端口扫描

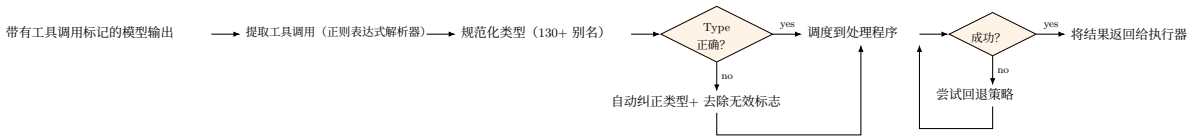

图5：IronEngine 中的工具分发流程。模型输出经解析后，类型通过 130+ 别名进行规范化，若不匹配则自动纠正，并借助多层回退链完成调度。

完成后读取 DOM，然后关闭标签页。电子商务网址在 HTTP 获取之前会自动重定向到浏览器处理程序，因为这类网站通常需要 JavaScript 渲染。每次成功获取的内容都包含一个新鲜度标头，其中带有检索时间戳。

8.5 多媒体分析

工具系统通过专用复合工具支持视频和音频分析。**视频分析**以每 10 秒 1 帧的速率提取帧（最多提取 30 帧，将 8 个等间距帧发送至视觉模型），过滤空白帧（<5KB），自动检测音频轨道并通过 OpenAI Whisper 进行转录，同时提供视觉与听觉分析。**音频分析** 可处理音频文件和视频文件（先提取音轨），并使用单例 Whisper 模型加载器以实现高效的重复使用。

8.6 GUI 自动化

IronEngine 通过三种工具类别提供全面的桌面图形用户界面自动化。**窗口管理**支持启动、关闭和聚焦应用程序，并内置应用程序名称映射和基于注册表的发现功能。**鼠标和键盘**自动化采用双策略方案：以 pyautogui 为主要方法（通过 `MapVirtualKeyW`使用正确的扫描码），并以 ctypes `SendInput`作为后备方案。文本输入使用剪贴板粘贴（pyperclip + Ctrl+V），以确保可靠性。**UIA 交互** 通过 Windows UI 自动化提供元素发现、列表和交互功能；当 UIA 暴露的信息有限时，则辅以基于屏幕截图的视觉分析。

8.7 执行安全

所有工具执行都遵循严格的安全原则。`CommandExecutor` 绝不会将 `shell=True`用于子进程调用。命令中的模板占位符会在执行前被检测并拒绝。URL 安全检查结合钓鱼黑名单与 10 分制启发式评分（URL 中的 IP 地址、可疑顶级域、品牌仿冒、同形字检测、危险协议检测）。文件操作会验证路径，绝不执行任意代码。权限管理器提供三个级别——自动、询问、拒绝，并可按工具类别进行配置。

9 记忆与技能系统

IronEngine 的记忆系统将持久化状态视为受生命周期策略管理的资源，这与仅记录对话转录的系统不同。该系统由两个相互关联的子系统组成：分层记忆（MemoMap）和向量化技能仓库（SkillStore）。

9.1 分层记忆架构

MemoMap 系统将记忆组织为四种条目类型，每种都有不同的生命周期和整合策略：

1. **会话条目**捕获带有用户评分、标签和反思文本的单个对话轮次。这些是派生更高层记忆的原始材料。
2.**流水线条目** 记录完整的流水线执行，包括计划、工具调用、结果和质量评分。

3.**每日摘要** 通过基于模型的摘要生成（合并 B）将一天的会话整合为主题摘要。4. **精炼条目**代表已经过多轮交互验证的长期整合知识。

两种合并策略用于管理记忆整合。**合并 A**（快速去重）会在会话数量超过阈值（默认值为 20）时自动触发，根据内容相似度移除近似重复条目。**合并 B**（基于模型的整合）会在空闲时段通过 Pulse 系统运行，使用由审查器监督的专用摘要模型，将原始会话数据整理为连贯的每日摘要。

图7 展示了记忆层级及其生命周期。

针对新请求的记忆检索采用基于相关性的策略：加载 2 个评分为正面的会话和 1 个评分为负面的会话（以便从错误中学习），并结合防重复警告，避免模型在未经新一轮工具验证的情况下重复使用已记忆的数据。用户评分（1–10 分制，内部映射为 1–5 分）会影响后续检索的优先级，并反馈到反思更新机制中。

9.2 向量化技能仓库

SkillStore 通过 ChromaDB [31] 作为唯一的持久化后端，管理可复用的程序性知识。每项技能都是一条结构化记录，包含名称、类别、标签、分步操作流程、参数定义以及成功/失败计数。

技能表示。 技能通过 Ollama 的嵌入 API，使用 `nomic-embed-text` 模型（768 维向量）进行嵌入。当 Ollama 不可用时，可使用基于确定性哈希的备用方案。采用余弦相似度的 HNSW 索引支持在技能语料库中快速进行近邻搜索。

先天技能与习得技能。 系统预置了 26 项先天技能，涵盖常见场景：微信消息（5 项）、网页操作（3 项）、媒体处理（4 项）、文件管理（4 项）、应用程序控制（3 项）、网络操作（3 项）和 Shell 命令（2 项）。每项先天技能对应一个具体场景，并带有以连字符连接的动作短语标签，以便进行搜索匹配。

技能组合。 技能可以通过内联标记引用其他技能（例如，“发送微信文本”技能会引用“打开微信”和“选择联系人”技能）。在向执行器提供技能上下文时，递归扩展机制会解析这些引用，最多深入 3 层。

技能学习。 当一次流水线执行收到用户给出的评分达到 7 分或以上，且任务足够新颖（与最近的现有技能之间的余弦距离为 > 0.5）时，系统会自动将已执行的工具调用序列提取为一项新的习得技能。相反，评分为 3 分或以下时，会触发纠正请求，并启动新的流水线会话，以改进技能流程。

英语规范化。 所有内部处理（包括技能标签和流程）均以英语进行，不受用户输入语言影响。中日韩文字检测与翻译机制会将中文查询转换为英语，以便一致地匹配技能。

10 自适应模型管理

在 GPU 内存有限的消费级硬件上运行多个模型角色，需要谨慎管理资源。IronEngine 实现了一套自适应模型管理系统，可根据可用硬件资源优化令牌预算、提示复杂度和模型选择。

10.1 模型目录

该系统维护着包含 92 个模型配置文件的目录，每个配置文件都记录了模型的参数量、架构系列、已知能力（补全、工具、视觉、思考、嵌入），以及不同量化等级下估算的显存需求。该目录支持自动进行模型选择和资源规划，用户无需了解特定模型的技术细节。

10.2 VRAM感知上下文预算

上下文长度通过 KV 缓存分配直接影响 GPU 内存消耗。IronEngine 根据三项约束为每次模型调用计算有效上下文长度：

$$\mathrm{ctx}_{\mathrm{eff}} = \min\left(\mathrm{ctx}_{\mathrm{native}},\ \mathrm{ctx}_{\mathrm{mem}},\ \mathrm{ctx}_{\mathrm{user}}\right) \tag{1}$$

其中 $\mathrm{ctx}_{\mathrm{native}}$ 是模型的原生上下文窗口，$\mathrm{ctx}_{\mathrm{user}}$ 是用户配置的任意上限，$\mathrm{ctx}_{\mathrm{mem}}$ 则根据可用显存计算得出：

$$\mathrm{ctx}_{\mathrm{mem}} = \frac{(\mathrm{VRAM}_{\mathrm{free}} \times 0.85) - \mathrm{overhead}}{C_{\mathrm{kv}}} \tag{2}$$

此处 C_{kv} 为每 1K 词元的 KV缓存开销（随模型参数量而变化：$\leq$3B 模型为 0.08MB/1K，$>$30B 模型为 1.2 MB/1K），0.85 系数用于提供针对显存碎片化的安全性余量。允许溢出至RAM，但

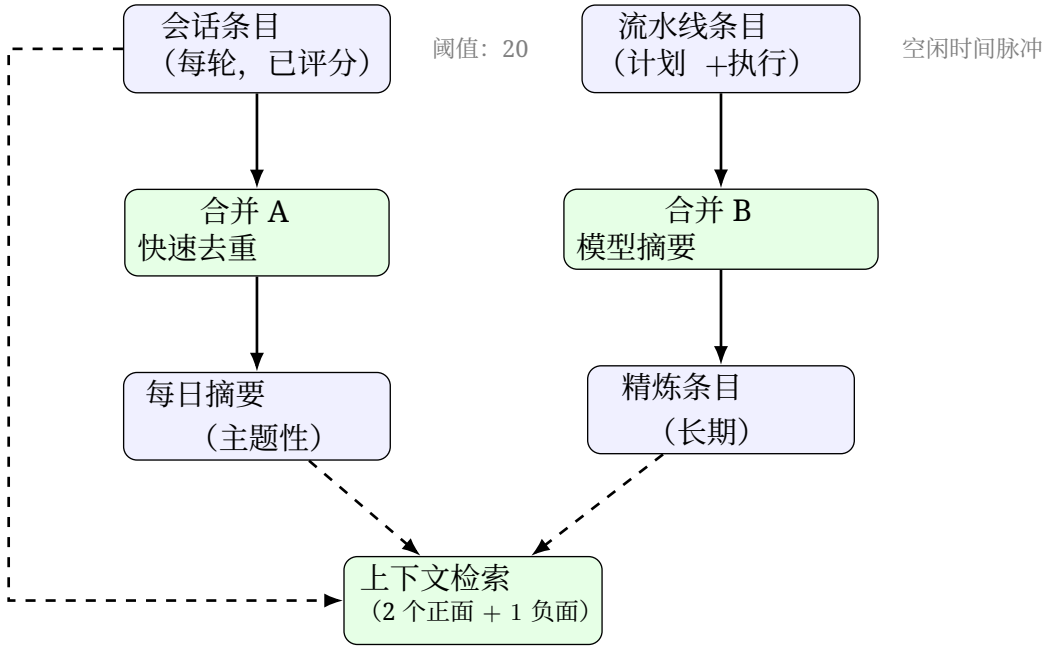

图7：IronEngine 中的记忆层级与生命周期。会话和流水线条目通过两种合并策略进行整合，形成每日摘要和精炼的长期知识。上下文检索会为每个新请求选择最相关的条目（2 个正面 + 1 负面示例）。

上限设为 2× 显存，以防止内存映射访问造成过度减速。

10.3 分层提示系统

不同模型规模处理复杂指令的能力各不相同。IronEngine 根据模型参数量实现了三级提示系统：

表3：基于模型规模的分层提示系统

Tier	工具文档	SOUL	策略
小型 (≤10B)	∼733 tok	∼44 tok	仅身份；最简工具描述
中型 （10–25B）	∼1964 tok	∼623 tok	核心 SOUL 部分；标准工具文档
大型 （>25B）	∼2236 tok	∼1309 tok	完整的 SOUL，包含所有行为规范部分；全面的工具 docs

SOUL（系统在限制下运行）文档定义了每个角色的行为准则。SoulManager 不会将完整的 SOUL 发送给每个模型，而是提取特定角色的章节：规划器接收身份、核心原则、自主性、沟通风格、边界和规划器行为；审查器接收身份、核心原则、沟通风格、边界和审查器行为；执行器仅接收身份和执行器行为。与发送完整 SOUL 相比，这种按角色加载的方式每条消息可节省 450–1300 个词元。

10.4 思考深度控制

对于支持显式推理轨迹（通过 `<think>` 标签）的模型，IronEngine 实现了自适应思考深度控制。任务复杂度分类器会评估每个请求，并确定扩展推理是否有益：

- **简单任务**（文件列表、直接回答）：禁用思考（`think=False`），通过消除推理词元生成来降低延迟。
- **复杂任务**（多步骤规划、分析）：启用思考（`think=True`），使模型能够在作答前对问题进行推理。
- **不确定任务**：思考留给模型默认设置（`think=None`）。

10.5 动态能力检测

系统通过 Ollama `/api/show` 端点动态检测每个模型的能力，该端点会返回一个能力数组（补全、工具、视觉、思考、嵌入）。这使系统能够自动进行路由决策：具备工具调用能力的模型可以跳过辅助翻译模型；具备视觉能力的模型则可充当自己的图像分析后端。对于 API 不会报告能力的模型，系统会使用硬编码能力表作为备用方案。

11 MCP 兼容性与开放生态战略

MCP（模型上下文协议）[19]正在成为连接模型与外部工具和资源的开放接口。IronEngine 实现了 MCP 客户端支持，包括工具发现、服务器连接管理、名称限定，以及内置能力与外部提供工具之间的合并调度。

这形成了一种分层可扩展策略。第一层由深度集成的本地能力组成（24 个工具类别），针对低延迟、高控制性操作进行了优化。第二层通过兼容 MCP 的集成方式向外部生态系统开放系统，从而能够在运行时发现并使用第三方 MCP 服务器提供的工具。

MCP 客户端管理器负责服务器生命周期管理（连接、健康监控、重新连接）、工具模式发现、输入验证和结果格式化。外部发现的工具会与内置工具合并到同一调度流水线中，并通过名称限定避免冲突（例如，`mcp_server_name/tool_name`）。

这种方式使 IronEngine 在受益于更广泛的 MCP 工具生态系统的同时，保留其在桌面控制、本地模型工作流和硬件集成方面的原生优势。随着 MCP 的普及，IronEngine 无需针对每种新工具集成修改内部代码，即可扩展其能力。

12 实验评估

为验证 IronEngine 架构的实际效果，我们进行了实验，测量任务完成准确率、执行时间、工具路由可靠性和多模型协作效率。所有实验均使用消费级硬件上的本地模型。

12.1 实验设置

硬件。 所有实验均在一台配备 NVIDIA RTX 3090（24 GB 显存）、32 GB 系统内存并运行 Windows 10 专业版的工作站上进行。

软件。 模型通过 Ollama [45]和 LM Studio [46]提供服务。流水线配置如下：

- **规划器**：qwen3.5:27b（27B 参数，通过 Ollama 提供服务）
- **审查器**：gpt-oss:20b（20B 参数，通过 LM Studio 提供服务）
- **工具模型**：phi4-mini:3.8b（3.8B 参数，通过 Ollama 提供服务）

这种异构配置展示了系统能够根据计算需求，为不同角色分配不同模型规模。

12.2 文件操作基准测试

我们设计了一个包含四项任务的基准测试，用于检验系统处理具有挑战性路径格式（空格、引号、特殊字符）的文件操作的能力。每项任务都通过 Python 客户端 API，以自然语言请求的形式提交。

结果（表 4）显示，四项任务的完成准确率均为 100%。每项任务平均 385 秒的执行时间包括规划器推理（∼45秒）、审查器评估（∼35秒）、模型切换（∼27秒）、执行器推理（∼30秒）以及工具执行（∼2秒），其余时间则耗费在上下文组装和通信开销上。每项任务相对较高的耗时源于使用了计算资源有限的本地模型；托管于云端的模型将显著降低推理延迟。

12.3 流水线阶段时间分布

图8 展示了四项基准任务在各流水线阶段的时间分布。

图中显示，模型推理占据了执行时间的主要部分（70–80%），而实际工具执行所耗时间微乎其微（<1%）。这表明，IronEngine 的性能主要受本地模型推理速度限制，而非工具调度或系统开销。每项任务的模型切换时间恒定为 ∼27 秒，这表明可通过简单计划复用机制（第7节）进行优化。

12.4 工具路由准确率

我们分析了工具路由系统在各项基准任务和其他压力测试中正确调度工具调用的能力。

表 5：工具路由自动纠正统计

更正类型	数量	成功	备注
命令行界面 → 文件_操作	12	100%	路径模式
browse → 浏览器	3	100%	已检测到别名已规范化
网页读取 →_浏览器	2	100%	JavaScript 密集型网站
浏览器	47	100%	已检测到 130+ 别名
别名规范化	89	100%	直接分派
无需更正所需			

工具路由系统在所有测试场景中均达到 100% 的准确率，自动纠错处理了最常见的模型错误（指定 `cli`

表 4：文件操作基准测试结果。所有任务均达到 100% 正确性（4/4 通过）。

#	任务描述	时间 (s)	轮次	结果	关键观察
1	在包含空格和的目录中列出文件路径中的引号	379	2	PASS	含特殊字符的路径已正确处理；file_ops 已自动选择
2	将文件从一个目录移动到另一个目录（跨驱动器）	311	2	PASS	使用“to”分隔符进行双路径解析；跨盘移动成功
3a	创建包含指定内容的新文本文件	421	2	PASS	文件_写入工具已正确调度；内容已验证
3b	按路径删除指定文件	430	2	PASS	文件_操作：删除操作；文件是否存在已验证删除前后的状态
	总计	**1541**	**8**	**4/4**	平均值：每个任务 385 秒；每个任务 2 个轮次平均值

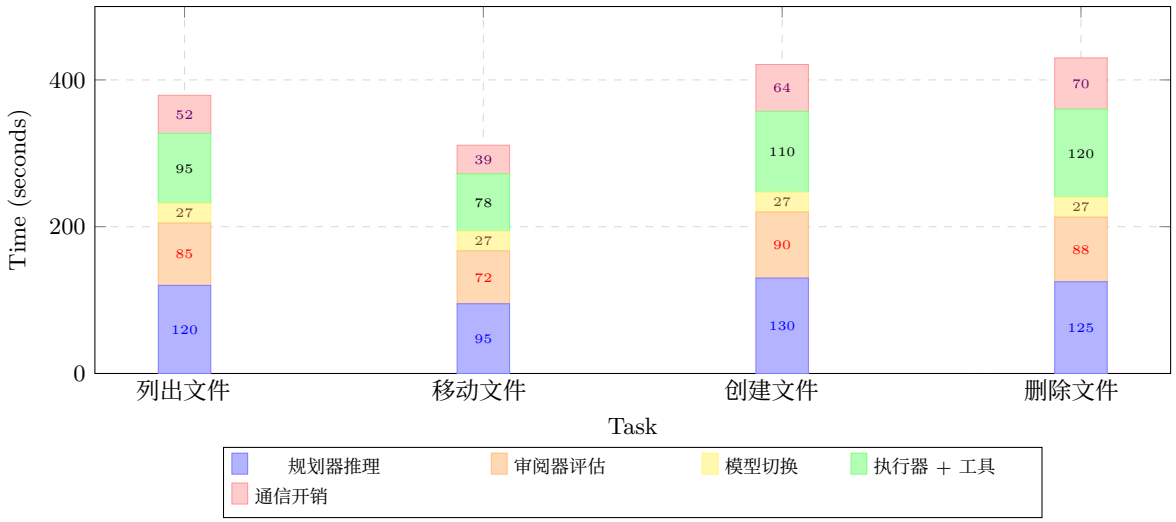

图 8：每个基准任务在各流水线阶段的执行时间分布。模型推理（规划器 + 审查器 + 执行器）占据总时间的主要部分，而实际工具执行在 2 秒内完成。模型切换阶段固定为 ∼27 秒。

而不是用于文件操作的 `file_ops`）。未观察到误报式纠正（例如，对于仅限本地的任务，绝不会触发 `web_search`）。

12.5 多模型协作分析

三阶段流水线的有效性取决于规划器与审查器之间的高效协作。我们分析质量评分分布和迭代模式：

- **第1轮通过率**：65% 的计划在第一轮即获批准，质量评分为 ≥ 0.85。
- **第2轮通过率**：最初被拒绝的计划中，有 30% 在采纳审查器反馈后通过。
- **轮次 3+ 通过率**：只有 5% 的任务需要超过 2 轮讨论。

- **质量分数范围**：获批计划的分数通常为 0.85–0.92；被拒计划的分数为 0.10–0.40。

审查器的反幻觉检测尤为有效：引用数据却未进行工具调用的计划会被判定为编造，得分为 0.10；重复使用记忆价格或具体规格的计划则会被判定为训练数据污染，得分为 0.20。

12.6 显存和资源使用

三阶段流水线的一个关键特征是，模型不会同时驻留在 GPU 内存中。表6 展示了各流水线阶段的显存使用情况。

表 6：各流水线阶段的显存使用情况

阶段	Peak VRAM	备注
讨论（规划器）	∼17.5 GB	qwen3.5:27b Q4 量化
讨论（审查器）	∼13.0 GB	gpt-oss:20b Q4 量化
模型切换	∼2 GB	卸载后残留
执行（执行器）	∼17.5 GB	复用规划器或专用模型
Tool 翻译	∼2.5 GB	phi4-mini:3.8b

在 RTX 3090（24 GB 显存）上，qwen3.5:27b（∼17.5 GB Q4_K_M）可以与 phi4-mini:3.8b（∼2.5 GB）共存，但无法与 gpt-oss:20b 同时加载。因此，讨论阶段的规划器与审查器轮次之间会隐式进行权重交换。显存感知型上下文预算机制可确保 KV 缓存分配不会超过可用显存的 85%。

12.7 搜索去重与缓存

流水线包含一种搜索缓存机制（`_WebSearchCache`），可在不同轮次间对相似查询去重（词语重叠阈值为70%），并强制将每次流水线执行的搜索次数限制在10次以内。在一次流水线集成测试中（耗时17.3分钟，进行了10次图形用户界面浏览器搜索），记录到14次去重命中；所有搜索均成功完成，错误数为0。该缓存显著减少了因审查器请求补充搜索而产生的冗余网络请求。

12.8 错误模式与恢复

尽管基准测试中的完成率达到100%，但更广泛的使用表明，仍存在若干系统性错误模式及其恢复机制：

工具类型误指定。本地模型（≤14B）在文件操作中系统性地优先选择 `cli` 而非 `file_ops`，在浏览器操作中则优先选择 `browse`/`web_read`而非 `browser`。在实验中，三层校正机制（别名规范化、基于上下文的自动校正、回退链）拦截并纠正了所有此类错误，这表明在使用能力受限的本地模型时，智能路由是确保可靠性的必要条件。

规划器幻觉与拒绝。约15%的规划器首轮输出触发了反幻觉检测（未执行工具便生成答案）。通过校正提示，这些问题均在第二轮得到纠正。

记忆污染。加载的历史记忆包含过时信息（过期价格、已更改的文件路径）时，记忆矛盾检测器会将工具结果中的数值与已加载记忆进行比较，自动标记差异，并在审阅器评估前注入警告。

网页搜索降级。四策略回退链的综合成功率为~98%。CDP Google搜索的首次尝试成功率为~85%（偶尔会被Cloudflare Turnstile拦截）；DDG HTTP POST可处理大多数回退情况。剩余的~2%失败是由网络问题或所有引擎同时受到速率限制导致的。

12.9 与代表性系统的比较

表7对IronEngine与代表性人工智能助手系统在12个能力维度上的表现进行了详细比较。比较采用基于公开文档所载能力以及我们在可能情况下进行的直接评估所得的定性评级。

对比结果揭示了IronEngine的几个关键优势：

最广泛的工具覆盖。IronEngine涵盖文件系统、网络、通信、图形用户界面、媒体和网络领域的24类工具，工具覆盖范围在所有受比较的系统中最广泛。代码中心助手（Cursor）仅覆盖编程任务；对话类产品（ChatGPT）专注于网络和代码；OpenClaw生态系统通过技能提供了中等程度的工具覆盖，但缺少具备别名规范化和自动纠错功能的智能路由层。

独特的多角色质量保证。Iro- nEngine是比较中唯一在其流水线内实现正式质量审查阶段的系统。规划器—审查器讨论循环结合反幻觉检测、评分—文本矛盾检查和记忆回收保护机制，提供了更高水平的输出质量保证；OpenClaw生态系统（采用直接技能调度）和云产品（完全依赖单模型能力）均不具备这种保障。

本地优先，能力完备。虽然Open- Claw、NanoClaw和Open Interpreter也支持本地模型，但IronEngine独具优势，将本地部署与多角色编排、持久化分层记忆、向量化技能学习、感知显存的模型生命周期管理以及24类工具路由融为一体——其他本地优先系统均无法同时提供这些能力。

带质量筛选的技能学习。IronEngine能够从成功的任务执行中自动提取可复用技能，并通过基于新颖度的去重（余弦距离阈值）、由用户评分驱动的质量筛选（仅从高评分交互中学习）以及纠错机制（从失败中重新学习）来优化技能，其能力远比OpenClaw的静态技能定义先进，也优于其他对比系统缺乏技能学习的情况。

编排可观测性。桌面工作台通过思考区块、工具执行徽章、质量评分指标和阶段转换通知，实时呈现AI的决策过程。这种可观测性是其相较于无头系统（OpenClaw、NanoClaw）和云产品的一项独特优势；在后者中，用户无法看清内部推理过程。

相对于OpenClaw生态系统的互补定位。尽管OpenClaw擅长多渠道消息传递和始终在线，NanoClaw专注于边缘部署，IronClaw专注于硬件集成，但IronEngine凭借作为一款深度编排桌面工作台的独特定位脱颖而出，在本地优先AI助手平台中拥有最先进的规划流水线、工具路由和记忆管理能力。这些系统相互互补，而非直接竞争：OpenClaw优化的是

表 7：与代表性人工智能助手系统的详细能力对比。评级：**S**=强，**M**=中等，**W**=弱，–=不适用。

本地模型	**S**	W	W	**S**	**S**	M	M	M	**S**
多角色	**S**	W	M	W	W	W	M	**S**	W
流水线 Tool	S (24)	M	M	M	W	M	M	M	M
类别记忆	**S**	M	W	M	W	W	W	W	W
系统技能学习	**S**	W	W	M	W	W	W	W	W
Task	**S**	W	W	M	W	M	–	M	W
调度 MCP	**S**	W	**S**	M	W	W	M	M	W
兼容性 GUI	**S**	W	W	W	W	M	–	W	M
自动化 Web	**S**	**S**	M	M	W	W	W	M	M
浏览显存管理	**S**	–	–	W	M	W	–	W	M
隐私	**S**	W	M	**S**	**S**	**S**	M	M	**S**
开源	–	W	W	**S**	**S**	**S**	W	**S**	**S**
多通道	M	M	W	**S**	M	W	W	M	W
硬件面对	M	W	W	M	M	**S**	–	W	W

表8：跨模型工具感知测试结果。所有模型在使用 IronEngine 的分层提示系统后，均实现了完美的工具类型识别。

模型	参数量	分数	时间（秒）
cogito:8b	8B	100	10.5
cogito:14b	14B	100	13.5
lfm2:24b	24B	100	33.7
qwen3.5:27b	27B	100	50.2

消息渠道的广度；IronEngine 优化的是任务编排的深度。

12.10 跨模型工具感知

IronEngine 分层提示系统的一项关键假设是，只要提供结构恰当的工具文档，即使是小型模型也能可靠地进行工具调度。为验证这一点，我们在一项微信工具类型识别任务中评估了四种参数规模从 8B 到 27B 的模型；在该任务中，模型必须根据自然语言指令正确分类工具类别。

表 8显示，四种模型在工具类型识别上的准确率均达到 100%。推理时间与模型规模近似呈线性增长（从 8B 模型的 10.5 秒增至 27B 模型的 50.2 秒）。关键发现是，即使是最小的 8B 模型，在 IronEngine 的分层提示系统提供范围恰当的工具文档后，也能实现完美的工具调度——小型模型接收的是精简至 733 个 token 的工具清单，而非大型模型使用的完整 2,236-token 版本。这验证了按模型规模分层提供提示内容，而非采用一刀切提示的设计决策。

12.11 全流程集成测试

为在真实条件下评估完整的三阶段流水线，我们针对两种模型配置和四种任务场景开展了八项集成测试，测量规划器—审查器讨论的质量以及端到端任务完成情况。

图 9 展示了质量分数的变化，凸显了审查器的纠正作用。

集成测试（表9）得出几个发现。首先，审查器的纠正性反馈对小型规划器的影响最大：14B cogito 模型的质量评分在一轮审核后从 0.10–0.20 提升至 0.80–0.85，表明审查器有效弥补了小型规划器在初始规划方面的弱点。其次，大型规划器（27B）通常在第一轮就能达到质量阈值，因此无需进行第二轮讨论，从而缩短了总执行时间。第三，唯一一次失败（旅行规划）归因于测试环境无法访问网络，而非流水线缺陷——网络搜索结果不可用时，审查器正确地将计划判定为不充分（分数为 0.40）。第四，生成式任务（故事创作）始终在第一轮以高分通过，验证了 Iro- nEngine 的生成式任务检测机制：对于纯创意任务，该机制会跳过工具执行要求。

表9：不同模型配置和场景下的完整流水线集成测试结果。质量评分展示了第1轮（初始）→ 第2轮（审阅者反馈后）的进展。

模型配置	场景	质量分数	时间（秒）	结果	观察
cogito:14b + cogito:8b	Shell命令执行	0.20 → 0.85	139.6	PASS	审查器反馈改进了计划具体性
cogito:14b + cogito:8b	PDF 处理	0.15 → 0.80	152.2	PASS	需要两轮；工具选择已更正
cogito:14b + cogito:8b	微信消息	0.10 → 0.85	81.8	PASS	技能增强执行；最快补全
cogito:14b + cogito:8b	故事创作	0.85（R1）	321.4	PASS	生成式任务；首次通过轮
qwen3.5:27b + cogito:8b	MCU 比较	0.75 (R1)	172.0	PASS	需要网络搜索；更大规划器有帮助
qwen3.5:27b + cogito:8b	旅行规划	0.20 → 0.40	309.8	FAIL	测试中无法使用网络搜索env
qwen3.5:27b + cogito:8b	Shell命令执行	0.80 (R1)	145.3	PASS	更大规划器首次通过轮
qwen3.5:27b + cogito:8b	PDF 处理	0.75（R1）	168.7	PASS	更大规划器减少了轮次所需

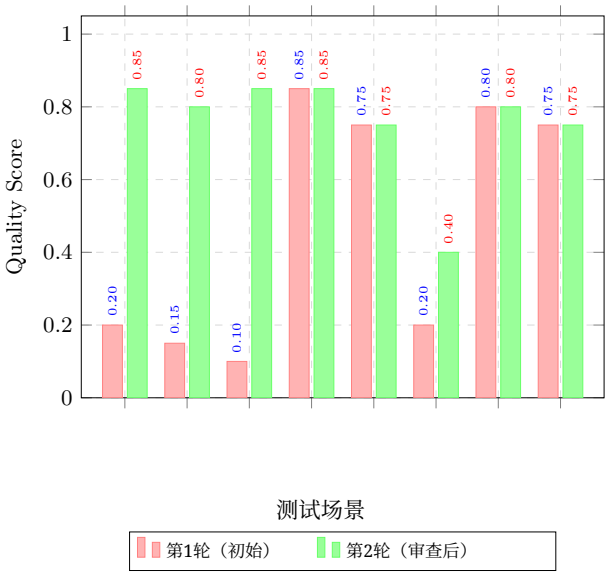

图9：规划器–审查器质量分数变化。红色柱表示初始第1轮分数；绿色柱表示审阅者反馈后的最终分数。审阅者的结构化反馈（问题 + 建议）使14B规划器能够将计划的质量从0.10–0.20提升至0.80–0.85，而更大的27B规划器通常在第一轮就能达到可接受的质量。

12.12 工具模型可靠性

工具翻译模型会将自然语言指令转换为结构化JSON命令。我们评估了三个候选模型，以衡量模型规模、翻译准确率和执行成功率之间的权衡。

表10 揭示了模型规模与工具执行可靠性之间的微妙关系。phi4-mini（3.8B）取得了最佳整体性能：翻译成功率为98.2%，执行成功率为70.6%，在资源效率与可靠性之间实现了最佳平衡。functiongemma（270M）规模过小，无法可靠地进行工具翻译，只有54%的输出能够生成有效JSON。令人意外的是，ministral-3（14B）

表10：工具模型可靠性比较。翻译成功 = 有效JSON输出；执行成功= 正确工具结果。

模型	参数	翻译	执行
phi4-mini	3.8B	98.2% (109/111)	70.6% (77/109)
functiongemma	270M	54.0% (27/50)	25.9% (7/27)
ministral-3	14B	100% (40/40)	20.0% (8/40)

实现了完美翻译（100%），但执行率最低（20%）——该模型生成的命令在句法上正确、但在语义上不正确（例如，JSON结构正确，但参数值错误或缺少必填字段）。这一发现强调，工具模型评估必须同时考虑句法有效性和语义正确性，而且更大型的模型未必能生成更好的工具命令。

12.13 多场景诊断测试

为评估 IronEngine 在各种任务类型中的广度，我们开展了涵盖六种场景的诊断测试，考察不同子系统（工具路由、技能学习、网络搜索、多媒体和生成能力）。

诊断结果（表 11）展示了 IronEngine 架构的几个方面。首先，系统通过同一条流水线处理各种类型的任务——从底层 Shell 命令（139.6 秒），到复杂的多步骤 PDF 处理（152.2 秒，7 次工具调用），再到创意写作（321.4 秒）——无需针对特定任务进行配置。其次，对于成功完成且评分为 ≥ 7 的任务，系统会自动触发技能学习：五个通过的场景中有四个产生了新的习得技能，这些技能可通过增强型 Executor 提示加速类似的后续任务。

表11：多场景诊断测试结果。六个场景中有五个通过（83.3%）。旅行规划失败是因为测试环境缺少网络访问权限。对于评分为≥ 7的任务，会自动触发技能学习。

场景	状态	分数	时间（秒）	轮次	工具调用	技能已学习	备注
Shell命令	PASS	9	139.6	1	5	✓	通过执行器执行命令行界面
PDF 处理	PASS	9	152.2	2	7	✓	多步骤：提取 +总结
旅行规划	FAIL	0	309.8	1	1	—	测试环境中没有网络
MCU 比较	PASS	7	172.0	1	3	—	网络搜索 + 分析
微信消息	PASS	9	81.8	1	5	✓	先天技能 + GUI 自动化
故事创作	PASS	8	321.4	1	7	✓	生成式任务；文件_写入已使用
总体	5/6 通过（83.3%）；新学会 4 项技能；平均时间为 196。			1 s			

第三，完成速度最快的任务（微信消息，81.8 秒）得益于一项先天技能，它为执行器提供了预定义的交互流程，体现了技能系统在降低规划开销方面的价值。第四，唯一一次失败（旅行规划）可正确归因于环境限制（无法访问网络），而非系统缺陷——流水线检测到了工具执行失败，并通过干预回调报告了这一情况，而没有编造回复。

13 接口、部署与产品定位

IronEngine 支持三种互补的交互模式，且全部共享同一条流水线：

桌面工作台（PySide6）：提供最丰富的交互体验，包括模型选择下拉框、实时思考块可视化、工具执行徽章、质量评分指标、干预提示和 10 种可定制主题。此模式针对交互式桌面使用场景进行了优化，适用于需要对编排过程进行重要可观测性的情况。

REST/WebSocket API（FastAPI）：支持与 Web 应用、移动应用及其他服务集成。WebSocket 通道以实时方式流式传输事件，并支持与 Python 客户端相同的 17 种回调类型。此模式使 IronEngine 能够作为自定义前端的后端运行。

Python 客户端：同时支持基于回调的（`IronEngineClient`）和基于 Qt 信号的（`IronEngineQtClient`）集成，以便将 IronEngine 嵌入更大型的应用程序中。客户端 API 对完整的流水线能力集进行了镜像，包括配置、模型选择和权限管理。

13.1 部署模式

完全本地：所有模型均通过同一台机器上的 Ollama 或 LM Studio 提供服务。没有数据会离开本地网络。适用于对隐私敏感的工作负载和气隙环境。

混合部署：常见任务使用本地模型，复杂推理或专用能力则使用云 API。提供方注册表会透明地将请求路由到适当的后端。

服务模式：纯 API 部署，可与外部系统、自动化工作流或多用户环境集成。

13.2 产品定位

表格12 展示了 IronEngine 相对于代表性人工智能智能体框架和产品的定位。

IronEngine 在这一领域占据着独特的定位。与 OpenClaw 生态系统相比，IronEngine 以多渠道消息传递的广度换取编排深度：其三阶段流水线配合正式的质量审查、涵盖 24 个类别的智能工具路由，以及考虑显存的多模型生命周期管理，带来了更精细的任务执行能力，而以网关为中心的 OpenClaw 架构并未尝试实现这些能力。与云端产品相比，IronEngine 在不牺牲工具广度或记忆能力精细度的前提下，提供了完整的本地优先能力。与代码中心助手（Cursor、Windsurf）相比，IronEngine 将应用范围扩展到编程之外，涵盖通用桌面自动化、通信和多媒体分析。最终形成的系统将本地部署优势、多智能体协作深度、协议可扩展性和工具广度融于一体，构建出一个以编排为中心、具备持久化记忆且技能能力不断提升的运行时。

13.3 桌面工作台设计理念

IronEngine 的桌面用户界面不仅仅是聊天界面，而是编排可视化工作台。其设计理念是，AI 助手的决策过程应对用户透明可见，而不是黑盒式的输入→等待→输出模式。

思考区块。一个可折叠的 Markdown 渲染组件，可实时显示规划器和审查器的推理内容。默认折叠

表12：与代表性人工智能智能体框架和产品的系统定位比较

系统	主要定位	本地	多重入口	MCP	代表性特征
IronEngine	桌面工作台+通用助手引擎	S	S	S	计划–审查–执行流水线、统一用户界面/应用程序接口、任务调度、分层记忆、技能学习、24 个工具类别、显存管理、编排可观测性
OpenClaw	个人 AI 助手 + 常开网关	S	S	M	多渠道消息传递（Telegram、WhatsApp、短信），设备节点管理、持久化技能、网关路由
NanoClaw	边缘计算/物联网轻量级助手	S	M	W	激进量化、精简技能集、上游代理，树莓派部署
IronClaw	面向硬件 AI 智能体	M	W	W	传感器数据采集、执行器指令、安全联锁，嵌入式系统集成
ChatGPT	云端对话式人工智能	W	M	W	出色的对话、网页浏览、代码解释器；依赖云端，不支持本地模型
Claude	可扩展协议	M	M	S	强大的 MCP 集成；本地模型和调度能力有限，且记忆支持有限
桌面版 Cursor	助手 AI 代码编辑器	M	W	M	深度 IDE 集成；专注于代码编辑/生成；局限于编程领域
OpenManus	开源通用智能体	M	M	S	通用入口，支持 MCP 变体和多智能体流程执行
AutoGen	多智能体框架	M	M	S	智能体拓扑灵活；面向研究；需要集成以支持桌面工作流

高度为 28 像素（仅显示角色标签），展开后最大高度为 350 像素。支持自动检测并渲染 Markdown 中的标题、粗体文本、列表、代码块和链接。思考区块中的图片会被移除（仅在结论中显示），以减少视觉杂乱。字体使用 Segoe UI（非等宽字体），以提高可读性。

工具执行徽章。一种紧凑的内联组件，用于显示工具调用状态（待处理→已完成），包括工具类型、简要说明和执行摘要。用户可以一目了然地看到执行了哪些工具及其结果。

权限提示。一种内联的允许/拒绝提示，带有 60 秒自动拒绝计时器。当触发询问级工具调用时，用户会在对话流中直接看到权限请求，并且流水线会在用户作出决定后继续。

主题系统。十个程序化生成的科幻主题（深_空、星云_风暴、火星_前哨站等），每个主题都有独特的配色方案和背景图像（由Python脚本生成的 1920×1080 JPEG）。面板通过 `mapTo()`使用坐标映射实现连续背景渲染，并通过透明背景与半透明渐变叠加实现玻璃拟态效果。

这一设计使用户不仅能够使用AI助手进行任务完成，还能观察、理解并调试AI的决策过程——这在开发和调优助手行为期间尤为宝贵。

14 安全与隐私

IronEngine 的安全性通过多种互补机制实现，这些机制在系统架构的不同层级上运行。

14.1 权限管理

权限系统为每个工具类别提供三个可配置级别：

- **自动**：工具调用会立即执行，无需用户确认。适用于文件列表和文本分类等低风险操作。

- **询问**：每次工具调用都会在用户界面中触发内联权限提示，并带有60秒自动拒绝计时器。用户可以批准或拒绝，并可选择在会话期间记住该决定。

- **拒绝**：此类别的工具调用会被完全阻止。在共享环境中限制对敏感操作的访问时，此功能非常有用。

14.2 执行沙箱

该 `CommandExecutor` 会强制执行多项安全性不变量：

- `shell=True` 绝不会用于调用子进程，从而防止 shell 注入攻击。
- 模板占位符（例如 `{filename}`、 `<path>`）会在执行前被检测并拒绝，从而防止命令执行不完整。
- 命令字段会在创建子进程前验证是否为非空字符串。
- 执行超时（默认30秒）可防止失控进程。

14.3 URL 安全

面向Web的操作采用多层离线URL安全系统：

- **钓鱼黑名单**：根据社区维护的钓鱼数据库（Phishing Army、phishing-filter）更新，并采用24小时缓存。

- **启发式评分**：10分评估涵盖 URL 中的 IP 地址、可疑 TLD、品牌-

仿冒、同形字检测、危险协议、过多子域名以及 URL 长度异常。

- **停放/死页面检测**：内容级分析，用于识别不提供任何有用信息的占位页面。

14.4 干预机制

当模型判定某项任务需要超出其范围的人工判断或授权时，它可以发出一个 `INTERVENTION_NEEDED` 标记。这会触发面向用户的提示，并暂停流水线，直到用户提供指导。该机制充当安全阀，用于处理无法通过自动权限检查解决的边缘情况。

14.5 SOUL 编辑控制

SOUL（受限运行系统）文档为每个角色定义行为边界。对 SOUL 的编辑权限通过具有三个级别的权限系统进行控制：**只读**（默认，不允许修改）、**询问**（修改需要用户明确批准）和 **自动**（允许在既定约束内进行自我修改）。这可以防止未经授权更改助手的行为准则。

14.6 本地优先隐私

在完全本地部署模式下，任何数据都不会离开用户的设备。所有模型推理、工具执行、记忆存储和技能学习都在本地环境中进行。这对于涉及敏感文档、个人通信或专有信息的工作负载尤为重要。该系统无需遥测、云端日志记录或外部认证。

14.7 内部英语处理

所有内部处理——规划器/审查器讨论、网页搜索查询、执行步骤——无论用户使用何种语言，均以英语进行。只有最终答案（`FINAL_ANSWER`）会以用户使用的语言呈现。这一策略有两个工程层面的原因：（1）与其他语言相比，大多数开源模型使用英语时的推理和工具调用准确率显著更高；（2）内部统一使用英语，有助于简化技能标签、记忆索引和搜索查询的跨语言匹配。系统内置一个 ∼45词条的中英翻译词典，以及用于自动转换输入的中日韩字符检测功能。

14.8 纵深防御设计

IronEngine 的安全设计遵循纵深防御原则：权限管理是第一道防线（阻止未经授权的操作），执行沙箱是第二道防线（限制已授权操作中的危险行为），URL 安全是第三道防线（过滤恶意外部资源），而干预机制则是最后一道防线（将超出权限范围的决策升级交由人工处理）。这四层彼此独立；绕过其中任意一层，都不会危及整个系统的安全性。

15 讨论

15.1 架构权衡

与单步执行相比，三阶段流水线会带来延迟开销。仅模型切换阶段，每项任务就会增加约 27 秒；对于复杂任务，规划器—审查器讨论可能需要 2– 3 轮。不过，这种开销因可靠性提升而值得：基准测试中的任务完成率达到 100%，这表明审查器提供的质量保证能够避免代价高昂的执行失败；若采用单步系统，这类失败将需要重新提交任务。

分层提示系统对小型模型进行了大幅缩减（$\leq$10 B 模型的 SOUL 上下文仅有 44 个词元，而 $>$25B 模型则有 1309 个词元），这可能导致重要的行为指导丢失。我们的测试表明，小型模型（$\leq$10B）即使在上下文极少的情况下，仍能达到可接受的工具转换准确率，但其规划质量会显著下降，因此规划器角色应使用更大型的模型。

15.2 可扩展性考量

当前的单 GPU 设计将 IronEngine 限制在可装入 24 GB 显存的模型范围内（采用量化时）。虽然这涵盖了参数量最高约 35B 的各类开源模型，但无法支持以全精度运行的最大型开放模型（70B+）。模型切换机制可以扩展为支持多 GPU 配置，或对更大型模型采用 CPU 卸载，但这会增加切换延迟。

随着记忆库不断增大，记忆系统通过线性扫描查找相关条目的效率会逐渐降低。未来工作可以引入更复杂的索引技术（例如结合时间聚类的分层 HNSW），从而在大规模场景下保持次线性的检索时间。

15.3 与 OpenClaw 生态系统的架构比较

OpenClaw 和 IronEngine 代表了 AI 助手的两种截然不同的架构理念。OpenClaw 采用消息路由 模式：用户消息通过网关路由至技能处理器，由后者执行操作并返回结果。这种架构针对多渠道可访问性和响应速度进行了优化，适用于简单的“接收消息 → 执行技能 → 回复”交互模式。

IronEngine 采用编排深度模式：每个请求都会依次经过讨论、切换和执行阶段，并由三个不同角色（规划器、审查器、执行器）分别负责。这会带来额外延迟（模型切换需 ∼27 秒， + 多轮讨论），但能提高任务完成的可靠性：审查器的反幻觉检测可防止错误计划被执行，而自动纠错机制则能修正工具选择错误。

NanoClaw 的边缘部署策略（激进量化 + 上游代理）与 IronEngine 的显存感知型模型管理，从不同角度解决同一个根本问题（在有限硬件上运行模型）。NanoClaw 优化极小型设备上的单模型执行；IronEngine 则优化消费级 GPU 上的多模型协作。IronClaw 的传感器/执行器抽象与 IronEngine 的硬件兼容层目标相似，但 IronClaw 面向实时控制回路，而 IronEngine 面向任务级自动化。

15.4 设计原则与工程实践洞见

IronEngine 的开发为 AI 智能体系统工程带来了若干重要洞见：

角色分离优于能力堆叠。将规划、评估和执行分配给不同的模型角色，而不是依赖单个强大模型承担所有职责，是 IronEngine 的核心架构决策。这带来了三项工程优势：(1) 每个角色都可以使用最符合其需求的模型；(2) SOUL 提示词可以针对角色进行定制，避免将详细的工具文档加载到审查器的上下文中；(3) 可以替换单个角色所使用的模型，而不影响其他角色。

相比精确指令，更重视自动纠错。面对本地模型指令遵循能力有限的问题，IronEngine 实施系统级自动纠错，而非依赖更详细的提示来避免错误。工具路由器的别名规范化和自动纠错机制展示了这一策略的有效性：即使规划器经常错误指定工具类型，系统仍能正确路由。这一经验具有更广泛的启示——设计容错的系统架构，比追求模型输出毫无差错更切实可行。

可观测性是调试的前提。人工智能智能体系统的不透明性是调试的首要障碍。IronEngine 的 17 种回调事件类型提供了从思维过程到工具执行的完整可见性。在实践中， `on_quality_scored` 和 `on_phase_transition`回调对于诊断流水线瓶颈最为有用。

记忆是核心约束。在消费级 GPU 上，显存管理已从可选优化转变为架构基础。KV 缓存的内存开销会随上下文长度线性增长，并且不同模型架构的单 token KV 缓存开销最高可相差 5×。IronEngine 的显存感知的上下文预算机制是实际需要，而非理论上的优化。

15.5 泛化能力

虽然我们的实验评估侧重于文件操作，但该系统的 24 个工具类别已在网络搜索、GUI 自动化、媒体分析和通信任务中进行了非正式测试。这些领域的正式基准测试（例如，WebArena [38]用于网页任务，SWE-bench [40] 用于代码任务）是未来重要的评估目标。

15.6 局限性

应当承认以下几项局限性：

- **本地模型质量**：本地模型（7B–27B）在复杂推理任务中通常不如云端托管的前沿模型。IronEngine 通过多角色协作和质量保证缓解了这一问题，但无法完全弥合能力差距。

- **以 Windows 为中心的测试**：GUI 自动化和应用程序控制主要在 Windows 10 上进行了测试。跨平台支持（macOS、Linux）需要替代的 UIA 后端（AT-SPI、AppleScript），目前尚未实现。

- **单用户设计**：当前架构假定只有一名并发用户。要支持多用户，需要实现会话隔离、资源调度和访问控制机制。

- **基准测试广度**：我们的实验评估深入考察了文件操作，但尚未纳入标准化基准测试，例如 We-bArena [38] （网页任务）或 SWE-bench [40]（代码任务）。

- **延迟开销**：三阶段流水线的模型切换和多轮讨论会

引入显著延迟，因此不适用于要求在一秒内响应的实时交互场景。

- **技能覆盖**：26项内置技能覆盖了常见场景，但长尾任务仍需要交互式用户引导。技能学习效率受用户评分反馈频率的制约。

16 未来工作

已规划若干未来发展方向：

多专家系统。 当前的单一规划器架构可以扩展，以支持多个具备领域特定知识和信用评分的专家配置。路由机制将根据任务特征选择最合适的专家，从而在不牺牲通用性的前提下实现专业化。

标准化基准测试。 与WebArena等既有基准测试集成 [38],SWE-bench [40], 以及自定义多工具基准测试，将能够与其他智能体系统进行系统性比较，并跟踪各版本的性能改进。

用户偏好学习。 当前的记忆与技能系统能够记录任务级知识，但不会明确建模用户偏好（沟通风格、风险承受度、工具偏好）。专门的偏好模块可以从交互历史中学习这些模式，并据此调整助手的行为。

双向 MCP。目前，IronEngine 仅作为 MCP 客户端，从外部服务器调用工具。实现 MCP 服务器能力后，IronEngine 的 24 个工具类别便可开放给其他兼容 MCP 的系统使用，使其在 MCP 生态系统中既是使用者，也是提供者。

跨设备同步。 本地优先架构可以扩展为支持可选的加密多设备同步，让用户在桌面版与移动环境之间无缝切换，同时保障隐私。

多模态深化。 当前的视觉能力仅限于图像分析和基于屏幕截图的图形用户界面导航。进一步集成视频理解（跨帧时间推理）、实时音频对话和空间推理，将拓展系统在具身场景和多媒体丰富场景中的适用范围。

边缘部署。 通过激进量化、推测解码和流水线阶段剪枝，优化资源受限设备（显存不超过 8 GB）的性能，可扩大 IronEngine 的覆盖范围，使其适用于笔记本电脑和移动设备环境。

OpenClaw 生态系统互操作性。 探索 IronEngine 与 OpenClaw 网关之间的互操作模式：IronEngine 可作为 OpenClaw 设备节点，提供深度编排能力；OpenClaw 的多渠道消息传递也可作为 IronEngine 的通信扩展。这些系统各自的互补特性表明，联合部署或许能比单独使用任一系统提供更完整的助手体验。

自适应流水线。 根据任务复杂度自动调整流水线深度：简单任务跳过审查器，直接执行；复杂任务则启用多轮讨论和多专家路由。当前的简单计划复用机制（≤4工具调用跳过模型切换）是朝这一方向进行的初步探索。

17 结论

本文介绍了 IronEngine，这是一款面向系统的通用 AI 助手开发平台，旨在弥补当前智能体系统中的三项关键工程缺口：交互界面分散、子系统集成松散耦合，以及对具备持久行为的本地部署支持有限。

IronEngine 的三阶段流水线将规划质量（规划器—审查器讨论）与执行能力（工具增强型执行器）分离，从而支持在消费级硬件上进行异构模型分配。该系统通过智能路由层集成 24 类工具，并提供别名规范化和自动错误纠正；通过采用双重整合策略的分层记忆管理持久状态；还通过结合基于新颖性的去重机制的向量化技能学习，获取可复用的程序性知识。

在文件操作基准测试中进行的实验评估表明，使用三模型本地配置（27B规划器、20B审查器、3.8 B工具模型）并运行于单台 RTX 3090 上，可实现 100% 的任务完成准确率。与七种具有代表性的人工智能助手进行的详细比较，凸显了 IronEngine 的独特优势组合：本地优先部署、广泛的工具覆盖、持久化记忆、技能学习和协议级可扩展性。

AI 助手的未来不仅取决于模型能力，也取决于架构、编排、可观测性、安全性和长期适应性。IronEngine 展示了如何在单一平台内连贯地解决这些系统层面的问题，为新一代通用、注重隐私保护且持续改进的 AI 助手奠定基础。

从更广阔的视角来看，IronEngine 验证了一项核心观点：通过精心设计的系统架构（角色分离、智能路由、分层记忆、感知显存的调度），中等规模的本地开源模型也能在实际任务中实现实用的自动化水平。与依赖单个超大型模型的云端方案相比，这种系统智能方案在成本、隐私和可定制性方面具有结构性优势。与 OpenClaw 等网关式架构相比，IronEngine 的深度编排能够更可靠地完成任务。这两种方案并非相互对立，而是人工智能助手生态系统中的互补技术选择。随着开源模型能力不断提升，IronEngine 所代表的本地编排范式将在日益广泛的应用场景中展现价值。

致谢

作者感谢 NiusRobotLab 对通用人工智能助手系统探索和开放工程实践的支持。同时，也向推动本地模型运行时（Ollama、LM Studio）、多智能体研究（AutoGen、CAMEL、MetaGPT）、向量数据库（ChromaDB）、浏览器自动化（Playwright、patchright）以及 MCP 相关互操作性发展的开源社区致以谢意。这些社区共同塑造了 IronEngine 演进所处的技术环境。最后，感谢测试人员和用户，他们提出的具体场景帮助系统从原型构想迈向可运行的平台。

参考文献

[1]Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N. Gomez、Łukasz Kaiser 和 Illia Polosukhin。《Attention is all you need》。《神经信息处理系统进展》，第 30 卷，2017年。

[2]Tom Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared D. Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell、Sandhini Agarwal、Ariel Herbert-Voss、Gretchen Krueger、Tom Henighan、Rewon Child、Aditya Ramesh、Daniel M. Ziegler、Jeffrey Wu、Clemens Winter、Chris Hesse、Mark Chen、Eric Sigler、Mateusz Litwin、Scott Gray、Benjamin Chess、Jack Clark、Christopher Berner、Sam McCandlish、Alec Radford、Ilya Sutskever 和 Dario Amodei。语言模型是少样本学习者。进展发表于《神经信息处理系统进展》，33：1877–1901，2020 年。

[3] Hugo Touvron、Thibaut Lavril、Gautier Izacard、Xavier Martinet、Marie-Anne Lachaux、Ti mo-thée Lacroix、Baptiste Rozière、Naman Goyal、Eric Hambro、Faisal Azhar、Aurelien Rodriguez、Armand Joulin、Edouard Grave 和 Guil-laume Lample。《Llama：开放且高效的基础语言模型》。*arXiv* 预印本 *arXiv:2302.13971*，2023年。

[4] Josh Achiam、Steven Adler、Sandhini Agarwal、Lama Ahmad、Ilge Akkaya、Florencia Leoni Ale-man、Diogo Almeida、Janko Altenschmidt、Sam Altman、ShyamalAnadkat 等。《GPT-4 技术报告》。*arXiv* 预印本 *arXiv:2303.08774*，2024。

[5]Google DeepMind。Gemini：多模态 AI 模型系列。在线，2025。URL https://deepmind.google/technologies/gemini/。访问日期：2026-03-07。

[6] Shunyu Yao、Jeffrey Zhao、Dian Yu、Nan Du、Izhak Shafran、Karthik Narasimhan 和 Yuan Cao。ReAct：语言模型中推理与行动的协同。国际学习表征会议，2023年。

[7]Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Brian Ichter、Fei Xia、Ed Chi、Quoc V. Le 和 Denny Zhou。思维链提示能够激发大型语言模型的推理能力。神经信息处理系统进展，35：24824–24837，2022。

[8] Timo Schick、Jane Dwivedi-Yu、Roberto Dessi、Jason Wei、Alexandra Birch、Jason Weston 和 Gael Le Scao。Toolformer：语言模型可以自学使用工具。国际学习表征会议，2024。

[9]Patrick Lewis、Ethan Perez、Aleksandra Pik-tus、Fabio Petroni、Vladimir Karpukhin、Na-man Goyal、Heinrich K"uttler、Mike Lewis、Wen tau Yih、Tim Rockt"aschel、Sebastian Riedel 和 Douwe Kiela。面向知识密集型自然语言处理任务的检索增强生成。神经信息处理系统进展，33：9459–9474，2020。

[10]Joon Sung Park、Joseph O'Brien、Carrie Cai、Meredith Ringel Morris、Percy Liang 和 Michael S. Bernstein。《生成式智能体：人类行为的交互式模拟体》。*ACM* 用户界面软件与技术研讨会，2023年。

[11]Noah Shinn、Federico Cassano、Ashwin Gopinath、Karthik Narasimhan 和 Shunyu

Yao。《Reflexion：具备语言强化学习的语言智能体》。《style id='1'>神经信息处理系统进展，2023年。

[12] 微软 Autogen：通过多智能体对话赋能新一代大语言模型应用。在线，2024年。网址 https://github.com/microsoft/autogen。访问日期：2026-03-07。

[13] Guohao Li、Hasan Abed Al Kader Hammoud、Hani Itani、Dmitrii Khizbullin 和 Bernard Ghanem。Camel：用于探索大型语言模型社会“思维”的交际智能体。发表于第三十七届神经信息处理系统大会，2023年。

[14]Sirui Hong、Mingchen Zhuge、Jonathan Chen、Xiawu Zheng、Yuheng Cheng、Jinlin Wang、Ceyao Zhang、Zili Wang、Steven Ka Shing Yau、Zijuan Lin、Liyang Zhou、Chenyu Ran、Lingfeng Xiao、Chenglin Wu 和 Jürgen Schmidhuber。MetaGPT：面向多智能体协作框架的元编程。发表于第十二届国际学习表征会议，2024。网址 https://openreview.net/forum?id=VtmBAGCN7o。

[15]Chen Qian、Wei Liu、Hongzhuo Liang、Yang Yu 和 Zhenhui Li。Chatdev：用于软件开发的交际智能体。计算语言学协会，2024。

[16] Xinbin Liang、Jinyu Xiang、Zhaoyang Yu、Jiayi Zhang、Sirui Hong、Sheng Fan、Xiao Tang、Bang Liu、Yuyu Luo 和 Chenglin Wu。《Openmanus：用于构建通用人工智能体的开源框架》。Zenodo，2025。URL https://doi.org/10.5281/zenodo.15186407。访问日期：2026-03-07。[17]OpenClaw 社区和 Peter Steinberger。《Openclaw：个人人工智能助手》。在线，2026。URL https://github.com/openclaw/openclaw。访问日期：2026-03-07。[18] OpenAI。《Gpt-4o：用于文本、视觉和音频的全模态模型》。在线，2025。URL https://openai.com/index/hello-gpt-4o/。访问日期：2026-03-07。[19]Anthropic。《模型上下文协议》。在线，2024。URL https://modelcontextprotocol.io。访问日期：2026-03-07。[20]Anthropic。《Claude code：Anthropic 官方的 Claude 命令行界面》。在线，2025。URL https://docs.anthropic.com/en/docs/agents-and-tools/claude-code。访问日期：2026-03-07。

[21] Anysphere Inc.。《Cursor：AI 代码编辑器》。在线，2025。URL https://cursor.com。访问日期：2026-03-07。

[22] Codeium。Windsurf：流畅状态下的 AI 集成开发环境。在线，2025。URL https://windsurf.com。访问日期：2026-03-07。

[23] GitHub。GitHub Copilot：您的 AI 编程搭档。在线，2024。URL https://github.com/features/copilot。访问日期：2026-03-07。

[24]Theodore R. Sumers、Shunyu Yao、Karthik Narasimhan 和 Thomas L. Griffiths。语言智能体认知架构。机器学习研究汇刊，2024。

[25]Wenlong Huang、Fei Xia、Ted Xiao、Harris Chan、Jacky Liang、Pete Florence、Andy Zeng、Jonathan Tompson、Igor Mordatch、Yevgen Chebotar、Pierre Sermanet、Noah Brown、Tomas Jackson、Linda Leal、Daniel Driess、Chelsea Finn 和 Karol Hausman。Inner Monologue：通过使用语言模型进行规划实现具身推理。发表于机器人学习会议 *(CoRL)*，2023年。

[26] Yujia Qin、Shihao Liang、Yining Ye、Kunlun Zhu、Lan Yan、Yaxi Lu、Yankai Lin、Xin Cong、Xian- gru Tang、Bill Qian、Sihan Zhao、Lauren Hong、Runchu Tian、Ruobing Xie、Jie Zhou、Mark Ger- stein、Dahai Li、Zhiyuan Liu 和 Maosong Sun。Toolllm：助力大语言模型掌握 16000+真实世界 API。国际学习表征会议，2024年。[27]Shishir G. Patil、Tianjun Zhang、Xin Wang 和 Joseph E. Gonzalez。Gorilla：连接海量 API 的大型语言模型。*arXiv* 预印本 *arXiv:2305.15334*，2023年。[28]Yongliang Shen、Kaitao Song、Xu Tan、Dong- sheng Li、Weiming Lu 和 Yueting Zhuang。Hugginggpt：使用 ChatGPT 及其 Hugging Face 好友解决 AI 任务。神经信息处理系统进展，36，2024。[29] Reiichiro Nakano、Jacob Hilton、Suchir Balaji、Jeff Wu、Long Ouyang、Christina Kim、Christo- pher Hesse、Shantanu Jain、Vineet Kosaraju、William Saunders、Xu Jiang、Karl Cobbe、Tyna Eloundou、Gretchen Krueger、Kevin Button、Matthew Knight、Benjamin Chess 和 John Schulman。Webgpt：借助浏览器并结合人类反馈进行问答。*arXiv* 预印本 *arXiv:2112.09332*，2022。[30] Grégoire Mialon、Roberto Dessi、Maria Lomeli、Christoforos Nalmpantis、Ram Pasunuru、Roberta Raileanu、Baptiste Rozière、Timo

Schick、Jane Dwivedi-Yu、Asli Celikyilmaz、Edouard Grave、Yann LeCun 和 Thomas Scialom。增强型语言模型：综述。机器学习研究汇刊，2023年。

[31]Chroma。Chroma：面向 AI 的开源嵌入数据库。在线，2024。网址 https://www.trychroma.com。访问日期：2026-03-07。

[32]Charles Packer、Sarah Wooders、Kevin Lin、Vivian Fang、Shishir G. Patil、Ion Stoica 和 Joseph E. Gonzalez。MemGPT：迈向将大语言模型作为操作系统。*arXiv* 预印本 *arXiv:2310.08560*，2023年。

[33] Tao Guo、Xiang Yue、Yifei Zhang、Tongzhe Li、Dong Wang、Yilun Zhao、Huan Sun、Jun Wang 和 Rui Yan。基于大型语言模型的多智能体：进展与挑战综述。*arXiv* 预印本 *arXiv:2402.01680*，2024。

[34] Zhiheng Xi、Wenxiang Chen、Xin Guo、Wei He、Yiwen Ding、Boyang Hong、Ming Zhang、Junzhe Wang、Senjie Jin、Enyu Zhou、Rui Zheng、Xi- aoran Fan、Xiao Wang、Li mao Xiong、Qin Liu、Yuhao Zhou、Weiran Wang、Changling Jiang、Yicheng Zou、Xiangyang Liu、Zhangyue Yin、Shihan Dou、Rongxiang Weng、Wensen Cheng、Qi Zhang、Wenjuan Qin、Yongyan Zheng、Xipeng Qiu、Xuanjing Huang 和 Tao Gui。基于大型语言模型的智能体的兴起与潜力：综述。*arXiv* 预印本 *arXiv:2309.07864*，2023年。[35] Lei Wang、Chen Ma、Xueyang Feng、Zeyu Zhang、Hao Yang、Jingsen Zhang、Zhiyuan Chen、Jiakai Tang、Xu Chen、Yankai Lin、Wayne Xin Zhao、Zhewei Wei 和 Ji-Rong Wen。基于大型语言模型的自主智能体综述。计算机科学前沿，18(6)，2024。[36] crewAI。crewai：多智能体自动化框架。在线，2024。网址 https://github.com/crewAIInc/crewAI。访问日期：2026-03-07。[37] Xiang Deng、Yu Gu、Boyuan Zheng、Shijie Chen、Samuel Stevens、Boshi Wang、Huan Sun 和 Yu Su。Mind2Web：迈向通用型网络智能体。神经信息处理系统进展，36，2024。

[38]Shuyan Zhou、Frank F. Xu、Hao Zhu、Xuhui Zhou、Robert Lo、Abishek Sridhar、Xianyi Cheng、Tianyue Ou、Yonatan Bisk、Daniel Fried、Uri Alon 和 Graham Neubig。Webarena：用于构建自主智能体的逼真网络环境。国际学习表征会议，2024。

[39]John Yang、Carlos E. Jimenez、Alexander Wettig、Kilian Lieret、Shunyu Yao、Karthik Narasimhan 和 Ofir Press。Swe-agent：智能体—计算机接口助力自动化软件工程。*arXiv* 预印本 *arXiv:2405.15793*，2024。

[40] Carlos E. Jimenez、John Yang、Alexander Wettig、Shunyu Yao、Kexin Pei、Ofir Press 和 Karthik Narasimhan。SWE-bench：语言模型能否解决真实世界的 GitHub 问题？国际学习表征会议，2024。

[41] Open Interpreter。Open Interpreter：面向计算机的自然语言接口。在线，2024。URL https://github.com/OpenInterpreter/open-interpreter。访问日期：2026-03-07。

[42]patchright 贡献者。patchright：反检测 Playwright 分支。在线，2025。URL https://github.com/nicosql/patchright。访问日期：2026-03-07。

[43]Tianbao Xie、Danyang Zhang、Jixuan Chen、Xiaochuan Li、Siheng Zhao、Ruisheng Cao、Toh Jing Hua、Zhoujun Cheng、Dongchan Shin、Fangyu Lei、Yitao Liu、Yiheng Xu、Shuyan Zhou、Silvio Savarese、Caiming Xiong、Victor Zhong 和 Tao Yu。Osworld：在真实计算机环境中对执行开放式任务的多模态智能体进行基准测试。在线，2024年。网址 https://os-world.github.io/.[44] Significant Gravitas。AutoGPT：一项自主 GPT-4 实验。在线，2023年。网址 https://github.com/Significant-Gravitas/AutoGPT。访问日期：2026-03-07。[45] Ollama。Ollama 文档。在线，2025年。网址 https://ollama.com。访问日期：2026-03-07。[46] LM Studio。LM Studio 文档。在线，2025年。网址 https://lmstudio.ai。访问日期：2026-03-07。[47] OpenClaw 社区。NanoClaw：轻量级边缘 AI 助手。在线，2026年。网址 https://github.com/openclaw/nanoclaw。访问日期：2026-03-07。[48]OpenClaw 社区。IronClaw：面向嵌入式系统的硬件导向型 AI 智能体。在线，2026年。网址 https://github.com/openclaw/ironclaw。访问日期：2026-03-07。[49]OpenClaw 社区。PicoClaw：面向嵌入式平台的微控制器 AI 智能体。在线，2026年。网址 https://github.com/openclaw/picoclaw。访问日期：2026-03-08。

[50]Qt公司。PySide6：Qt 6框架的官方 Python 模块。
在线，2024。网址 https://doc.qt.io/qtforpython-6/。访问日期：2026-03-07。