HUAWEI

Coactive Agent 技术白皮书

面向开放世界的 Agent 范式

COACTIVE AGENT TECHNICAL WHITE PAPER

Coactive Agent 华为云 AI 创新 LAB

版本 V1.0 | 2026 年 7 月

RECOMMENDATION

推荐语

张民 哈工大（深圳）特聘校长助理，计算与智能研究院院长，ACL/AAIS Fellow

在业界普遍将下一代 Agent 简化为“更高自主性”的当下，这份白皮书给出了更为清醒而深刻的回答：Agent 的未来不是把人移出工作流，而是让人的经验、判断与创造力持续进入工作流。白皮书提出的 Coactive 范式，将 Interactive 与 Proactive 耦合为“不确定处共创、成熟处承接”的双态协同，并以环境感知、智能主体、认知底座与元进化闭环构成完整的系统架构，把人机协同从理念推进为可工程化、可治理、可持续成长的体系。人机协同的深度，取决于理解与交互的深度。这份白皮书为通往人机智能深度融合的未来提供了一份兼具前瞻性与工程可行性的路线图，值得研究者与产业界认真研读。

肖仰华 复旦大学未来技术研究院副院长，计算与智能创新学院教授
上海市数据科学重点实验室主任

智能体的价值不在于把人排除在外，而在于让人的经验、判断与创造力更好地融入其中。华为云这份白皮书提出的“协同共处”范式，把行业从“越自主越好”的认知误区，拉回到“交互式”与“主动式”协同耦合的正确路线，直面开放世界中目标漂移、上下文缺失、授权失控等真问题。其“环境感知 × 智能主体 × 认知底座 + 元进化闭环”的整体架构，辅以贯穿智能体全生命周期的安全体系，为智能体在企业侧的可信落地给出了扎实路径。这是一份既有范式高度、又见工程细节的诚意之作。

刘知远 清华大学计算机系长聘教授，面壁智能联合创始人

从 Reactive 到 Proactive，再到 Coactive 的技术范式，驱动着 Agent 智能体技术体系持续突破，指向人机共生、包容向善的人文未来。在当前的社会背景下，Coactive 对于如何构建“以人为本”的数字发展框架有着重要的研究意义。期待工业界与学术界的研究者共创 Coactive 的未来。

赵新奎 浙江大学软件学院百人计划研究员，博士生导师

自大模型技术问世以来，学术界与产业界持续探索人机协作的理想路径，单向问答、全自动自主智能等各类范式均已有实践。尽管探索方向各有侧重，但技术演进的核心目标始终不变：更好地服务于人、赋能文明。Coactive Agent在自主性与可控性之间达成精妙平衡，令智能体成为人类可靠的协作伙伴，全面感知真实情境、深度参与决策、高效协同完成任务。这一创新范式有望推动人机协作从单纯的工具属性或替代关系走向伙伴共生关系，为生产力的跃升与协作体验的革新提供新的可能。

温颖 上海交通大学人工智能学院 副教授，上海创智学院 全时导师

如果说人类文明的核心力量来自合作，而不是单一目标下的服从，那么下一代 Agent 也不应被理解为孤立执行命令的自动化机器。Coactive Agent 的意义，在于把人、AI、环境、记忆和制度边界放回同一个真实任务现场，让 AI 在开放世界中成为人的智能放大器，而不是人的替代者。这份白皮书提供了一种更成熟的判断：AI的未来不是把人移出工作流，而是让人类以更强的协作能力进入新的文明阶段。

骆昱宇 港科大（广州）助理教授，博士生导师

随着大模型智能体从“能够完成任务”走向“持续进入真实工作”，行业面临的关键问题已不再只是如何提升 Agent 的自主性，而是如何让其在复杂、动态、开放的环境中，与人形成长期、可信、可控的协作关系。

本白皮书提出 Coactive Agent 范式，将 Interactive 的实时共创与 Proactive 的主动执行有机结合，并围绕共享任务现场、可信承接、协作唤醒、长期记忆与持续进化，构建了一套较为完整的技术架构。其核心价值在于：既不把 Agent 局限为被动响应的工具，也不盲目追求将人排除在外的全自动化，而是让人的经验、判断与创造力持续进入工作流，同时将成熟、可验证、可授权的任务逐步交由 Agent 可靠承接。

这份白皮书既提供了面向下一代人机协作的系统性思考，也覆盖了环境感知、智能主体、认知底座、安全治理和典型应用场景等关键环节。对于关注智能体技术演进、企业级 Agent 架构与可信落地的研究者、开发者和产业实践者而言，具有很好的参考价值。谨此推荐。

PUBLICATION INFORMATION

编写说明

牵头编写单位	华为云
编写成员	姚婷 赵羽 王欣 彭潇 余佳容 郭良 尹青 胡斐然 李剑彪 丁来强
指导成员	王洪利 刘赫伟 黄毽 马会彬 杜恒 涂妍 赵英杰 沈彬 陈斌
版本信息	V1.0 · 2026 年 7 月
版权与使用	本文件用于技术交流与架构研讨，正式发布信息以最终版本为准。

适用范围与阅读对象

维度	说明
适用范围	面向 Coactive Agent 的产品规划、系统架构、工程实现、企业部署与可信治理。
目标读者	企业管理者、AI 产品与研发负责人、架构师、安全与合规团队，以及关注下一代人机协作的研究者。
使用建议	可按“行业问题—范式—架构—安全—应用场景”的顺序阅读，也可按专题直接进入相关章节。

本白皮书围绕 Coactive：Interactive × Proactive 双态协作展开，依次讨论行业洞察、愿景与顶层架构、智能主体、环境感知、认知底座、元进化、企业可信落地与应用场景，形成从协作范式到系统实现、价值验证和可信治理的完整论证。

CONTENTS

目录

引言

大模型让 Agent 成为 AI 进入真实工作场景的关键形态，但行业却常常将其未来误读为“自动化”：让 Agent 更主动、更自主、并尽可能减少人的参与。“自动化”固然能提升效率，却也会在复杂任务中放大目标误解、上下文缺失和行动时不可控等问题。真实任务不是目标明确、边界固定、环境稳定的封闭流程，它存在于一个目标、边界与环境持续变化的开放世界，推进任务依赖人与智能体不断进行判断、协商、确认、修正与经验沉淀。一旦将人排除在工作流之外，人的经验、智慧与灵感也就被排除在产物之外。

因此，AI Agent 的核心命题正在从“能否完成任务”转向“能否在开放世界中持续、可信、可控地与人协作”。下一代 Agent 的关键，不是把人移出工作流，而是让人的经验、判断和创造力持续进入工作流，并让成熟、可验证、可授权的任务片段由 Agent 可靠承接。人与工具之间从来不是简单的替代关系。工具的价值，在于扩展人的能力边界，而不是削弱人的主体作用。纸笔没有取代思想，而是让思想留存；乐器没有取代情感，而是让情感得以表达；AI 也应如此——它让人的灵感与智慧更容易被激发，并将人的知识、创造力与责任意识持续融入任务过程和最终成果。

基于这一判断，本白皮书提出 **Coactive Agent**：一种面向开放世界的智能体范式。Coactive 不是 Reactive（被动触发）到 Proactive （主动执行）的单线升级，而是 **Interactive × Proactive** 的协同耦合。Interactive 让人和 Agent 在同一工作现场中实时共处、共同思考、持续澄清、推演与修正，适用于目标尚未清晰、判断标准仍在形成、创造性与责任取舍较强的任务；Proactive 则让 Agent 在授权边界内主动准备、执行、验证和回退，适用于路径稳定、标准可验、边界明确、风险可控的成熟任务片段。二者在共享任务状态中相互增强、相互约束、动态切换。

为了支撑这一范式，本白皮书进一步提出**“环境感知 × 智能主体 × 认知底座”**的顶层架构，以及贯穿三层的元进化闭环。环境感知将聊天框之外的空间、应用、设备、操作和工具反馈组织为最小充分、低扰、可控、可追溯的任务现场；智能主体在 Interactive 与 Proactive 之间切换与并行，形成前台共创和后台执行的统一循环；认知底座将原始材料、用户偏好、流程经验、应用记忆和任务状态沉淀为可追溯、可修正、可复用的长期经验；元进化闭环则基于真实任务轨迹、失败反馈、评估结果和成本信号，持续优化环境感知、智能主体和认知底座的感知、交互、执行、记忆与授权策略。

这套体系的目标不是让 AI 替代人与世界的关系，而是让人从重复、低价值、可沉淀的事务中释放出来，回到创造、判断、沟通和决策等高价值环节。对企业而言，Coactive 也提供了一条更可信的部署路径：先协作，再沉淀；先可见，再授权；先实现低风险自动化，再逐步走向高价值主动承接。Agent的能力越强，就越需要把人的参与、权限边界、证据链、记忆治理与系统进化能力，内生到智能体的设计与运行范式之中。

第一章 解构 Agent 落地的结构性难题

BACKGROUND & INSIGHTS

AI Agent 正从概念验证迈向规模化商业落地。能力快速提升的同时，真实部署仍面临经验难进入工作流、轮次式交互不自然、记忆连续性不足，以及主动执行导致人的边缘化等结构性问题。本章结合市场趋势、能力演进与人机协作层级，对这些关键矛盾进行系统分析。

1.1 AI Agent 技术发展正在重塑生产力

AI Agent 技术正在经历从概念验证到规模化商业落地的关键转折期。2026 年 5 月 Gartner 最新预测，2028 年 33%的企业软件将内嵌 Agentic AI 能力，至少 15%的日常工作将由 AI 自主执行；每家 500 强企业将平均部署 150,000 个 AI Agent [1]。麦肯锡估计，Agentic AI 每年可创造 2.6-4.4 万亿美元价值，仅 Agentic Commerce 领域就将产生 3-5 万亿美元的交易 [2]。

这些数字释放出一个清晰信号：Agent 技术对生产力的重构已经开始。如图 1-1 所示，结合红杉资本的产业判断，以及 SpaceX 提交的 S-1 招股书所展现出的未来组织形态，可以看到 Agent 技术最具突破性的方向之一，正是“数字世界劳动力市场”[3]。AI 执行个体（AI Agent）正在逐渐成为未来生产力体系的重要组成部分。

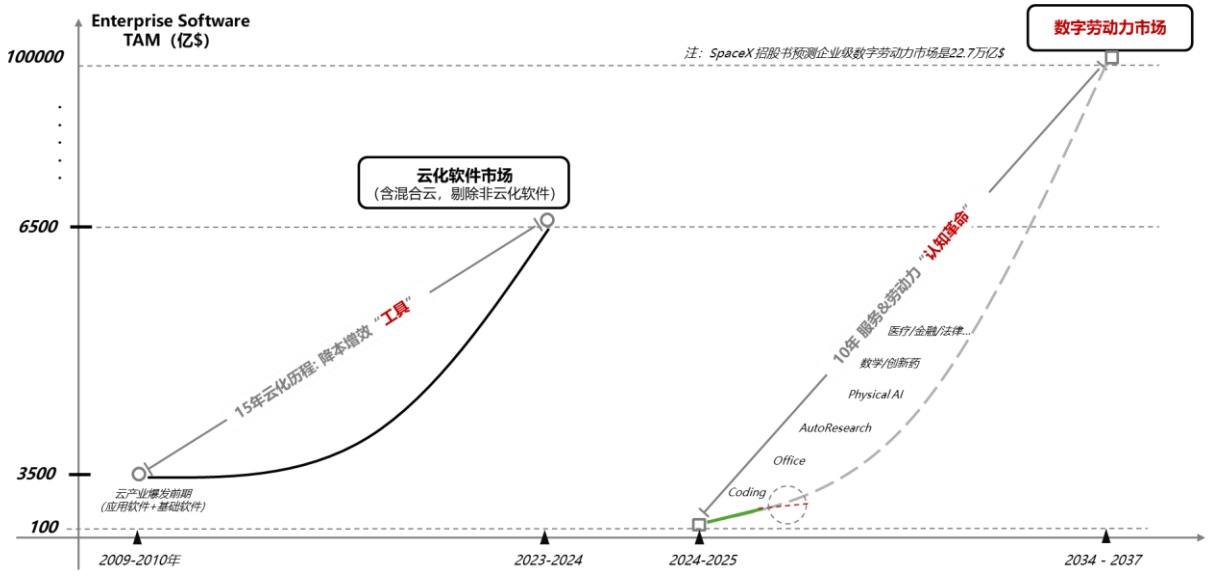


图 1-1 未来 10 年，Agent 将颠覆整个政企“数字劳动力市场”

AI Agent 技术生态正在快速发展。Claude Code、LangChain、OpenClaw、Hermes Agent 等技术框架和 Agent 应用，为构建复杂 Agent 系统提供了软件基础；与此同时，Prompt Engineering、Context Engineering、Harness Engineering，乃至 Loop / Environment Engineering 等以 Agent 为中心的新一代软件工程方法持续迭代，推动 Agent 的任务执行能力快速提升，并呈现出智能体“摩尔定律”式的发展趋势 [4]。

一方面，在经历了概念炒作的喧嚣后， AI Agent 的落地采用率正在逐渐告别高增长，进入理性回归的“平台期”。这并非意味着 Agent 技术的终结，而是市场从“为了 AI 而 AI”向“算总账、看实效”过渡的必然阵痛。Agent 技术概念快速更迭，但并没有解决记忆可靠性、被动工作模式、断代式接入和用户信任不足等问题，这些因素仍是制约 Agent 规模化部署的核心瓶颈。

另一方面，当前 Agent 的商业叙事经常把人及其工作岗位视为被替代的对象，把业务流全自动化视为最终方向。实际落地经验却表明，更有效的方式通常是评估员工与关键利益相关者对人工智能的接受程度及组织文化，选择与真实风险相匹配的自动化水平。现有的 Agent 系统虽然高效，但依然无法完整呈现人类独有的灵感、经验和创造力。这意味着，在技术奇点到来之前，完全由 AI 自动化决策驱动的方式必然面临信任与能力的双重瓶颈；相比之下，“人机协作”方式由于融合了人类的高阶认知，AI 的高效率特点，成为了当下更优的落地路径。

综上所述，当前 Agent 的落地形态仍处于探索期，而人机协作的 Agent 架构新范式将成为重要的变革力量。通过对人机协作层级进行建模，确立不同层级下人类与 Agent 的动态分工合作边界，是破解当前 AI 落地“非确定性”风险的关键。通过将人机交互抽象为“人类主导、AI 增强、系统自治”等分级架构，企业可以针对不同的业务场景，将复杂的业务流解构为标准化节点。在低容错场景引入人工卡点（Human-in-the-loop），在高频重复场景实施自动化托管。这种精细化的分工建模，不仅规范了 Agent 的调用权限和沙箱行为，更在组织内部确立了决策支持、增强到全自动化演进中的“责任防火墙”。

而典型的人机协作（Human-Agent Collaboration）分层模型，如图 1-2 所示：

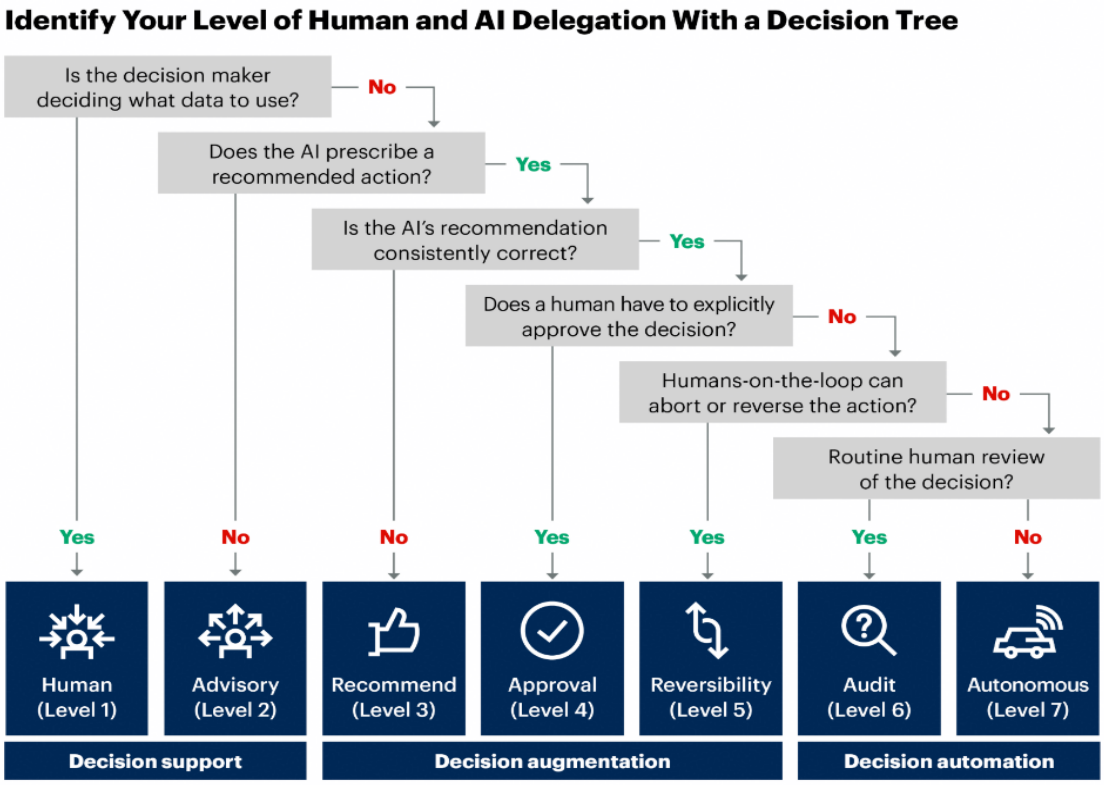


图 1-2 人机协作（Human-Agent Collaboration）的 7 个层级[5]

在 Gartner 给出的从人类独立执行，到 Agent 完全自主运行的分层模型中，可以看到人机协作（Human-Agent Collaboration）主要聚焦于 Level 2 - Level 6，其中 Agent 在这几层级（阶段）分别起到了“建议，推荐，确认，复盘，审计”等作用。这说明现阶段的人机协作，仍主要围绕“经验，判断，自主决策等过程”展开，依然是让 Agent 充当人类高级认知能力的“智囊”或“执行代理”。然而，将模型

抽象的“推荐与确认”转化为企业真实的生产力工具，绝非将 Agent 系统部署、运行起来那样简单。当深入复杂、高并发的行业真实项目时，会发现现有的 Agent 系统能力，还无法稳定支撑起“端到端”完整业务闭环。具体而言，当前的 Agent 系统在底层架构和交互机制上，依然暴露出以下五个直接阻碍其走向商业闭环的结构性难题。

1.2 现场断裂：Agent 仍被困在聊天框，难以进入真实任务环境

在 Agent 技术迈向实用化落地的进程中，单纯依赖大模型自主决策的“幻觉”开始破灭。我们逐渐意识到，现有的智能系统在面对高复杂、高模糊以及高动态的真实商业世界时，往往表现出一种“缺乏常识的精确”或“死板的逻辑正确”。这逼迫行业不得不重新审视人机协作的本质。在探讨当前大语言模型与智能体（Agent）落地的深水区时，一个核心的痛点日益凸显：人类的经验、判断和创新性难以持续进入工作流。尽管 Agent 系统在自动化和执行力上展现出巨大潜力，但在实际的复杂业务场景中，人类智慧与机器执行之间仍存在着难以逾越的鸿沟。这一困境主要体现在以下三个底层逻辑上：

Prompt/Context 模式难以承载行业直觉：人类专家的核心能力多为经验知识，依赖直觉与长期积累；而 Agent 主要依赖 Prompt / Context（文本向量）嵌入人类意图，难以承载行业直觉，导致人类的核心能力无法有效传授给 Agent。

Agent 单向工作模式导致实时判断力断层：目前的工作流通常是“人写好 Workflow / Harness / Loop → Agent 执行 → 产出结果”。在复杂、多阶段任务中，人类无法在 Agent 思考的“中间步骤”实时、动态地介入并修正其认知。缺少“双向对齐”机制，容易造成实时判断力在工作流中的断层。

人类非结构化创新与 Agent 高度结构化系统的矛盾：Agent 工作流高度结构化和逻辑化。当人类产生新的灵感或商业点子时，现有 Agent 系统很难立即、灵活地调整工具链和行动策略，去捕捉并放大这一创新。

1.3 协作断裂：轮次式交互让人的判断难以持续进入过程

Turn-Based 交互本质上延续了计算机控制台（Console）时代的命令行逻辑，违背了人类大脑的流式思考方式和团队协作习惯。这种“说完再想、答完再改”的模式难以形成流畅的人机协作，并在天然协作链路中暴露出三个明显的不自然：

中断式等待：当用户产生新想法时，必须等待模型输出完上一段内容，或者强行中断。等待会破坏人的专注流（Flow State），当再次轮到用户发言时，最初的灵感往往已经打折。

反馈滞后：用户发出指令后，模型在黑盒中连续思考并一次性给出结果。如果模型在早期步骤就理解偏离，仍可能继续完成后续工作，造成沟通成本与计算资源的双重浪费。

缺少默契：人类真正的协作是“边看、边聊、边动手”。Turn-Based 模式迫使人把模糊直觉翻译成极度精确的文字指令，这种生硬转换会削弱经验和直觉中那些微妙、难以言传的部分。

1.4 行动断裂：主动执行缺少授权边界、过程可见与可接管机制

前沿技术已经部分实现 Proactive Agent（主动式智能体）：系统无需用户逐步发号施令，也能主动监控数据、发现问题、规划路径并执行任务。虽然这种从“被动响应”到“主动包办”的跨越显著提高了自动化效率，却也带来更隐蔽的风险——人的边缘化（Human Excluded）。把人排除在任务完成过程之外的 Proactive 设计，主要暴露出以下三个深层问题：

产生“过程黑盒”：当 Agent 自动分析数据、写好报告，甚至直接向客户发送邮件时，人类往往只看到“已完成”的通知，无法了解执行中发生了哪些微调、妥协或逻辑跃迁。结果一旦偏离，人类需要花费数倍精力复盘，甚至可能面对无法挽回的后果。

被动同意：人的工作退化为机械地查看报告、点击“同意”，不仅削弱判断参与，也可能造成职业价值感下降和精神内耗。

人类技能与经验退化：当基础岗位中的实践环节被过早全部自动化，人类的实操技能会逐渐退化；新手也失去了通过试错积累经验知识和直觉的机会。

1.5 认知断裂：记忆停留在后台检索，难以形成长期协作连续性

当前很多 Agent 记忆系统，本质上只是把历史记录打包存入 Vector DB（向量数据库）。只有等用户发问后，系统才在后台检索并拼接到回复中。这种“被动检索”机制使 Agent 难以像真正的助理一样持续理解人类：

缺乏实时交互：大多数 Agent 只有在一次会话彻底结束（Session Close）后，才通过独立 LLM 总结对话并提取关键信息，只能事后沉淀；系统无法根据人类刚说的一句话，实时更新对用户、任务和关系状态的认知。

缺乏逻辑延展与主动唤醒：当前跨会话（Cross-Session）记忆往往只是无差别拉取历史记录。Agent 无法像人脑一样，把前天讨论的商业点子、昨天阅读的行业报告和今天出现的新灵感组织成网状关联，也难以在当前任务中主动提醒用户存在历史冲突或遗漏。

缺乏跨应用连续性：文档、会议、邮件、浏览器和任务系统中的上下文通常被锁在不同产品或会话内。Agent 无法围绕同一目标维护连续的对象、状态与证据，导致每次切换应用都需要用户重新解释背景。

1.6 成长断裂：经验难以从一次协作沉淀为可复用、可治理的能力

当前很多 Agent 系统仍停留在单次任务模式：任务结束后，系统通常只留下结果文件、聊天记录或工具日志，却难以把目标澄清、上下文选择、工具路径、质量判断、用户偏好和失败教训，转化为下一次可继承的能力。这一问题主要体现在三个层面：

轨迹难以转化为洞察：真实协作会持续产生模型调用、工具调用、上下文装配、状态转移、用户打断和结果采纳等过程信号。但如果这些信号只停留在日志层，系统就难以判断任务从哪里开始偏航、哪些上下文真正有效、哪些工具调用带来了收益。结果是成功经验无法复用，失败教训也无法沉淀。

反馈难以形成改进依据：用户的一次修改、拒绝或采纳，往往包含新的偏好、质量标准或风险边界；一次失败任务，也可能暴露新的评估缺口。若这些反馈不能进入评估样本、验证器或回归测试，系统就无法判断哪些改造值得保留，哪些变化应该回滚。

改进难以成为受控进化：Agent 的能力不只取决于模型，也取决于 Prompt、工具、记忆、验证器、停止条件、Token 策略和主动触发规则等外部结构。如果这些结构只能依赖人工经验零散调优，而不能经过诊断、变异、评估、选择和沉淀，Agent 的成长就难以规模化。

更进一步，成长断裂的深层难点在于：进化机制本身也可能停滞。Agent 不只是需要被优化，还需要不断改进“如何发现问题、如何评价好坏、如何生成改造、如何选择保留、如何沉淀复用”这套机制。固定的观测指标可能无法解释新的失败模式，静态评估集可能覆盖不了真实任务变化，人工设计的改造策略也可能逐渐失效。如果驱动 Agent 成长的机制不能随着任务、工具、用户和组织规则一起演进，系统就只能在旧框架内重复优化，难以真正走向元进化。

1.7 本章小结：下一代 Agent 需要 Coactive 闭环

AI Agent 的核心矛盾已从“技术可行”转向“人机协同”。经验无法融入、交互断层、记忆割裂以及人类边缘化等现状表明：下一代 Agent 的进化方向绝非盲目追求更高的全自动化，而应致力于构建更自然、更具启发性的协同生态。必须将人类的行业经验、核心判断与创造性价值，深度锚定在高价值的业务链条中，同时完美融合 Agent 的智能与自主执行力。唯有让这种“人机智能交融”扎根于透明、可控且可追溯的信任基石之上，才能真正开启规模化落地的新纪元。

CHAPTER TAKEAWAY

下一代 Agent 不追求孤立自动化，而重在人机协同共创。

第二章 Overall：愿景、范式与顶层架构

VISION, PARADIGM & ARCHITECTURE

本章回答三个问题：我们相信怎样的 Agent 未来，Coactive 与行业主流范式有何不同，以及系统如何兑现这一判断。核心判断是：下一代 Agent 不是从 Reactive 到 Proactive 的单线升级，而是 Coactive——Interactive 与 Proactive 的协同范式，并由环境感知、智能主体、认知底座与元进化闭环共同支撑。

2.1 从自动化到共处：我们对下一代 Agent 的判断

随着大模型具备语言理解、工具调用、任务规划和多步执行能力，Agent 开始从“被操作的软件”转向“能替人推进任务的系统”。 这代表了 AI 能力的一次重要跃迁：AI 不再只回答问题，而开始进入任务本身。行业对 Agent 的典型定义，围绕“推理、规划、使用工具、保持记忆、执行多步任务”展开。NVIDIA 将 Autonomous AI Agents 描述为能够基于目标进行推理、规划和执行多步任务的 AI 系统，并强调它们以最少的人工输入执行任务[6]。AI 实验室普遍将“自主完成长任务的能力”视为模型和 Agent 最重要的能力[7]。Loop engineering[8]则是当前 Agent 自动化的一种工程实现，人不再逐条输入 prompt，而是设计一个由目标、工具、反馈和验证构成的循环，让 Agent 在循环中持续行动、观察和修正，直到任务完成。

然而，如果我们把 Agent 的未来简单理解为“越自动越好”，就会忽略一个更根本的问题：真实工作并不是一串可以一次性完整表达的指令。复杂任务往往伴随着目标澄清、判断调整、信息补充、审美选择、风险权衡和临场决策。人在这些环节中的价值，不只是发起任务，更是持续提供上下文、经验、偏好和判断。一旦人被自动化工作流排除，人的智慧、灵感、思考、顿悟也就无法在工作产物中承载。

因此，我们认为，下一代 Agent 的核心不应只是更强的自动化，更应是更高质量的人机共处。所谓共处，不是让 AI 永远等待人发号施令，人等待 AI 的响应；也不是让 AI 完全脱离人独立自主行动，而是让人和 Agent 能够处在同一个动态上下文中：人可以表达、停顿、修改、打断、确认和授权；Agent 可以观察、理解、补充、执行、反馈和沉淀。二者之间不是一次 prompt 与一次 response 的关系，而是一段持续演进的协作关系。想象你我展开一场对谈，当我开始表达，你的脑海中同步浮现出观点、情绪和策略；并非我表达完你才开始思考，而是我表达时你已在思考，我的语言、神色、手势、语气等等都成为了你判断因素；当我停止表达，你开始陈述思考的内容。如此往复，我们的对谈，形成了一种思维的流动。交互越丰富、越即时、越双向，理解就越深，协作的质量也越高。人与人如此，人与 Agent 也是如此。

这种判断也与人本 AI 的长期研究方向一致。Human-Centered AI 强调，高水平自动化并不必然意味着低水平人类控制；理想系统应同时提升自动化能力和人的控制力、创造力与责任感[7]。对企业级 Agent 来说，这一点尤其关键：Agent 越能做事，就越需要让用户保留理解、介入、修正和追责的能力。NIST AI Risk Management Framework 也将透明、可解释、可问责、隐私保护和安全可靠列为可信 AI 的重要特征[9]。

所以，本白皮书对下一代 Agent 的基本判断是：

> **Agent 的目标不是把人从工作流中移除，而是把人从重复、低价值、可沉淀的事务中解放出来，让人更稳定地留在创造、判断、沟通和决策等高价值环节。**

在这个判断下，Agent 的演进方向不是从“人操作软件”直接跳到“AI 替人完成一切”，而是形成一种新的协作关系：人在前台与 Agent 共同思考，Agent 在后台持续执行；人在创造性环节保持主导，Agent 将稳定经验沉淀为可复用能力。下一代 Agent 的价值，不只是完成任务，而是建立一段越用越懂、越协作越有能力的长期关系。

2.2 愿景与范式：Coactive = Interactive × Proactive

我们对下一代 Agent 关系的愿景是：人始终是主体，AI 像影子一样自然进入工作现场，连接文档、会议、表格、设备与团队，理解任务语境、人的节奏与判断边界；它在需要时及时帮助，在不需要时安静工作，并在长期协作中沉淀偏好、标准、流程和技能。这样的 Agent 不替代人与世界的关系，而是把人的智慧持续带入工作流，让每一次协作都更自然、更有效，也让可复用经验逐步成长为稳定能力。

当前行业对 Agent 的讨论，正在从 Reactive 走向 Proactive，如图 2-1 所示。Reactive Agent 等待用户指令，再根据指令完成任务；Proactive Agent 则试图根据环境、历史和目标，提前判断用户可能需要什么，并主动发起帮助。相关研究已经明确提出，LLM-based agents 不应只停留在被动响应模式，而应能够在没有显式指令的情况下预测和发起任务。Proactive Agent 一文提出了 Proactive Bench，并通过训练 reward model 来模拟主动帮助对人类是否合适的判断[10]。

我们认同 Proactive 是 Agent 的重要能力，但不认为 Agent 的未来可以被简单定义为 Proactive。原因在于，主动性只有在任务边界清楚、判断标准稳定、用户偏好已被理解、风险可控的情况下，才会真正产生价值。否则，Agent 很容易引入研究者已经指出的两类问题：false alarm，用户并不需要时 Agent 仍主动介入；early assistance，意图尚未明确时 Agent 过早出手[10]。过早的主动可能变成打扰，过度的自主可能带来误判。

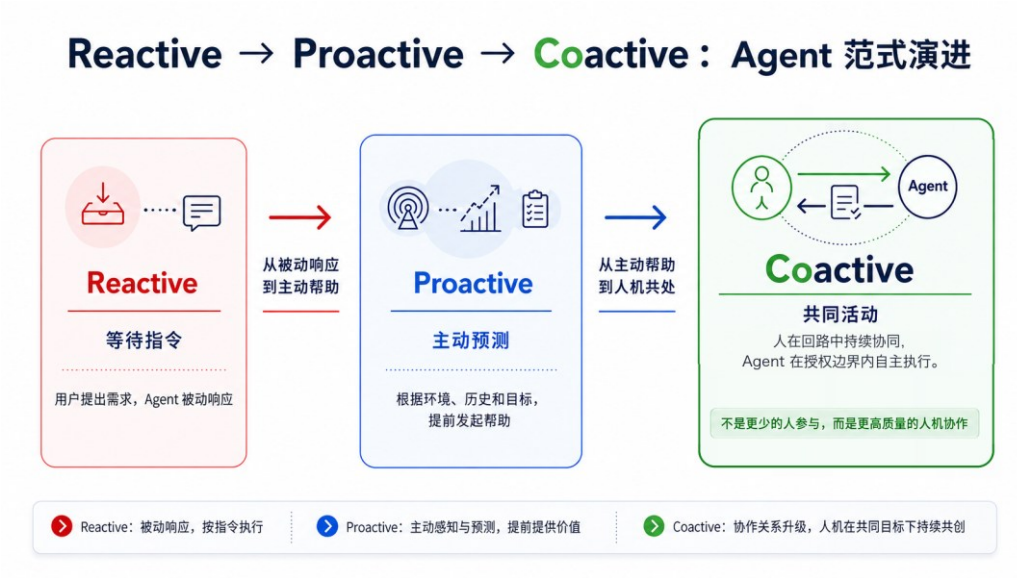


图 2-1 Agent 演进路径：Reactive -> Proactive -> Coactive

真实工作中，越重要的任务往往越不适合一开始就完全交给 Agent。写一份战略报告、设计一套产品方案、判断一个商业机会、形成一个复杂项目计划，这些任务并不是“把目标输入清楚，然后等待结果”就能完成。它们需要人与 Agent 像人与人一样协作，形成前文所说的那种"思维的流动"。

Thinking Machines Lab 对 Interaction Models 的论述，也指向同一类问题：当 AI 越来越强时，用户不应因为界面限制而被挤出工作流。AI 协作需要支持实时、多模态、双向、可打断的交互，让人能够持续澄清、反馈和参与[11]。换句话说，智能的扩展不应只体现在“AI 自己做更多事”，还应体现在“AI 更好地与人一起做事”。

因此，我们提出 **Coactive** 双态协同范式。

> **Coactive = Interactive × Proactive**

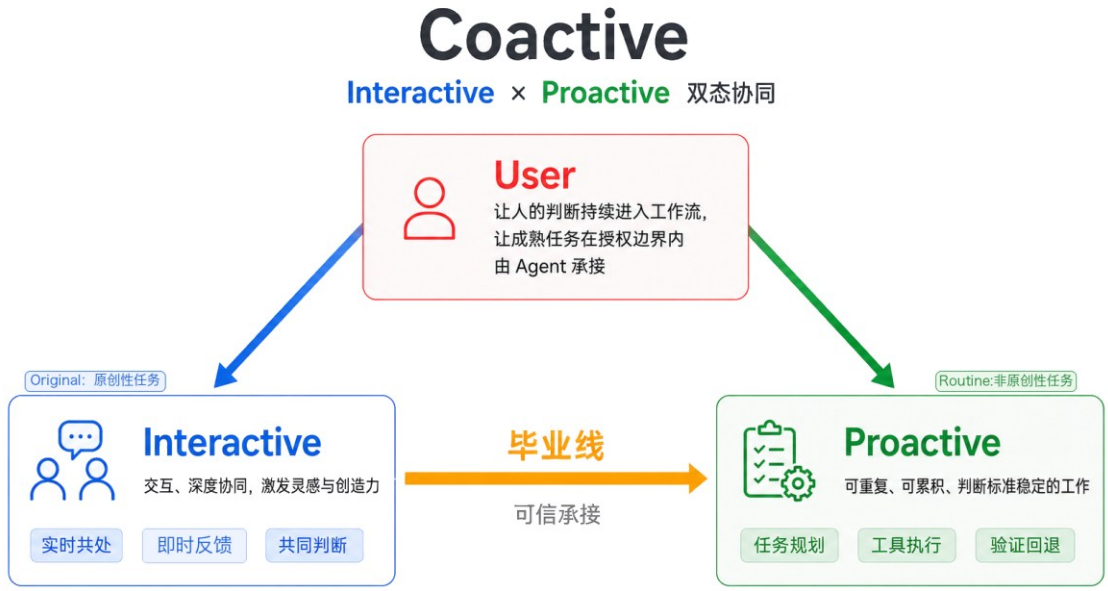


图 2-2 Coactive：Interactive × Proactive 双态协同

Interactive x Proactive 代表 Interactive 与 Proactive 的协同耦合，如图 2-2 所示，Coactive 范式包含两种相互协作的状态：

第一种是 Interactive。它不是聊天交互的延伸，而是一种面向任务现场的实时协作机制。Agent 通过多模态环境感知、共享任务状态和低延迟反馈，持续接收用户表达、编辑、浏览、停顿、修正与授权等信号，并将这些信号在线写入目标、约束、假设、开放问题和下一步候选动作。它适用于目标尚未完全明确、判断标准仍在形成、结果依赖经验、审美和创造力的任务。Agent 的价值不是替人“想完”，而是在推理和行动循环中承接人的判断，使人的思考过程被实时放大。

交互和协作形态会直接影响理解深度。邮件交互可以留下清晰的文字，但节奏慢，许多隐含背景需要反复补充；电话沟通让语气、停顿和追问进入过程，理解变得更连续；面对面讨论则进一步把表情、

手势、白板、现场材料和即时反馈纳入共同思考。Interactive 要表达的正是这种差异：交互越即时、越双向、越接近共同在场，Agent 就越能理解人的意图、判断和未说出口的约束，人也越容易把 Agent 当作可以一起推敲问题的协作者，而不只是一个等待指令的问答入口。

第二种是 Proactive。它不是无条件的主动包办，而是一种受证据、授权和治理边界约束的主动执行机制。Agent 基于已沉淀的流程、偏好、质量标准、工具路径、风险规则和上下文证据，在限定范围内持续进行任务规划、工具调用、状态跟进、结果验证和异常恢复。它适用于可重复、可累积、判断标准相对稳定、风险可控的工作，例如信息整理、状态跟进和例行交付。

Proactive 的关键不是让 Agent 预判一切，也不是追求“越主动越好”，而是把已经稳定的工作从人的持续注意力中转移出来，并保持可暂停、可恢复、可解释、可验证和可审计。任务越清晰、路径越成熟、标准越可验证、风险边界越明确，Agent 就越适合在后台承接更多执行任务；一旦目标变化、证据不足、风险升高或授权不清，系统应主动降级，回到 Interactive 重新形成判断。

一个任务从 Interactive 状态迁移到 Proactive 状态的过程称为“毕业”，触发这一迁移所需达到的最低可信标准称为**“毕业线”**。“毕业线”用于判断：在特定任务、特定环境和特定授权范围内，Agent 是否已经具备相对独立推进任务的能力。“毕业”是 Interactive 转向 Proactive 的证据驱动迁移机制：当一类任务在 Interactive 中被多次协作、修正和验证后，系统识别其中稳定的能力片段，包括常用材料来源、判断标准、输出格式、审批规则、用户偏好和质量边界，并将其沉淀为可复用的流程、技能、记忆和主动策略。任务一旦满足毕业线条件，就从“需要人与 Agent 共同探索”转向“Agent 主动承接，人保留监督与介入权”。任务该走哪条路，由用户分流；何时从 Interactive 进入 Proactive，由真实协作中积累的证据决定。

这就是 Coactive 的核心：

> **原创性任务 Interactive 共创，非原创性任务 Proactive 自主承接，两者经毕业线实现可信迁移。**

这里的“原创性”不是狭义的艺术创作，而是指目标、判断、策略、审美和价值选择尚未稳定的工作；“非原创性”也不是低价值，而是指已经可流程化、可复用、可衡量、可授权的工作。前者需要人留在回路，后者应该尽可能下放给 Agent。这样，人不是被 Agent 替代，而是从重复性负担中被释放出来，回到更需要人类经验和创造力的位置。我们将其概括为：人负责创造和判断，Agent 负责沉淀和执行。

2.3 顶层架构：环境感知 × 智能主体 × 认知底座与元进化闭环

为了支撑 Coactive Agent，我们将下一代 Agent 系统抽象为三层顶层架构：环境感知、智能主体、认知底座，以及贯穿三层的元进化闭环，如图 2-3 所示。

第一层是**环境感知**。它负责把人的任务现场转化为 Agent 可理解的上下文。过去，Agent 主要通过聊天框获得用户输入；未来，Agent 需要理解更丰富的现场信号，包括用户正在看的屏幕、正在编辑的

文档、正在参与的会议、正在操作的应用、正在输入的文本、正在说出的语音，以及可能体现注意力和情绪变化的设备信号。这些信号可以来自生活与工作中的多类入口，包括耳机、麦克风、摄像头、鼠标、键盘和屏幕等设备，也包括浏览器、办公软件、IDE、会议系统和终端窗口等应用环境。环境感知的目标不是收集更多数据，而是让 Agent 与人面对同一个现场，理解任务正在何处发生、对象如何变化，以及何时应该观察、回应或行动。

图 2-3 环境感知 x 智能主体 x 认知底座与元进化闭环

第二层是**智能主体**。智能主体依照 Coactive 范式具备 Interactive 和 Proactive 两种协作状态。Interactive 模式让主体在前台保持实时共处，负责上下文承接、澄清追问、即时反馈和共同思考，使人的经验、创造力和判断能够持续进入任务过程；Proactive 模式让主体在授权边界内推进后台任务，负责长程任务、工具调用、跨应用操作、动态 workflow 和可授权的主动执行，把稳定、重复、可验证的部分从人的注意力负担中释放出来。同一个主体中两种协作状态可以切换，当任务仍需共同判断时，主体保持交互在场；当任务已经清晰、稳定、可授权时，主体转入主动推进。Thinking Machines Lab 提出的 interaction model 与 background model 分工[11]，也可以理解为这种模式承载方式：前台保持实时在场，后台承接更深推理、工具调用和长期任务，共同服务同一个 Coactive 主体。

第三层是**认知底座**。认知底座不应被理解为知识库或检索系统的增强版，而是支撑 Agent 长期协作的认知运行环境。它一方面回应 Interactive：通过交互式记忆，在用户表达、编辑和反馈的过程中同步准备相关上下文，让 Agent 能更自然地承接人的历史判断、项目背景和偏好，而不是等一次输入结束后再从零检索；另一方面回应 Proactive：把反复验证过的判断标准、任务状态、稳定流程和授权边界沉淀整理为可追溯、可复用、可修正的长期经验，使 Agent 的主动执行有依据、有边界、可治理。一个高质量认知底座首先要保存可信的原始依据，再在此基础上形成可调用的经验、偏好、流程和判断边界；所有派生理解都应能够回到来源，接受验证、纠正和治理。这样，Agent 才不只是每次重新回答问题，而是在一次次协作中保持认知连续性，并逐步形成稳定、可信、可进化的长期能力。

在三层架构之外，还需要一个持续运行的**元进化闭环**。环境感知让系统看见真实现场，智能主体在现场中交互和执行，认知底座沉淀可追溯、可调用、可修正的协作经验；元进化则根据真实协作中的结果、反馈、失败、修正和成本变化，持续优化 Agent 的感知策略、交互策略、执行策略、记忆策略和主动触发机制。

由此，Coactive Agent 不只是一次性响应请求的工具，而是一种长期成长的协作关系。它的目标不是最大化自动化，而是把最美好的交互方式带进每个人和 Agent 的关系中：让人的创造力、判断力和责任感被尊重、被承载、被放大，让 Agent 在稳定、重复、可授权的事务中可靠承接，使人与 AI 在一次次共同工作中越协作越默契，越沉淀越有能力。

2.4 本章小结

Coactive 的核心判断是：下一代 Agent 不是从 Reactive 到 Proactive 的单线升级，而是 Interactive 与 Proactive 的协同范式。环境感知让 Agent 进入真实现场，智能主体组织前台共创与后台执行，认知底座保持长期连续，元进化闭环让协作方式在真实使用中持续优化。

> CHAPTER TAKEAWAY
>
> **下一代 Agent 不是更自动，而是更好地与人共处。**

第三章 智能主体：Coactive 的双态协作机制

INTELLIGENT SUBJECT

本章介绍 Coactive 的智能主体设计：采用双态协作机制，Interactive 承接前台的实时共处与共同判断，Proactive 承接成熟任务的后台主动执行，毕业线是 Interactive 与 Proactive 之间的迁移标准。两种运行状态围绕同一共享任务现场运行，使目标、对象和授权在前台共创与后台执行之间保持连续。

3.1 Coactive 视角下的智能主体能力需求

Agent 的主流形态长期以 Reactive 为主：用户给出指令，系统完成一次响应。Reactive 解决“响应”：用户给出背景、目标和约束，Agent 在当前上下文中生成结果。它适合轻量问答和边界明确的信息处理，但任务状态、对象变化、结果检查和下一步触发仍主要由用户维护，因此只能提升单次效率，难以支撑连续任务。

为突破单轮响应，行业开始将 Agent 设计为围绕目标、工具、状态、验证、记忆和停止条件持续运行的长程循环，推动工程实践从 Prompt Engineering 走向 Loop Engineering[8]。Proactive 解决“推进”：Agent 不再等待用户逐轮提示，而是在既定目标和约束下持续行动，适合输入明确、过程可拆解、结果可验证、失败可回滚的任务。面向开放世界的真实任务，不同于封闭执行。复杂任务的目标、成功标准和风险边界往往在过程中逐步形成，需要人与 Agent 持续讨论、比较和取舍。若 Agent 过早进入主动执行，容易带来过程不可见、纠偏困难和责任边界不清等问题。

Coactive 解决“协作”。Horvitz 在 Mixed-Initiative 研究中指出，好的人机系统需要在自动化服务与人的直接操作之间平滑配置主动权[12]。Coactive 把这一原则放进开放任务的持续运行中，将智能主体设计为双态协作机制。Interactive 承接需要人参与判断的开放任务，Proactive 承接证据充分、边界清晰、可自主推进的成熟能力片段。两种状态围绕同一任务现场运行，并通过“可信承接”和“协作唤醒”管理两种状态的迁移：条件成熟时进入自主推进状态，目标变化、证据不足或风险升高时回到协作状态。

3.2 Coactive 智能主体范式：Interactive 和 Proactive 的双态协同

Coactive 由 Interactive 与 Proactive 两种运行状态构成，并通过“毕业线”管理二者之间的能力迁移，这个过程如图 3-1 所示。Interactive 面向目标尚未稳定的阶段[13]，强调实时共处与共创：Agent 与人处于同一任务现场，持续承接用户正在操作的对象、正在表达的偏好、不断修正的意图以及需要确认的判断，并将这些信息写入过程状态，直到正确目标、判断标准与执行路径逐步清晰。Proactive 面向成熟阶段，负责在授权边界内规划、执行和验证：当一类任务经过反复验证，形成稳定的质量标准和执行路径后，其中可独立验证的能力片段便可沉淀为后台执行；它继承 Interactive 中形成的目标、约束和授权边界，调用工具、Agent App 与动态 Workflow 推进任务，并将结果与证据带回同一任务现场。

毕业线管理 Interactive 与 Proactive 之间的双向迁移。一方面，“可信承接”判断能力片段何时具备进入受控主动承接的条件，使任务能够从共同探索、静默准备、草稿执行、确认后执行逐步走向条件自主；另一方面，“协作唤醒”判断后台执行何时必须将判断权交回前台——无论是因为目标漂移、证据不足，还是风险升高。由此，Coactive 的能力迁移不再是单向自动化，而是覆盖升级、执行、验证与回流的生命周期治理。

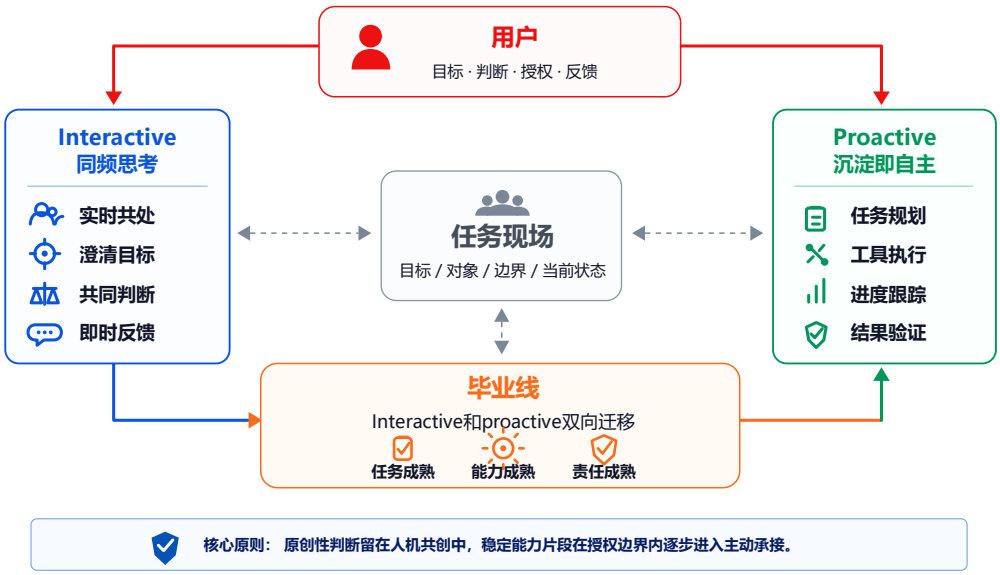


图 3-1 Coactive 的统一运行范式：Interactive、Proactive 与毕业线

Coactive 的机制突破不只是让 Agent 更实时或更主动，而是让 Agent 根据任务是否仍需人的判断，在 Interactive 与 Proactive 之间有依据地迁移，并保持任务连续。其关键机制如表 3-1 所示。

表 3-1 Coactive Agent 的关键机制突破

关键机制突破	机制说明	关键技术	Agent 能力
人与 Agent 从消息交互到共享同一任务现场	人与 Agent 不再只交换离散消息，而是自动围绕当前目标、对象、约束、进展和证据持续协作	链接对象及协议（3.3.2）、实时交互运行时（3.3.3）	Agent 能够理解任务正在何处发生、哪些对象和信息发生变化，并减少用户反复解释背景
从单轮响应到 Interactive x Proactive 双态协同	Interactive 负责实时共处与共同判断，Proactive 负责成熟任务的后台推进，两种状态围绕同一任务现场运行	前台交互模型与后台深度模型、可打断控制（3.3.3）	Agent 能够在人需要参与时保持在场，也能在任务成熟后承接执行，并保持目标和进度连续
从单次 Agent 调用到 Agent 可持续的主动执行	成熟任务片段经过沉淀，能够跨阶段、跨应用和跨会话持续推进	Agent App Runtime（3.4.2）、Proactive Workflow（3.4.3）、主动任务挖掘（3.4.4）、主动介入分级（3.4.5）	Agent 能够持续维护任务状态，在授权边界内规划、执行、验证、暂停和恢复
从单向自动化到协作唤醒的能力迁移	任务条件变化时把判断权交还用户	过程状态（3.5.1）、三域成熟度与可信承接（3.5.2）、协作唤醒（3.5.3）	Agent 能够说明为何承接、执行到哪里、为何停下，并根据证据扩大、收缩或撤销承接范围

任务现场是前台与后台共同面对的任务事实、行动边界和结果证据，上述机制围绕同一个任务现场运行。模型可以切换，Agent 可以被委派，工具可以分布运行，但它们围绕同一任务现场协同：哪些目标已被确认，当前允许执行什么，哪些动作需要授权，何时应从 Proactive 回到 Interactive。

表 3-2 能力形态与转换策略的职责

能力 / 机制	核心职责	主要产物	切换或调整条件
Interactive	承接现场、澄清目标、组织材料、比较方案、形成共同判断。	明确目标、共享约束、开放问题、判断标准和人的最新取舍。	局部目标与标准趋于稳定，或出现需要后台验证和准备的工作。
Proactive	围绕目标规划、行动、验证、汇报，并维护跨时间任务进展。	真实任务产物、阶段进展、异常、风险和可验证执行证据。	目标变化、证据不足、风险升高、用户介入或任务阶段完成。
毕业线	判断能力片段能否从 Interactive 迁移到 Proactive，并支持协作唤醒、降级和回流。	能力状态、适用范围、触发条件、评估基线、授权边界与回流条件。	成熟度上升则扩展承接；环境、质量、风险或信任变化则收缩。

两种运行状态既可以顺序切换，也可以前后台并行。如表 3-2 人在前台讨论方向时，Agent 可以在后台检索材料或分析数据；后台产生关键发现后，再将证据带回前台。实时交互模型使前台持续接收音频、文本和操作事件，并在时间中实时理解和回应；Coactive 将这种实时性扩展到任务级协同，使前台保持关系和认知连续，后台承接不必占用用户持续注意力。无外部影响的检索和材料准备可以静默进行，改变任务对象的编辑应保留版本和可见状态，对外发送、权限变化或不可逆动作必须进入确认[9]。系统需要说明为何转入后台承接，也需要说明为何回到 Interactive。

3.3 Interactive：面向共同判断的实时协作

Interactive 是 Coactive 的前台协作状态，面向目标还需讨论、判断标准不清晰、需要人类经验和创造性参与的开放任务。它让人和 Agent 面对同一任务现场，在连续时间中共同形成问题、组织材料、比较方案，并把人的判断实时整理成后续可能的任务。

3.3.1 连续共处：从轮次对话到共同判断

传统对话式 AI 以回合为基本单位：用户完成输入，系统完成推理与输出，用户再在下一轮修正。这一结构适合边界清晰的问答，却会把真实协作中连续发生的信号压缩为离散消息。用户表达过程中的补充、中途修正和对象切换只能在下一轮重新描述，模型也容易沿着已形成的假设继续推进。Coactive 希望能改变这一时间结构，核心思路在于 Agent 能够在用户表达尚未完成时开始理解，在意图逐渐稳定时同步准备，在方向变化时及时调整，并在需要深度分析时把工作交给后台而不退出现场。Interactive 的目标，是让人和 Agent 在同一任务现场中持续形成共同判断。

Interactive 的整体设计包含四个层面：

1）连续输入与现场承接，主体持续接收语音、文本、屏幕操作和应用事件，并整理为与当前任务相关的状态增量。2）链接对象与多应用嵌入。文档、表格、会议、网页和代码通过链接对象协议接入，使人和 Agent 面对同一对象和同一任务现场。3）实时交互模型。以低延迟实现实时的交互，并在静默跟随、轻量提示、澄清追问、请求确认和后台委派之间切换。4）后台深度模型与工具执行。长检索、复杂推理和方案生成可以在后台并行执行。

这个架构让 Interactive 和用户在同一任务现场同频思考。用户不必反复说明刚才在做什么、材料为何重要、上一版为何被否定。系统根据用户当前打开的文档、选中的段落、正在浏览的网页和会议中刚确认的结论，持续更新任务现场信息。Interactive 会识别用户所处的协作阶段，并有不同的交互重点。探索阶段，用户表达不完整、对象频繁切换，Agent 帮助整理问题和补充候选路径，实现认知反馈。编辑阶段，用户操作集中在具体对象上，Agent 贴近段落或代码块给出局部建议，并保证修改可撤销。核验阶段，Agent 关注证据来源、版本和数据冲突。确认阶段，Agent 说明拟执行动作、影响范围和回退方式。委派阶段，通过毕业线把结论交给 Proactive，但前台仍保留当前对象和用户最新判断可持续交互。

3.3.2 任务现场同频：链接对象与多应用嵌入

链接对象与多应用嵌入解决如何实现任务现场同步的问题。它把用户正在处理的文档、表格、会议和代码组织为可寻址、可订阅的工作对象，使实时交互模型面对的是带对象、状态和权限的任务现场，而不仅是语音和文字。具体的从环境感知到任务现场的实现可以参考第四章，本节重点在这些信息在 Interactive 中的使用。

真实工作天然分布在多个界面中：用户在浏览器里查资料，在 Word 中组织观点，在 Excel 中核对数据，在会议里形成新结论，在 IDE 中处理代码。传统软件将这些视为不同应用中的操作片段；Coactive Agent 需要把它们组织为同一任务现场中的连续信号。支撑多应用交互的关键，是链接对象及其接入协议。链接对象区别于普通 URL 和文档复制，它是 Coactive 任务现场中可寻址、可关联、可操作的工作对象。文档、网页、表格区域、会议片段和代码位置，都通过统一协议进入和退出任务现场。

以客户季度复盘为例，一个场景走完全程：

1. 用户打开 Excel 中的客户数据表格。Excel 插件向平台**注册**这个对象：声明自己是什么（客户数据工作表）、属于谁（当前用户/工作空间）、在哪个场景下被打开（季度复盘任务）。
2. 注册完成后，Excel 插件提交**功能手册**。手册声明两类信息：它可被调用的操作（读取指定区域、写入单元格、筛选数据——各带参数、成本和风险级别）；以及它可被订阅的信号（数据变化、选区切换、表格关闭——各带频率特征和语义级别）。
3. Agent 在任务现场中通过**对象发现**看到这张表格上线了。它接着**查询手册**，了解到可以读取表格数据、可以订阅数据变化信号。

4. Agent 决定**订阅**"数据变化"信号，这意味着表格数据每次更新都会实时并入当前 Interactive 上下文，Agent 能感知用户正在编辑什么。
5. 用户在表格中更新了一个客户的续约状态。Excel 插件按手册声明的信号**发布**事件："B12 单元格从'待确认'改为'已续约'"。
6. Agent 收到这个变化，判断它与当前复盘报告相关。它通过**功能调用**向 Excel 发出操作请求："读取 A1:F20 区域的完整数据"，用于更新报告中的客户状态汇总。
7. 当用户关闭这份表格，Excel 插件**注销**该对象，退出任务现场。Agent 知道这个数据源暂时不可用了。

在线期间，Excel 插件通过心跳维持活跃状态——如果心跳中断，平台知道对象可能已经崩溃或断线，不再向它发送调用。该场景描述的十个协议原语归纳如表 3-3 和表 3-4 所示 。

表 3-3 链接对象接入面五原语

原语	作用
注册 register	声明身份、归属和场景，进入任务现场。
功能手册 manifest	声明可被调用的操作，以及可被订阅的信号。
心跳 heartbeat	维持在线状态和健康度。
事件发布 publish	按手册声明的信号主动推送变化。
注销 deregister	退出任务现场。

表 3-4 Agent 应用面五原语

原语	作用
对象发现 discover	查找当前在线对象及其状态。
手册查询 query_manifest	了解目标对象的可用操作和信号。
功能调用 invoke	向对象执行操作，即 Agent → 对象。
信号订阅 subscribe	声明关注哪些事件，并指定消费语义。
退订 unsubscribe	释放不再关注的信号。

这套协议的关键在于两条方向相反的通路。功能调用从 Agent 流向对象，用于执行操作和改变状态；信号订阅从对象流向 Agent，让任务现场中的变化主动流入判断过程。订阅的消费语义分为两类：1）将信号并入实时交互上下文，用于 Interactive 的低延迟共处；2）用于触发后台任务重新评估，支撑 Proactive 的长程执行和毕业线中的协作唤醒。

链接对象使 Agent 能理解"用户现在打开的网页是报告证据""会议里刚形成的这句话是新约束"、"Excel 中这个区域是月报数据源"。这些对象共同构成任务现场，Interactive 的上下文由此成为持续更新的共同任务现场。

工程实现来自两类入口。第一类是**多应用插件**。Office、浏览器、IDE 和会议软件可以通过插件暴露结构化对象和事件。例如文档插件提供段落、批注和版本差异；表格插件提供区域、公式和数据变化；会议插件提供发言人、议题和确认语句；项目系统插件提供任务状态和审批边界。第二类是**屏幕与系统级 AI 探针**。对于无法安装插件或没有 API 的软件，屏幕探针可以对当前窗口、视觉布局和用户操作进行轻量理解，补齐结构化对象覆盖不到的信息；敏感场景下仅输出经过本地过滤的语义片段。

这种设计带来三点价值。其一，上下文由系统持续维护，减少用户反复解释背景。其二，多应用状态被组织为同一任务现场，用户跨软件切换时 Agent 仍围绕同一目标和证据工作。其三，感知与行动处在同一治理链路中，Agent 基于对象能力和权限边界选择提醒、准备、执行或等待。

3.3.3 实时交互模型：多模态流式理解与低延迟反馈

实时交互模型解决"连续现场如何被理解、压缩和反馈实现同频的实时交互"的问题，它与链接对象协议共同工作，把多应用现场中的对象变化转化为模型可处理的条件输入，并实现同频的实时交互。传统实时交互常被简化为语音识别更快或流式输出更快。问题在于，这种轮次式时间轴中用户没有说完时，模型通常还没有进入有效推理，模型开始生成后，它对现场变化的感知又会被冻结，直到生成结束或被外部机制强行打断。真实工作中的用户可能一边说话一边选中文档段落，一边修改表格一边补充口头判断，也可能在看到建议后立即打断或切换对象。若模型仍以完整 prompt 为推理起点，就会慢半拍，若每个输入变化都触发回答，实时能力就会转化为持续打扰。

Coactive 在这一节的核心方案，是建立一套实时交互运行时。它不把实时性等同于更快返回一句话，而是把用户表达、屏幕变化、光标选区、会议语音、文本输入、应用事件和对象状态组织成连续的现场增量；在用户意图尚未完全稳定时提前计算；在需要深度推理时委托后台；在用户打断、对象切换或风险升高时立即取消、降级或重排。其目标可以概括为边输入边准备，边生成边感知，边执行边可打断。

该机制的核心实现是前台交互模型与后台深度模型的分层协同。前台交互模型负责低延迟理解、回应时机和交互控制。后台深度模型承接异步推理、检索和工具调用。多模态输入通过各自转换模型转换后进行整合，实现多模态信息的处理。流式预提取模块在用户意图形成过程中提前准备记忆等相关信息。如图 3-2 所示，实时交互模型核心包含五个要素。

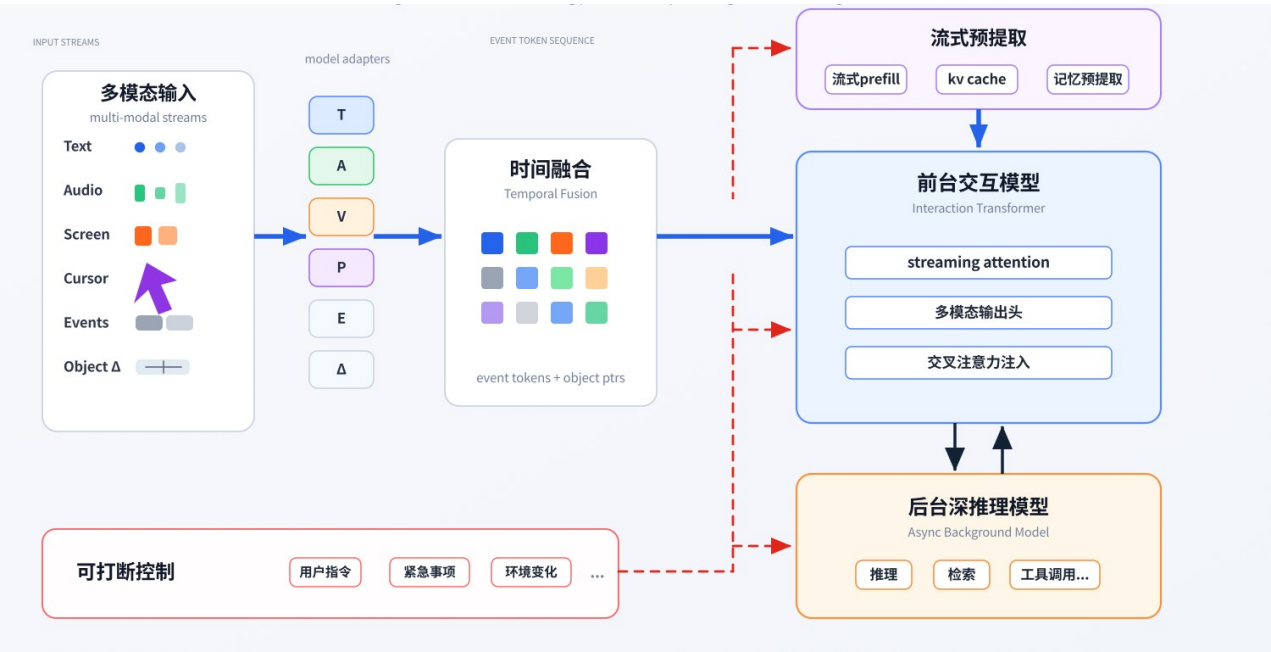


图 3-2 实时交互模型的运行时架构：多模态流式理解、预取计算、后台深推理与可打断控制

多模态输入模块接收任务现场中同时发生的多路信号，将语音片段、文本增量、屏幕事件和应用信号输入，通过多模态编码器转换为模型可处理的事件序列。之后多模态适配器对齐不同输入的特征和粒度，应用事件和对象变化成为结构化状态 token。再通过时间融合把适配后的事件按顺序、重叠关系和因果关系组织起来，对齐不同模态信息间的时间粒度，并给不同的模块提供不同粒度的任务现场信息。**流式预提取**为模型实时交互提前准备记忆等信息，低延迟不只来自更快的生成速度，也来自更早的准备。等用户意图变得明确时，前台模型已经拥有一部分可用上下文，后台模型也可以从已准备的材料继续推理。**前台交互模型**承担实时共处中的实时交互和节奏判断。它读取事件 token 序列和预提取结果，判断此刻应当保持静默、给出局部提示、提出澄清、请求确认，还是把重任务交给后台。**后台深度模型**承接交互中需要一些时间的工作，比如跨文档检索、复杂推理、方案生成、工具调用和版本校验等。通过前台和后台的分工，实现实时不中断的交互体验。**可打断控制**是实现连续交互的关键，在 Interactive 阶段，用户思路可能不成熟，会多次和模型产生来回讨论。可打断控制对后台任务进行协调，根据用户输入和任务现场变化的实时情况调度前台和后台任务是否继续执行或者回到交互。底层基础设施也需要配合。输入侧支持多源事件流的时间戳对齐和优先级调度；上下文侧支持增量编译和 KV cache 复用，使语义逐步稳定时可以提前准备；执行侧支持后台任务的取消令牌和回流校验；服务侧区分低延迟前台路径和高算力后台路径。两条路径共享过程状态。

通过这种设计，Agent 可以在用户意图形成过程中完成现场对齐和候选准备，明确需求出现时不必从零开始。对于复杂任务，后台模型提供深度推理和工具能力，前台仍然保持用户可见的节奏控制。用户一旦打断，旧的输出、cache 和后台任务都会被同步收束。前台模型也会根据任务语义选择静默、提示、澄清、确认或后台委派，保持合适的交互节奏。实时交互模型 Interactive 产生的判断能够沉淀，现场变化、讨论过程、后台结果等记录都会为毕业线的判断提供信息。

3.4 Proactive：面向成熟任务的主动执行状态

Proactive 是 Coactive 中的任务执行态，面向目标清晰、路径稳定、结果可验证的成熟任务片段，主动发现任务及持续完成任务。Proactive 继承 Interactive 形成的目标、质量标准和权限边界，并把执行证据写回过程状态。经毕业线确认任务片段成熟后，完成任务目标，并在条件变化时回到 Interactive。通过这种分工， 既能释放人的沟通注意力，也能实现可监督、可打断和可审计的任务执行。

3.4.1 沉淀后的主动执行：任务规划、工具行动与结果验证

主动式 Agent 研究通常将 Proactive 描述为在没有明确指令时预测需求并发起帮助[10]。在真实工作中，更有价值的主动能力是持续跟进一项任务，并准确判断哪些工作值得承接、何时介入、以何种强度推进，以及何时将判断交还前台。Proactive 的核心是把成熟任务片段转化为可监督、可打断、可恢复的后台推进过程。在企业和个人工作场景中，大量任务不需要每一步都由人亲自操作，却需要被可靠跟进。例如，会议结束后整理纪要并同步到项目管理工具；写报告时收集资料、整理引用并生成不同版本；做销售分析时持续追踪客户动态和关键节点。Proactive 在授权边界内承担任务责任，同时把目标变化和高风险动作及时带回 Interactive。

Proactive 的整体设计包含五个运行环节。1）目标承接：读取 Interactive 已确认的目标、成功标准和停止条件。2）Agent App 承载：将成熟能力片段封装为有生命周期的可运行应用，明确启动条件、权限边界和回调规则。3）Proactive Workflow 推进：用长程任务动态脚本记录阶段、依赖和检查点，使任务跨会话继续运行。4）主动任务挖掘：通过长程任务状态差异发现下一步主动任务。5）主动介入分级：根据任务价值、风险等级和授权证据，在静默观察、轻量提示、请求确认和条件自主之间选择合适强度。

Proactive 的成熟度不取决于主动提示数量，也不取决于后台任务跑得多远，而取决于它能否可靠承担成熟任务责任，承接前有目标和证据，执行中有状态和权限，交付时有结果和回执，异常时有回流和恢复。

3.4.2 Agent App：从单次调用到可运行应用

当前主流 Agent 平台中的 Skill，通常指 Agent 可调用的一项能力：有名称、有参数定义、有返回格式，调用一次、返回结果、交互结束。Coactive 面对的长程任务通常持续数天甚至数周，需要多阶段执行、持久状态和条件回调。整理会议纪要、跟踪项目承诺、准备客户复盘等任务，无法由单次调用的能力单元完整承接。

从传统 Skill 到 Agent App，核心跨越在于运行形态。开发者描述任务能力和工作流程，平台自动补齐交互入口、状态回写和协作接口，把一段能力封装成可持续运行、可分享、可定制的应用。这里的 Agent App 可以兼容 Skill App 表达：Skill 强调能力来源，Agent App 强调运行形态和产品承载。两者的差异如表 3-5 所示。

以产品规划会议为例。传统 Skill 调用"会议纪要"能力，传入会议录音，返回一份文本摘要，交互结束；后续待办分配和逾期提醒不再由系统跟进。Agent App 则启动一个"会议跟进"应用：会后 30 分钟内提取承诺，进入确认阶段；之后每日检查待办进展，跨越录音、聊天消息和项目管理系统；当客户交付待办逾期且会议被提前时，Agent App 通过 Decision Packet 回调到 Interactive，由用户选择延期、改派或升级，随后继续执行。

表 3-5 Agent App 与传统 Skill 的能力边界

要素	Agent App 描述	传统 Skill
能力目标	说明解决的问题、适用场景与不适用场景。	通常仅有函数描述。
多阶段 Workflow	定义任务如何分阶段推进，以及阶段间如何转换。	缺失。
状态模型	描述核心实体如何流转，如提取、确认、执行中、完成。	缺失。
触发与暂停	定义何时自动启动、暂停、恢复或终止。	缺失。
权限边界	明确允许执行、需要确认和禁止的动作。	部分支持。
验证方式	说明结果由人工确认、系统校验或混合校验。	通常缺失。
回调接口	定义协作唤醒条件，将问题交回 Interactive。	缺失。

Agent App Runtime 是支撑 Agent App 运行的平台层，负责长程应用的可靠启动、暂停、恢复和治理。它覆盖五个层面。**生命周期管理**：启动、暂停和终止应用实例，中断后恢复到上次检查点。**状态持久化**：将任务阶段和验证结果写成结构化状态，支持跨会话延续。**平台能力反调**：让运行中的 Agent App 调用模型推理、搜索和链接对象操作。**回退与副作用管理**：区分可逆动作和外部副作用，失败时回退到安全状态。**分享、定制与治理**：使应用可复制、可发布、可审计和可回滚。Agent App Runtime 让成熟能力从一次性工具调用升级为平台可管理的运行时应用。它承接 Interactive 中被反复验证的能力片段，也为 Proactive Workflow 提供运行载体。

3.4.3 Proactive Workflow：长程任务的动态脚本与可恢复执行

Proactive 的长程任务不能只依赖模型在上下文窗口中记住全部过程。任务一旦跨越数天、多应用和多轮反馈，单一上下文滑窗会同时承载顶层目标、中层编排和叶子节点的语义生成，窗口宽度只能覆盖流程片段，旧语义不断被挤出，容易产生目标漂移和幻觉积累[14]。

Proactive Workflow 的核心设计原则是：流程的归变量，变化的归模型。脚本负责固定流程骨架，模型负责当前步骤的语义理解和生成。脚本记录阶段结构，保存每一步的输入、输出和检查点；模型处理当前一步需要的语义判断，例如搜索、抽取、验证或冲突消解。顶层流程状态和中层编排被外化到脚本与状态文件中，模型上下文只承载当前步骤的最小必要信息，滑窗负担接近 O(1)。

一个典型 Proactive Workflow 分为三层。顶层是跨全周期流程状态，记录任务目标、全局成功标准和用户授权边界。中层是阶段编排状态，管理检索、抽取、验证和输出等阶段的输入输出和检查点。叶子层是原子化语义任务，由模型完成当前一小步的理解或判断。三层结果均需外化为结构化状态文件，使任务可以断点续执行、跨会话恢复和事后审计。此外，脚本不应把模型、搜索和外部系统写死在业务逻辑里，而应通过 Runtime 调用统一能力。Workflow 只表达"这一阶段需要搜索""这一阶段需要验证"，具体能力来源由 Runtime 提供。Workflow 与 Agent App 的关系可以概括为：Agent App 是可运行能力的产品单元，Workflow 是其内部推进长程任务的动态脚本。Agent App 管理启动、暂停、权限和回调；Workflow 管理阶段、依赖和检查点。二者结合，才能让 Proactive 跨时间持续推进，并在条件变化时安全停下。

Proactive Workflow 必须与毕业线相结合。Workflow 运行得越久，越需要持续写入执行状态和证据，使用户知道当前做到哪里、为什么这样做、失败时从哪里恢复。只有当 Workflow 的目标、路径和授权边界都被证据化，相关能力片段才具备从 Interactive 渐进迁移到 Proactive 的基础。

3.4.4 主动任务挖掘：基于长程任务状态差异的需求发现机制

传统主动任务挖掘从历史的对话和任务执行记录中发现可能需要提前或者主动协助用户完成的任务。当前方案常把用户输入切成单句或单轮对话，再从当前片段判断是否存在可执行事项。这种方法在短会话中有效，但在长时间、多项目并行的真实工作场景中会失效：同一个需求可能分散在会议、文档、消息和工具调用中；同一句话也可能同时更新项目状态、时间点和用户偏好。

Coactive 将主动任务的发现对象从当前片段中的显式动作提升为长程任务的状态差异。如图 3-3 所示，系统先将用户的工作项目、周期流程和沟通承诺、甚至日常生活周期事件等建模为可持续维护的长程任务，再基于长程任务的目标、现状和风险状态推导下一步主动任务。

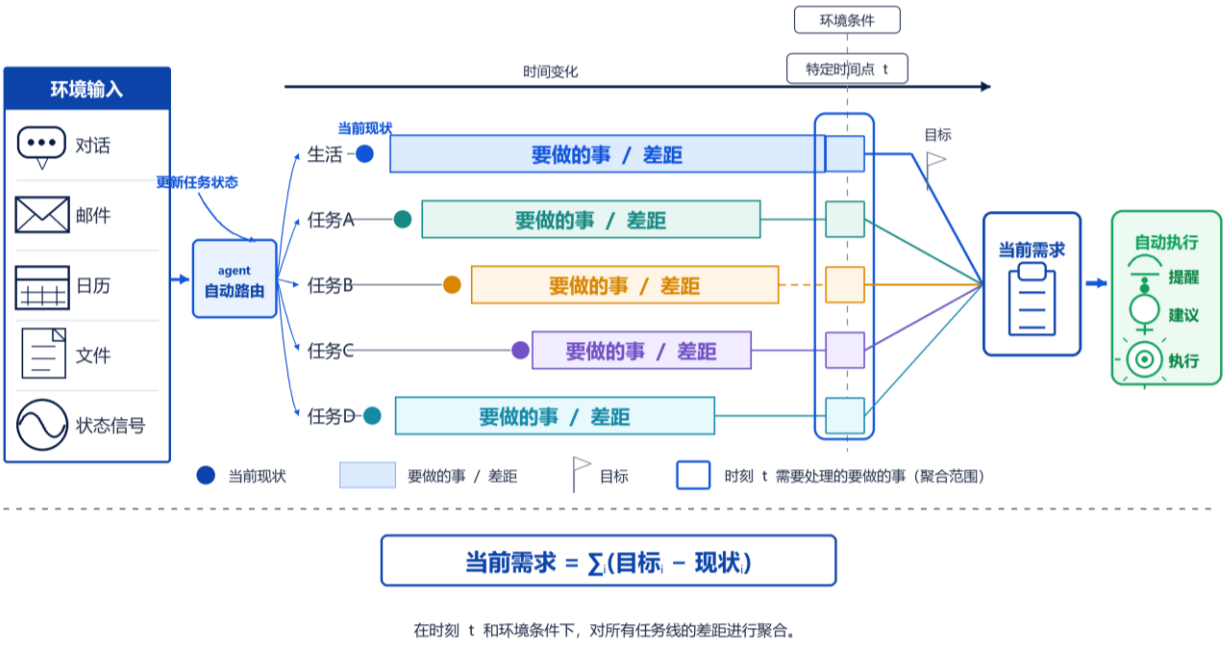


图 3-3 主动任务挖掘：在特定时间点与环境条件下聚合长程任务的状态差异

通过这种方式，可以将每个长程任务封装为类似 tool card 的轻量描述，包含任务目标、当前状态和适用路由条件。模型处理复杂交互记录时，复用已有 tool 选择能力，将新证据路由到最相关的长程任务中，避免依赖人工枚举的需求类型。路由后的任务本地分析器只更新该任务的状态，例如某个计划节点是否完成、某个外部依赖是否仍未闭环、某个文件是否已生成但未审核。碎片化的聊天、文件和工具记录由此持续汇聚到稳定任务状态中，减少孤立行动建议。

任务状态更新后，系统通过"目标—现状—时机—风险"差异算子生成主动任务。只有目标与现状之间存在可行动缺口、时间窗口正在收窄或风险等级上升时，系统才形成下一步候选输出。主动任务因此成为长程任务状态的一次可解释推导结果。实验中，基于该长程任务建模与状态差异推导的方案，在 Proactive 评测集上相较原始 Proactive 方法取得约 20%的 F1 指标提升[10]，说明该方法能更稳定地把长期上下文转化为有效的主动任务发现。

主动任务挖掘产生的是候选任务队列，并非直接执行命令。候选任务只有在目标、证据和权限满足毕业线要求时，才进入更高等级的 Proactive 承接；一旦状态差异指向目标漂移或风险升高，系统应转化为协作唤醒，而非继续扩大后台执行。

3.4.5 主动介入分级：从静默准备到条件自主

在 Proactive 状态中，Agent 面临的核心问题是主动到什么程度。所有潜在需求都立即推送，会使主动服务变成打扰；目标尚未稳定或授权边界不清时直接执行，则会形成越权。Coactive 将主动性设计为连续等级链：从静默观察、静默准备、轻量提示、请求确认、确认后执行到条件自主，每一级对应不同的证据和授权要求。

这一机制将"发现潜在任务"和"打扰用户"拆成两个连续判断。先判断长程任务内部是否产生行动价值：任务目标、依赖状态和时间窗口是否指向值得处理的主动任务。再判断当前任务现场是否适合介入，用户注意力状态、最近交互节奏和历史反馈是否允许此刻提示。

通过分级治理，Proactive 可以避免两类常见失败。过度主动：在用户阅读、连续编辑或刚刚拒绝类似建议时反复打断。越权主动：在外部发送、审批或不可逆操作中绕过用户确认。低置信或高打扰成本的事项保持静默；信息充分但仍需人判断的事项进入轻量提示；目标稳定、路径成熟、动作可验证且失败可回滚的任务片段，才进入确认后执行或条件自主。可信承接决定能力片段能否升级到更高主动等级；协作唤醒决定条件变化时是否降级或回流。分级治理把这些迁移落实为每一次提示、执行和暂停的运行策略。

3.5 双态迁移：可信承接与协作唤醒

毕业线是 Interactive 与 Proactive 之间的迁移标准，用于判断哪些能力片段可以从共同判断中沉淀出来，进入受控主动承接；也用于判断后台执行在什么条件下必须停止或回到前台。它是能力片段在开放世界中的生命周期边界。双态迁移机制包含两个方向。可信承接：当目标、执行路径、质量标准和用

户信任逐步稳定后，能力片段可以从 Interactive 进入 Proactive。协作唤醒：当 Proactive 执行中出现目标漂移、证据不足或风险升高时，系统把后台状态压缩为 Decision Packet，将判断权交还 Interactive。

毕业线的关键突破在于建立以能力片段为单位，基于证据、渐进释放且可以回退的双向迁移机制。过程状态记录目标、执行、证据、风险和待决问题，是 Interactive 与 Proactive 保持连续的公共介质；任务、能力和责任三域成熟度用于判断能力片段是否具备可独立验证、可授权和可回退的承接条件；可信承接根据证据逐步释放主动等级，协作唤醒则在目标漂移、证据不足、风险升高或权限变化时暂停后台执行，并通过 Decision Packet 将任务事实、可选路径和风险带回前台。由此，Agent 的能力迁移可以被解释、追踪和撤销，而不是一旦自动化便持续扩大自主范围。

3.5.1 过程状态与能力片段：前后台连续与能力抽取

双态迁移的两个方向都依赖同一个底座：过程状态。链接对象协议解决"有什么在场"，Agent App Runtime 解决"能力如何运行"，过程状态解决"事情推进到哪里"。没有过程状态，Interactive 与 Proactive 会断线：前台已确认的目标无法约束后台执行，后台发现的异常也无法回到前台继续判断。

过程状态不同于消息历史。消息历史记录"说过什么"，过程状态记录"共同活动已经形成了什么"：当前目标和成功标准是什么，哪些约束仍然生效，任务推进到哪个阶段，哪些证据已经查询，哪些决策还悬而未决。链接对象说明工作对象如何进入现场；过程状态说明这些对象围绕同一任务产生的目标、进展和责任关系。过程状态是 Interactive 经验升华为 Proactive 能力的中间层。如**错误!未找到引用源。**所示，用户在 Interactive 中的表达、采纳、拒绝和授权，需要被整理为可迁移的结构化产物。**执行目标**说明要完成什么、做到哪里停。**判断标准**说明什么是好结果。**材料关系**说明哪些来源可信、哪些被排除。**行动边界**说明什么可静默准备、什么需确认、什么禁止。**用户偏好**说明哪些工作方式可以复用。**开放问题**说明哪些部分尚未具备主动承接条件。

表 3-6 过程状态的五类信息

状态类型	记录内容	作用
目标状态	当前目标、完成标准、优先级、停止条件和生效约束。	确保后台推进方向与前台判断一致。
执行状态	任务阶段、最近检查点、已完成动作、阻塞原因和下一步候选。	支持前台和后台从同一进度接续。
证据状态	已查询来源、工具回执、结果版本、数据矛盾和未解决问题。	支撑承接判断和事后审计。
风险状态	不可逆操作、敏感数据、外部影响范围、权限期限和回滚方案。	判断哪些动作必须请求确认或升级审批。
决策状态	待决问题、候选方案、系统建议和各方案风险。	为协作唤醒生成 Decision Packet。

这些结构化产物进一步连接到 Proactive 的运行单元。稳定目标可以进入 Workflow 的目标状态；稳定操作路径可以成为 Agent App；反复验证的阶段结构可以成为长程任务动态脚本；用户对提醒时机和风险阈值的反馈可以成为主动策略；成功和失败案例可以成为评估样本。Proactive 执行也会更新过

程状态。后台完成阶段产物、发现数据冲突、遇到权限不足或触发风险边界，都应写回同一份过程状态。下一次用户回到前台时，看到的应是当前做到哪里、为什么停下、有哪些选项，而非零散日志。

因此，过程状态是前台与后台的连续性协议，也是毕业线运行的公共介质。可信承接基于过程状态中的目标和证据完成升级；协作唤醒基于过程状态生成判断回流，将判断权交还前台。

3.5.2 毕业线：从 Interactive 到 Proactive 的可信承接

一项任务不能因为做过一次就交给 Agent 自动执行。真实工作里，表面相似的任务常常包含不同风险：同样是整理会议纪要，一次只是内部同步，一次可能涉及客户承诺；同样是生成周报，一次只是草稿，一次可能要对外发布。执行路径相同，不代表承接边界相同。可信承接回答的是：什么条件下，一个能力片段可以从 Interactive 移交到 Proactive。

承接的基本单位是可独立验证、可授权、可回退的能力片段。如图 3-4 所示一个能力片段能否被承接，需要经过任务、能力和责任三域成熟度判断。任务成熟度关注目标和成功标准是否充分稳定，目标仍在形成或材料缺口明显时，应继续留在 Interactive。能力成熟关注执行路径是否可重复和可恢复，结果是否有稳定的质量标准。流程依赖临场判断或失败无法恢复时，只能静默准备或草稿执行。责任成熟度关注权限、影响和回滚链是否明确，动作影响外部对象或对外承诺时，需要缩小适用范围或进入确认审批。

从 Interactive 到 Proactive 的过程应渐进释放。共同探索阶段，由人与 Agent 形成目标与标准。静默准备阶段，主体整理材料但不改变外部状态。草稿执行阶段，主体生成可检查产物。确认后执行阶段，关键动作置于明确授权之下。条件自主阶段，主体在限定范围内承接低风险、可验证的成熟任务。每一级都保留回退到更早阶段的可能。Interactive 中的经验需要转化为 Proactive 可运行产物。执行目标说明目标和停止条件。Agent App 记录稳定操作路径和回调条件。Workflow 描述阶段、依赖和检查点。经验记忆保存用户偏好和质量要求。主动策略定义触发条件和介入等级。评估样本包含成功案例、失败案例和验收标准。授权边界明确可静默、需确认和禁止的动作。

OpenAI 在 Tax AI 场景中展示了类似逻辑：系统通过专家反馈和生产轨迹持续改进，在六周内将达到 75%字段正确率的退税材料比例从约四分之一提升到 86%[15]。能力不是一次性判断，而是在证据积累中渐进释放。

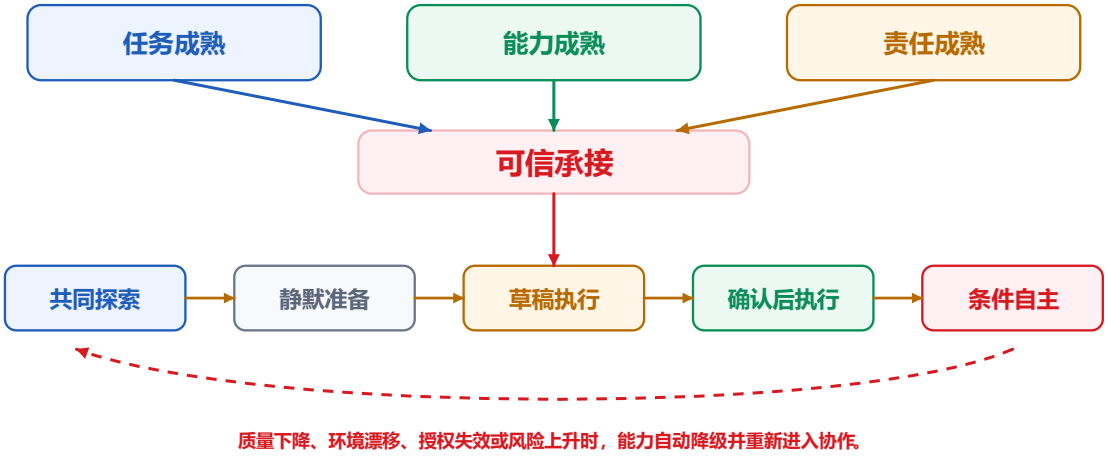


图 3-4 毕业线中三种成熟维度和可信承接

组织政策、数据源和用户偏好都会变化，表面重复也可能被误判为真正成熟。稳健原则是：以最小适用范围承接，以多源证据验证，以持续回归维持，以保守默认处理不确定性。一旦质量下降、授权失效或风险上升，智能主体必要时应退回 Interactive 重新形成目标与边界。

3.5.3 协作唤醒：从 Proactive 到 Interactive 的协作交互

可信承接解释任务如何从 Interactive 进入 Proactive，协作唤醒解释 Proactive 何时必须将判断权交还 Interactive。开放世界不会因任务片段被承接而停止变化。后台执行时，目标可能漂移，证据可能不足，风险可能升高，用户意图可能变化。此时继续推进会越过人的判断权。

协作唤醒回答什么条件下 Proactive 必须停下来。触发条件来自任务现场的变化。目标漂移：新材料或外部事件与原目标不一致，系统需要确认是否调整方向。证据不足：关键来源缺失或数据冲突，系统需要让用户决定补充材料还是暂停。风险升高：动作可能影响敏感数据或不可逆结果，系统必须请求确认。权限变化：对象下线或授权过期，系统需要重新授权。用户意图变化：用户在其他现场给出新偏好或新优先级。多主体分歧：不同 Agent 或工具给出冲突判断，系统应把分歧点带回 Interactive。

唤醒被触发后，后台状态需要压缩成一个人可以直接判断的决策单元，即 Decision Packet。它回答五个问题：后台发生了什么，为什么需要人判断，现在有哪些选项，系统推荐什么，各选项的风险是什么。用户看到的不应只是"任务失败，错误码 403"，而应看到任务事实、影响范围、可选路径和回退方案。

以"会议跟进" Agent App 为例。后台发现某项客户交付待办逾期 3 天，且客户会议被提前到明天。它不应自动升级，也不应只弹出逾期提醒，而应生成 Decision Packet：当前待办完成度、客户影响、可选动作（延期、改派、升级或先发说明）、系统推荐路径，以及每个选项对时间和风险的影响。用户选择后，过程状态随之更新：目标是否调整，Workflow 是否重排，主动策略是否收紧。

从 Proactive 回到 Interactive 的信息分为四类。结果回流来自阶段完成或部分失败，回到前台后由用户验收并决定是否提升承接等级。异常回流来自工具失败或权限不足，回到前台后澄清原因或修改计

划。风险回流来自不可逆动作或影响他人的操作，回到前台后请求授权或收缩范围。经验回流来自用户采纳、拒绝或接管，回到前台后更新记忆和主动策略。多主体分歧尤其需要协作唤醒[16]，多个 Agent 或工具给出冲突判断时，主体不应以模型置信度在后台静默裁决。它应说明分歧来源：数据源冲突、工具返回不一致，还是风险规则冲突。最终判断权交还用户，并在决策后写回过程状态。

协作唤醒的难点是时机。唤醒过早会使 Proactive 变成另一种打扰，唤醒过晚则可能越过人的判断权。Coactive 将这类张力显化：当前为什么停下，继续运行的风险是什么，人需要判断哪一件事。后台既不把所有内部状态原样推给用户，也不把风险藏在"继续执行"之后；它把关键变化组织为可判断的选项，并说明每个选项会如何影响目标和后续主动等级。

双态迁移使 Interactive 与 Proactive 形成持续循环：Interactive 把模糊判断转化为稳定经验，Proactive 把稳定经验转化为可验证行动；行动结果再回到 Interactive，让人修正目标、确认边界，并决定是否扩大或撤销承接范围。成熟的 Coactive 主体能够解释当前选择，并在边界失效时安全回退。

3.6 本章小结

Coactive 智能主体以 Interactive、Proactive 和双态迁移为核心。Interactive 让人的经验和判断持续进入任务，面向目标未稳、需要共同创造的开放任务；Proactive 在授权边界内承接成熟任务，负责长程执行、结果验证和阶段汇报；双态迁移的可信承接和协作唤醒，使升级与回流都基于证据。成熟的 Coactive 主体能够说明当前为什么处在某一状态，能够把前台共创和后台执行放在同一目标和证据下运行，也能够在边界失效时安全回退。由此，智能主体从一次性响应工具转变为可运行、可沉淀、可治理的协作主体。

> **CHAPTER TAKEAWAY**
>
> **Interactive 与 Proactive 共享同一任务现场，由双态迁移管理可信承接和协作唤醒。**

第四章 环境感知：从聊天框到任务环境

ENVIRONMENT AWARENESS

Coactive 环境感知不止是让 Agent 拥有更长的上下文窗口和收集更多数据，而是定义了 Agent 与人的协作发生在怎样的世界中，把用户所处的任务现场组织成 Agent 可理解、可响应、可沉淀的上下文。本章说明 Agent 如何在授权范围内接入屏幕、文档、会议、应用、设备，并把开放世界的环境投影为当前任务最小充分、可行动且可追溯的任务现场和 Agent 可理解的上下文。

4.1 Agent 环境感知面临的挑战

当前 Agent 工程已经形成多种扩展上下文的方法。长上下文窗口让模型能够读取更多文本，RAG 让模型从知识库中检索相关材料，工具让模型直接处理用户资料，MCP 等协议让 Agent 可以连接并感知工具、数据源和外部系统。这些技术都扩展了 LLM 可感知的世界，但它们主要解决的是某一时刻“模型在一次调用前能看到什么”，无法支持 Agent 面对真实世界中跨工具、跨文件、跨会话、跨时间的持续性复杂任务。传统 Agent 在环境感知中常见的失败模式如表 4-1 所示。

在开放世界中，同一段信息在不同任务中价值完全不同，同一个用户动作在不同阶段可能意味着确认、犹豫、探索或放弃。情境感知计算的经典研究指出，Context 不只是位置、时间等传感器信息，而是任何能够刻画实体处境、并影响交互相关性的资料；Dey 对 context-aware system 的定义进一步强调，系统提供的信息或服务是否相关，取决于用户正在进行的任务[17]。

表 4-1 Agent 环境感知中常见的失败模式

失败模式	典型表现	Coactive Agent 的环境感知
上下文遗漏	用户没有显示说明任务目标、会议结论、历史决策或当前约束	持续跟踪、维护任务状态、关键对象和来源指针
任务对象混淆	Agent 不知道“这个表”“上一版”“刚才那段”“客户说的那个方案”指什么	建立链接对象图、版本关系和指代解析
状态过期	旧结论、旧数据、旧权限或旧任务阶段仍被使用，触发越权或错误行动	持续维护任务时间线、最近变化、有效期和状态刷新机制
主动误触发	过多的环境信息输入，导致 Agent 在用户不需要时打扰，或在证据不足时主动行动	基于任务价值、风险、置信度和时机控制主动性

当任务跨越多个文件、会议、应用和时间节点时，用户实际承担了任务状态管理者的角色，而 Agent 只是被动接收压缩后的片段。Coactive 环境感知需要跨过的正是这条边界，它需要把任务连续性从用户的口头补充中移出来，由系统持续维护状态、对象、边界和结果证据。

4.2 设计原则：相关、低扰、可控、可追溯

环境感知能力越强，越需要先明确设计原则。否则，它很容易从“理解现场”滑向“过度采集”，从“主动帮助”滑向“持续打扰”，从“共同工作”滑向“用户被系统观察”。在 Coactive Agent 中，环境感知不是为了让 Agent 显得无所不知，而是为了让用户相信：系统只在合适范围内理解自己正在做的事。

相关 决定哪些信息可以进入任务现场。Agent 不需要知道一切，只需要知道与当前任务相关的上下文。环境感知系统应围绕任务目标、对象身份、用户意图和当前阶段进行过滤，把全量环境收敛为最小

充分表达。判断相关性的关键，不是信号是否存在，而是它是否会改变当前任务的理解、行动、风险或结果验证。

低扰 决定环境信息何时呈现或触发交互。感知的价值是降低用户反复解释背景的负担，而不是制造新的注意力负担。系统需要判断什么时候出现、什么时候等待、什么时候只在后台准备。能安静完成状态更新时，不应把每一次观察都转化为提醒、追问或弹窗；只有当信息改变任务判断、风险上升或用户可能需要帮助时，才应进入前台交互。

可控 决定当前任务允许访问什么、改变什么。用户和组织应当知道 Agent 正在感知什么、为什么需要这些信息、这些信息会被用于什么任务，以及如何暂停、撤回或删除。可控还意味着边界清晰：环境感知必须与权限系统、隐私策略、企业数据边界和审计机制配合，个人记忆、应用记忆和组织知识之间不能被无差别打通。

可追溯 决定系统如何检查、修正和恢复。环境感知产生的状态、建议、行动依据和记忆候选，都应带有来源、时间、对象、权限和置信度标记。当 Agent 基于环境做出建议或触发行动时，用户应能够回到具体文件、会议片段、操作记录或用户确认。NIST AI RMF 将有效可靠、安全、可解释、隐私增强和问责透明等列为可信 AI 的重要特征；对环境感知而言，这些要求最终要落实到来源记录、权限边界和行动回执[9]。

四项原则之间也存在优先级。当完整性、主动性与用户控制冲突时，可控与可追溯优先；当推断证据不足时，系统应保持为候选态势，而不是写成确定事实；当环境信息不影响任务判断时，应留在后台或被任务现场投影丢弃，不进入前台打扰。

Coactive 环境感知的核心机制，是把开放世界中持续变化的环境信号加工为当前任务所需的任务现场。基于上述设计原则，我们设计了：1）AI 设备、AI 应用与工具、Agent 探针：通过链接对象及接入协议提供对象身份、状态、事件、可执行操作和调用回执；2）环境接入层：通过语义分层与压缩，将连续信号整理为结构化事件、状态快照、语义片段和可执行动作变化；3）任务现场投影：进一步完成信息抽取、对象对齐和约束确定，只保留会改变任务理解、行动选择、风险判断或结果验证的信息。Agent 不只是“看到更多”，而是能够判断当前任务发生在哪里、哪些变化值得响应、允许采取什么行动，以及行动依据能否追溯。

4.3 面向开放世界的 Coactive 环境感知机制

Coactive Agent 包含一个核心目标：让 Agent 接入开放世界的真实任务环境，而不是只停留在聊天框中。为了达成此目标，Coactive 环境感知模块需要基于 Agent 对象链接协议接收来自开放世界的信息。在用户授权的范围内，Agent 实时感知用户将要进行的任务，并跟随任务进入不同设备、应用和空间，在复杂开放世界中保持任务连续性。

表 4-2 Coactive 环境感知获取信息的三种接入对象

接入对象	定义	样例
AI 设备	Agent 连接物理世界和人类活动现场的终端节点。	未来的 AI 车载系统、工业设备、AI 耳机、AR 眼镜等边缘智能设备。
AI 应用与工具	Agent 连接数字工作世界的对象来源和行动入口。	未来的 AI 浏览器、企业系统、IDE、Office 软件等。
Agent 探针	面向非 AI-native 的设备与应用，从系统间接获取环境信息。	PC/手机探针从系统获取屏幕、音频、文件事件、鼠标键盘操作等信号。

围绕这个目标，本章引入以下三类 Coactive 环境感知的接入对象：AI 设备、AI 应用与工具和 Agent 探针，它们的模式定义及样例如表 4-2 所示。

我们将“任务现场”定义为 Coactive Agent 围绕当前任务所需要的全量信息。Coactive 环境感知需要把三类对象产生或获取到的环境变化，组织成当前任务真正需要的任务现场。AI 设备、应用与工具可以原生上报结构化数据；Agent 探针则为非原生环境补齐信号获取和结构化事件生成能力。经过隐私管理后的三类输入最终都进入任务现场投影，由投影层判断当前任务需要知道什么、允许做什么、依据是什么以及结果如何验证，最终生成 Coactive Agent 可用的任务现场。具体流程如图 4-1 所示。

图 4-1 全量任务环境如何被动态投影为当前任务的任务现场

以下两个样例分别说明：三类环境接入对象如何产生或获取信号，以及最小任务投影如何消费这些结构化事件，进一步说明环境感知如何把真实环境中的信号转化为 Agent 可理解、可沉淀的任务上下文。

4.3.1 AI 设备、AI 应用与工具和 Agent 探针：让 Agent 接入开放世界

以会议协作为例，传统 Agent 往往只能在会后读取一段转写文本，再生成摘要或待办。这种方式忽略了会议中的大量现场状态：谁在主导讨论，哪句话只是发散想法，哪句话已经形成确认，用户是否正在看共享屏幕，某个沉默是等待、犹豫还是分歧。

在 Coactive 环境感知中，会议现场可以由三类对象共同接入。AI 耳机、会议屏和 AR 眼镜等 AI 设备，直接上报发言人变化、确认语句、设备状态和用户操作；会议系统、文档、白板和浏览器等应用与工具，主动暴露共享材料、参会状态、编辑对象和操作回执；当某些桌面应用、网页或外设并不具备原生接入能力时，Agent 探针作为一个应用获取屏幕、音频、键鼠、窗口焦点和系统事件，并把它们转成结构化事件。

在这个场景中，Coactive 环境感知关注的不止是会议里说了什么，而是这些信号如何改变任务。例如，一位负责人说“这个方案我们先按 B 版推进”，这不是普通转写文本，而可能意味着任务状态发生变化；用户随后打开报价表，也不仅仅意味着用户开始操作表格，而可能说明当前讨论对象已经从方案方向转向成本核算。

三类环境接入对象在 Coactive 环境感知中获取的信号和应用场景如表 4-3 所示。

表 4-3 环境接入对象的信号获取样例

接入对象	获取环境信号的方式	环境信号样例
AI 设备	通过链接对象协议主动上报设备状态、传感器事件、语音片段、用户操作等事件。	AI 耳机上报音频、确认发言人的语句、打断、沉默；AR 眼镜上报用户的视觉焦点、视频流等数据。
AI 应用与工具	通过链接对象协议主动暴露操作对象、工作状态、工具能力、调用结果和可执行的操作。	Office 应用上报文档某段被修改，是否在同时操作 Excel 等应用；AI 浏览器读取用户正在访问的页面、停留时间，主动识别用户感兴趣的内容。
Agent 探针	面向非 AI-native 环境获取屏幕、音频、键鼠、窗口焦点、系统通知等系统事件，并通过对接连接协议上报。	PC 探针通过捕获用户的键鼠输入、屏幕捕捉，确认用户焦点窗口和感兴趣的内容；手机探针获取用户的位置坐标、语音输入等信息。

因此，三类环境接入对象的共同价值，是把 Agent 从聊天框带入真实任务现场，使 Agent 能够感知人、设备、应用和物理空间中与任务相关的状态变化，其中 AI 设备和应用与工具负责原生上报，Agent 探针负责为非原生环境补齐信号获取和结构化事件生成能力。

4.3.2 任务现场投影：从环境噪声中抽象出当前任务的相关信息

环境接入对象从开放世界中获取的信息包含大量噪音，但最终 Coactive Agent 需要的是当前的任务和任务所需的最小上下文，因此我们需要通过任务现场投影获得真正相关的环境信号。

以文档写作为例，真实环境中存在大量噪声：用户 PC 上打开了多个版本的 docx、excel 和 ppt 等应用、多组浏览器页面，同时还带着耳机播放音乐。任务现场投影需要消费来自 AI 设备、应用与工具、Agent 探针的结构化事件，首先识别到用户正在编辑的焦点文档应用，判断哪些对象影响当前写作，哪些事件只是背景噪声。例如，用户长时间停留在某个段落来回修改，可能说明当前任务是结构调整；浏览器和文档来回切换，可能说明用户正在核对引用；与当前写作无关的通知和窗口变化则应被丢弃。

4.4 Coactive 环境感知的信息加工链路：从环境信号到任务现场

Coactive 环境感知的核心是“构造任务现场”：系统需要持续识别任务现场的变化，再将这些变化组织为任务中的对象、状态、边界和证据，最终交付 Agent 所需的结构化上下文。这个过程可以分为如下两个核心环节：

1. 环境信号采集：从 AI 设备/应用/工具和 Agent 探针中获取原始信号，转化为结构化事件

2. 任务现场投影：对环境信号进行过滤、去噪、对象对齐，围绕 Coactive Agent 的当前任务重新组织现场上下文

4.4.1 环境信号采集：链接对象协议、语义分层与压缩

链接对象协议 真实工作环境由大量对象组成：文档、会议视频、鼠标键盘操作、打开的应用和用户行为等。Agent 如果只能读取一段文本或一张截图，就无法稳定判断“这个表”、“上一版”、“客户说的那个方案”到底指向什么。环境接入层的任务，是让这些对象以统一的身份、状态、事件和能力进入系统。

因此，首先需要一层通用**链接对象协议**。它不要求所有设备和应用采用同一种底层实现，但要求它们以一致的方式向探针暴露三类能力：对象自动发现和接入、可执行的操作、可获取的数据。更具体地说，连接协议是一条运行链路：连接端注册对象，声明对象能力，Coactive Agent 通过网关发现对象和读取工具说明，随后把工具调用路由回接入端执行，并接收结构化结果。AI 设备和 AI-native 应用可以直接上报这些信息；Agent 探针则为非原生环境生成等价的结构化事件。链接对象协议的通信内容与典型样例如表 4-4 所示。

这三类能力构成链接对象协议的主体。自动发现和接入解决“环境里有什么”；可执行操作解决“Agent 可以做什么”；可获取数据解决“Agent 可以知道什么”。在工程实现上，它们可以落地成“连接对象 + 对象工具 + 调用回执”的机制。

链接对象协议可以直接把 AI 设备、应用和工具从“黑盒”变成 Agent 可直接理解的工作对象。Agent 探针作为一个应用，也按照同样约束上报自己从非原生环境中获取并结构化后的事件。PC 探针可以管

理桌面应用、浏览器、本地文件和外接 AI 设备；手机探针可以管理移动应用、通知、通话和随身 AI 设备。

基于三类环境接入对象，环境感知和行动能力被放在同一条协议链路里：同一个连接对象既能暴露可获取数据，也能暴露可执行操作；同一个工具调用既能产生任务所需数据，也能产生动作回执和证据。例如：Office 文档不只是文件内容，还暴露了用户当前的鼠标焦点、正在编辑的内容，甚至可以让 Agent 插入到用户的思考、写作过程中。

表 4-4 链接对象协议内容及样例

协议能力	通信内容	典型样例
对象身份状态的自动发现和接入	当前终端上有哪些应用、设备、账号和对象可以接入	对象类型、对象幂等键、展示名、当前对象、在线状态、心跳、连接端、权限范围
Agent 可执行的操作	Agent 或探针在授权边界内可以对对象做什么	对象工具、工具说明、参数 schema、读写副作用、二次确认、调用超时、结果回传
Agent 可获取的事件与数据	探针可以从对象中持续或按需获得什么信息	内容摘要、元数据、状态、事件、版本、选区、焦点、通知、转写、设备状态、工具回执

语义分层与压缩 用户在一分钟内可能发生多次窗口切换、选区、输入、撤回、保存和浏览；会议中可能同时出现发言、打断、共享屏幕、聊天评论和待办确认。环境接入层需要在本地维护一组轻量状态，用于判断环境是否发生了有意义的变化，以及变化是否需要上报给 Agent。探针上报给 Agent 的事件内容可以归类为以下四种：

1. **结构化事件**，描述外部发生了什么。例如，文档某节被修改、会议出现确认、工具调用失败、用户授权某个动作。结构化事件要求对象明确、时间明确、来源明确。

2. **状态快照**，描述当前环境处在什么状态。例如，当前活跃应用、正在编辑的对象、连接设备、可用能力、权限边界和最近版本。状态快照用于让 Agent 在行动前获得最新环境条件。

3. **语义片段**，用规则、小模型或 LLM 判断事件语义，将一段行为或多源事件的压缩成有意义的片段。例如，用户持续对文档的键鼠操作可以压缩为“用户正在围绕当前章节的工程设计进行结构调整”。

4. **可执行的动作变化**，描述现在可以做什么、不能做什么、哪些动作需要确认。例如，当前文档允许写入草稿但不允许覆盖源文件，会议纪要可读但客户 CRM 更新需要二次授权。

通过这种语义归类和压缩，三类环境接入对象把开放世界中的信息转化为 Agent 可接收处理的结构化事件。

4.4.2 任务现场最小化：从结构化事件到最小任务现场

任务现场投影负责把 AI 设备、应用与工具、Agent 探针上报的结构化事件、状态快照、能力变化和语义片段，进一步组织成与用户当前任务相关的任务现场。开放环境包含大量噪声，探针输出也是持续流动的，但 Coactive Agent 真正需要的是一个围绕当前任务生成的最小任务现场。

任务现场投影可以分为三步：

1. 信息抽取：基于规则和模型抽取任务目标和当前阶段，从探针输出的结构化事件中选择真正相关的信息。相关性的判断不仅要看关键词相似度，还要看这条信息是否会改变任务理解、行动选择、风险判断或结果验证。

2. 对象对齐：把来自不同来源的信息对齐到同一组对象和任务状态上。例如，会议里确认的结论、文档中的修改和表格中的数据变化，可能都指向同一个客户方案任务。对齐依赖对象图、版本图、时间线和任务状态机。

3. 确定约束：把权限、风险、确认要求和回滚能力写入任务现场。任务现场需要告诉 Agent 当前可执行的操作、是否需要确认、失败后如何恢复。

最终输出的任务现场应包含的关键要素如表 4-5 所示。

表 4-5 任务现场的关键要素及样例

任务现场关键要素	内容	作用
当前任务目标	当前任务目标	让 Agent 不偏离任务主线
关键对象与版本	当前文档、章节、文件、会议、表格等对象及版本关系	避免对象混淆和版本错误
任务阶段与阻塞	当前处于构思、修改、审核、执行还是交付阶段	判断下一步应追问、建议、执行还是等待
用户确认与开放问题	哪些判断已确认，哪些问题仍未定	区分事实、假设和候选方案
约束及行动边界	当前可读、可写、可调用、需确认和禁止的动作	支撑受控执行
证据回溯	来源、时间、对象、权限、置信度和回执	支撑解释、审计、复盘和记忆筛选

4.5 Coactive 的环境底座：共享现场、主动触发与经验沉淀

环境感知的最终价值，在于让 Coactive 从一种协作理念变成可运行的系统结构。Coactive 不是让 Agent 在聊天框里更积极地回应，而是让人和 Agent 进入同一个任务现场：Interactive 可以在现场中同频思考，Proactive 可以在现场中受控执行，基于毕业线判断能力是否成熟。

因此，环境感知在 Coactive Agent 中承担三重作用：为 Interactive 提供共同现场，为 Proactive 提供触发依据与行动边界，为任务提供是否“毕业”的证据，见图 4-2。它把屏幕、文档、会议、应用事件、操作轨迹和工具反馈组织为同一个任务现场，使“共同活动”有共同事实基础。

对 **Interactive** 来说，环境感知让 Agent 能更自然地参与人的思考过程。用户无需反复解释“我正在改哪一段”“刚才否定了什么”“这份材料与哪次讨论相关”，屏幕、选区、版本、会议讨论、停顿、修改和确认都成为同一任务现场中的信号。Agent 的追问因此可以更少、更准，反馈也更贴近当前任务。

对 **Proactive** 来说，环境感知提供主动性的触发条件和边界条件。Agent 不应凭空主动，而应根据任务状态、对象变化、时间节点、用户偏好和授权规则判断是否需要行动。例如，会议中形成明确待办时记录，用户反复执行同一类整理动作时建议沉淀为技能，文档引用缺少来源时提醒补证，长程任务到达节点时汇报进展。主动性由环境信号、主体判断和授权边界共同产生。

对毕业线来说，环境感知决定哪些经验具有沉淀价值。不是所有现场信息都应该进入记忆。高质量的环境感知需要区分临时噪声、任务状态、用户偏好、稳定流程和可复用技能。它既是记忆写入的前置过滤器，也是未来主动协作的证据来源。

图 4-2 环境如何同时支撑 Interactive、Proactive 与 毕业线

本节最重要的范式是工作的连续性：前台 Interactive 中被确认的目标、边界和修改，会直接约束后台 Proactive 的下一步；后台执行产生的结果、异常和新材料，又会回到同一任务现场，成为下一轮交互、记忆写入或能力升级的证据。共同现场不等于共同权限。环境可以把事实共享给人和 Agent，但哪些动作可执行、哪些内容可记忆、哪些结论可外发，仍必须由智能主体在治理机制下作出判断。

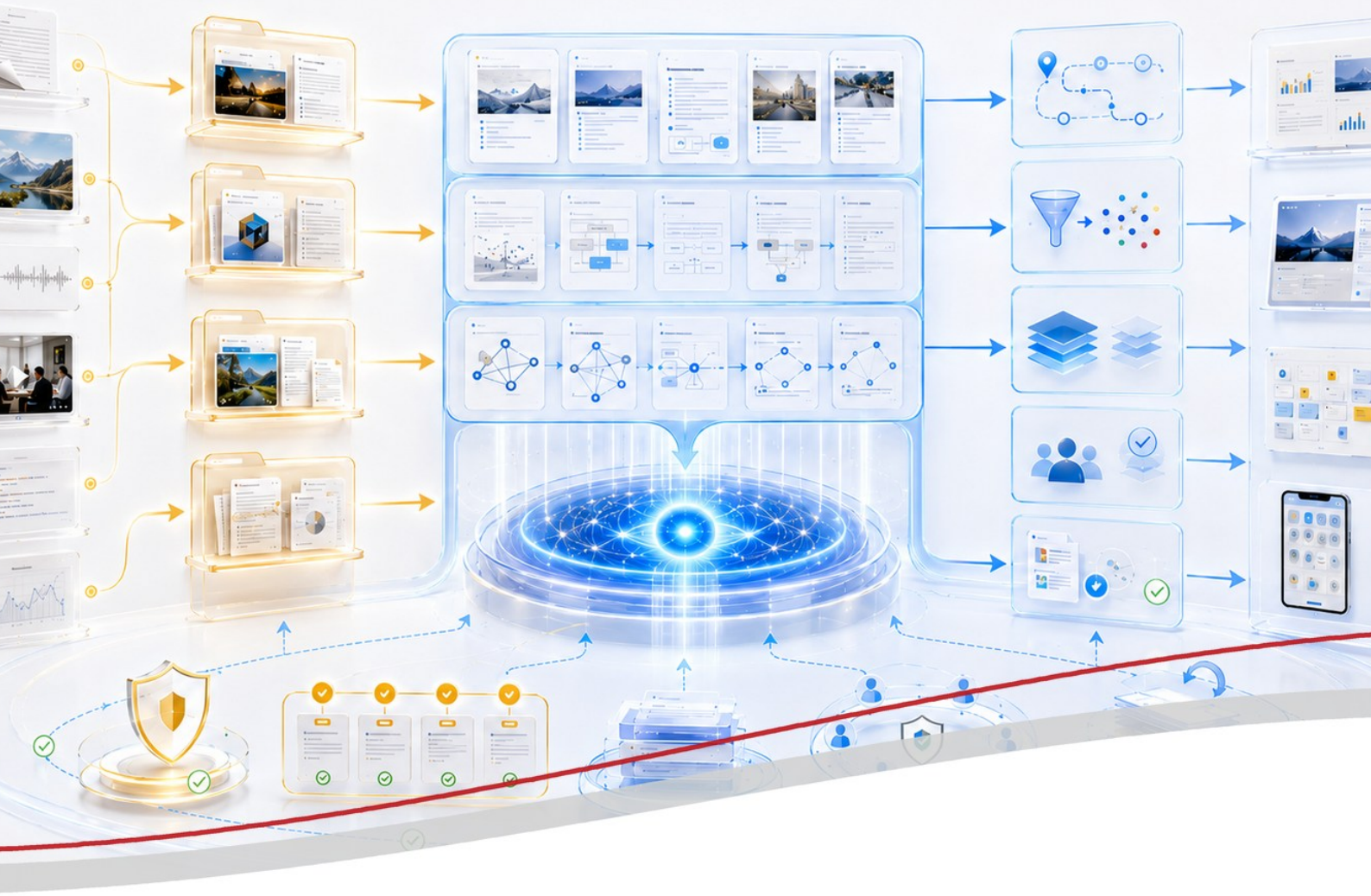
4.6 本章小结

环境感知是 Coactive Agent 进入真实工作的第一层能力。它把聊天框之外的设备、软件、文件、会议、操作、时间和任务状态组织起来，让 Agent 能够理解用户所处的任务环境。

从系统角度看，环境感知为智能主体提供实时上下文、可行动边界和结果证据，为认知底座提供可沉淀材料；从用户角度看，它减少重复解释背景的负担，让人与 Agent 能够面对同一个任务现场进行协

作。由此，第四章交付给后续系统的并不是一份庞大的数据集合，而是一套经过任务约束的运行条件：当前共享什么事实、开放什么动作、保留什么证据。这个输出足以支撑主体协作，又不会提前替代主体的判断职责。

> **CHAPTER TAKEAWAY**
>
> **环境感知让人和 Agent 面对同一个真实、可控、可追溯的任务现场。**



第五章 认知底座：Agent Memory

Agent Memory

Agent Memory Runtime 是支撑 Agent 长期协作的认知运行时。它不是任务结束后的归档或生成前的一次检索，而是贯穿交互、执行、沉淀与进化的系统能力，使长期经验能够可信、低成本、低打扰、可审计地被使用。

5.1 记忆成为运行时

Agent Memory 的行业讨论已经不再停留在简单知识库或 RAG。Generative Agents 将 memory、reflection 和 planning 组织起来，让智能体能够基于过往经验形成连续行为；MemGPT 则把记忆视为一种虚拟上下文管理问题，通过不同记忆层级之间的调度突破上下文窗口限制[18][19]。这些工作共同说明：长期协作需要的不只是“查到一段相关文本”，而是一套贯穿执行、沉淀和反馈的运行机制。

对 Coactive Agent 来说，记忆首先不是一个外置存储，而是人与 Agent 共同活动中的运行时能力。Interactive 需要记忆在用户表达过程中提前准备项目背景、历史判断和偏好；Proactive 需要记忆为后台执行提供依据、边界和可追溯记录；可信承接需要记忆把一次任务中稳定下来的目标、质量标准和工具路径沉淀下来；元进化则需要记忆把任务成败、用户修正和系统反馈变成后续优化的信号。

因此，记忆要回答的不是单一检索问题，而是一组运行时问题，即如何让 Agent 在正确时机、正确边界内使用长期经验。记忆成为运行时，意味着高频更新：环境感知在变、任务状态在变、用户判断在变、智能主体自身的经验在变，记忆必须以接近任务节奏的频率被写入、修正和作废。当前存在两条路线：参数化记忆将经验固化进模型权重，通过训练、微调或偏好学习吸收过往——参数化记忆具备流畅的泛化能力，但它无法回答一条判断从何而来，也无法在出错时回滚单条经验；明文记忆将经验保留为可读、可引用、可版本化的材料——写入即生效，修正可定位到条目，来源可追溯，权限与边界可治理。运行时对更新频率、可修正性与可审计性有更高的要求，因此明文记忆更适合成为支撑长期协作的运行时底座；参数化记忆则更适合承载低频变化的稳定能力。

5.2 Interactive Memory：让记忆进入实时交互

Agent Memory Runtime 与传统记忆系统最大的差别，不是“记得更多”，也不是“检索更准”，而是记忆进入运行时，参与实时交互。

传统记忆通常像一个后台资料库：用户说完需求之后，系统再理解意图、查询索引、拼接上下文，然后调用模型生成回答。这种方式本质上是事后检索，适合问答式的资料查找和背景补充，却很难支撑 Coactive Agent 所需要的实时共处——当记忆只在用户提交完整指令后才被调用时，Agent 总是慢半拍：人已经在表达、判断和行动，记忆却还停留在后台等待触发。

运行时记忆改变的是时间轴：它让记忆在用户输入、编辑、说话、切换页面、调用工具和推进任务的过程中就开始工作。如图 5-1 所示，传统记忆调用是一条串行链路，记忆查询、上下文拼接和模型预填充都发生在用户完成输入之后；Interactive Memory 则把用户输入看作连续的 input stream——随着 input 1、input 2、input 3、input 4 逐步出现，系统同步判断候选意图，提前预取项目背景、历史决策、应用对象、权限规则和用户偏好，并在语义相对稳定时进行 prefill 或等价的 KV Cache 预准备。它的关

键不是单个检索器更快，而是把原本发生在提交之后的记忆查询和模型准备，折叠进用户表达过程中的时间间隙，从而降低首 token 延迟和第一次有效反馈的等待感。

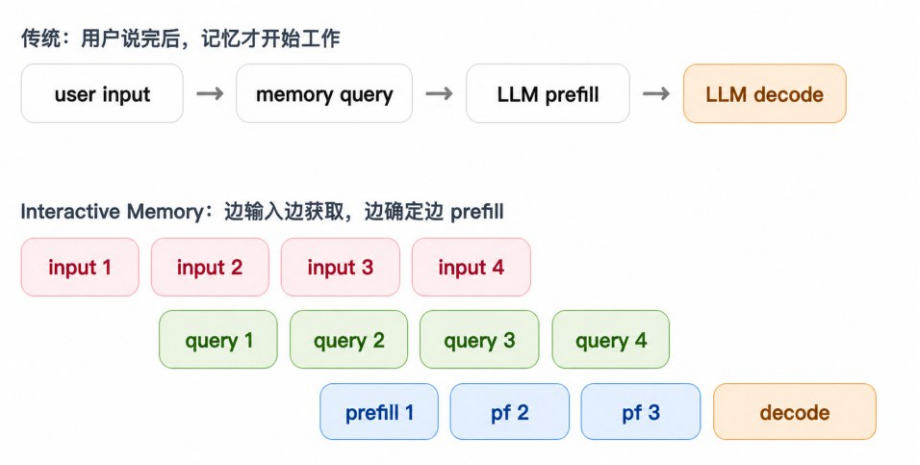


图 5-1 传统记忆与交互式记忆的对比

Interactive Memory 的核心逻辑在于，记忆不再只是回答前的补充材料，而是协作过程中的实时参与者。传统记忆像"查档案"，只有用户发问，记忆才被召回；Interactive Memory 更像“在场协作者”，能在用户形成判断的过程中补证、提醒、推荐、追问，甚至纠偏。例如，当用户正在写一句对外承诺时，Agent 可以提示历史口径是否一致；当用户引用一个客户案例时，Agent 可以检查是否授权、是否需要脱敏；当用户表达含混时，Agent 可以先追问项目、对象或版本，而不是等生成错误结果后再事后纠正。

因此，运行时记忆区别于传统记忆的核心关注点是"记忆什么时候出现"：有些记忆应该在输入过程中预取，有些应该在计划生成前进入上下文，有些应该在工具调用前装载，有些只应作为可追溯来源保留。但实时交互不意味着记忆可以无边界地抢先介入。输入过程中的预取带有投机性，系统必须能够丢弃错误候选、合并重复结果、避免把未经授权的记忆提前暴露给模型，并在不确定时等待更多上下文或向用户澄清。好的运行时记忆，应该既敏捷又克制。

用户最终感受到"Agent 正在和我同步理解"，就是记忆进入运行时的作用：传统记忆服务于一次回答，运行时记忆参与一段协作。对于 Coactive Agent 来说，记忆只有进入实时交互，才真正从后台能力变成共处能力。

Runtime Coordinator。Interactive Memory 并不是一项孤立的缓存技巧，而是记忆与执行在时间轴上咬合的一个实例。将它一般化，就得到 Agent Memory Runtime 的通用协同层——Runtime Coordinator。它统筹的对象不止用户输入流与记忆预取，还包括模型服务的 KV Cache 与 prefill 窗口、沙箱的初始化与工作空间挂载、工具调用前的对象与权限装载，以及多 Agent 协作中的记忆交接时序。这些场景面对的是同一形态的问题：让正确的记忆，在正确的时机，以正确的粒度，进入正确的执行环节。Interactive Memory 是这一机制在实时交互场景下最直观的呈现；同样的协同，也发生在每一次后台任务启动、每一次沙箱创建与每一次 Agent 交接之中。

5.3 原文为终审：明文记忆的机制

通读当前的明文记忆方案，我们发现并不存在一种单一、优雅、永远最优的记忆解法。明文记忆工作能总结的一个现实是：真实工作中的记忆召回会长期依赖多路机制组合。文件路径提供确定地址，全文检索提供精确匹配，向量召回提供语义相似，图关系提供多跳关联，时间线提供状态演进，结构化状态提供任务约束，应用对象和权限策略提供操作边界。不同任务、不同应用、不同 Agent 所需要的线索并不相同，系统很难用一种统一检索器解决所有问题。

这不是明文记忆路线的缺陷，而是它必须面对的工作现实。企业级协作里的记忆不是均质知识库：一个表格公式、一次浏览器操作、一条用户确认、一段工具调用日志，都可能在不同任务里成为依据。召回精度可以依赖多路索引不断提升，但这些索引本身不应该成为不可挑战的事实本体。因此，本章提出一个核心判断：

> **原文为终审，索引为派生结构。**

原文不只包括用户显式输入的文字，也包括环境感知系统传来的各种环境信息。它们是 Agent 记忆的原始材料，也是发生争议、冲突、过期或低置信时可以回到的事实锚点。系统可以基于这些原始材料生成摘要、标签、向量、图谱、时间线、任务卡、用户偏好和流程片段，但这些派生结构只作为索引层、加速层和解释层，不能替代原始材料本身。

因此我们对于一个统一的记忆存储基座持怀疑态度。一个成功的 Agent 产品，往往会把自己的任务目标、交互方式、工具使用、质量标准和主动执行边界一起写进记忆机制里。两个成功 Agent 未必能共用同一套派生记忆结构：同一段会议记录，在办公 Agent 看来可能是待办事项，在研发 Agent 看来可能是需求变更。它们对“什么重要”“如何抽取”“何时复用”“能否主动执行”的判断都不同。所以，跨 Agent、跨平台真正稳定可共享的，不是向量库、图谱、summary、偏好画像或某种统一抽取模板，而是可读、可迁移、可重新解析的原始材料，以及围绕这些原始材料建立的地址、来源、权限、版本和引用机制。因此明文记忆难以统一记忆接口，也难以统一派生结构，但原文是可以共享的媒介。

沿着“原文为终审”的判断，选择明文记忆后首先要解决三类系统问题。第一，是多种检索方式的一致性：同一段经验会被摘要、向量、图谱、时间线和结构化状态同时索引，系统必须知道这些派生结构来自哪一批原文、是否同步更新、发生冲突时以什么为准。第二，是记忆的隔离：不同应用、不同项目、不同权限域中的经验不能被混入一个平面池，否则召回越丰富，越可能带来噪声、越权和误用。第三，是记忆的共享：主 Agent、子 Agent、沙箱和协作 Agent 需要共享必要上下文，但这种共享不能依赖把所有历史复制进 prompt，而应在最小暴露、可审计引用和权限边界下完成。

下面以我们的实现为例，说明如何提供一个以原文为终审的 Memory Base，以及这三类问题如何在机制层被解决：文件系统承载原文，一致性保存点统一多路索引，应用级记忆完成认知空间隔离，软链接工作空间完成共享与权限治理。

5.3.1 Memory Base：把原文写成一种文件系统规格

既然原文是终审依据，Memory Base 的第一项职责就是承载原文。Coactive Agent 的协作关系不能只存在于模型上下文里。模型上下文是短暂的，任务现场却是长期的；单次会话是局部的，用户工作却跨越多个项目、应用与时间段。因此，Agent Memory Runtime 需要一个稳定的共同载体，将 session、project、resource、app 等上下文原文组织起来。

我们选择文件系统作为这一载体，并将其抽象为 Agent Memory Filesystem。它不是普通文件存储，而是一套人和 Agent 都能读写、引用、版本化和治理的记忆规格。文件系统为长期协作提供了最朴素也最可靠的共同载体：以稳定地址组织 session、project、resource 的上下文原文，以目录表达归属，以 Markdown 与 frontmatter 承载人和 Agent 都可解析的结构，以链接表达引用关系。

业界也已经出现以目录、Markdown 与结构化元数据表达可迁移知识的探索。Google Open Knowledge Format 用 Markdown 文件、YAML frontmatter 和 Markdown 链接表示知识，使知识既能被人阅读，也能被 Agent 解析和迁移[20]。但 Agent Memory Filesystem 面向的不是单个文件，而是长期协作中的经验、状态、依据与边界所形成的目录组织规范，一种文件系统规格。例如，以 session、project、resource、workspace、apps、common tools 、common skills 切分目录。最终目的是原始材料有稳定地址，派生结构有来源指针，Agent 和人都能回到原文判断一条记忆是否仍然有效。

5.3.2 一致性保存点：以 Commit 统一多路索引

明文路线的多路检索带来一个必须正面解决的系统问题：同一段经验会同时投影到摘要、图谱、向量、时间线、结构化状态等多个派生结构中。若这些派生结构各自独立更新，就可能出现不一致：图查到的内容与结构化状态查到的内容互相矛盾，摘要仍保留旧判断，时间线却记录了后续修正；向量索引已经召回某段经验，权限策略却不允许在当前任务使用。此时系统不能让多个派生结构互相争夺事实地位，而必须回到原文和保存点。

为此，重要记忆的写入应形成一致性保存点。保存点做两件事：第一，以原文为依据生成阶段性摘要；第二，以同一批原始内容同步更新所有派生索引，包括图谱、向量、时间线、结构化状态等。两者完成后，系统写入一条提交记录，说明本次写入的范围、依据与影响。这一机制在理念上接近版本管理中的提交，也接近分布式事务中的两阶段提交。关键不在于具体工具，而在于原始变更、派生更新与写入原因被绑定为同一个可审计事务。它回答的是：哪些原始材料发生了变化，哪些派生结构被更新，更新依据是什么，来源指针在哪里，谁或什么机制触发了写入。

由此，记忆可以被审计、回滚、恢复与实验；派生结构也可以在必要时被重建，而不是沉淀成无法挑战的黑箱。记忆从“模型好像记得”变成“系统知道它如何记得”。

5.3.3 应用级记忆

真实工作中的记忆不是均质的。文档、演示、表格、浏览器、邮件与 IDE 具有不同的对象、工具、权限与质量标准。PPT 里的“风格不错”可能指叙事结构和版式，表格里的“这里不对”可能指公式、数据源或口径，IDE 里的“按上次方式处理”可能指代码路径、测试命令或部署习惯。表格场景沉淀的工具经验与技能片段，不能直接搬用到演示场景。

因此，每个应用都应拥有自己的认知空间。应用级的对象、工具经验、技能与偏好，应存放在专属的持久化目录结构之下，并通过 app、permissions 等字段保留归属与访问边界。系统不仅要知道“这是一条有用经验”，还要知道它来自哪个应用、关联什么对象、依赖什么工具、受什么权限约束、适用于什么质量标准。

每个应用有独立的记忆认知空间让跨应用协作成为可控连接，而不是把所有记忆混入一个平面记忆池后的无边界混用。写 PPT 时可以引用会议结论和用户偏好，但不能误用表格里的计算规则；浏览器里的资料可以支持写作，但不应自动突破文档权限；IDE 经验可以帮助工程任务，但不应被泛化成办公任务的默认策略。

5.3.4 软链接工作空间：记忆共享与权限治理

长期协作还要求记忆在多个执行主体之间共享：主 Agent 与子 Agent 之间，Agent 与沙箱任务之间，协作的多个 Agent 之间。常见做法是把相关内容拼入上下文传递，代价是上下文膨胀、语义稀释与重复拷贝。对于一个明确任务，更好的方式是创建一个受控工作空间，只把相关 session、project、resource、app 记忆以引用方式暴露给当前执行环境。

Agent Memory Filesystem 可以利用文件系统的软链接构造工作空间：为一项任务创建 workspace 目录，将相关的原文 session 与 project 以软链接挂入。原文变动时，链接内容随之更新，且不产生额外拷贝；权限治理随之变得简单清晰，处理某一工作空间的沙箱，只能看到该空间之下的内容，其余记忆天然不可见。

软链接工作空间的价值，不在于某个文件系统技巧本身，而在于它提供了一种受控引用机制。工作空间不是记忆复制，而是对相关原始材料的任务级引用集合。共享与隔离由同一个机制完成：sub-agent 或 sandbox 能获得完成任务所需的上下文，同时无法越过 workspace 看到主 Agent 的全部历史。这样，记忆共享不再依赖把所有东西塞进 prompt，而是通过最小必要暴露和可审计引用完成。

以“原文为终审”这一原则的机制展开——文件系统提供稳定地址，一致性保存点保证多路派生结构可以被审计和重建，应用级记忆保留对象与权限边界，软链接工作空间让多 Agent 和沙箱在最小暴露下共享必要原文。明文记忆的地基是建立一套围绕原始材料的治理机制：原文可以被读取，派生可以被重建，来源可以被追溯，边界可以被控制，错误可以被修正。在这个地基之上，原始材料稳定存在、派生

索引可以替换，系统才需要进一步决定当前任务应该写什么、读哪里、何时读，以及以什么粒度把记忆交给模型和工具。

5.4 记忆的读写：Memory Router 与 Compiler

当记忆被写成可读、可追溯、可治理的明文材料之后，新的问题随之出现：记忆源会越来越多，检索方式也会越来越多。系统可能同时拥有文件路径、全文检索、向量召回、图谱关系、时间线、结构化状态、应用对象、用户偏好和组织策略。单一检索器无法承担这样的复杂度，把所有结果粗暴塞进上下文也只会制造噪声和成本。

明文记忆一旦变得丰富，核心问题就变成了“当前任务应该读哪类记忆、从哪里读、什么时候读、以什么粒度交给模型”。因此我们的判断是，明文记忆路线并不是押注某一种“最强检索器”，而是承认真实工作需要多路检索协同：文件路径提供确定地址，全文检索提供精确匹配，向量召回提供语义相似，图关系提供多跳关联，时间线提供状态演进，结构化状态提供任务约束，应用对象和权限策略提供操作边界。Agent Memory 的关键创新不在于再增加一种检索方式，而在于让 Memory Router 把这些检索方式组织成任务相关的调度机制，在准确性、成本、权限和时机之间做出取舍。

因此，Agent Memory Runtime 需要两个运行时角色：

Memory Router 负责选择路径。它要回答三个问题：怎么写、去哪读、何时读。一次交互结束后，哪些内容只是临时上下文，哪些应该写入项目记忆，哪些应该写入用户偏好，哪些应该写入应用边界，哪些应该沉淀为稳定流程，哪些只应保留原文但不生成派生结构？同样一句话，在不同任务中含义不同。用户说“这版风格不错”，可能只是对当前 PPT 的反馈，也可能是对某类客户汇报风格的长期偏好。Router 必须结合任务、项目、应用、权限和确认机制判断写入层级。

读取也是如此。当前任务需要用户长期偏好、项目历史、应用记忆、组织知识、工具经验还是外部资料？这些来源的可靠性、权限和时效不同，不能只按相似度排序。记忆也不总是在用户提交完整指令后才有用：在用户输入过程中，可以根据前缀和语义片段提前预取候选记忆；在计划生成前，需要读取目标和约束；在工具调用前，需要读取应用对象和权限规则；在任务完成后，需要读取写入策略来决定如何沉淀。

Memory Compiler 负责组织结果。它的核心不是总结得更短，而是让记忆以正确粒度进入运行时。某些内容应该进入 prompt，某些内容适合进入 prefill 或缓存，某些内容只需要保留为可按需查询的索引，某些内容则不应进入当前任务。好的编译器会减少 token 成本、降低延迟、抑制噪声，同时保留回到原文的能力。

例如，当用户说“按照上次给投资人那版逻辑重新整理这个 PPT”时，一个好的运行时不会只检索“投资人”和“PPT”两个关键词。Router 应定位“上次给投资人那版”对应的项目和版本，读取 PPT 相关的叙

事结构、版式偏好和修改历史，检查当前文件对象和权限边界；Compiler 再把这些材料组织成当前任务可用的上下文包。如果系统无法确定“上次”指哪一次，就应提出澄清，而不是自信地使用错误历史。

Router 与 Compiler 让记忆从被动检索变成主动调度。它们是 Interactive 与 Proactive 之间的关键连接器：前台交互中形成的判断，经由路由被写入合适位置；后台执行时需要的经验，又通过路由和编译被取回到合适时机。

5.5 Self-Improve：记忆在使用中自我优化

长期协作中，记忆面对的世界并不静止：材料在变、判断在更新、边界在移动。昨天有效的检索组合，今天可能因项目阶段变化而失准；沉淀下来的流程片段，会随工具与应用的演进而过时。因此 Agent Memory Runtime 的最后一块拼图，是让记忆在使用中持续自我优化——Memory Self-Improve。

优化的信号来自使用本身：任务成败、用户修正、检索命中与落空、工具调用失败、计划被打断、输出被采纳或废弃。这些信号回流之后，作用于三类对象：路由策略——哪类任务应读哪些来源、写入哪个层级；编译策略——什么内容以什么粒度进入上下文；触发策略——预取与主动执行在什么条件下启动。

需要强调的是，自我优化的对象是派生结构与调度策略，而不是原文本身。原文作为事实基准不被触碰，这使得任何一次优化都可审计、可回滚、可重新实验——Memory Self-Improve 因此是受治理的进化，而非不可控的漂移。

Memory Self-Improve 也是元进化闭环在记忆子系统的落点：交互与执行产生反馈，反馈优化记忆的组织与调度，更好的记忆又支撑更高质量的交互与执行。记忆由此不只是被使用的资产，而是随协作一起变强的能力。

5.6 Agent Memory Runtime 架构总览

原文是记忆的基础，索引只是派生——派生结构天然具有多样性与变化性，会随任务形态、应用场景与检索技术的演进而不断更替。因此，在我们的实践系统中，重点关注：原文文件存储规范（Agent Memory Filesystem）、Memory Router、Memory Self-Improve 与 Runtime Coordinator。Agent Memory Runtime 的整体架构如图 5-2 所示。

自下而上看，存储底座以文件系统为事实基准，承载原文、工作空间与应用级记忆；向量、知识图与时序作为派生索引与之并列，可重建、可替换。实现层由四个关键组件构成：Content Manager 管理原文与版本，并以一致性保存点绑定摘要与多路索引的同步更新；Memory Router 负责写入分层、读取选路与时机调度，其下游的 Memory Compiler 完成粒度编译；Memory Self-Improve 依据使用反馈优化路由、编译与触发策略；Runtime Coordinator 统筹输入流、记忆查询、KV Cache 与 prefill、沙箱初始化与多 Agent 交接的时序，Interactive Memory 是实时交互场景下一个核心实现。向上，文件系统挂

载、API 与 CLI 构成统一的核心接口，支撑 Session / Workspace、Agent 插件、浏览器插件与模型网关等接入方式，最终向用户呈现为认知反馈与协作沉淀等外部能力。原文为真相、派生结构为索引的核心原则贯穿全部层次。

图 5-2 Agent Memory Runtime 运行时架构

5.7 认知底座如何支撑 Coactive

Agent Memory Runtime 是 Coactive 范式成立的关键。

对 **Interactive** 来说，记忆让实时交互不再从空白开始。用户不需要反复解释项目背景、个人偏好、历史决策和应用状态；Agent 可以在用户表达过程中同步准备上下文，在不确定时提出更好的问题，在共创时提供更贴近历史的材料和结构。

对 **Proactive** 来说，记忆让后台执行有依据、有边界、可追溯。Agent 可以根据已沉淀的流程、工具经验、应用对象和授权策略主动推进稳定任务，同时保留可审计记录和用户介入入口。

对**“毕业线”**来说，记忆是任务从 Interactive 走向 Proactive 的沉淀机制。一个任务能否被可信承接，取决于系统是否已经积累了足够稳定的目标、判断标准、工具路径、应用对象、质量边界和用户偏好。没有记忆，就没有可复用经验；没有可复用经验，Proactive 就只能靠模型临场发挥。

对 **元进化** 来说，记忆提供了反馈基础。任务成败、用户修正、检索命中、工具调用失败、计划被打断、输出被采纳或废弃，都可以成为后续优化记忆路由、编译策略和主动触发机制的信号。

> Agent Memory Runtime 的价值，不是让 Agent 记得更多，而是让 Agent 知道什么值得记、何时使用、如何证明、如何修正，并如何在长期协作中变成更稳定的能力。

5.8 本章小结

Agent Memory Runtime 的价值不在于保存更多信息，而在于以可信、低成本、低打扰、可审计的方式使用长期经验。原始材料构成事实锚点，派生结构负责索引与加速，Router 决定何时读写何种记忆，Compiler 把记忆组织成适合当前任务的上下文，从而支撑 Interactive、Proactive 与持续进化。

> **CHAPTER TAKEAWAY**
>
> **记忆的价值不在于记得更多，而在于在正确时机以正确方式被使用。**

第六章 元进化闭环：Coactive Agent 的持续成长机制

META-EVOLUTION

当 Agent 已经能够进入真实现场、与人交互、主动执行并沉淀经验之后，系统还需要从每一次协作、失败、反馈与成本浪费中持续成长。AgentReflex 作为运行时之上的元进化控制面，把轨迹、评价、诊断、变异、选择与复用组织成可治理的成长闭环。

当 Agent 进入真实任务现场之后，它面对的环境不再是静态问题集，而是持续变化的任务、用户、工具、流程和组织规则。一次成功的任务执行，只能证明 Agent 在当前条件下完成了目标；能否在下一次相似任务中表现更稳定、在新的任务分布下快速适应、在失败之后形成可复用经验，才是企业级 Agent 走向规模化的关键。

Coactive Agent 的价值，不止于让 Agent 与人共同完成任务，更在于让这段协作关系持续成长。环境感知让 Agent 看见现场，智能主体让 Agent 与人协作并推进任务，认知底座让经验得以沉淀；但真正决定 Agent 能否长期演进的，是一套能够自我更新的元进化机制。它不仅驱动 Agent 的行为能力持续提升，也持续改造“如何观测、如何评估、如何生成改造、如何选择保留、如何沉淀复用”这些进化机制本身。

本章提出 AgentReflex 作为 Coactive Agent 的元认知进化机制。AgentReflex 的核心机制，是在面向任务完成的运行回路之外，建立一个面向系统成长的第二控制回路。它位于 Agent Runtime、业务系统和认知底座之上，以真实任务轨迹、用户反馈、评估结果、成本消耗和失败模式为输入，将 Agent 的运行过程转化为可观测、可评估、可改造、可复用的进化资产。更重要的是，AgentReflex 将可观测、评估、Harness、记忆和进化资产本身纳入进化对象，使 Agent 系统从“能力可进化”进一步走向“进化机制可进化”。最终达成让 Agent 的每次变化都有证据、经过评估和选择、支持灰度与回滚，并保留版本谱系。由此，Agent 不仅能够在协作中变好，也能够持续改进发现问题、评价变化和沉淀经验的方法。本章核心观点是：

> **Coactive 让 Agent 与人共同工作，AgentReflex 让 Agent 在共同工作中持续成长，元进化让驱动成长的机制本身持续成长。**

6.1 从持续改进到元进化

企业级 Agent 的规模化应用，会把系统从“能不能完成一次任务”的阶段，推向“能不能持续稳定交付”的阶段。在试点阶段，Agent 能够写报告、查知识、改代码、跑流程，就足以证明技术可行；但进入真实生产环境后，任务类型会不断扩展，工具接口会持续变化，组织规则会动态调整，用户偏好也会在长期协作中逐步显现。此时，单次任务成功率已经不足以衡量 Agent 的长期价值。

传统 Agent 系统容易停留在单次运行模式。一次任务成功了，经验没有稳定继承；一次任务失败了，教训没有转化为评估样本；一次用户纠正了，偏好没有进入记忆；一次成本浪费了，Loop 结构没有被改造。系统不断产生运行痕迹，却没有形成可被继承的成长机制。Agent 看似越来越能做事，实际仍然在大量相似任务中重复试错。另一方面，自我改进回路不只作用于记忆和技能，也作用于工作流结构，Aflow 等工作尝试探索 Agent 结构本身的搜索和进化[21]，这也是当前 agent 探索的重要方向。

Coactive Agent 的核心路径是“先协作，再沉淀；先可见，再授权；先低风险自动化，再高价值主动承接”。这一路径天然要求系统具备持续成长能力。复杂任务中的目标、边界、偏好和判断标准，往

往无法在初始指令中一次性表达完整，必须通过交互澄清、执行反馈、结果修正和经验沉淀逐渐形成。只有当这些过程被系统性吸收，Agent 才能从“完成任务”走向“理解工作方式”，再进一步形成可授权的主动能力。

元进化的关键，在于把成长机制本身纳入进化范围。Agent 的第一层进化发生在行为层，表现为交互方式更自然、工具调用更准确、记忆策略更有效、任务执行更稳定、Token 使用更经济。第二层进化发生在机制层，表现为观测对象会根据新的失败模式扩展，评估样本会根据真实任务持续生长，Harness 改造策略会根据历史验证结果调整，进化资产会根据适用边界和效果反馈进行升级、降级或退役。

因此，元进化不是简单的自动优化，也不是让 Agent 无边界地修改自身。它是一种可治理的成长机制：以真实证据驱动变化，以持续评估形成选择压力，以灰度和回滚控制风险，以谱系管理沉淀经验，以机制自我修正适应新的任务环境。AgentReflex 的目标，正是让 Agent 系统不仅能够变好，而且能够越来越会变好。

6.2 AgentReflex 的元进化架构

AgentReflex 可以理解为 Coactive Agent 的第二控制回路。第一控制回路面向任务完成：Agent 感知环境，与人交互，调用工具，执行任务，沉淀经验。第二控制回路面向系统成长：AgentReflex 观察任务协作过程，评价结果与过程质量，诊断失败和浪费，生成候选改造，选择有效变化，并将验证后的经验沉淀为可复用资产。

这套机制的核心不在于增加一个平台界面，而在于建立一组可持续运转的进化能力。它需要让运行过程可解释，让评价标准可生长，让改造策略可验证，让经验资产可继承，让进化机制可自我修正。只有当这些能力形成闭环，Agent 的每一次执行才不只是一次消耗，而是一次潜在的学习事件。

AgentReflex 的元进化架构可以概括为四个相互耦合的机制。

6.2.1 AgentReflex 的核心闭环

传统 AgentReflex 的核心闭环可以概括为如图 6-1 所示。这条链路不是一次性的线性流程，而是持续运行的数据飞轮和进化飞轮。它也不要求每次任务都完整触发重型诊断与改造流程。对于低风险、低价值、稳定运行的任务，系统可以只产生轻量观测和抽样评估；对于高价值、高风险、高失败率或高成本任务，系统则进入完整的诊断、变异、验证与沉淀流程。AgentReflex 需要在成本和收益之间建立分层机制：不是所有 Trace 都值得深挖，但所有关键 Trace 都应该可追溯。

Run 阶段，Agent Runtime 或 Loop Engine 执行真实任务。任务可能来自内容共创、企业知识问答、代码修复、会议纪要、跨应用办公、自动巡检、流程办理或多 Agent 协作。Agent 在任务过程中留下模型调用、工具调用、上下文装配、状态转移、验证结果和用户反馈。Run 的关键要求是事件可采集。

系统不要求 Runtime 使用某个固定实现，但需要输出任务 ID、Agent ID、Loop 版本、模型版本、工具调用、上下文来源、状态转移、验证结果和成本信息。没有这些最小元数据，后续评估、诊断和谱系管理就会失去基础。

图 6-1 传统 AgentReflex 的核心闭环

Observe 阶段，AgentLens 将这些事件结构化为可回放、可分析、可归因的行为轨迹。系统不仅记录发生了什么，还要解释 Agent 为什么这样行动，哪些上下文影响了判断，哪一步开始偏航，哪些工具调用带来了收益或浪费。Observe 的重点是把原始事件转化为行为语义。一次工具调用不只是“调用成功”，还应该被标注为“检索上下文”“验证假设”“修改状态”“恢复失败”或“输出交付物”。只有进入行为层，系统才能进行跨任务聚类、失败归因和经验复用。

Evaluate 阶段，AgentEval 把任务结果、过程质量、成本效率、安全约束和用户反馈纳入持续评估。它不仅关注最终答案是否正确，也关注过程是否稳健、行为是否可解释、成本是否合理、边界是否可控。Evaluate 的价值在于把主观反馈转化为可复用的选择压力。用户说“这次不好用”只是原始反馈，系统需要进一步判断问题来自正确性、完整性、风格偏差、上下文遗漏、工具误用、主动打扰，还是成本过高。不同问题会指向不同的进化方向。

Diagnose 阶段，系统识别失败模式、成本热点、质量回归和能力短板。某类任务失败可能来自上下文缺失，某类成本浪费可能来自重复工具调用，某类质量问题可能来自 Verification 过弱，某类长任务卡死可能来自 Stop Condition 失效。诊断的目标不是简单定位一个错误点，而是识别 Agent 行为链路中的结构性偏差，并将其转化为可以被改造的对象。

Mutate 阶段，AutoHarness 基于诊断结果生成候选变异。变异对象可以是 Prompt、Tool、Memory、Workflow、Context Layout、Router、Guardrail、Evaluation Hook、Stop Condition、Token Strategy 或主动触发规则。Mutate 的关键不在于自动修改一切，而在于生成受控、可验证、可

回滚的候选方案。每个候选变异都需要声明作用对象、预期收益、潜在风险、适用任务族、验证方法和回滚方式。这样 AutoHarness 才是工程化的进化组件，而不是不可控的自动改配置脚本。

Select 阶段，Eval Runner 对候选变异进行对比评估，并结合离线 Benchmark、线上灰度、成本指标、安全约束和用户反馈选择更优版本。Select 是 AgentReflex 形成选择压力的核心环节。变化本身不等于进步，只有通过真实或仿真的任务验证，并证明收益大于风险的变异，才值得被保留。对于企业级 Agent，Select 还需要支持灰度发布、人工审批、风险分级和快速回滚，确保进化过程始终处于治理边界之内。

Preserve 阶段，Evolution Genome 将通过验证的变异沉淀为 Reflex Gene 或 Reflex Capsule，并记录触发条件、适用环境、验证结果、收益指标、风险边界和谱系关系。Preserve 的价值在于把一次局部优化升级为可继承资产。一次有效的上下文排序调整、一次成功的工具选择策略、一个稳定的验证器、一个成本显著下降的 Token 策略，都可以成为未来任务复用的进化单元。

Reuse 阶段，Gene Reuse Engine 在新的任务、新的失败模式或新的成本热点出现时，检索历史进化资产，并推荐给新的 Agent 或新的任务族使用。到这一步，一个 Agent 的经验才真正变成组织级能力。复用不是简单复制，而是基于任务类型、模型版本、工具环境、业务边界和风险等级进行匹配、适配与再验证。

这个闭环的价值，不在于让系统自动改得更多，而在于让每一次改造都带着来源、证据、验证和继承关系。AgentReflex 追求的是可治理的进化：每一次运行都可观察，每一次评价都可解释，每一次变异都可验证，每一次沉淀都可追溯，每一次复用都可审计。

6.2.2 可观测自进化：从固定指标到动态认知

传统可观测面向相对稳定的软件系统，观测对象通常是服务、接口、日志、指标、链路和资源消耗。Agent 系统的行为边界更加开放，失败也更难归因。一次看似失败的输出，可能源于目标理解偏差、上下文选择错误、工具调用路径不当、记忆污染、验证条件过弱、多 Agent 交接失真，或者主动触发时机不合适。固定指标只能覆盖已知问题，很难解释不断涌现的新型失败模式。

可观测自进化要求 AgentLens 不断调整“看什么、怎么看、如何归因”。当系统发现某类失败无法被现有指标解释时，观测对象应随之扩展。例如，从工具调用成功率扩展到工具选择理由，从 Token 总量扩展到重复上下文率、无效重试率和缓存不友好片段，从任务完成率扩展到用户修正密度、主动建议采纳率、人机往返轮次和任务中断点。观测体系的价值，不在于记录更多数据，而在于持续提升系统发现问题、解释问题和定位问题的能力。

不同任务场景也需要不同观测模型。内容共创任务更关注结构演化、风格一致性和用户修正轨迹；研发任务更关注上下文命中、代码影响面、验证闭环和回归风险；企业办公任务更关注跨应用状态、权

限边界和主动触发准确性；多 Agent 协作任务更关注委派深度、子任务交接质量、协同成本和责任归因。随着任务形态变化，观测体系也必须同步调整。

因此，可观测不是元进化的外部工具，而是元进化机制自身的一部分。它既观察 Agent，也在真实失败和反馈中更新自己的观察方式。

6.2.3 评估自进化：从静态测试到持续选择压力

评估决定进化方向。没有评估，系统只能产生变化，却无法判断变化是否真正有效。传统 Eval 多依赖固定测试集、固定 Rubric 和发布前抽样，适合做版本门禁，却难以覆盖企业真实环境中的长尾任务、隐含偏好、新工具、新流程和新风险。

评估自进化要求 AgentEval 从静态测试集升级为持续生长的选择压力系统。线上 Badcase、用户反馈、人工 Review、安全事件、高价值成功轨迹，都应成为评估样本池的来源。当某类失败反复出现，系统需要将其转化为新的 Eval Case；当用户持续修正某类表达或输出结构，系统需要将其沉淀为新的 Rubric；当 Agent 虽然得到正确结果却过程绕路、成本过高或风险失控，评估标准也要从结果正确扩展到过程稳健、成本合理、边界可控。

评估标准的演化，直接影响 Agent 能力的演化。白皮书共创任务的评估，可以从结构完整进一步扩展到论证连贯、概念一致、风格统一和引用可信；研发任务的评估，可以从代码可运行扩展到影响面可解释、测试覆盖充分、变更风险可控；企业办公任务的评估，可以从任务完成扩展到权限合规、打扰可控和跨应用状态一致。随着 Agent 能力提升，选择压力也需要从单一结果质量，扩展到过程质量、成本效率、安全边界、用户体验和长期可维护性。

AgentEval 的角色由此发生变化。它不再只是发布前的质量门禁，也成为系统持续进化的方向盘。它决定哪些变化值得保留，哪些变化应当回滚，哪些能力可以获得更高授权，哪些主动行为必须重新退回人与 Agent 的交互过程。

6.2.4 Harness 自进化：从人工调优到受控变异

Harness 是模型之外所有影响 Agent 行为的外部结构，包括 Prompt、工具描述、上下文布局、记忆读写、Workflow、Router、Guardrail、验证器、停止条件、权限策略、Token 策略和主动触发规则。Agent 能力能否稳定落地，很大程度上取决于这些外部结构是否能够适应真实任务的复杂性。

Harness 自进化的核心，是让系统持续学习“什么样的改造在什么条件下有效”。同样是上下文遗漏，有些场景适合增加检索召回，有些场景适合调整上下文排序，有些场景需要引入工作空间状态，有些场景则应该要求 Agent 先澄清目标。同样是 Token 成本过高，有些场景需要压缩 Prompt，有些场景需要减少无效工具调用，有些场景需要改造多 Agent 交接格式，有些场景需要调整缓存友好的上下文布局。有效改造并不存在放之四海而皆准的固定模板，必须结合任务类型、模型能力、工具环境和风险边界进行选择。

AutoHarness 的价值不是自动修改配置，而是生成带有证据、边界和验证计划的候选变异。每一个候选改造都需要说明来源于哪类失败或浪费，预期解决什么问题，可能影响哪些任务和 Agent，需要用哪些 Eval Case 验证，以及效果不佳时如何回滚。经过评估、灰度和人工治理之后，真正有效的改造才会进入进化资产库。

随着历史变异不断积累，AutoHarness 自身也会进化。它会逐步学习不同失败模式与不同改造策略之间的关系，优化候选变异生成方式，调整验证策略，并改进进化资产的复用条件。系统不只是改造 Agent，也在学习如何更有效地改造 Agent。

6.2.5 进化资产自进化：从经验沉淀到能力谱系

通过验证的改造需要被沉淀为可复用资产。AgentReflex 将这些资产组织为 Reflex Gene 和 Reflex Capsule。Gene 是较小粒度的进化单元，可以是一条上下文排序规则、一个工具选择策略、一个记忆写入条件、一个验证器、一个 Token 优化技巧；Capsule 是较大粒度的能力封装，通常由多个 Gene 组合而成，面向一类稳定任务或一类复杂失败模式。

进化资产的价值，在于将一次局部优化转化为组织级能力。一个团队在白皮书共创中沉淀的结构推演、风格对齐和引用检查经验，可以封装为内容共创 Capsule；一个研发 Agent 在跨仓修改中形成的仓库理解、影响面识别和测试验证策略，可以封装为研发 Capsule；一个高成本任务中验证有效的上下文去重、缓存友好布局和多 Agent 状态压缩策略，可以封装为 Token Efficient Capsule。通过 Gene 和 Capsule，Agent 的经验不再停留在单次任务、单个专家或单个团队内部，而可以跨任务、跨 Agent、跨组织场景复用。

进化资产本身也具有生命周期。随着模型版本、工具接口、业务规则、用户偏好和任务分布变化，过去有效的 Gene 可能失效，过去稳定的 Capsule 可能退化，过去高收益的 Token 策略可能带来新的质量损失。Evolution Genome 需要持续记录每个资产的来源、适用条件、验证结果、收益指标、风险边界、依赖关系和版本谱系，并根据真实任务表现对其进行升级、降级、合并、拆分或退役。

这种谱系化管理，使 Agent 的成长从零散经验积累，升级为可治理的能力生态。系统不仅知道哪些经验有效，也知道它们为什么有效、在哪里有效、何时失效、如何迁移、如何回滚。

6.3 元进化如何支撑 Interactive 与 Proactive 的转换

Coactive Agent 的关键机制，是 Interactive 与 Proactive 之间的动态转换。一类任务在早期往往需要人和 Agent 共同探索：人提供目标、偏好和判断，Agent 提供资料、结构、反例和执行支持。随着协作次数增加，任务路径逐渐稳定，质量标准逐渐清晰，风险边界逐渐明确，部分能力就可以从 Interactive 进入 Proactive，由 Agent 在授权范围内主动承接。

AgentReflex 是双态迁移背后的判断系统。它通过可观测机制记录任务路径是否稳定，通过评估机制判断结果和过程是否可靠，通过 Harness 变异机制优化执行结构，通过进化资产机制沉淀可复用能力。只有当任务路径稳定、风险边界清晰、效果经过验证、回滚机制可用时，一类能力才适合进入 Proactive 能力池。

“毕业线”并没有固定阈值，而是随任务成熟度、用户信任、组织风险偏好、评估能力和工具可靠性动态调整。某些任务一开始必须保持人在回路，但当系统积累了足够多的成功轨迹、稳定 Eval、低风险 Gene 和可回滚 Capsule 后，可以逐步提高主动性。相反，某些已经进入 Proactive 的能力，一旦出现新的失败模式、风险事件或用户反馈下降，也应该自动降级，重新回到 Interactive 中进行澄清、修正和再沉淀。

由此，元进化不仅支撑 Agent 能力增长，也支撑授权边界的动态治理。它让系统能够回答：哪些任务仍需人机共创，哪些任务可以主动承接，哪些主动能力需要收敛，哪些经验已经成熟为可复用能力。对于企业级 Agent，这种动态授权机制比单纯提升自动化程度更关键，因为它同时兼顾了效率、质量、风险和用户信任。

6.4 企业级价值：从单点智能到组织能力

元进化闭环的最终价值，不在于构建一个更复杂的 Agent 平台，而在于将 Agent 的运行过程转化为组织能力。每一次任务执行、每一次用户反馈、每一次失败修复、每一次成本优化，都可以成为组织学习的一部分。Agent 不再只是消费知识和工具的执行者，也成为沉淀流程、积累经验、发现风险和优化机制的参与者。

对于企业而言，AgentReflex 带来三类核心价值：

提升 Agent 规模化运行的稳定性。通过持续观测、动态评估和受控改造，系统能够更早发现失败模式，更快形成修复方案，并避免相似问题在不同 Agent、不同团队和不同任务中反复出现。

提升 Agent 能力成长的可治理性。所有进化动作都需要证据、评估、灰度、回滚和谱系管理，避免自进化滑向不可解释的配置漂移。企业可以清楚知道一个策略从哪里来、解决了什么问题、在哪些环境中有效、影响了哪些 Agent，以及出现问题时如何回退。

提升组织经验的复用效率。过去依赖专家个人经验、团队文档和零散配置的能力，可以被结构化为 Gene 和 Capsule，跨任务、跨团队、跨 Agent 复用。随着使用规模扩大，系统不只是部署了更多 Agent，也沉淀出更强的组织级智能资产。

因此，元进化闭环将 Agent 平台从“任务执行基础设施”升级为“组织学习基础设施”。谁能让 Agent 跑起来，只解决了智能应用的起点问题；谁能让 Agent 在真实世界中越跑越稳、越协作越懂、越沉淀越强，才真正掌握企业级 Agent 的长期竞争力。

6.5 本章小结

Coactive Agent 提出了一种面向下一代 Agent 的协作范式：Agent 在人与 AI 的共同活动中，把实时交互、主动执行和长期沉淀组织成一个统一系统。AgentReflex 进一步回答了这套系统如何持续成长的问题。

它将 Agent 的运行过程转化为学习过程，将失败反馈转化为选择压力，将有效修复转化为进化资产，将局部经验转化为组织能力。更重要的是，它把可观测、评估、Harness 和进化资产本身纳入持续进化范围，使系统不仅能够驱动 Agent 变好，也能够持续改进“如何驱动 Agent 变好”。

最终，Coactive 与 AgentReflex 共同构成下一代 Agent 的完整路径：Coactive 让 Agent 与人共同工作，AgentReflex 让 Agent 在共同工作中持续成长，Meta-Evolution 让“如何成长”这件事本身持续成长。

> **CHAPTER TAKEAWAY**
>
> **元进化的核心，不只是 Agent 能够持续变好，而是驱动 Agent 变好的机制本身也能够被观测、被评估、被改造、被选择和被沉淀。**

第七章 Coactive 范式下智能体的全生命周期安全体系

LIFECYCLE SECURITY FOR COACTIVE AGENTS

Agent 越能自主行动，安全越不能停留在外围防护。本章基于 Coactive 范式，重新阐述了智能体全生命周期安全体系的变革：安全的基本单元从“一次请求及其响应”转向“持续变化、可以作用并能够返回证据的任务环境”。

7.1 安全范式迁移：从“墙”到“场”

7.1.1 传统安全范式的失效

传统 Agent 安全以“墙”为隐喻：筑起权限边界的高墙，拦截越界行为，保护内部资产。这种范式在可控、确定性的系统中有效，但在 Coactive Agent 时代面临结构性失效。

Coactive Agent 的核心判断是：下一代 Agent 不是更自动，而是更好地与人共处。Agent 既不是等待指令的 Reactive，也不是一味替人完成的 Proactive，而是在人与 AI 的共同活动中，把实时交互、主动执行和长期沉淀组织成一个统一系统。这意味着安全的基本单元从“一次请求及其响应”转向“持续变化、可以作用并能够返回证据的任务环境”。

7.1.2 新安全范式：“场”的安全

Coactive 范式下，安全的隐喻从“墙”转向“场”。场是人与 Agent 共同工作的任务环境，安全应内生于场的每一个运行条件中，而非外挂于场的边界上。具体而言：

从“拦截越界”到“共处可信”：安全不再只是拦截越界行为，而是保障人与 Agent 在共同活动中的信任基础——人可以信任 Agent 的行为边界，Agent 可以信任人的判断输入。

从“事后审计”到“行为内生”：安全不再是事后追溯的审计日志，而是内嵌于 Agent 行为的每一步——从感知到执行，从记忆读写到能力沉淀，安全约束与业务逻辑同步并行。

从“静态规则”到“可信承接安全”：安全策略不再是固定的黑白名单，而是随可信承接动态调整——任务在 Interactive 阶段和 Proactive 阶段有不同的安全边界，越自主越需要越严格的安全约束。

> **安全不是牢笼，而是共处的地基。人与 Agent 共处的任务环境，就是安全应该内生的场。**

7.1.3 Coactive 安全的三条原则

原则一 共处即安全。安全不是对 Agent 的牢笼，而是人与 Agent 共处的地基。安全的目标不是最大化自动化，而是让人的判断、经验和创造力持续进入工作流，同时让 Agent 在授权边界内可靠承接。

原则二 原文为真相，派生为索引。记忆的原始材料是事实锚点，派生结构只是索引。每一条安全判断、每一次行为迁移、每一次能力承接，都必须可以追溯到原始证据。安全的可信度，建立在证据链而非信任链上。

原则三 越自主越约束。Agent 的自主性与安全约束不是零和博弈，而是正向关系。任务满足可信承接条件进入 Proactive，安全约束就越严格。自主执行的范围越大，证据链、审计日志、回滚能力的要求就越高。

7.2 Coactive Agent 的五大安全支柱

基于 Coactive = Interactive × Proactive 的协作范式，我们将 Coactive Agent 安全体系定义为以下五大支柱，如表 7-1 所示。每一根支柱都回答一个核心安全问题，并提出明确的安全观点：

表 7-1 Coactive Agent 的五大安全支柱

支柱	核心观点	回答的安全问题	安全能力映射
① 共处可信	人与 Agent 共处的信任基础，而非牢笼的约束墙	人如何信任 Agent?	权限边界 + 用户可打断
② 记忆主权	记忆是 Agent 的持久层，主权归属必须明确	记忆归谁、谁能访问、边界在哪?	隐私保护 + 来源追溯 + 知识隔离 + 记忆边界
③ 行为可控	行为可控是可信承接的安全内洽，而非事后补救	Agent 做了什么、能否回滚?	可审计 + 可回滚
④ 进化护栏	进化必须有证据、有选择、有边界、可追溯	Agent 越用越好，但谁来守护进化?	元进化安全约束
⑤ 推理安全	Token 流即安全流，从输入到输出的连续治理	推理过程中的安全如何保障?	安全护栏

7.2.1 支柱一：共处可信——人与 Agent 共处的信任基础

Coactive 范式下，人与 Agent 的关系不是“指令者与执行器”，而是“共同活动中的协作者”。这种关系的安全基础不是牢笼的约束墙，而是共处的信任地基。信任不是先决条件，而是在共处过程中持续产生的结果：人可以信任 Agent 的行为边界，Agent 可以信任人的判断输入。

身份与权限：数字员工与 Coactive 权限：每个用户与 Agent 必须完成唯一身份认证，基于“数字员工身份”进行最小权限授权。系统根据预设岗位角色模板自动分配最小必要权限集。关键创新在于：任务在 Interactive 阶段和 Proactive 阶段有不同的权限边界，越自主越严格。

用户可打断：“机主防、人主控”的共处机制：Coactive 的核心是“原创性交互，非原创性自主”。这意味着人始终保持对任务的最终决策权。用户可打断不是简单的“暂停”按钮，而是一套完整的共处机制：任何敏感操作均需人工授权后方可执行；当 Agent 从 Interactive 转入 Proactive 时，必须解释为什么转换、当前已授权的范围是什么；当质量下降、上下文漂移、授权过期或风险上升时，Agent 应主动降低自主级别、回到 Interactive。

> **核心观点：安全不是牢笼，而是共处的地基。信任不是先决条件，而是共处过程中持续产生的结果。**

7.2.2 支柱二：记忆主权——记忆是 Agent 的持久层，主权归属必须明确

记忆机密运行与动态脱敏：Agent 记忆数据在传输与存储阶段均采用高强度加密算法，实现全生命周期密文保护，仅在运行时即时解密。通过机密计算架构（CCA），将解密与运算严格限制在机密计算

环境中，云平台超级管理员也无法访问或篡改运行中的记忆数据。同时对输出文本实施动态语义脱敏，自动遮蔽企业核心代码、高敏财务数字、凭据密钥等。

记忆来源追溯：原文为真，派生为索引：这是 Coactive 范式下最具特色的安全能力。Agent Memory Runtime 的第一原则是“原文为真，派生为索引”：原始材料（会话转写、文档版本、工具调用结果、用户确认和修正意见）是事实锚点；派生结构（摘要、标签、向量、图谱、偏好画像）只是索引和加速层，永远不能替代原始材料。每一条记忆条目都必须标记来源信息（产生时间、触发任务、调用工具、输入源），实现全链路证源。Agent 不能只说“我记得你偏好这种风格”，而必须能追溯到哪个历史任务、哪次文档修改、哪次用户反馈支持了这个判断。

企业知识隔离与应用记忆边界：真实世界的记忆不是均匀的——文档、PPT、表格、浏览器、邮件和 IDE 有不同的对象、工具、权限和质量标准，它们不应被混在一个扁平的记忆池中。Agent Memory Filesystem 通过 scope、app、permissions、source 字段保留记忆归属和访问边界。企业知识库之间实施严格的访问隔离，不同租户的知识资产不可互透。每个应用实例拥有独立的记忆空间（Per-App Memory），应用间记忆不可互访，记忆生命周期与应用实例绑定。

> **核心观点：记忆不是后台检索系统的增强版，而是 Agent 的持久层。记忆的主权归属、访问边界和治理规则，必须与 Agent 的身份、任务和约束体系绑定。**

7.2.3 支柱三：行为可控——行为可控是可信承接的安全内洽

全流程审计：共享任务状态即审计对象：在 Coactive 范式下，审计的对象不再只是孤立的操作日志，而是共享任务状态的完整轨迹。共享任务状态是 Coactive 智能主体的核心架构对象，包含当前目标、已确认约束、人的最新判断、执行计划与进度、授权与风险、结果证据、当前运行模式。审计日志不可篡改，加密留存与哈希校验，支持按时间、角色、模式切换、操作类型多维度查询。

操作回滚与自适应降级：当 Agent 执行导致异常结果时，系统支持将环境回滚到执行前的一致状态。这与 Coactive 的“可逆性”设计原则一致：当质量下降、上下文漂移、授权过期或风险上升时，Agent 应主动降低自主级别、回到 Interactive。同时具备自适应降级能力：异常时自动切换到更保守的执行策略，待风险解除后自动恢复。

Proactive 的三域六维成熟度模型与安全级联：任务成熟度（目标成熟+上下文成熟）、能力成熟度（流程成熟+质量成熟）、责任成熟度（风险成熟+信任成熟）。只有三域均达标的任务片段，才能进入 Proactive。进入 Proactive 后，安全约束自动升级：审计粒度更细、回滚能力强制、证据链必备。

> **核心观点：行为可控不是事后补救，而是可信承接的安全内洽。Agent 的每一次行为迁移，都必须带着证据、带着边界、带着回滚方案。**

7.2.4 支柱四：进化护栏——进化必须有证据、有选择、有边界、可追溯

Coactive Agent 的元进化闭环（AgentReflex）让 Agent 在真实协作中持续成长。但进化本身必须受到安全约束，否则自我优化可能导致行为漂移、安全边界侵蚀、责任链断裂。进化护栏的四条铁律：

1. **进化必须有证据：**任何修改必须来自真实任务轨迹、用户反馈、评估结果和安全事件。不允许无证据的策略变更进入生产。

2. **进化必须有选择压力：**每一次 Prompt 重写、工具描述调整、记忆策略修改、主动触发规则变更，都必须经过 Eval、灰度、对比和回滚。

3. **进化必须有边界：**不同任务、用户、组织和模型版本有不同的适用经验。一个有效策略不能简单复制到所有场景。进化资产（Reflex Gene/Capsule）必须记录适用条件、验证结果、风险边界和版本谱系。

4. **进化必须可追溯：**企业级 Agent 自我进化必须知道策略来自哪里、什么失败触发了它、经过哪些验证、影响哪些 Agent、在哪个版本有效、出问题如何回滚。

核心观点：Agent 越用越好，但谁来守护进化？元进化不是“自动修改自己”，而是可治理的成长机制。

7.2.5 支柱五：推理安全——Token 流即安全流，从输入到输出的连续治理

在 Coactive 范式下，推理安全的边界从“单次请求-响应”扩展到“持续协作的任务环境”。Token 流不仅穿越单次推理的多个阶段，还穿越 Interactive 与 Proactive 的模式切换。如表 7-2 所示，安全护栏必须覆盖五个阶段。

表 7-2 安全护栏的五个阶段

护栏阶段	核心能力与 Coactive 安全要求
用户输入安全	语义识别 + 越狱拦截 + 角色边界硬隔离；在 Interactive 中保护人的意图纯净性，在 Proactive 中防止指令劫持
上下文安全	多轮诱导检测 + 算力滥用防护 + 推理深度控制；防止共处过程中的语义污染向 Proactive 渗透
RAG 检索安全	知识库静态扫描 + 检索行为审计 + 上下文可信度评估；确保记忆读取的每一条知识都有可信度标记和来源指针
工具调用安全	调用生成约束 + 参数安全校验 + 注入防护
输出内容安全	流式合规检测 + 企业敏感数据动态脱敏 + 安全重定向；确保共处产物不泄露企业机密、不违反合规边界

核心观点：Token 在运行中持续穿越输入解析、上下文构建、知识检索、工具调用与内容生成，推理安全是围绕 Token 生命周期的连续治理，而非孤立模块防护。

7.3 五大支柱如何支撑 Coactive Agent 协作

五大支柱不是独立的安全模块，而是围绕 Coactive 协作范式形成的内生安全体系。它们在两个关键的协作场景中协同作用：

Interactive 场景：共处可信，保障人的意图纯净性和 Agent 的行为边界；推理安全，保护输入输出的双向安全；记忆主权，确保 Interactive Memory 的预取不会暴露未授权记忆。

Proactive 场景：行为可控，确保可信承接后的自主执行有证据、有边界、可回滚；进化护栏，确保元进化的每一步都受到安全约束；推理安全，确保工具调用经过授权校验。

Coactive 安全的核心机制，是让安全约束与任务现场、过程状态和能力迁移同步运行，而不是只在输入和输出边界进行一次拦截。共处可信通过身份、最小权限、用户可打断和随可信承接动态调整的授权边界，保障人与 Agent 的协作关系；记忆主权通过原文追溯、访问隔离和应用记忆边界控制长期经验的归属与使用；行为可控将过程状态、执行证据、审计、回滚和自适应降级嵌入 Proactive 执行；进化护栏要求每次策略变化都有证据、评估、灰度、回滚和版本谱系；推理安全则覆盖输入、上下文、RAG 检索、工具调用和输出的连续链路。由此，Agent 在 Interactive、Proactive 和可信承接之间迁移时，权限、证据、审计和回退要求能够随主动程度同步变化。

7.4 本章小结

本白皮书基于 Coactive = Interactive × Proactive 的协作范式，将 Agent 安全从“墙”的范式迁移到“场”的范式，并将 Coactive Agent 安全体系定义为五大支柱：共处可信、记忆主权、行为可控、进化护栏、推理安全。

其核心逻辑是：安全内生于共处的地基（共处可信），植根于记忆的主权基础（记忆主权），内嵌于可信承接的行为约束（行为可控），守护于进化的每一步（进化护栏），流转于推理的每一个环节（推理安全）。

未来，随着 Agent 从“更自动”向“更好地与人共处”演进，安全体系也将从“规则驱动的边界防护”向“证据驱动的内生安全”演进。最终目标是实现“共处而信，共处而安，共处而进”的安全愿景——让人永远留在创造性事务里，把可重复、可沉淀的部分交给 Agent 自主成长，而安全是这一切的内生地基。

CHAPTER TAKEAWAY

Coactive Agent 安全的核心，不是为 Agent 筑起更高的墙，而是在人与 Agent 持续共处的任务场中，构建可追溯、可约束、可回滚、可进化的内生安全体系。

第八章 应用场景：面向开放世界的应用案例

APPLICATION SCENARIOS

前序章节已经阐述 Coactive 的核心范式：同一个智能主体在 Interactive 与 Proactive 两种状态之间切换——不确定处共创，成熟处承接；证据充分时推进，风险升高时唤醒。本章把这一范式放回真实业务中检验：四个场景各回答一个具体问题，前三个场景分别证明一项关键机制，第四个场景在一个完整的服务现场中验证全部机制如何协同运转。

开放世界中的任务很少以完整指令出现。目标会在操作中逐步清晰，判断分散在每一次修改、停顿、确认和拒绝里；组织经验则分散在文档、系统和一线人员的头脑中。为了说明 Coactive 在这种环境里解决什么问题，本章按四个问题展开：

1. 当目标还在变化时，Agent 如何参与共创，而不过早进入自动化？
2. 当任务可以托管时，Agent 凭什么获得授权？
3. 当用户正在撰写和表达时，记忆如何在输入时刻进入判断？
4. 一次方案讨论结束后，协作中形成的共识如何转化为可直接使用的应用能力？

四个场景中，Agent 的姿态保持一致：观察真实任务现场，在判断形成时给出建议，由人决定是否采纳；证据积累到一定程度后，再把稳定片段交给后台执行。前三个场景分别对应 Interactive、Proactive 和 Interactive Memory；第四个场景把记忆召回、需求结构化、后台长程任务和能力沉淀放在同一次方案讨论中验证。

本章的判断标准也很明确：协作过程是否可验证、可接管、可回退，并能形成后续可复用的能力。四类场景与机制的对应关系如图 8-1 所示。

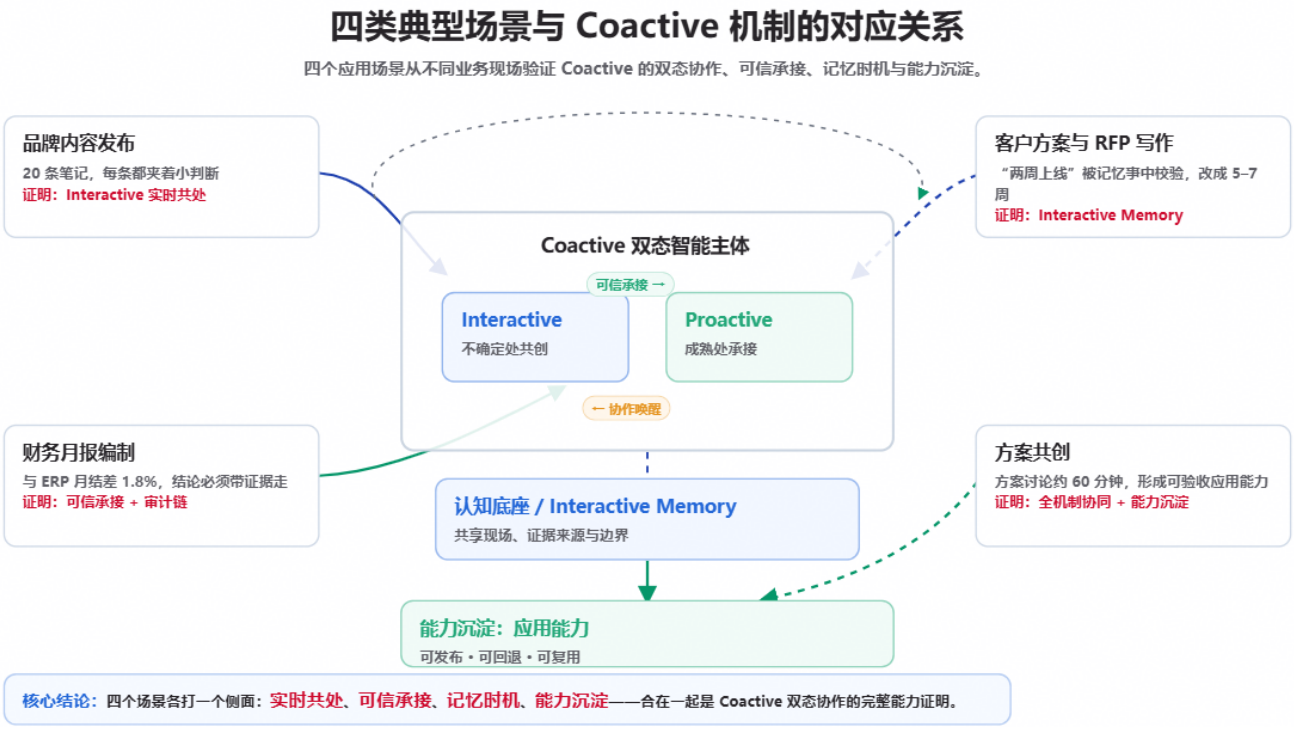


图 8-1 四类重点场景与 Coactive 机制的对应关系

8.1 品牌内容发布：从人工运营到智能协同的自动化演进

内容发布是高频、低容错的日常任务。运营人员手里有一张待发布表格、一组商品图片和几个平台账号，要把当天的新品和活动发布出去。表面看，任务只是复制、粘贴、上传和提交；实际操作中，每一步都夹着判断：标题是否自然，图片是否带水印，文案会不会触发平台规则，哪张图适合做封面。

8.1.1 场景中的硬问题

下午，运营人员打开在线表格。表格里有二十条待发布笔记，每条都包含商品名称、卖点、原始文案、图片路径和目标平台。他从第一篇开始处理：复制文案，粘贴到发布页，上传图片，调整标题和标签。

Agent 进入同一个发布现场，不要求用户回到聊天框重新描述需求。它能看到剪贴板内容、光标位置、图片文件和平台反馈。运营人员停在标题输入框时，Agent 给出两条更接近品牌语气的标题建议；图片拖入页面后，Agent 提醒角落可能有第三方水印；平台弹出违禁词提示时，Agent 定位触发词，并给出可替换表达。运营人员采纳其中一个标题，换掉水印图；另一条建议没有被采用，Agent 也不继续追问。

这个场景的难点在于，用户的判断无法提前完整写成指令。删掉一个过度营销的词，保留一句更口语的描述，拒绝一张合规但不自然的封面，这些动作本身就是判断信号。Interactive 让 Agent 在信号出现时理解和修正；等整篇内容完成后再检查，很多判断已经错过发生位置。

8.1.2 传统方案的不足

普通 Copilot 可以改写文案，但看不到发布页面、平台报错和用户刚做出的取舍。用户需要把现场重新讲一遍，协作会退回到事后描述。RPA 能搬运表格内容，却无法判断水印、标题承诺和封面选择。纯 Proactive Agent 如果一开始就批量提交，可能把未确认的文案和图片一次性发出去。内容发布的自动化要从第一篇里的人工判断方式开始，随后才能进入批量提交。

8.1.3 Coactive 如何解决

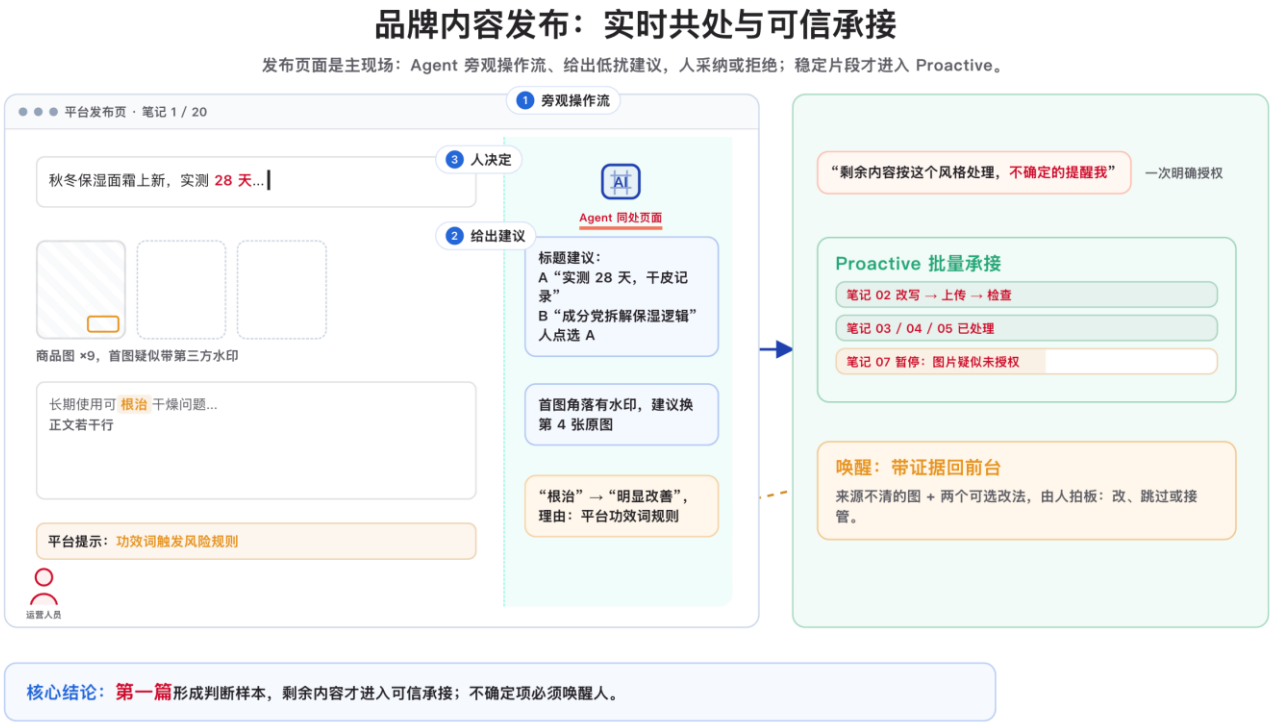


图 8-2 品牌内容发布：实时共处、共同判断与可信承接

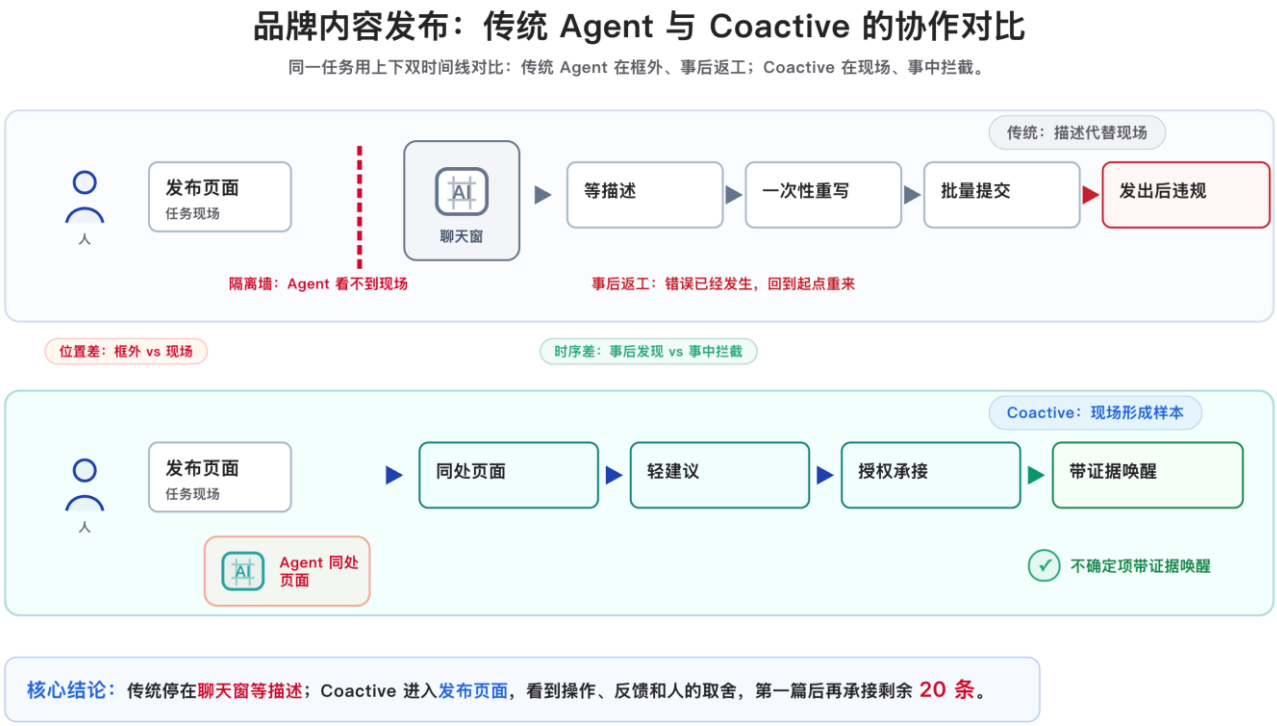


图 8-3 品牌内容发布：传统 Agent 与 Coactive 的协作对比

Coactive Agent 先进入 Interactive 状态，与运营人员共享同一个发布现场：在线表格、剪贴板、素材文件夹、发布页面、平台规则和用户的每一次手动修改，都进入同一个过程对象。用户复制、粘贴、拖拽、删除、停顿、采纳、拒绝，这些信号持续更新“这次发布应该怎么做”。

提示保持低扰、可打断。用户刚粘贴文案，Agent 只标出两处可能过度承诺的短语，不重写整篇；用户把其中一句改回自己的表达，Agent 随即降低建议强度；用户拒绝某个建议，Agent 将这个选择记入当前任务的偏好和边界。记录内容具体到本次任务的可执行判断：标题保持克制的品牌语气；图片默认先查水印；涉及价格和功效的句子必须人工确认。

第一篇顺利发布后，运营人员明确授权：“剩余内容按这个风格处理，不确定的地方提醒我”。Agent 将稳定片段迁入 Proactive：逐条读取表格，轻量改写，上传图片，检查禁用词，按平台格式填充标签，并把每篇结果写回表格。过程仍在同一个任务列表中可见，用户可以暂停、接管或修改某一篇。

一旦出现新的不确定性，例如图片疑似未授权、文案触发平台风险提示、卖点可能构成功效承诺，Agent 停下当前条目，把问题、来源和可选改法带回前台。整个过程如图 8-2 所示。

8.1.4 该场景证明的 Coactive 价值

日常内容发布场景的 Coactive 判据如表 8-1 所示。这个场景展示了 Interactive 的工作位置：它进入人的操作流，在修改和犹豫发生时理解上下文。批量发布只是共同判断之后的承接动作。传统 Agent 与 Coactive 在这一场景中的协作差异如图 8-3 所示。

表 8-1 日常内容发布场景的 Coactive 判据

判据	在日常内容发布中的体现
Interactive 如何发生	Agent 与用户同处发布页面，持续理解复制、粘贴、拖图、停顿、修改、采纳和拒绝等操作信号
共同判断如何形成	标题语气、图片水印、违禁词替换、封面选择和功效承诺边界，都在第一篇的操作中被即时讨论和修正
哪些片段可承接	已确认的轻量改写、图片上传、水印检查、禁用词替换、标签填充和结果回写进入 Proactive
什么事件触发唤醒	图片疑似未授权、平台报错、功效承诺过度、封面选择不确定、素材缺失
回退路径	暂停当前条目、保存草稿、跳过风险内容、回到人工编辑、只继续发布已确认内容

8.2 财务月报编制：从人工汇总到可审计承接的智能工作流

财务经营月报包含 Excel 工作簿、公式填充、图表生成、异常解释，也延伸到 Word 公告、PPT 汇报和邮件通知。企业需要确认每个数字、公式、图表和结论都有来源、版本、口径和责任人。这个场景用于验证从 Interactive 到 Proactive 的可信承接：可验证片段进入后台执行；公式异常、口径冲突或对外风险升高时，证据回到财务人员面前。

8.2.1 场景中的硬问题

每月初，财务专员打开一份包含 12 个部门 sheet 的月度经营数据工作簿。数据来自 ERP 收入表、费用系统报销明细、预算系统年度目标，以及业务团队补充的人工调整表。她需要在 Excel 中完成格式统一、合计公式、同比环比和部门趋势图，再基于同一套数据生成 Word 公告、管理层 PPT 和邮件草稿。

开始阶段并不复杂。她在第一个部门 sheet 中选中表头，让 Agent 加粗居中；在合计列填入 SUM 公式；对收入、成本和毛利率生成柱状图。Agent 在 Excel 侧边栏中执行，她可以用鼠标选区、语音补充或 Ctrl+Z 撤销。她把图表配色改成企业模板蓝，要求添加数据标签；Agent 记录的是本月月报模板的一部分，不把它泛化为长期审美偏好。

难点出现在异常归因。第一个部门 sheet 的收入合计与 ERP 月结表相差 1.8%；市场费用环比上升 42%；一个项目收入仍挂在旧部门名下。如果 Agent 继续生成图表和摘要，月报会把未解释的异常包装成确定结论。财务人员必须知道差异来自数据版本、公式范围、部门映射，还是业务真实变化。

格式和公式可以交给后台，结论必须带着证据走。

8.2.2 传统方案的不足

数据分析 Agent 能读表格、出图表，但常常说不清改了哪个单元格、用了哪条公式、引用了哪个数据版本。RPA 可以按固定模板处理 12 个 sheet，却难以判断“部门名称变更”是口径更新还是数据错误。纯

Proactive Agent 如果在差异未解释时继续生成 Word、PPT 和邮件，会把 Excel 中的小风险放大成管理层判断风险。

财务场景对 Agent 的要求可以概括为三点：会做；能说明为什么可以做；知道做到哪里必须停，并把证据交给对应负责人。

8.2.3 Coactive 如何解决

Coactive Agent 先在 Excel 现场建立共享过程对象：当前工作簿、12 个部门 sheet、ERP 与预算的数据版本、公式规则、图表模板、人工调整记录和审批人，都进入同一个任务现场。财务专员处理第一个 sheet 时，Agent 记录选区范围、公式写入位置、图表字段、撤销记录和确认口径。

图 8-4 财务月报编制：可信承接、异常唤醒与审计链

首个 sheet 的处理路径稳定后，承接才开始。进入后台前，任务先经过可信承接条件：数据版本对得上，公式范围经过确认，口径有出处，责任人明确。四项齐备后，Agent 才把可验证片段迁入 Proactive：依次处理剩余 11 个部门 sheet，统一格式，填充公式，生成图表，校验跨 sheet 合计是否等于汇总表，并把每一步操作写入审计链。

发现异常时，Agent 停下当前片段并唤醒财务人员。提示内容给出一组可判断的证据：哪个单元格与 ERP 月结表不一致，差异金额是多少，当前公式引用了哪些行列，旧部门名称对应哪个新部门，继续生成会影响哪张图、哪段 Word 说明、哪页 PPT。财务人员确认原因后，Agent 更新口径继续推进。

Excel 月报完成后，同一条证据链延伸到跨应用产物：Word 公告中的关键结论引用对应单元格和数据版本，PPT 图表保留来源 sheet 与生成时间，邮件只作为草稿进入审批，不自动发送。跨应用输出只复制经过确认的证据链，未解释的风险不进入下游材料。整个过程如图 8-4 所示。

8.2.4 该场景证明的 Coactive 价值

表 8-2 财务月报场景的 Coactive 判据

判据	在财务月报中的体现
共享现场如何建立	Excel 工作簿、12 个部门 sheet、ERP/预算版本、公式、图表、Word/PPT/邮件草稿和审批意见进入同一过程对象
哪些片段被承接	格式统一、SUM 公式、同比环比、图表生成、跨 sheet 校验、Word/PPT 草稿和邮件草稿等可验证片段
什么事件触发唤醒	ERP 合计差异、费用异常无法解释、部门映射缺失、公式范围错误、关键结论缺少审批、邮件发送风险升高
回退路径	回到上一数据版本、撤销某次公式或图表变更、仅保留草稿、标记待确认项、人工审批后继续

财务月报场景的 Coactive 判据如表 8-2 所示。财务月报场景验证的是可信承接的边界：每一次后台执行都有证据、责任和回退路径。传统 Agent 与 Coactive 在这一场景中的协作差异如图 8-5 所示。

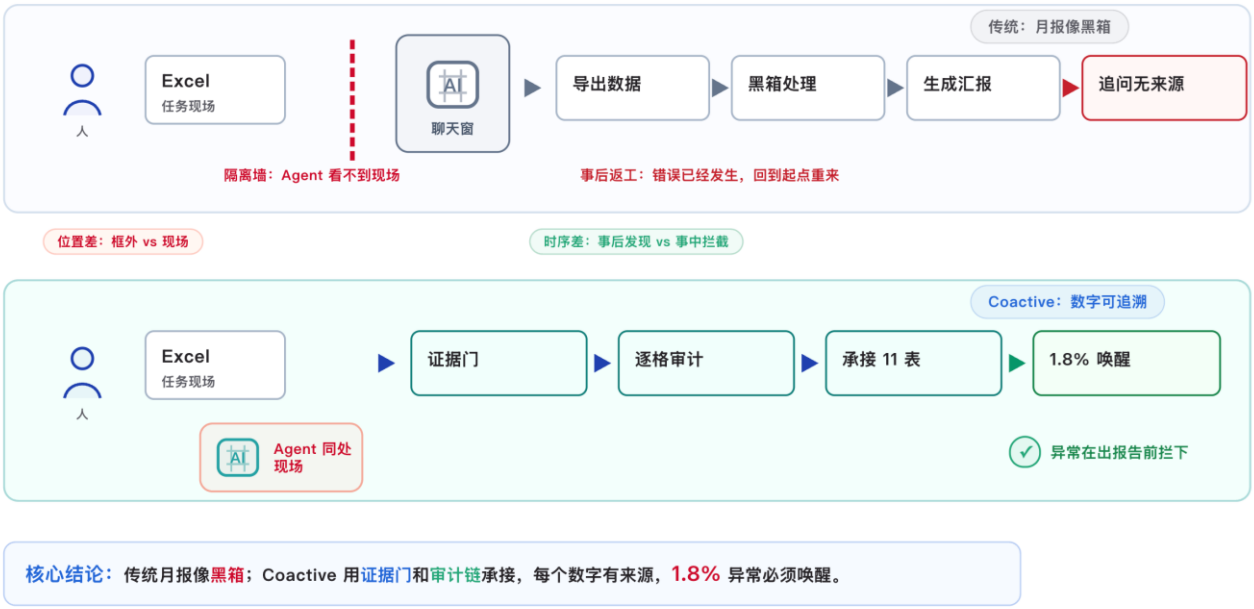


图 8-5 财务月报编制：传统 Agent 与 Coactive 的协作对比

8.3 客户方案与 RFP 写作：从事后检索到实时记忆参与的协同编制

客户方案和 RFP 响应中的高成本错误，往往发生在文本形成的瞬间：交付周期写得过短，案例未经授权就被引用，灰度能力被写成正式商用能力，旧版 SLA 被复制进正式响应。这个场景验证 Interactive Memory：记忆在用户输入和表达时进入现场，给出推荐、补证或事中校验。

8.3.1 场景中的硬问题

售前团队收到一份大型客户 RFP，要求五个工作日内提交完整方案。客户关注安全、部署周期、行业案例、服务 SLA、价格模型和数据合规。团队过去做过类似项目，但材料分散在历史投标、客户邮件、产品路线图、交付复盘和法务模板中。有的案例可以复用，有的价格策略已经变化，有的能力仍在灰度，有的承诺必须由交付负责人确认。

组织经验有价值，但必须在写作和判断发生时出现。等全文生成后再审查，错误承诺已经进入草稿，纠偏成本和沟通风险都会上升。

8.3.2 传统方案的不足

普通 RAG 和知识库检索的时序通常固定为：用户说完，系统检索，模型生成。检索即使准确，也慢半拍；用户已经写下承诺、复制案例、确定口径，系统才提示材料冲突。文档生成 Agent 也常把问题推到生成后检查：输出看似完整，却无法在关键句形成前介入。

把两种时序放在一起，差异如图 8-6 所示：

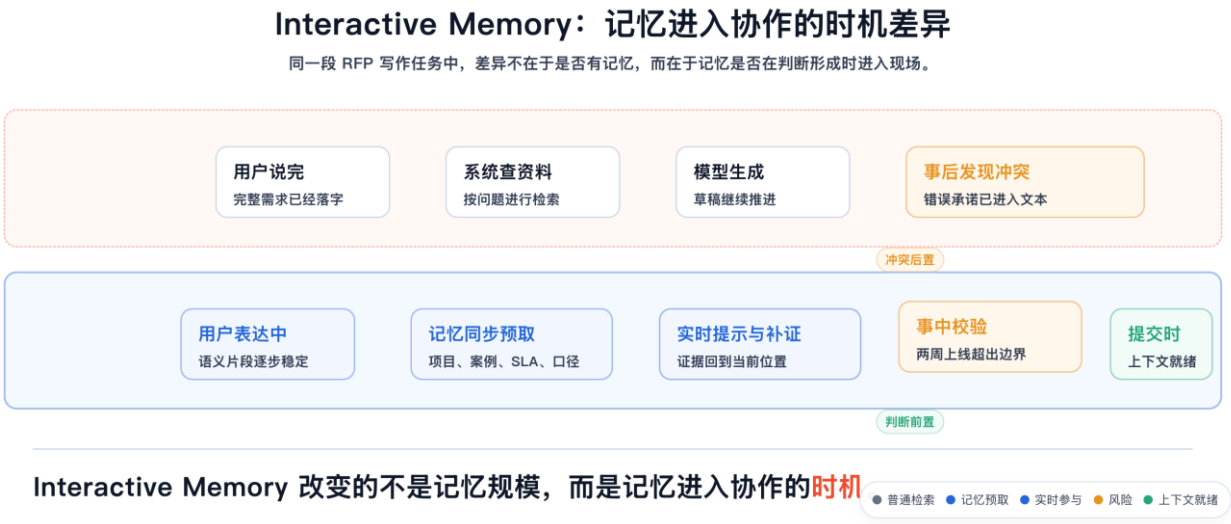


图 8-6 Interactive Memory 改变的是记忆进入协作的时机

8.3.3 Coactive 如何解决

Coactive Agent 把 RFP 原文、客户背景、历史方案、产品文档、法务模板、交付复盘和评审意见组织为任务现场。用户在前台撰写时，Interactive Memory 持续观察输入流中的稳定语义片段，包括客户名称、行

业关键词、部署模式、SLA 条款、案例引用和价格口径，并根据当前位置预取相关记忆。输入期预取、记忆编译和模型预准备把后台记忆调用前移到表达过程中；在正文层面，用户写到哪里，相关记忆就到哪里。

当售前顾问写下“预计两周完成私有化部署”时，Agent 立即进行事中校验：过去三个同类项目平均交付 5 到 7 周，其中一次延期来自数据迁移审批；交付负责人在上次复盘中明确标记“不得承诺两周上线”。Agent 建议改写为“完成环境评估后给出详细排期，标准交付周期约 6 周”，或在投标文本中写成“5–7 周、分阶段交付”。承诺还没进入定稿，组织记忆已经回到判断现场。

用户粘贴历史案例时，Agent 提示该案例是否已授权用于公开投标、是否需要脱敏、是否仍适用于当前行业。引用产品能力时，Agent 核对产品路线图和审批口径，仍在灰度的能力建议写为“可试点验证”。回答 SLA 条款时，Agent 比对最新服务模板与旧销售口径，发现冲突就要求法务或交付确认。用户可以采纳、忽略或修改这些提示，每一次选择都会留下新的协作证据：哪些提示被采纳，哪些口径被确认，哪些材料被标记为不可复用。

来源清楚、口径一致、责任人明确之后，已确认内容交给 Proactive 承接：生成需求响应矩阵，补齐标准章节，统一术语，检查遗漏，整理评审问题。整个过程如图 8-7 所示。

图 8-7 RFP 写作中的 Interactive Memory：记忆进入协作的时机差异

8.3.4 该场景证明的 Coactive 价值

RFP 写作场景的 Coactive 判据如表 8-3 所示。RFP 写作场景验证的是记忆介入时机。普通检索像查档案，Interactive Memory 更接近一名熟悉组织历史的协作者：当一句话即将成为对外承诺时，它把证据和边界放回当前输入位置。传统 Agent 与 Coactive 在这一场景中的协作差异如图 8-8 所示。

表 8-3 RFP 写作场景的 Coactive 判据

判据	在 RFP 写作中的体现
记忆何时进入现场	用户输入客户名、需求、承诺句、案例引用和 SLA 条款时，记忆同步预取并准备提示
记忆如何参与判断	以推荐、补证、风险提醒和事中校验的形式出现在当前输入位置，避免事后才返回一堆检索结果
哪些片段可承接	需求矩阵生成、标准章节补齐、术语统一、遗漏检查、引用核验和评审问题整理等已确认内容
什么事件触发唤醒	交付周期超边界、案例授权缺失、灰度能力被写成正式承诺、旧口径与最新模板冲突、证据不足
回退路径	回到来源原文、撤销或改写当前段落、标记为待确认、转交产品/法务/交付评审、保留多版候选表达

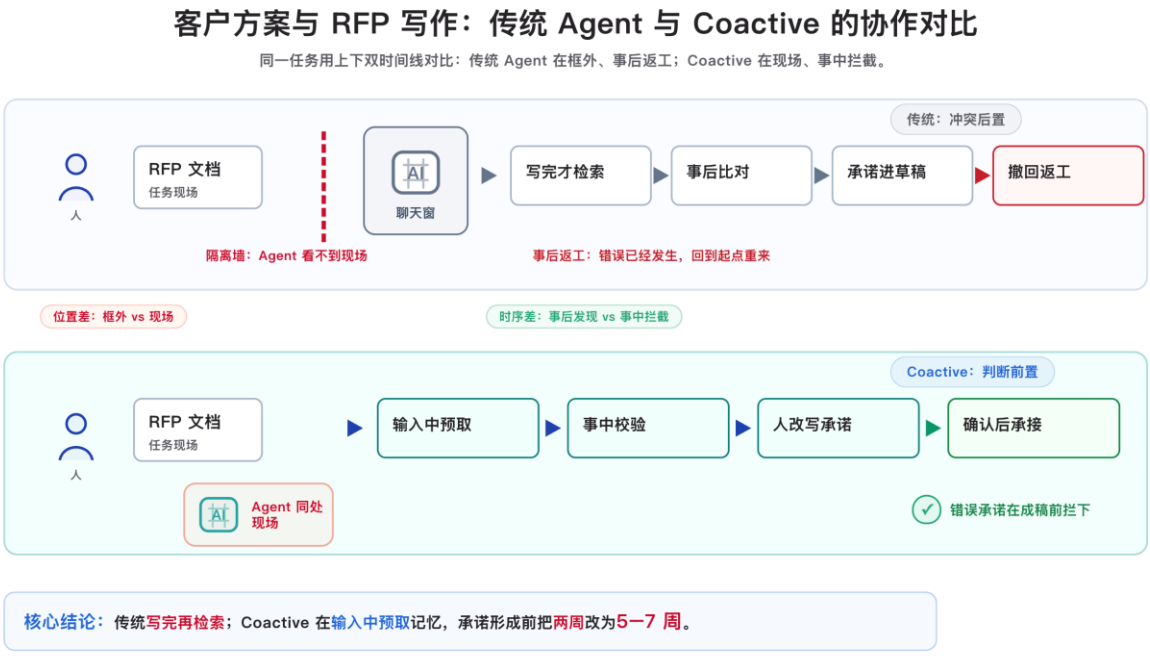


图 8-8 客户方案与 RFP 写作：传统 Agent 与 Coactive 的协作对比

8.4 方案共创与能力沉淀：从方案讨论到可复用应用能力

前三个场景分别验证单项机制。第四个场景把这些机制放回同一个方案讨论现场，并回答一个更后置的问题：协作结束后，组织留下了什么。

一次方案讨论结束后，会议纪要会被归档，待办事项会被分配，但讨论中形成的需求结构、约束条件和可复用方案，通常还要等待数周开发周期才会变成可用产品。这个场景验证能力沉淀：在同一次协作中，Agent 召回组织记忆、实时结构化需求、委托后台长程任务、处理中途变更，并把讨论结果转化为可使用的应用能力。

8.4.1 场景中的硬问题

一位技术顾问带着 Coactive Agent 与医院客户进行第二轮方案讨论。第一轮已经交付病理咨询与科室推荐应用，正在院内运行。客户这次提出新需求：患者到院后能否用 3D 导航完成全流程，从挂号、取号、签到，到找诊室、付费、取药，一路获得指引。

生成一个导航页面只是表层任务。客户列出 6 个导航节点，每个节点的位置类型不同：窗口、自助机、签到台、诊室、柜台、药房窗口。院区存在电梯、楼梯、无障碍通道等多条路线，现场还可能在讨论中追加约束。顾问和客户希望在同一次会议里完成需求确认、方案设计、中途变更和交付验收；在传统流程中，这通常会被拆成会议纪要、需求评审、排期、开发、测试和再次验收。

方案讨论本身就是一个正在形成的任务现场。客户提到“上次方案”时，记忆要能出现；客户描述新需求时，结构化卡片要能形成；顾问授权后，后台任务要能推进；中途变化出现时，已经完成的工作不应被整体推倒重来。

8.4.2 传统方案的不足

会议记录工具能转录对话，却不知道“上次方案”指的是哪个版本，也无法判断这句话应该唤醒哪段组织记忆。项目管理系统能拆解任务，却不能在讨论进行时同步结构化需求并启动后台工作。代码生成 Agent 能写出导航页面，但当客户中途追加“付费机要按排队推荐、轮椅通道要标注”时，往往只能重新生成受影响部分，难以保留已完成的建模和路径规划。

传统流程把讨论、设计、实现和验收拆在不同时间里。Coactive 关注的是在边界清楚、授权明确、可回退的前提下，把这些环节组织进同一个协作过程。

8.4.3 Coactive 如何解决

顾问桌面上的 Coactive Agent 以 Interactive 状态参与讨论。客户提到“上次方案”时，Agent 从组织记忆召回上一轮方案的上下文，包括病理咨询模块、科室推荐逻辑、已部署版本和当前限制，并以结构化卡片呈现：“已找到上次方案，你想在此基础上叠加全流程 3D 导航，对吗？”，这类动作属于跨会话记忆在讨论现场的唤醒，并非简单关键词命中。

客户逐一列出 6 个导航节点，Agent 实时将口头描述整理为带位置类型的导航目标卡片。顾问确认后，Agent 将结构化需求委托给 Proactive 后台。这是一次明确的 Interactive 到 Proactive 切换。后台并行推进多个子任务：3D 平面图建模、6 个目标位置坐标采集、路径规划、楼层切换逻辑、室内定位策略、患者端 UI 设计。顾问和客户继续讨论其他事项，大屏右侧显示后台进度。

客户中途追加需求：“门诊楼多个付费机要按候诊人数动态推荐，还有电梯位置和无障碍通道也要标注”。Agent 识别新约束后更新正在进行的后台任务：保留已完成的平面图建模和路径规划，只新增两个受影响子任务。这里体现的是长程任务的增量更新能力，后台执行可以在变化中继续推进，每次变更无须从头开始。

方案完成后，产物作为应用能力包回到 Interactive 侧，经过校验、注册、安装和激活，生成一个二维码。客户当场扫码，手机上出现从挂号处到各科室的 3D 路线指引。从需求提出到应用可用，都发生在同一次会议中。该应用只处理导航和指引，不参与挂号、付费等业务交易，这些仍由医院既有系统处理。

8.4.4 该场景证明的 Coactive 价值

方案共创场景的 Coactive 判据如表 8-4 所示。方案共创场景验证的是多项机制的连续运转。会议现场、组织记忆、后台承接和能力沉淀被放进同一个可控过程；人保留确认权，已确认的部分继续推进为可用能力。传统 Agent 与 Coactive 在这一场景中的协作差异如图 8-9 所示。

表 8-4 方案共创场景的 Coactive 判据

判据	在方案共创场景中的体现
共享现场如何建立	上一轮方案记忆、客户口头需求、顾问确认、6 节点结构化卡片、后台任务进度组织为同一讨论现场
哪些片段被承接	需求结构化、平面图建模、路径规划、UI 设计、应用能力打包与激活
什么事件触发唤醒	客户提到”上次方案”触发记忆召回、客户中途追加约束触发后台任务更新
回退路径	回到顾问人工记录和方案整理；应用能力未通过校验时保留草案；后台任务可随时暂停或取消

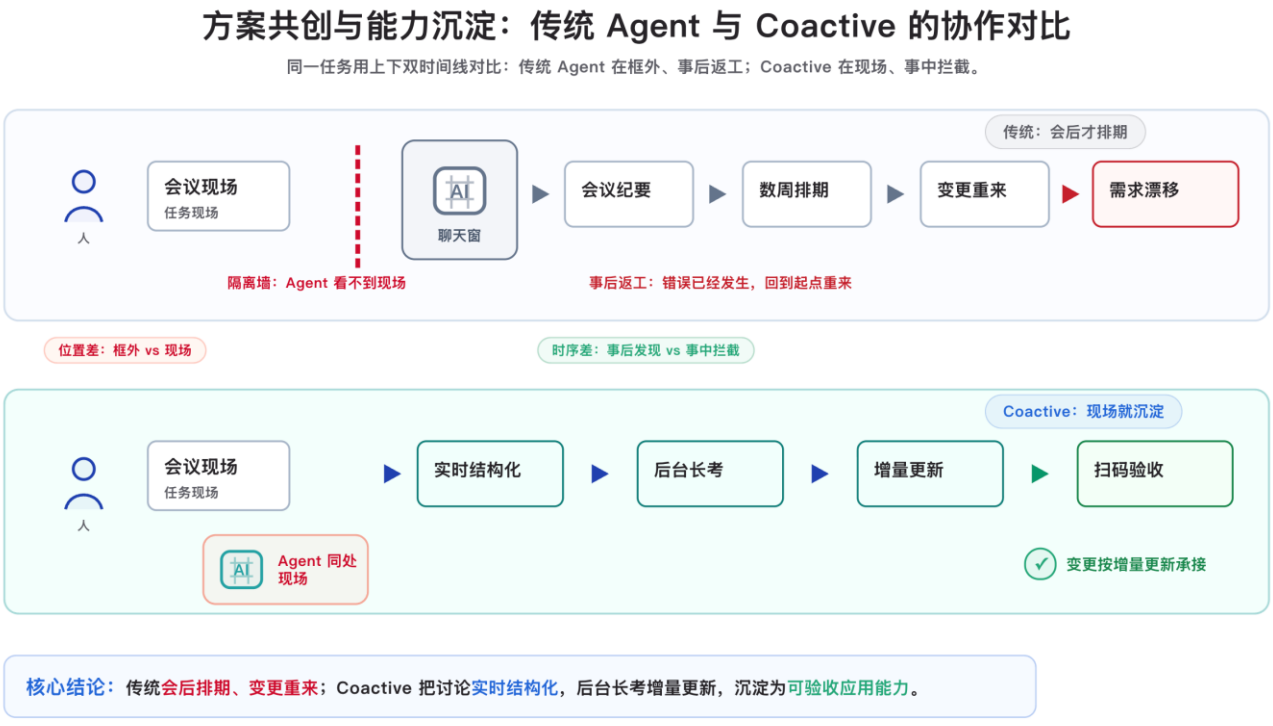


图 8-9 方案共创与能力沉淀：传统 Agent 与 Coactive 的协作对比

8.5 本章小结

如表 8-5 所示，四个场景分别检验 Coactive 的不同侧面。品牌内容发布强调 Interactive 在操作现场放大人的判断；财务月报强调可信承接需要承接条件和审计链；RFP 写作强调记忆必须进入输入时刻；方案共创则把记忆召回、需求结构化、后台长程任务和能力沉淀放在同一个过程里。

表 8-5 四类协作痛点场景与 Coactive 机制总览

场景	证明的 Coactive 优势	解决的硬问题
品牌内容发布	Interactive 实时共处与共同判断，稳定后可信承接	高频操作中夹杂表达、合规与风格判断，传统工具无法在操作过程中理解和调整
财务月报编制	可信承接、异常唤醒、审计链	财务结果必须可解释、可追溯、可审批、可回退，不能黑箱生成
客户方案与 RFP 写作	Interactive Memory 在输入过程中实时推荐、补证与事中校验	组织经验分散、口径易过期，用户写作时容易形成错误承诺
方案共创与能力沉淀	记忆召回、实时结构化、Proactive 长程任务、中途更新与能力沉淀的完整闭环	方案讨论中的共识要等数周开发才能变成产品，中途变更导致推倒重来

这些场景也对应第六章提出的企业级价值：财务月报的可信承接与审计链支撑规模化运行的稳定性；各场景中的回退路径支撑能力成长的可治理性；从方案讨论到可扫码使用的 3D 导航应用能力，体现组织经验的结构化复用。

CHAPTER TAKEAWAY

四类场景指向同一结论：Coactive 让协作过程可验证、可接管、可回退、可形成能力。不确定处共创，成熟处承接，讨论中的共识进入后续可用的应用能力。

参考资料

[1] Shah, A. “Gartner Sees Untamed Growth in Agentic AI.” Computerworld, 2026-04-30. https://www.computerworld.com/article/4165686/gartner-sees-untamed-growth-in-agentic-ai.html

[2] Schumacher, K., Keutel, M., Kluge, P., Giebel, K., and Wendler, T. “Europe’s Agentic Commerce Moment: Decision Influence Is Here; Execution Is Coming.” McKinsey & Company, 2026-03-02. https://www.mckinsey.com/capabilities/quantumblack/our-insights/europes-agentic-commerce-moment-decision-influence-is-here-execution-is-coming

[3] Team Sequoia. “AI Ascent 2026.” Sequoia Capital, 2026-05-08. https://sequoiacap.com/article/ai-ascent-2026/

[4] Kwa, T., West, B., Becker, J., et al. “Measuring AI Ability to Complete Long Software Tasks.” arXiv:2503.14499, 2025.

[5] Herschel, G., and Pidsley, D. “Human-AI Delegation Framework for Decision Augmentation.” Gartner, 2024-09-19. https://www.gartner.com/en/documents/5775515

[6] NVIDIA. “Autonomous AI Agents.” NVIDIA Glossary, n.d. https://www.nvidia.com/en-us/glossary/ai-agents/

[7] Shneiderman, B. “Human-Centered Artificial Intelligence: Reliable, Safe & Trustworthy.” International Journal of Human–Computer Interaction, 2020.

[8] de Oliveira, D., and Segner, M. “Getting Started with Loops.” Claude by Anthropic, 2026-06-30. https://claude.com/blog/getting-started-with-loops

[9] NIST. “Artificial Intelligence Risk Management Framework (AI RMF 1.0).” NIST AI 100-1, 2023.

[10] Lu, Y., Yang, S., Qian, C., et al. “Proactive Agent: Shifting LLM Agents from Reactive Responses to Active Assistance.” arXiv:2410.12361, 2024.

[11] Thinking Machines. “Interaction Models: A Scalable Approach to Human-AI Collaboration.” Thinking Machines Lab, 2026-05-11. https://thinkingmachines.ai/blog/interaction-models/

[12] Horvitz, E. “Principles of Mixed-Initiative User Interfaces.” CHI Conference on Human Factors in Computing Systems, 1999.

[13] Amershi, S., Weld, D., Vorvoreanu, M., et al. “Guidelines for Human-AI Interaction.” CHI Conference on Human Factors in Computing Systems, 2019.

[14] Anthropic. “Building Effective Agents.” Anthropic Engineering, 2024-12-19. https://www.anthropic.com/engineering/building-effective-agents

[15] Srinivasan, A., Shamdasani, S., Araujo, A. F., and de Wasseige, J. “Building Self-Improving Tax Agents with Codex.” OpenAI Engineering, 2026-05-27. https://openai.com/index/building-self-improving-tax-agents-with-codex/

[16] Khatua, A., Zhu, H., Tran, P., et al. “CooperBench: Why Coding Agents Cannot Be Your Teammates Yet.” arXiv:2601.13295, 2026.

[17] Dey, A. K. “Understanding and Using Context.” Personal and Ubiquitous Computing, 2001.

[18] Park, J. S., O’Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. “Generative Agents: Interactive Simulacra of Human Behavior.” ACM Symposium on User Interface Software and Technology, 2023.

[19] Packer, C., Wooders, S., Lin, K., et al. “MemGPT: Towards LLMs as Operating Systems.” arXiv:2310.08560, 2023.

[20] GoogleCloudPlatform. “Open Knowledge Format (OKF) SPEC.” Version 0.1 Draft, n.d. https://github.com/GoogleCloudPlatform/knowledge-catalog/blob/main/okf/SPEC.md

[21] Zhang, J., Xiang, J., Yu, Z., Teng, F., et al. “AFlow: Automating Agentic Workflow Generation.” International Conference on Learning Representations, 2025.

华为技术有限公司

www.huawei.com