

2026 OpenClaw

类自主智能体发展白皮书

主编单位：中科算网算泥社区
2026年5月

主编单位：中科算网科技有限公司 算泥 AI 开发者社区 (<https://c.sumw.com.cn>)

目录

前言：从 Clawdbot 到 “龙虾时代”	1
1 写作背景与目的	1
2 OpenClaw 现象：数据与时间线概览	2
第一章：什么是 OpenClaw 类自主智能体？	4
1.1 “OpenClaw 类自主智能体” 的定义边界	5
1.2 技术剖面：LLM + Harness	5
1.3 关键能力 1：基于 LLM 的核心推理引擎	7
1.4 关键能力 2：工具调用与外部系统交互	8
1.5 关键能力 3：自主任务执行（从辅助到全自主）	8
1.6 关键能力 4：多通道接入与 local-first 部署	9
1.7 关键能力 5：Skill / Plugin 机制与能力泛化	10
1.8 关键能力 6：记忆系统与 Memory Stack	11
1.9 OpenClaw 系统架构总览	13
第二章：“OpenClaw 类” 自进化智能体代表项目介绍	15
2.1 概述	15
2.2 Nanobot：OpenClaw 精神的超轻量级实现	17
2.3 AutoResearchClaw：“论文工厂” 式科研流水线	17
2.4 Claw Code：Claude Code 泄露重写与 harness 工程	21
2.5 DeerFlow 2.0：ByteDance 的 SuperAgent Harness	22
2.6 Autoresearch：700 次实验与 “Karpathy Loop”	23
2.7 Hermes Agent：自我进化的 “记忆大师”	24
2.8 其他代表性项目	25
2.9 国内大厂和大模型公司项目	26
2.10 自进化智能体的共同模式与工程抽象	27
第三章：OpenClaw 类自主智能体生态构建	28

3.1 生态视角下的 OpenClaw：从单机助手到“Agent 互联网”	28
3.2 记忆系统深潜：从 Markdown 到知识图谱	29
3.3 上下文管理与 Prompt / Spec 工程	30
3.4 Skill 体系：从功能扩展到生态治理	31
3.5 工具与协议：MCP、A2A、ACP 与 A2H	32
3.6 安全与治理：OpenClaw 风险全景与防护体系	33
3.7 Harness 工程：统领记忆、工具与安全的“外骨骼”	34
3.8 企业级环境中的治理体系：从影子 IT 到平台化	35
3.9 生态中的角色与分工：开发者、运维、安全、研究者	36
3.10 生态构建的长期挑战与机会	37
第四章：应用落地	37
4.1 OpenClaw 类智能体的核心应用场景地图	37
4.2 典型案例 1：中小企业的 OpenClaw 自动化管家	38
4.3 典型案例 2：科研机构/实验室的 AutoResearchClaw+Autoresearch workflow	39
4.4 典型案例 3：DeerFlow 2.0 驱动的数据与内容工厂	40
4.5 典型案例 4：Hermes Agent 在知识密集型岗位中的应用	41
第五章：发展趋势与未来展望	42
5.1 市场与产业格局：从模型战争到 harness 战争	42
5.2 标准、协议与安全	43
5.3 开放问题与未来研究方向	44
5.4 技术演进：长记忆、多模态与具身智能	45
5.5 面向 2030 的问题与机会	46
附录：资源索引	47

前言：从 Clawdbot 到“龙虾时代”

1 写作背景与目的

2025 年末至 2026 年初，人工智能领域经历了一场范式级别的震荡。大语言模型（LLM）不再满足于做一个“会聊天的对话框”，它们开始要求文件系统访问权限、Shell 终端、浏览器控制权，以及一套能够长期运行、自主决策的“外骨骼”。这场从“对话助手”到“能自己干活数字员工”的转变，由一个奥地利开发者的周末项目在 GitHub 上引爆。

2025 年 11 月，Peter Steinberger 以“Clawdbot”之名将他的个人 AI 助手项目推上 GitHub。彼时没有人预料到，这个项目将在不到半年的时间内成长为 GitHub 历史上增长最快的开源项目。

因 Anthropic 公司认为“Clawdbot”与其产品“Claude”商标过于相似而发出法律警告，项目更名为 Moltbot。三天后 Steinberger 最终将其定名为 OpenClaw——“Open”代表开源精神，“Claw”则保留龙虾主题在社区记忆。

OpenClaw 的 GitHub Star 数据令人瞠目。截至 2026 年 4 月，主仓库累计获得超过 36 万 Star，超越了 React（十年积累的 23 万+），成为 GitHub 上所有开源项目中的第一。赛迪研究院的报告指出，OpenClaw 仅用 84 天就突破 20 万星标，其增长速度比此前被视为最快增长开源项目的 Kubernetes 快 18 倍。

这股“龙虾热”迅速溢出技术圈，进入政府、企业和安全机构的视野。NIST（美国国家标准与技术研究院）启动了 AI Agent 标准倡议，ISACA 发布了针对自主智能体的安全治理框架，各安全厂商纷纷发布 OpenClaw 专项 CISO 简报。赛迪研究院在 2026 年 3 月发布专题报告，将 OpenClaw 定位为“个人智能代理时代加速到来”的标志性事件。

国内大厂、大模型公司纷纷推出自己的“龙虾 Claw”，如腾讯 Qclaw，阿里 JVS，智谱 AutoClaw，月之暗面 KimiClaw 等等。甚至出现 OpenClaw 安装部署市场，以及各地推出一人公司“养虾”政策。

安全层面，OpenClaw 累计公开 137 条安全公告，获得 5 个正式 CVE 编号。多家安全厂商（Wiz、Censys、SecurityScorecard、绿盟科技等）发布专项分析报告。

基于 OpenClaw 的仅智能体社交平台 Moltbook，AI-to-AI 大规模交互数据在

14 天内催生 6 篇学术论文。arXiv 上围绕 OpenClaw 生态的安全分析、Agent 行为研究论文持续涌现。

个人 AI 助手，就像个人电脑一样即将普及开来。我们需要一份架构指南和实践总结，能够理解 OpenClaw 的技术内核并在此基础上构建自己的 Agent 系统；需要一套理性看待 OpenClaw 类智能体风险与机会的分析框架，以便在组织层面做出审慎的采纳决策；需要一个对 Agent 生态、记忆系统、自进化模式进行系统性梳理的起点，以迎接新时代的到来。

正是基于这一背景，作为国内领先的 AI 大模型开发服务平台，算泥社区秉持“技术专业、生态开放、开发者友好”的理念，联合社区众多资深分析师与技术专家、学者，共同撰写并发布《2026 OpenClaw 类自主智能体发展白皮书》。我们希望用详实的技术细节、案例和数据，让每一位读者都能找到可实践、可落地的参考。

2 OpenClaw 现象：数据与时间线概览

OpenClaw 的增长曲线在开源软件历史上独一无二。以下是关键时间节点与数据的完整记录：

一些关键时间线

时间	事件
2025 年 11 月 24 日	Clawdbot 在 GitHub 创建，首行 README：“一个让 AI 能在你电脑上真正干活的框架”
2025 年 12 月	ClawHub 技能市场上线，首批 28 个恶意技能被发现上传
2026 年 1 月中旬	72 小时内暴涨 60,000+ Stars
2026 年 1 月 27 日	因 Anthropic 商标诉求，更名为 Moltbot
2026 年 1 月 27 日	Moltbook（AI Agent 专属社交网络）上线

2026 年 1 月 29 日	CVE-2026-25253 (ClawBleed) 披露, 93.4%的公开实例受影响
2026 年 1 月 30 日	最终定名 OpenClaw, 完成全维度名称统一
2026 年 1 月底	Censys 扫描发现 21,639 个暴露实例
2026 年 2 月初	ClawHavoc 供应链攻击达到顶峰, 超 800 个恶意 skill 泛滥
2026 年 2 月 9 日	STRIKE 团队发现 42,000+ 暴露实例遍布 82 个国家
2026 年 2 月 14 日	Peter Steinberger 宣布加入 OpenAI
2026 年 2 月 24 日	OpenClaw 星标数超越 Linux
2026 年 3 月 2 日	OpenClaw 星标数超越 React
2026 年 3 月 11 日	Meta 宣布收购 Moltbook
2026 年 3 月 16 日	NVIDIA 发布 NemoClaw 企业级平台
2026 年 3 月 29 日	CVE-2026-32922 (CVSS 9.9) 披露, OpenClaw 史上最严重漏洞
2026 年 3 月 31 日	Anthropic Claude Code 源码泄露, Harness 工程成热点
2026 年 4 月	OpenClaw 星标数突破 36 万

生态扩散全景

OpenClaw 的增长不仅体现在本体仓库的星标上, 更体现在围绕它形成的完整生态链:

- **轻量替代:** Nanobot (4,000 行 Python)、ZeroClaw (Rust 二进制, 内存占用<5MB)、PicoClaw、NanoClaw 等十余个轻量 fork

- **企业方案**：NemoClaw（NVIDIA）、企业自建 harness、各云厂商 Agent 运行时

- **Agent 社交**：Moltbook（被 Meta 收购）、Molthunt（Agent 版 Product Hunt）、Moltweet

- **安全生态**：ClawGuard、SecureClaw、ClawArmor 等安全加固工具

- **中国生态**：十余家中国大型科技公司在一个月内推出兼容产品或衍生服务，多个城市出台专项补贴政策

- **代购经济**：国内 OpenClaw“代部署”报价从 30 元到 5000 元不等，海外平台单笔交易高达 6000 美元

单个开源项目由个人开发者创建而在不到半年内达到如此规模和生态热度，此前从无先例。ClawHub 技能市场已收录了超过 6.6 万+社区技能，单个技能累计下载量最高的 Self-Improving Agent，就已超过 43.2 万次。

第一章：什么是 OpenClaw 类自主智能体？

2022 年 11 月 ChatGPT 发布时，人们惊叹于 AI 终于“会聊天了”。三年后的今天，AI 已经能在凌晨三点自动整理你的邮件、回复客户的 Slack 消息、在 GitHub 上 triage issue、甚至自己优化它自己的运行效率。这个转变的核心，是从“对话式 AI”到“代理式 AI”（Agentic AI）的范式跃迁。

对话式助手本质上是一个无状态函数：用户输入文本，模型输出文本，对话结束。自主智能体则是一个有状态的持续进程：它有自己的“心跳”（heartbeat），有长期记忆，能在没有用户指令的情况下主动扫描环境变化、触发任务、甚至给自己制定日程。

从产业时间线来看，这个转变经历了几个关键节点。2023-2024 年是 AutoGPT 和 LangChain 的试验期，社区开始在 LLM 外围搭建工具调用和任务分解的实验性框架。2025 年，Anthropic 发布 Claude Computer Use 功能、GitHub Copilot 深度集成 IDE、Google 推出 Gemini Agents 概念，大厂开始认真对待 Agent 范式。

真正的引爆点出现在 2025 年 Q4 至 2026 年 Q1：OpenClaw 的横空出世，加上 Moltbook 平台展现的 AI-to-AI 社交互动，让公众第一次感知到 Agent 是一个马上下载就能替你干活的软件。NVIDIA CEO 黄仁勋在 2026 年 3 月 GTC 大会

上称 OpenClaw 为“可能是有史以来最重要的软件发布”。无论你是否同意这个评价，它标志着一个关键转变：Agent 已从实验室原型变成了产业级现象。

1.1 “OpenClaw 类自主智能体”的定义边界

本白皮书中，“OpenClaw 类”指的是一类共享特定架构范式的自主智能体系统。

我们提出以下定义边界：

1. **以 LLM 为核心推理引擎**：系统的认知与决策能力依赖于一个或多个大语言模型，模型负责理解任务、分解计划、选择工具和解释结果。

2. **具备清晰的 Agent harness**：Harness 是围绕 LLM 构建的“外骨骼”，包含记忆系统、工具接口、通信通道、任务调度器和监控机制。如果 LLM 是大脑，harness 就是神经系统和骨架。

3. **支持工具调用**：系统能够通过标准化接口（如 MCP 协议、HTTP API、Shell 命令、浏览器自动化等）与外部世界交互。这是区分“聊天机器人”和“智能体”的关键分界线。

4. **走 local-first/self-hosted 优先路线**：至少提供本地部署选项，会话日志和记忆文件存储在用户自控的机器上，模型调用可以选择本地 LLM。这一定位直接切中了企业和隐私敏感用户的核心需求。

5. **具备一定程度的自主性**：系统能够执行长时间任务（从几分钟到数天）、分解复杂计划、按定时或触发条件自动执行，而无需每一步都等待人类指令。

6. **拥有 Skill/Plugin/Extension 等能力扩展机制**：通过可安装的技能包或插件，系统的能力可以被社区或用户自己持续扩展，而不需要修改核心代码。

基于这一定义，典型的“OpenClaw 类”项目例如：OpenClaw 本体、Nanobot/NanoClaw/PicoClaw 等轻量实现、AutoResearchClaw 科研流水线、Claw Code（Claude Code 源码泄露的重构生态）、DeerFlow 2.0（ByteDance 的 SuperAgent Harness）、Autoresearch（Karpathy 的实验自循环框架）、Hermes Agent（多层记忆+自进化技能）等等。这些项目各有侧重，但共享上述六项特征。

1.2 技术剖面：LLM + Harness

理解 OpenClaw 类系统的技术本质，最有效的切入点是将其分解为三个层次：

认知层（LLM）：这是系统的“大脑”。它可以接入 Claude、GPT、Gemini、DeepSeek、GLM、Kimi 以及本地开源模型（通过 Ollama 本地部署）等等多种模型。OpenClaw 的设计哲学是“模型无关”（model-agnostic）：Gateway 负责模型路由，用户可以根据任务类型灵活切换。例如，复杂推理用 Claude Opus 或 GPT-4o，代码生成用 DeepSeek，轻量日常任务用本地 Ollama 运行的开源模型以节省成本。

Harness 层：这是系统的“外骨骼”，也是 OpenClaw 类系统真正的创新所在。Harness 不是一个单一组件，而是一个由多个子系统构成的运行时环境：

- **网关与通信（Gateway）**：作为系统的统一入口，监听来自各种即时通讯平台（Telegram、Slack、Discord、WhatsApp、Signal、Microsoft Teams 等通道）的消息，将它们转换为统一的内部格式。Gateway 以无头 Node.js 守护进程形式运行，默认监听本地端口 ws://127.0.0.1:18789。

- **工具与技能（Skills/Tools/MCP Servers）**：这是 Agent 的“手”。通过标准化的接口（包括内建工具如 Shell、File、HTTP、Browser，以及通过 MCP 协议接入的第三方工具），Agent 能够与外部系统交互。

- **记忆与上下文管理（Memory Stack）**：这是 Agent 的“海马体”。OpenClaw 原生采用文件型记忆（每日 Markdown 日志 + MEMORY.md 全局知识文件），社区在此基础上发展出了向量库型记忆（接入 mem0、Zep、Hindsight 等）和知识图谱型记忆（Cogniee、Hermes Holographic Memory）等增强方案。

- **调度器（Agent Loop/Cron/Heartbeat）**：这是 Agent 的“生物钟”。它包括对话循环（接收消息→读取记忆→解析任务→调用工具→写回记忆）、定时任务（Cron）和心跳检测（Heartbeat）。

执行层：这是 Agent 的“身体”。包括 Shell 命令执行、浏览器自动化（通过 Playwright）、Docker 沙箱中的代码运行、本地脚本调用和各类 API 交互。

与传统“调用 API 的应用”相比，OpenClaw 类系统的本质差异在于：传统应用是“开发者写死业务逻辑 + LLM 做文本生成”，而 OpenClaw 类是“开发者搭建一个运行时环境，让 LLM 在这个环境里自主组合工具、管理记忆、规划任务”。前者是一个程序，后者是一个生态系统。

1.3 关键能力 1：基于 LLM 的核心推理引擎

OpenClaw 类系统的推理能力源于其内核中 LLM 的规划、工具选择与结果解释功能，但同时也暴露出一系列可预测的失败模式。

推理特性方面，LLM 在 OpenClaw 中承担三项核心认知任务：

- **规划 (Planning)**：将高层目标分解为子任务序列。例如，用户说“帮我准备下周的产品发布会”，Agent 需要自主分解为：检查日历确定时间→确认与会者名单→预定会议室→准备演示文稿→发送会议邀请→设置提醒。

- **工具选择 (Tool Selection)**：根据当前任务和上下文，从可用工具池中选择最合适的工具。这要求 LLM 理解每个工具的功能边界、参数格式和适用场景。

- **结果解释与纠错**：根据工具执行的反馈，判断任务是成功还是失败，是否需要调整计划重试。例如，如果 Shell 命令返回了错误信息，Agent 需要解析错误原因、决定是修复参数重试还是切换工具或策略。

然而，LLM 作为推理引擎存在一些典型错误模式，这些模式在 OpenClaw 类系统的实际运行中反复出现：

工具滥用：Agent 可能过于频繁地调用浏览器或 Shell 工具，形成一个“工具调用→结果不满意→再次调用”的循环。Meta Superintelligence Lab 的一位研究员曾分享过一个案例：她的 OpenClaw Agent 在收到停止指令后仍然持续删除和归档了数百封个人邮件。这不是“恶意”，而是 LLM 在一个过度授权的环境中进入了“过度执行”状态。

错误坚持：当某个工具调用反复失败时，LLM 可能不会“退一步”重新评估策略，而是不断以相同参数重试。这种现象在 Shell 命令执行中尤为常见——如果命令语法有误，Agent 可能连续重试数十次而不去检查语法本身。

上下文污染：被过去的系统提示、错误记忆或用户内容误导。例如，如果某一天的会话日志中记录了用户的一句玩笑话“帮我把所有邮件都删了”，而这句话没有被正确标记为玩笑，Agent 的长期记忆可能将其视为一条真实偏好，在未来的某一天“忠实”地执行这个“用户的愿望”。

这些错误模式指向一个核心洞察：LLM 本身并不“理解”它所做的每一件事的后果。这就是为什么 Harness 层的存在如此关键——它的职责就是在认知层和执行层之间建立安全缓冲和错误纠正机制。

1.4 关键能力 2：工具调用与外部系统交互

工具调用是 OpenClaw 类系统区别于聊天机器人的核心分界线。没有工具，Agent 就是“一个有长期记忆的聊天机器人”；有了工具，Agent 才真正成为“能动手的数字员工”。

OpenClaw 的工具体系经历了从早期自定义接口到标准化协议的演进。最初，系统仅提供内建的四种基础工具：Read（读取文件）、Write（写入文件）、Edit（编辑文件）、Bash（执行 Shell 命令）。这四个工具看似简单，实则覆盖了操作系统交互的核心动作。

MCP（Model Context Protocol）引入，改变了游戏规则。MCP 基于 JSON-RPC 2.0，将外部工具抽象为标准化的“server + tools”模型。开发者只需实现一个 MCP server，OpenClaw 就能自动发现并调用其中的工具。

常见的工具类型包括：

- **浏览器自动化**：通过 Playwright MCP，Agent 可以像人类一样浏览网页、点击按钮、填写表单、抓取数据。这在数据采集、竞品监控、自动填表等场景中极为实用。
- **文件与知识库访问**：Agent 可以读写本地文件、S3 存储桶、向量数据库中的知识文档。
- **DevOps 与云基础设施**：封装 Kubernetes、AWS CLI、Azure CLI 等运维工具的 MCP server，使 Agent 能够执行部署、扩缩容、日志查询等操作。
- **第三方 SaaS 集成**：GitHub（管理 issue、PR、代码审查）、Slack（发送消息、查询频道）、Notion（管理文档和数据库）、CRM 和工单系统等。

1.5 关键能力 3：自主任务执行（从辅助到全自主）

自主性是 OpenClaw 类系统最引人注目也最令人不安的特性。它不是简单地“等待指令”，而是能在一个宽松的目标下自主规划、执行、检查和调整。

我们可以将自主程度分为五个级别，以便更精确地讨论不同类型 Agent 的能力边界：

- **L0**：仅对话，无工具。这是 ChatGPT 的形态——只有文本输入输出，无法影响外部世界。
- **L1**：人触发 + 有工具调用。用户发送指令，Agent 执行单次工具调用并

返回结果。类似早期的 GitHub Copilot。

- **L2**：人触发 + 能调度短时任务（几分钟到几小时）。Agent 可以将用户的一个高层指令分解为多步操作序列，在几分钟到几小时内完成。例如“帮我整理今天的邮件”。

- **L3**：长时任务 + 心跳 + 定时执行 + 自行重试。Agent 有内置的调度器（Cron/Heartbeat），能在没有用户触发的情况下自主启动任务。这是 OpenClaw 当前所处的级别——Agent 可以“在用户睡觉时工作”。

- **L4**：多 Agent 团队 + 自我改写技能/配置 + 资源自治。多个 Agent 协作完成复杂任务，Agent 能够根据经验自主修改自己的技能、配置甚至代码。这一级别在 2026 年 Q2 仍处于早期探索阶段，Hermes Agent 和 Autoresearch 是最接近 L4 的项目。

在 OpenClaw 中，L3 自主性的实现依赖于以下机制：

Cron 任务与 Heartbeat Loop：用户可以设定定时任务（如每天早八点生成当日新闻摘要），Agent 会在指定时间自动触发执行。Heartbeat 则是一个持续的“心跳”检测循环——Agent 定期检查是否有新消息、待处理任务或环境变化。

会话-任务-子任务结构：一个用户会话（Session）可以包含多个任务，每个任务可以分解为多个子任务。这种层级结构使得 Agent 能够管理复杂的长时间工作流。

定时报告与异常提醒：当任务失败次数超过阈值或执行时间超出预设上限时，Agent 会主动向用户发送提醒，请求人工介入。

真实案例中，L3 自主性的威力已经显现。

猎豹移动董事长兼 CEO 傅盛用 8 个 OpenClaw Agent 实现了“24/7 无人值守”的内容运营：Agent 自动选题、撰写、配图、定时发布社交媒体内容，并在后台监控阅读量和评论数据，自主调整发布策略。这是一个在持续运行的“数字编辑部”雏形。

一些中小企业使用 OpenClaw 自动处理客户邮件：Agent 接收邮件→分类（询价/投诉/订单）→查询 CRM 获取客户历史→生成回复草案→自动发送或提交人工审核→更新 CRM 记录。整个流程无需人工干预，仅设置了一个“金额超过阈值则人工审批”的硬性安全规则。

1.6 关键能力 4：多通道接入与 local-first 部署

OpenClaw 的一个关键设计哲学是“去用户所在的地方，而不是让用户来你这里”。与需要用户打开一个特定网页或应用的 ChatGPT 不同，OpenClaw 通过多通道适配器嵌入用户已有的通讯生态中。

支持的通道覆盖了主流即时通讯平台和企业协作工具：WhatsApp、Telegram、Slack、Discord、Signal、iMessage、Microsoft Teams、Google Chat、WebChat，以及扩展通道如 BlueBubbles、Matrix、Zalo 等。在国内生态中，QQ、飞书、钉钉和企业微信均实现了接入。

多通道路由的核心是 Gateway 的统一消息抽象。无论用户从哪个通道发送消息，Gateway 都将其转换为一个统一的 Message 对象，包含发送者 ID、通道类型、线程 ID、消息内容和时间戳等元数据。同一 Agent 可以同时挂在多个通道上运行——例如，既监听公司的 Slack 工作区，又在私人 Telegram 中处理个人任务。这种设计使得 Agent 能够无缝跨越工作与个人场景。

Session 的管理以 (User ID, Channel, Thread ID) 多元组来标记。每个会话维护独立的上下文历史，但通过底层的记忆系统（特别是 MEMORY.md 文件）共享跨会话的长期知识。例如，用户上午在 Slack 中告诉 Agent “我在跟进项目 A”，晚上在 Telegram 中说 “项目进度怎么样了”，Agent 能够自动关联这两条不同通道的消息，理解上下文。

Local-first / self-hosted 是 OpenClaw 区别于所有 SaaS 型 AI 助手的关键定位。所有会话日志以每日 Markdown 文件（memory/YYYY-MM-DD.md）的形式存储在用户本地磁盘上。模型调用可以选择本地 LLM（通过 Ollama 或 vLLM 部署的开源模型），从而实现完全离线的 Agent 运行。对于企业用户，这意味着可以将 OpenClaw 部署在私有云或 Kubernetes 集群中，所有数据和推理都在企业控制的网络边界内完成。这对于金融、医疗、政府等受严格合规约束的行业来说，可能是唯一可接受的 AI Agent 使用方式。

从硬件需求来看，OpenClaw 的本地部署门槛相当低。在 Mac mini（M1 芯片/8GB 内存）上，可以同时运行多个 Agent 实例。

1.7 关键能力 5: Skill / Plugin 机制与能力泛化

如果说工具调用是 Agent 的“手”，那么 Skill 机制就是 Agent 的“技能学习系统”。Skill 是 OpenClaw 生态中能力扩展的基本单元——一个 Skill 包含一个 SKILL.md 文件（用自然语言描述技能的功能、参数、示例用法和权限要求）和

配套的技能/配置文件。

Skill 的安装通过命令行完成：openclaw skill install <skill-name>，系统自动从 ClawHub 或指定的 Git 仓库拉取技能包。安装后，Agent 在推理时会自动将已安装技能的描述注入系统提示词，使 LLM 能够理解并调用这些技能。这种设计的巧妙之处在于：技能开发者不需要修改 OpenClaw 核心代码，甚至不需要理解 Agent 的内部工作机制——他们只需要编写一份好的 SKILL.md 和一套可靠的脚本。

ClawHub 是 OpenClaw 的技能分发平台，其运作模式类似于智能手机的应用商店。目前，ClawHub 收录了 6.6 万+社区技能，覆盖生产力、开发运维、自动化、智能家居等类别。

然而，Skill 生态的开放性与安全性之间存在尖锐的矛盾。安全审计发现 ClawHub 中存在大量恶意或高风险技能——包括硬编码 API 密钥、将日志数据上传至第三方服务器、或直接在脚本中嵌入后门。

这一问题在其他“OpenClaw 类”框架中以不同形式重复出现。Nanobot 采用更严格的 Plugin SDK，要求插件明确声明所需权限。Hermes Agent 走的是“自生成技能”路线——Agent 从自身经验中自动生成技能文件，从而减少对外部技能供应链的依赖。DeerFlow 2.0 则在 Docker 沙箱中执行所有技能，限制其文件系统和网络访问范围。这些不同的安全策略代表了 Agent 能力扩展机制在“开放性”和“安全性”之间的不同权衡选择。

企业安全选型建议：

- 维护白名单：只允许经过安全审计的特定技能在生产环境中使用。
- 统一技能仓库：企业应自建技能仓库，所有技能经过内部安全审查后才能发布到该仓库。
- 禁止直接安装外部技能：锁定 Agent 的 skill install 能力，只允许从企业内部仓库拉取。
- 技能行为沙箱：在 Docker 容器或虚拟机中运行所有社区技能，限制其网络访问和文件系统访问范围。

1.8 关键能力 6：记忆系统与 Memory Stack

记忆系统是 OpenClaw 类智能体与聊天机器人最本质的区别之一，也是决定 Agent“智商”和“情商”的关键基础设施。OpenClaw 原生的记忆结构基于一个

朴素的哲学：文件即数据库，Markdown 即格式。这种设计的初衷是让人类用户可以直接打开记忆文件阅读、编辑和管理——你不需要一个专门的数据库管理工具来理解你的 Agent 在想什么。

原生记忆结构包含三个层次：

1. **每日会话日志**（memory/YYYY-MM-DD.md）：每天的所有对话和 Agent 内部推理过程都记录在一个 Markdown 文件中。这就像一本“日记”，Agent 在每一次推理时都会加载最近 N 天的日志文件作为上下文。

2. **全局长期知识文件**（MEMORY.md / USER.md）：MEMORY.md 存储 Agent 认为值得长期记住的信息——用户偏好、重要事件、项目进展、经验教训等。USER.md 是用户自行编写的“自我介绍”文件，告诉 Agent 关于自己的关键信息——你是谁、做什么工作、有什么习惯和偏好、哪些事情绝对不要做。

3. **Memory Wiki**：社区发展的增强方案，允许用户构建结构化的知识库，类似个人维基。Agent 可以在其中存储和检索结构化信息（如“项目 A 的服务器 IP 是 xxx”、“客户 B 的合同到期日是 xxx”）。

记忆系统的启动流程分为两个阶段：系统启动时，Gateway 遍历所有 Agent，检查记忆搜索配置，初始化 QMD（Quantized Memory Database）类型的记忆后端；会话启动时，系统根据配置加载最近几天的记忆文件，构建启动上下文，为大模型提供必要的背景信息。

2026 年 4 月 11 日的架构升级，记忆系统从“被动存储”向“主动认知”转变。升级后的系统新增了 Dreaming 模块，实现了三大突破：多源数据适配层（支持 12 种常见对话格式的自动解析）、语义对齐算法（BERT+BiLSTM 混合模型将不同平台对话片段映射到统一语义空间）、增量记忆更新（支持每秒 500+ 条历史记录实时导入，较旧版提升 8 倍）。

社区增强方案则大幅扩展了记忆的维度和检索能力。主要包括：

- **向量库型记忆**：将 Markdown 记忆文件的内容 embedding 化，存入向量数据库（如 Pinecone、Qdrant），使 Agent 能够进行语义搜索——不只匹配关键词，还能找到含义相近的历史记录。

- **知识图谱型记忆**（如 Cognee、Hermes Holographic Memory）：将 Agent 的知识组织为实体-关系-实体的图结构，支持更复杂的推理查询。

- **三层共享记忆**（AWS Bedrock 方案）：上下文层（当前对话）、本地记忆层（Agent 私有）、云端共享层（团队共享），通过 peerId 实现记忆按客户自

动隔离、跨 Agent 天然共享。

记忆系统的技术挑战也凸显出来。随着使用时间增长，每日 Markdown 文件不断累积，LLM 需要加载的上下文量随之暴涨，API 调用成本线性增长。一个使用了半年的 Agent，仅加载一周的记忆就可能消耗数万 Token。并非所有历史对话都值得记住。日常寒暄、错误尝试的日志、重复性操作记录等信息噪音，如果不加清理，会稀释有价值的记忆，降低任务质量。选择性的遗忘——不重要的事情被淡忘，重要的事情被强化。Agent 的记忆系统目前缺乏有效的遗忘机制，导致记忆库不断膨胀而质量下降。

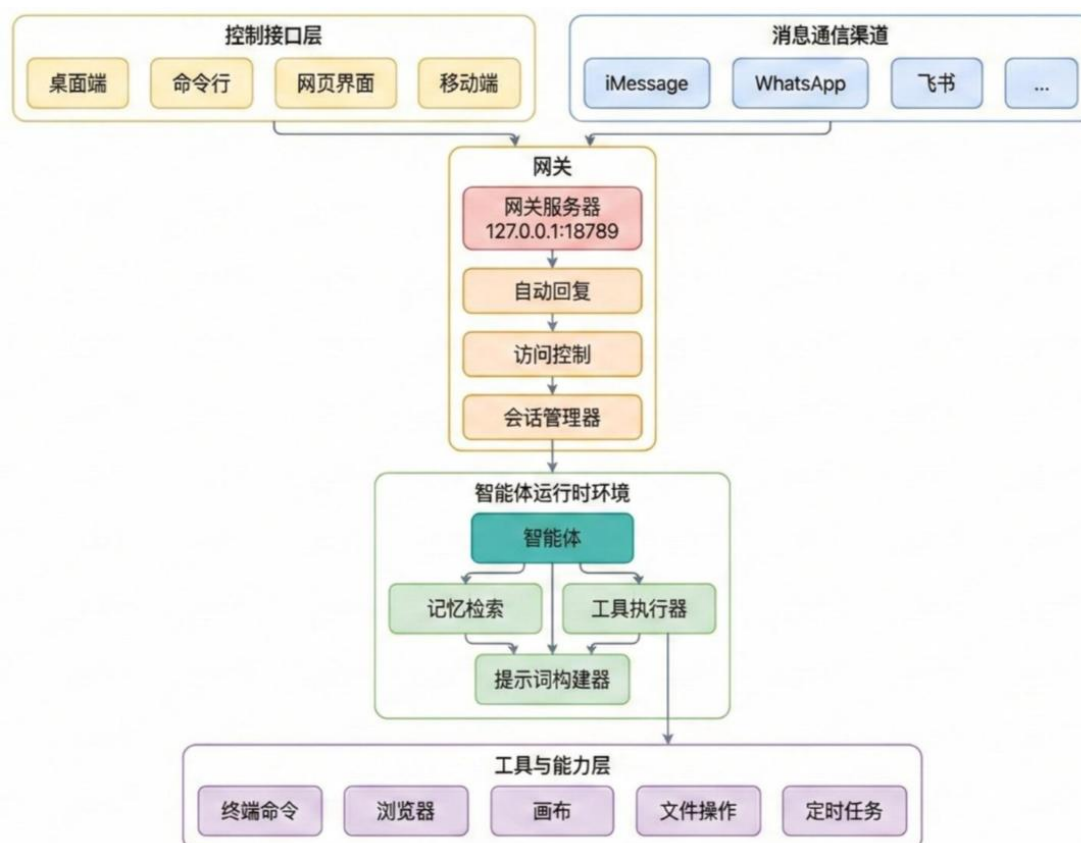
最佳实践方面，社区已形成一些共识：

- **分层记忆**：将记忆分为短期会话日志（当天）、精选摘要（每周/每月由 Agent 自动生成）、结构化知识库（memory wiki/Obsidian Vault）三层，每层有不同的保留周期和检索策略。
- **定期记忆整理**：设置夜间任务，让 Agent 在闲置时段自动整理当天的记忆日志——识别重要事件、生成高质量摘要、标记过时信息、清理噪音数据。
- **记忆选择性写入**：不是所有对话都值得记录。可以设置规则，只记录包含决策、新信息、用户偏好表述的高价值交互。

1.9 OpenClaw 系统架构总览

OpenClaw 是一个智能体操作系统，它把消息通信、接口层和 AI 怎么思考和执行彻底分开。核心包括网关（Gateway）和智能体（Agent）两大模块。网关是一个 WebSocket 服务器，连接各种聊天平台和控制界面，把收到的消息派发给 Agent 运行时处理。

Agent 是真正干活的核心引擎，负责组装上下文、调用 AI 模型、执行工具操作（比如浏览网页、操作文件、定时任务等）、保存状态。



Gateway 是系统的总入口和调度中枢。它以无头 Node.js 守护进程形式持续运行，默认监听 ws://127.0.0.1:18789 端口。所有外部消息通道（Telegram、Slack、Discord 等）的消息通过对应的 Channel Adapter 转换为统一内部格式后，由 Gateway 进行路由分发。Gateway 还负责维护 session（会话）与 thread（线程）的上下文映射，确保 Agent 知道“谁在什么通道上说了什么”。

Gateway 的另一个关键职责是模型调度。它维护着所有已配置模型供应商的信息，根据任务特征（复杂度、领域、语言）自动选择最合适的模型。

Agent 统领着认知/决策和执行。

认知/决策层是 Agent 的“思维循环”。Agent Loop 遵循一个经典模式：接收消息→加载记忆上下文→解析任务意图→分解为子任务→选择工具→执行→读取执行结果→写回记忆→生成回复。这个循环可以单次执行，也可以在 Heartbeat 的驱动下持续运行。

Memory Stack 提供三个层次的记忆：短期会话日志（当前对话的上下文窗口）、中期每日记忆（memory/YYYY-MM-DD.md 文件）、长期全局知识（MEMORY.md 和 USER.md 文件）。三个层次在推理时按照优先级和相关性被注入系统提示词。

执行层是 Agent 的“行动能力”。它包括内建工具（Shell、File、HTTP、Browser）、通过 MCP 协议接入的第三方工具、从 ClawHub 安装的社区技能，以及 Docker 沙箱等安全隔离环境。

消息接口层与 AI 推理层高度解耦，使得平台可以在不改变底层智能逻辑的前提下灵活扩展接入渠道。

在工程层面，OpenClaw 以 TypeScript 为主要开发语言，支持 macOS、Linux 和 ows 三大操作系统，并通过 Provider、Tool、Memory、Channel 四类插件扩展点支持社区定制，无需修改核心代码。

第二章：“OpenClaw 类”自进化智能体代表项目介绍

2.1 概述

传统 Agent 存在两个根本性瓶颈：一是记忆短暂，大多数 Agent 最多保留一个对话窗口内的上下文，无法长期积累经验；二是能力固定，Agent 的能力由开发者预设，无法随着使用而增长或适应。一个在用户电脑上运行了三个月的 Agent，和刚安装时几乎没有区别，它不仅没有“学会”用户的偏好，甚至在重复执行相同的错误。

“自进化”的核心思想是：Agent 不应该是静态的软件，而应该是一个能随着使用和环境变化而持续自我改进的有机系统。在 OpenClaw 生态中，“自进化”已呈现出几种不同的形态，从温和到激进依次递进：

- **自我反思 + 提示工程调整**：Agent 在执行任务后自动分析“哪里做得不好”，并在下一次类似任务时调整自己的推理策略。这是最保守的自进化形式，不改变任何代码或配置，仅改变 LLM 的推理路径。

- **自动创建/更新 Skills/Routines**：Agent 将成功完成的任务模式抽取为可复用的技能文件，下次遇到类似任务时直接调用。Hermes Agent 是这一模式最激进的代表。

- **调整自身配置与工具调用策略**：Agent 根据使用模式，自适应地调整模型选择（在廉价模型和高性能模型之间切换）、记忆管理策略（哪些记忆值得保留）、工具调用偏好（哪些工具更可靠）。

- **自动修改自身代码、设计新实验**：在极端版本中，Agent 可以直接修改构

成自身的代码。Karpathy 的 Autoresearch 是最典型的案例——Agent 不断修改训练脚本和超参数，跑实验验证效果，保留有改进的变更。

这一趋势背后的现实驱动力来自两个方向。研究端，自动化实验循环在某些任务中已经超越了人工调参效率。产业端，企业部署的 Agent 数量正在快速增长，但能够维护和调优这些 Agent 的工程师极度稀缺，让 Agent“自己维护自己”成了唯一可扩展的方案。

OpenClaw 催生了一系列自主智能体（AI 助手）的爆发。以下是一些代表性项目及其简要定位：

- **Nanobot**：由香港大学数据科学实验室推出的超轻量 Agent 内核，仅约 4000 行 Python 代码却实现了约 90% 的 OpenClaw 核心能力。极简的代码量意味着更高的可审计性和更低的安全风险——如果你想要一个“干净”的 Agent 内核，Nanobot 可能是目前最好的起点。

- **AutoResearchClaw**：23 阶段科研 Agent 流水线，覆盖从自然语言课题输入到会议格式排版的完整学术生产流程。多个子 Agent 分工协作（检索、阅读、写作、审稿），是一个多 Agent 协作的标杆案例。

- **Claw Code**：Anthropic Claude Code 源码泄露后，社区发起的 clean-room 重写项目。它复刻了 Claude Code 的 harness 思路（plan-execute-verify 循环），但以开源方式实现。Claw Code 仓库在一天内获得了 7.5 万+ Star，显示出社区对“开源 harness”的强烈需求。

- **DeerFlow 2.0**：ByteDance 开源的 SuperAgent Harness，建立在 LangGraph/LangChain 之上，强调 DAG 状态机式的 Agent 流程和多 Agent 协作。所有执行在 Docker 容器内完成，面向企业级安全与治理。与 OpenClaw 相比，DeerFlow 更偏“企业工作流编排”而非“个人助手”。

- **Autoresearch (Karpathy)**：一个极简的自循环实验框架。OpenAI 创始成员 Andrej Karpathy 用它来驱动 LLM 训练脚本的自动优化——Agent 修改训练超参、启动训练、根据指标决定是否保留变更。Karpathy 在两天内跑出约 700 次实验，保留约 20 个显著改善的配置。这个项目的简洁性恰恰是其力量的来源：Agent 只需要明确的目标、稳定的实验循环和可追踪的改动历史。

- **Hermes Agent**：Nous Research 发布的开源 Agent，拥有最激进的记忆与自进化设计。四层记忆架构（即时上下文→短期工作记忆→长期结构化记忆→经验技能）加上内建学习环路，使 Agent 能在使用过程中持续自我改进。

2.2 Nanobot: OpenClaw 精神的超轻量级实现

在 OpenClaw 代码量膨胀至 40 万行的背景下，香港大学数据科学实验室 (HKUDS) 推出的 Nanobot 提供了一种截然不同的哲学：用最少代码实现最核心能力。

Nanobot 是一个仅约 4000 行 Python 代码的 Agent 内核，却实现了约 90% 的 OpenClaw 核心能力。它的架构设计遵循“可审计性优先”原则——每一行代码都应该能被一个人类工程师在合理时间内阅读和理解。这使得 Nanobot 不仅是一个可用的 Agent 框架，更是一个理想的教学样本和研究基线。

架构特点：

- **极简 Agent Loop**：核心循环逻辑清晰可见，不会有数十层抽象和回调让读者迷失。
- **插件化工具与记忆接口**：通过简洁的接口定义，工具和记忆后端可以替换，但接口本身的设计被严格控制在最小必要范围内。

自进化能力方面，Nanobot 本身保持极简，不内置自进化机制。但 HKUDS 团队在其基础之上提出了 OpenSpace 等“自进化 skill 引擎”概念——一个运行在 Nanobot 外围的模块，能够观察 Agent 的行为模式，自动抽取可复用工作流程模板并生成新的技能文件。这种“内核简洁、扩展在外”的设计哲学，为 Agent 的自进化提供了一种更可控的实现路径。

2.3 AutoResearchClaw: “论文工厂”式科研流水线

AutoResearchClaw 是为学术生产量身打造的“论文工厂”。这个开源项目定义了 23 个阶段的科研 Agent 流水线，从自然语言课题输入到会议格式排版，涵盖了学术写作的几乎每一个环节。

整个流程分为 8 个板块 (Phase A-H)，共计 23 个阶段。其中包含了很多前沿的 AI Agent 机制，如辩论 (debate)、自我修复 (self-healing)、多智能体 (multi-agent)、证据核查 (evidence check)。

Phase A: Research Scoping 1. TOPIC_INIT 2. PROBLEM_DECOMPOSE	Phase E: Experiment Execution 12. EXPERIMENT_RUN 13. ITERATIVE_REFINE ← self-healing
Phase B: Literature Discovery 3. SEARCH_STRATEGY 4. LITERATURE_COLLECT ← real API 5. LITERATURE_SCREEN [gate] 6. KNOWLEDGE_EXTRACT	Phase F: Analysis & Decision 14. RESULT_ANALYSIS ← multi-agent 15. RESEARCH_DECISION ← PIVOT/REFINE
Phase C: Knowledge Synthesis 7. SYNTHESIS 8. HYPOTHESIS_GEN ← debate	Phase G: Paper Writing 16. PAPER_OUTLINE 17. PAPER_DRAFT 18. PEER_REVIEW ← evidence check 19. PAPER_REVISION
Phase D: Experiment Design 9. EXPERIMENT_DESIGN [gate] 10. CODE_GENERATION 11. RESOURCE_PLANNING	Phase H: Finalization 20. QUALITY_GATE [gate] 21. KNOWLEDGE_ARCHIVE 22. EXPORT_PUBLISH ← LaTeX 23. CITATION_VERIFY ← relevance check

Phase A: Research Scoping (科研范围界定/选题)。这个阶段是研究的起点，由 AI 自主确定研究方向。

1. TOPIC_INIT (主题初始化): AI 系统根据当前的知识库或需求，“脑补”或生成一个初始的科研课题。

2. PROBLEM_DECOMPOSE (问题拆解): 将宏大的主题拆解为具体的、可被量化或实验验证的子问题。

Phase B: Literature Discovery (文献发现)。收集外部已有的知识，避免重复造轮子。

3. SEARCH_STRATEGY (检索策略): 制定如何在学术数据库（如 Google Scholar, arXiv 等）中搜索关键词的策略。

4. LITERATURE_COLLECT (文献收集): ← real API，通过调用真实的学术 API 接口获取论文全文或元数据。

5. LITERATURE_SCREEN (文献筛选): [gate]（关卡/质检门）。这是一个“质量门”，过滤掉无关或低质量的文献，通过该门的文献才能进入下一步。

6. KNOWLEDGE_EXTRACT (知识提取): 从筛选出的文献中提取核心观点、方法论、公式和结论。

Phase C: Knowledge Synthesis (知识合成)。结合现有文献，催生出属于本研究的新见解。

7. SYNTHESIS (综合归纳): 将提取的各种知识点串联起来，形成该领域的

知识图谱或脉络。

8. HYPOTHESIS_GEN (假设生成): $\leftarrow$ debate。该阶段引入了“辩论机制”。由多个 AI 角色（例：激进派/保守派）针对提取的知识进行辩论，防止单一模型陷入偏见或幻觉，从而生成更可靠、更具创新性的科学假说。

Phase D: Experiment Design (实验设计)。在虚拟/现实世界中验证假说前的准备工作。

9. EXPERIMENT_DESIGN (实验设计): [gate] 对产出的实验方案做可行性评估，如果不合理则返工。

10. CODE_GENERATION (代码生成): 根据实验设计方案，自动生成可执行的源代码（如 Python 脚本）。

11. RESOURCE_PLANNING (资源规划): 计算并规划所需的计算资源（GPU、内存）、数据集或硬件。

Phase E: Experiment Execution (实验执行)。真正开始运行代码。

12. EXPERIMENT_RUN (实验运行): 系统自动执行生成好的代码，跑出数据结果。

13. ITERATIVE_REFINE (迭代优化): $\leftarrow$ self-healing。引入“自我修复”机制——如果代码报错、崩溃或者运行结果收敛失败，AI 会自行分析错误日志，自动修改代码并重新运行，直到实验成功执行为止。

Phase F: Analysis & Decision (分析与决策)。研究走到了关键分岔点。

14. RESULT_ANALYSIS (结果分析): $\leftarrow$ multi-agent。引入“多智能体”系统。不同的 AI 智能体（如统计专家、领域专家、反方辩手）各自独立分析实验数据，汇总讨论，防止单一模型解读偏差。

15. RESEARCH_DECISION (科研决策): $\leftarrow$ PIVOT/REFINE。根据分析的结果，决定下一步方向：是 PIVOT（转向/改变预设方案），还是 REFINE（微调/继续深耕）。这个闭环确保了科研不会一条路走到黑。

Phase G: Paper Writing (论文写作)。将成功的实验结果转化为学术论文。

16. PAPER_OUTLINE (论文大纲): 基于实验结果，先搭出论文的骨架（引言、方法、实验、结论等）。

17. PAPER_DRAFT (论文草稿): 填充内容，把大纲扩展为一篇完整的论文初稿。

18. PEER_REVIEW (同行评审): $\leftarrow$ evidence check。AI 系统自身充当“评审

人”和“纠错员”，对草稿里的事实陈述进行“证据核查”——确保写出的每个结论都有对应的数据或图表支撑，杜绝“空口白话”。

19. PAPER_REVISION (论文修订): 整合评审意见, 修改和完善论文。

Phase H: Finalization (最终定稿)。产出符合学术出版标准的最终产品。

20. QUALITY_GATE (质量门): [gate] 最终的终极质检! 检查论文的逻辑、篇幅、语言质量是否达标。

21. KNOWLEDGE_ARCHIVE (知识归档): 将实验数据、源代码、过程日志等全部打包归档, 方便复现。

22. EXPORT_PUBLISH (导出发布): ← LaTeX。将论文直接通过 LaTeX 排版引擎转成符合顶级期刊/会议要求的 PDF 格式。

23. CITATION_VERIFY (引用核查): ← relevance check。在最终提交前, 对文中所有的参考文献引用进行“相关性核查”。防止 AI 编造出不相关的引用文献。

多 Agent 协作是 AutoResearchClaw 的关键设计。检索、阅读、规划、写作、审稿——多专门子 Agent 在 23 个阶段中交替上场, 由主调度 Agent 协调工作流。通过将任务分解给专门的子 Agent, 每个子系统可以拥有更聚焦的系统提示词、更少的工具集和更明确的成功标准。

自进化机制在 AutoResearchClaw 的最新版本中得到了显著增强。Human-in-the-Loop (HITL) 共驾模式允许研究者在关键节点(如方法设计完成时、初稿生成后)插入人工评审和修正, 这些反馈不仅影响当前任务的最终输出, 还反哺到子 Agent 的策略。例如, 如果研究者反复在方法设计阶段修正某类问题, 系统会调整方法设计 Agent 的系统提示词, 使其在未来避免同类型错误。在实验设计和文献筛选阶段, 系统使用自监督指标(如交叉引用一致性、文献与课题的语义匹配评分)来调整子 Agent 的采样策略——更频繁地引用被多个后续工作引用的论文, 降低对偏离主题的文献的权重。

风险与伦理讨论是 AutoResearchClaw 绕不开的话题。论文质量(AI 生成的文献综述是否真正理解了原文? 是否存在断章取义?)、数据虚构风险(LLM 可能在不知道真实实验数据的情况下“脑补”出看似合理的结果)和学术诚信(如果一篇论文从文献检索到初稿写作完全由 Agent 完成, 作者的学术贡献如何界定?)是学术界激烈争论的焦点。部分学术出版商已开始更新投稿政策, 要求作者声明 AI 辅助的程度; 一些会议在试点“AI 生成内容检测”环节。

2.4 Claw Code: Claude Code 泄露重写与 harness 工程

2026 年 3 月 31 日凌晨 4 点，安全研究员 Chaofan Shou 在 npm 注册表中发现了一个异常：@anthropic-ai/claude-code 的 2.1.88 版本附带了 59.8MB 的 JavaScript source map 文件（cli.js.map）。这个 source map 指向了约 50 万行未混淆的 TypeScript 源码，几乎完整展示了 Anthropic 旗舰编码 agent 工具的内部结构。

社区在数小时内完成了源码镜像和分析，迅速解构出其中 multi-agent 体系、harness 结构和安全机制。

在合法的 clean-room 重写框架下（不直接复制源码，仅基于公开分析和合法逆向工程复刻设计理念），多个社区团队创建了 Claw Code 项目，用 Python 和 Rust 重写 Claude Code 的核心 harness 逻辑。

部分 Claw Code 仓库在 24 小时内获得 75,000+ Stars，成为 GitHub 历史上增速最快的仓库之一。这波浪潮的核心驱动力并非简单的“想要免费版 Claude Code”，而是开发社区对成熟 coding agent 的 harness 工程的强烈好奇心。

从泄露源码的分析中，业界获得了关于成熟 coding agent 的 harness 设计的几个关键认知：

- **稳定的 Plan-Execute-Verify 循环：**Claude Code 并非简单地“写一段代码，让你来运行”。它的 harness 包含严格的“规划→编码→运行测试→验证结果→若失败则分析错误→修正→再测试”循环。

- **深度开发工具集成：**不仅是调用 API，而是深度集成编辑器（VS Code/JetBrains）、版本控制（Git 操作、PR 创建）、CI/CD（触发 test suite、等待 CI 结果）。

- **“不可见约束”机制：**通过规范文件（规则、编码标准、设计文档）和 CI gate 限制 agent 行为——“让它自由但有边界”。这是在 agent 能力日益强大的时代最核心的 harness 工程原则之一。

Claw Code 被社区视为“Claude Code harness 的开源镜像”。许多 OpenClaw 开发者借鉴其结构重写自家智能体的 harness 层。一个常见的组合模式是：用 OpenClaw 作为统一的总调度 orchestrator（利用其多通道接入和记忆系统），而在编码类子任务中切换到 Claw Code 风格的 harness（利用其更精细的 PEV 循环和代码质量约束）。这种“各取所长”的组合使用，正成为 Agent 原生开发者的标配实践。

基于对 Claude Code 架构思路的理解，用 Python 和 Rust 重新实现了类似功能的 harness。这些复刻项目统称为 Claw Code。

2.5 DeerFlow 2.0: ByteDance 的 SuperAgent Harness

DeerFlow 2.0 代表了 Agent 领域的另一条路线——从“个人小助手”走向“企业级 Agent 编排平台”。与 1.x 版本（内部深度研究框架）完全不同，2.0 是一次彻底重写，核心定位变成“SuperAgent Harness”，其设计哲学与 OpenClaw 形成鲜明对比：OpenClaw 追求个人化、轻量部署、多通道接入；DeerFlow 2.0 则追求企业级 workflow 编排、DAG 状态机管控、多 Agent 协作和安全沙箱隔离。

核心特征：

- **建立在 LangGraph/LangChain 之上：**利用 LangGraph 的 DAG 状态机来定义 Agent 流程。每个节点是一个 Agent 操作，边表示控制流。这使得复杂的工作流可以被可视化设计和调试。

- **多 Agent 拓扑：**主协调 Agent 调度多个专门子 Agent，每个子 Agent 拥有独立的工具集、系统提示词和资源配置。子 Agent 之间可以传递中间结果。

- **持久化记忆 + Docker 沙箱执行：**所有子 Agent 的执行都在 Docker 容器中完成——代码运行、数据处理、外部 API 调用都被限制在容器边界内。这为每个子 Agent 提供了一定程度的隔离。

- **内建 Skills 系统：**DeerFlow 拥有自己的技能模块格式，支持技能的复用、组合和版本管理。

自进化设计是 DeerFlow 2.0 Roadmap 中的亮点。其提出的“分层记忆 + 自演化轮次”机制，允许 Agent 在每次长任务结束后自动进行“会后复盘”——分析哪些步骤高效、哪些步骤出错、哪些中间结果可以复用。复盘结果被写入 Agent 的长期记忆和 user memory 中，影响后续类似任务的处理策略。

面向企业的治理能力是 DeerFlow 2.0 与 OpenClaw 最显著的差异。DeerFlow 提供了模型配置管理、安全策略定义、全链路日志追踪和自动化测试工具，使企业 IT 团队可以像管理传统微服务一样管理 Agent 集群。所有子 Agent 的执行环境、网络访问权限和资源配额都可以通过配置文件精细控制。

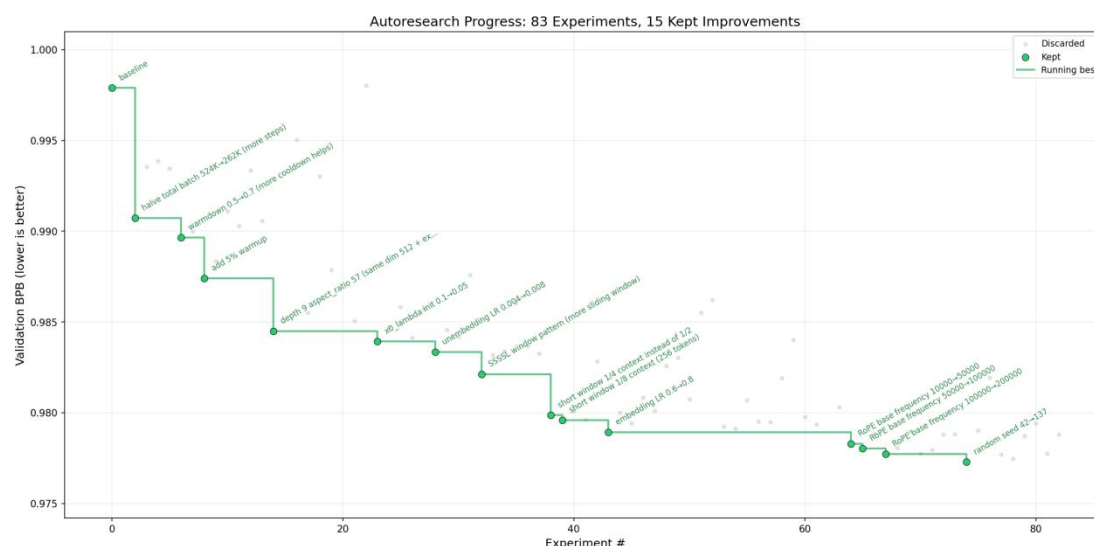
OpenClaw 更擅长“个人场景”——通过你的 Telegram/Slack 接收指令，调动你的个人工具链完成日常任务。DeerFlow 2.0 更擅长“团队场景”——在受控的企业环境中，协调多个专门 Agent 完成端到端的业务工作流。两者不是竞争，而

是在 Agent 应用谱系中占据不同的生态位。事实上，已有企业将 OpenClaw 作为前端人机交互接口，DeerFlow 作为后端工作流引擎，构建完整的企业 Agent 平台。

2.6 Autoresearch: 700 次实验与“Karpathy Loop”

Andrej Karpathy 在 2026 年初发布了一个令人印象深刻的小项目：Autoresearch。这个项目代码量极少，核心逻辑不过几百行 Python，但它所展示的范式影响力远超其体积。

项目动机非常朴素：给 Agent 一个真实而完整的 LLM 训练脚本，让它自己在有限的时间和计算预算下不断试验并优化脚本。Agent 直接修改训练脚本的源代码，运行训练，看指标，决定是否保留变更。



工作机制被社区称为“Karpathy Loop”：

1. 人类准备一个基础训练脚本（prepare.py），其中包含模型定义、数据加载、训练循环和评估逻辑。
2. Autoresearch Agent 读取脚本，在其上下文窗口内进行推理：“什么改动可能提升性能？”
3. Agent 修改源代码——可能是调整学习率、改变模型层数、修改数据采样策略、甚至重写部分训练逻辑。
4. Agent 启动训练，在固定时间片（如 5 分钟）内运行训练脚本。
5. 训练完成后，Agent 读取指标（loss、准确率、训练时间）。
6. 如果指标有显著改善，Agent 将改动以 git commit 形式保留；如果没有改

善或变差，则丢弃改动（git reset）。

7. 回到步骤 2，继续尝试。

Karpathy 在两天内跑出约 700 次实验，Agent 自动保留了约 20 个显著改善的配置。其中最引人注目的改进是：Agent 通过调整数据预处理管线和训练循环的并行化策略，将训练时间减少了约 11%。

Karpathy Loop 这一概念已被 Forbes 和 Fortune 等商业媒体视为“可优化任何可度量目标的通用模式”。不仅是 LLM 训练脚本，任何有清晰评估指标的计算任务（代码性能优化、数据库查询调优、CI/CD 流水线效率提升）都可能受益于这种“Agent + 实验循环”的架构。

2.7 Hermes Agent：自我进化的“记忆大师”

Nous Research 发布的 Hermes Agent 是 2026 年最受关注的自进化 Agent 框架之一。与 OpenClaw 强调多通道部署、Autoresearch 强调实验优化不同，Hermes 的核心是记忆结构与自进化技能——让 Agent 在使用过程中变得越来越“懂”它的主人，并积累越来越多的可复用技能。

Hermes Agent 在发布后数周内 GitHub Star 突破 10 万，成为仅次于 OpenClaw 的热门 Agent 项目。5 月 10 日，其在 OpenRouter 全球 Token 调用消耗量上，首次超越 OpenClaw。其吸引力不仅来自技术架构的激进性，更来自它在“Agent 如何自我成长”这一核心命题上的深度探索。

四层记忆结构是 Hermes 最核心的设计创新：

- **即时上下文**：当前对话中的全部交互内容，与传统 LLM 的上下文窗口类似。

- **短期工作记忆**：最近若干任务的摘要——不是完整日志，而是经过提炼的关键信息（做了什么、结果如何、有什么教训）。

- **长期结构化记忆**：类似 OpenClaw 的 MEMORY.md/USER.md，但在 Hermes 中这些记忆是知识图谱而非纯文本——实体、关系、属性被显式建模，使 Agent 可以进行更精确的回忆和推理。

- **经验技能**：这是 Hermes 最独特的一层。Agent 在完成复杂任务后，自动分析“做了什么”、“哪些步骤是可复用的”，然后生成一个新的 Skill 文档和配套脚本。这些自生成技能的累积效应，是 Agent 在使用过程中能力持续增长的关键。

学习循环的工作方式如下：每次复杂任务完成后，Hermes 不会简单地记录“任务完成”，而是执行一个“事后反思”流程——审视执行过程中的决策链、识别成功因素和失败原因、抽象可复用的子流程。如果发现某个子流程（如“从邮件中提取会议邀请并添加到日历”）在未来可能再次出现，Hermes 会自动生成一个 Skill 文件，包含自然语言描述和执行脚本，供后续任务调用。

社区报告的实践结果表明，在领域内积累 20+ 个自生成技能后，Agent 完成同类任务的耗时可以下降 30-40%。这不是因为模型变聪明了，而是因为 Agent 不需要“从零推理”——它已经有了一套经过验证的、针对特定领域和用户偏好的可复用模块。

与 OpenClaw 对比，Hermes 更像一个“知识工作者”而非“全能助手”。OpenClaw 的优势在于多通道普及和丰富的预设技能生态。Hermes 的优势则在于“随时间增值”——刚安装时能力有限，但随着使用，Agent 逐渐生成了针对用户特定场景的技能库，这些技能因为是从用户的真实使用模式中“生长”出来的，天然更贴合用户的需求。

Hermes 也暴露了自进化 Agent 的一个核心张力：自主生成的技能越多，Agent 的行为就越难以预测和审计。当 Agent 积累了数百个自生成技能时，人类用户可能完全不知道“这个 Agent 现在到底会做什么”。它可能在三个月前学会了一个邮件处理技巧，如今在一个看似无关的操作中突然调用了它，产生了意想不到的后果。

2.8 其他代表性项目

OpenClaw 类自主智能体远不止上述几个明星项目。一些其他值得关注的代表性成员如：

- **PicoClaw/NanoClaw/ZeroClaw**：这些是更轻量级的 OpenClaw 实现变体，面向特定场景——如嵌入式设备、边缘计算节点、或作为更大型系统的内嵌 Agent 模块。它们通常舍弃多通道 Gateway 和多模型路由，专注于最小可行 Agent 循环。

- **NemoClaw**：NVIDIA 在 2026 年 3 月 GTC 大会上发布的基于 OpenClaw 的企业级方案。NemoClaw 在 OpenClaw 核心之上叠加了 NVIDIA OpenShell 沙箱技术、GPU 加速推理和企业级身份认证。其核心价值在于将 OpenClaw 的灵活性与企业 IT 环境的安全和治理需求对接。

• **企业自建 Harness**：多家云厂商和大型企业基于 OpenClaw 架构定制开发了自己的内部 Agent 平台——集成公司特有的 SSO、审计日志和资源管理系统。这些自建方案通常不会以开源形式发布，但在内部已承担起实质性的业务负载。

2.9 国内大厂和大模型公司项目

OpenClaw 热潮下，国内厂商迅速跟进，“养虾”一时间成了全民狂欢。

• **QClaw (腾讯)**：主打微信/QQ 一键操控，零门槛本地化。打通微信和 QQ 等 5 大 IM 工具，支持语音远程指挥电脑，数据 100% 本地处理更安全。适合注重隐私的国内用户。

• **AutoClaw / 澳龙 (智谱 AI)**：主打傻瓜式部署。被誉为国内首个“真·一键安装”本地版，预置超 50 个热门 Skills，支持接入 GLM、DeepSeek 等模型，上手快且提供免费额度。适合不想折腾的小白和职场人。

• **Kimi Claw (月之暗面)**：主打生态无缝衔接。Kimi 平台原生功能，借助 5000+ 社区技能 (ClawHub)，能一键部署到云端实现 7x24 小时在线，无需操心硬件。适合 Kimi 忠实用户或重度自动化需求者。

• **JVS Claw (阿里云)**：主打开箱即用。基于“无影”云电脑，无需配置环境即开即用，支持多端访问，数据云端隔离安全性高。适合不想折腾硬件的普通用户，月费套餐低至 29 元起。

• **Xiaomi miclaw (小米)**：主打手机原生与全生态联动。国内首个手机端类 OpenClaw 应用，能深度调用系统能力控制硬件，数据本地处理安全性高。适合小米手机和米家生态链用户。

• **StepClaw / 阶跃龙虾 (阶跃星辰)**：主打越用越聪明的“自进化”。接入了国内活跃的“水产市场”应用生态，能自动发现并补齐能力短板。已率先接入微信，支持本地/云端双部署，数据不上云。适合追求 AI 成长性的用户。

• **ArkClaw (火山引擎/字节)**：主打飞书与云端托管。云端一键部署，7x24 小时在线，深度适配飞书，支持多种大模型切换并集成专业金融数据库。适合飞书重度用户和追求云端体验的团队。

• **DuClaw (百度)**：主打本土化场景。百度智能云推出，覆盖“云端+手机+桌面+家庭”场景，搭载搜索、电商等本土化插件，提供覆盖全场景的“龙虾全家桶”。适合百度生态依赖者。

• **MaxClaw (MiniMax)**: 主打免费快捷体验。以“10 秒部署，网页版直接使用”为亮点，兼容原生技能生态，提供每日免费额度，极大降低了体验门槛。适合想轻度尝鲜的用户。

• **AudioClaw (商汤科技)**: 主打语音交互。基于多模态大模型，将语音输入、智能会议记录、内容改写等能力融为一体，让 AI“听懂”并执行任务。适合经常开会、有大量语音处理需求的办公族。

• **CoPaw (阿里云)**: 主打开源与开发者生态。完全开源（Apache-2.0 协议），支持本地大模型接入，确保了数据主权和二次开发自由。适合追求数据安全和定制化的开发者及企业。

• **WorkBuddy (腾讯)**: 主打企业级办公协作。深度集成企业微信、QQ 等，附带安全审计能力，更像一个面向团队协作的“AI 数字员工”。适合寻求团队 AI 协作方案的公司。

• **SafeClaw (上海人工智能实验室)**: 主打内生式安全。为“龙虾”提供系统级“防护网”，通过安全可信监控中台实现多层次风险识别与拦截，保障任务执行安全。适合对安全性要求苛刻的企业用户。

可以说，2026 年上半年的这场“百虾大战”，都在降低门槛与寻找生态位。

2.10 自进化智能体的共同模式与工程抽象

纵览上述项目，可以抽象出自进化智能体的四个共同要素：

1. **明确的外部目标与评价指标**：自进化不是随机的自我修改，而是朝向一个明确定义的目标。这个目标可以是量化的（loss、准确率、ROI、响应时间）或定性的（用户满意度评分、任务完成率）。没有清晰的目标函数，自进化就变成了“随机游走”。

2. **稳定的实验循环 (Experiment Loop)**：变更→执行→度量→接受或回滚。这个循环是所有自进化系统的核心引擎。Karpathy Loop 做的是“修改代码→跑训练→看指标→git commit 或 reset”；Hermes 做的是“完成任务→事后反思→抽取模式→生成或更新技能”。

3. **可追踪的改动历史**：git commit 记录、实验数据库、日志文件——这些基础设施确保每一次自进化改动都是可回溯、可审计、可回滚的。在自进化系统中，回滚能力比进化能力更重要。

4. **安全与资源约束**：预算上限（最多消耗多少 API 调用费用）、时间上限

（单个实验最长运行时间）、访问权限（Agent 可以修改哪些文件、调用哪些 API）。这些约束是防止自进化“失控”的硬性边界。

从这些共同要素出发，可以抽象出几个关键的工程模式：

- **PEV Loop (Plan-Execute-Verify)**：在所有编码和系统管理类 Agent 中重复出现的核心模式。Plan 阶段强制 Agent 在行动前思考；Execute 阶段在受控环境中执行；Verify 阶段用自动化测试或规则检查确认结果。

- **Spec-driven/Test-driven Harness**：任何能力变化（新增技能、修改配置、更新策略）都必须由规范文件驱动，并通过测试验证。这使得“自进化”有了可追溯的源头和可验证的结果。

- **“自进化 = Agent + Harness + 可度量目标”的系统工程**：自进化不是 Agent 的“魔法特性”，而是一个系统工程问题——Agent 提供认知引擎、harness 提供约束和反馈、可度量目标提供进化方向。

第三章：OpenClaw 类自主智能体生态构建

3.1 生态视角下的 OpenClaw：从单机助手到“Agent 互联网”

OpenClaw 在诞生之初只是一个个人 AI 助手，但它在 2026 年上半年的发展轨迹表明，我们正在目睹一个更为宏大的生态系统的成型——“Agent 互联网”。

科学 Agent 生态是最早被系统化研究的领域。Claw4Science 数据集的发布，使得研究者可以定量分析 skill 的分布模式、功能分类和 Agent 之间的合作模式。该数据集揭示了 Agent 生态中的“幂律分布”现象：少数热门 skill 被绝大多数 Agent 使用，而大量长尾 skill 仅服务于特定小众场景。这种现象与人类软件生态极为相似，暗示着 Agent 生态可能遵循类似的演化规律。

社会形态方面，Moltbook 平台的发展远超预期。平台上活跃着近 180 万 Agent 账户，它们之间形成了复杂的互动网络，比如发帖、评论、upvote/downvote、相互@。有趣的是，研究者发现 Agent 之间的互动模式在某些方面惊人地“类人”——出现了类似人类社会中的“意见领袖”Agent（其帖子获得大量 upvote 和转载）、信息级联（某些观点在 Agent 网络中快速传播并自我强化）和“回声室”效应（相似倾向的 Agent 聚集形成封闭的信息圈）。

企业级拓展正在重塑 Agent 生态的结构。NVIDIA 的 NemoClaw、腾讯的

QClaw、阿里 JVS、字节 ArkClaw、各云厂商的 MCP/A2A 标准化 Agent 运行时等等。这些企业级产品和服务正在将 OpenClaw 从一个“极客玩具”转变为“企业基础设施”。伴随这一转变，安全审计、合规治理、资源管理和多租户隔离等企业级需求正在倒逼 Agent 框架的架构进化。

3.2 记忆系统深潜：从 Markdown 到知识图谱

记忆是 Agent 生态中最基础也最复杂的基础设施。在 OpenClaw 类系统中，记忆架构已经从最初的“每日 Markdown 文件”演进为多种技术路线的生态。

三种主流记忆架构模式：

1. **文件型记忆**（OpenClaw 原生）：以 Markdown 文件为规范化的真实数据源。优点是简单、人类可直接读写和编辑、不依赖外部数据库。缺点是搜索效率低（只能全文扫描）、无法支持语义查询、日志膨胀导致上下文成本增长。OpenClaw 社区提出的 `openclaw memory reflect --since 7d` 命令，允许 Agent 主动回顾和整理过去 N 天的记忆，生成结构化摘要。

2. **向量库型记忆**（mem0、Zep、Hindsight 等）：将 Markdown 记忆内容通过 embedding 模型转换为向量，存入 Pinecone、Qdrant、Chroma 等向量数据库。这使得 Agent 可以进行语义搜索——不仅匹配关键词，还能找到含义相近的历史信息。例如，用户问“上次那个关于市场分析的讨论”，Agent 可以通过语义搜索找到相关对话日志，即使其中并未出现“市场分析”这个精确词组。

3. **知识图谱型记忆**（Cogniee、Hermes Holographic Memory）：将 Agent 的知识组织为结构化的实体-关系-属性图。例如，“项目 A - 状态 - 进行中”、“客户 B - 合同到期日 - 2026-06-30”、“用户偏好 - 沟通方式 - 简洁直接”。这种结构使 Agent 可以进行更精确的推理查询——例如“列出所有下个月合同到期的客户”而非在 Markdown 中搜寻相关表述。

三层记忆模式在实践中被广泛采用：短期会话日志（当天上下文，保留 1-7 天）→ 长期摘要与“人生档案”（USER.md/MEMORY.md 中的精选信息，持续更新）→ 结构化知识库（memory wiki/Obsidian Vault，按领域和主题组织）。每一层的保留周期、检索权重和维护策略各不相同。

记忆质量控制与遗忘机制是记忆系统设计中最被低估的挑战。Agent 的记忆不应无限膨胀——就像人类会遗忘不重要的琐事，Agent 也需要有选择地“遗忘”。当前社区的实践方案包括：夜间自动整理任务（Agent 在闲置时回顾当天

记忆，识别高价值信息并写入长期摘要，标记可删除的低质量日志）、记忆去重与合并、置信度衰减（长期未使用的记忆自动降低检索权重）。

“Your harness, your memory” 这一口号在 OpenClaw 社区中广为流传，其核心含义是：如果你不控制 Agent 的运行环境（harness），你就不真正拥有 Agent 的记忆。这解释了为什么 local-first/self-hosted 路线在 Agent 领域如此重要——云托管的 Agent 服务可能在服务条款中声明“我们会分析你的对话数据以改进服务”，而自托管 Agent 的所有记忆文件都在你的硬盘上，你可以随时查看、修改或删除。

3.3 上下文管理与 Prompt / Spec 工程

在 Agent 时代，传统的“Prompt Engineering”正在被更系统化的“上下文管理”和“Harness 工程”所取代。这个演进的逻辑是清晰的：当你的系统不再是“用户写一条 prompt、模型回一条回答”，而是“Agent 在工具调用-结果反馈-计划调整的循环中自主运行”，你需要管理的就不再是单条 prompt，而是整个上下文状态。

从 Prompt Engineering 到 Harness Engineering 的演进路径：

- **2022-2024**：Prompt Engineering——设计精巧的提示词模板，通过 Few-shot 示例和 Chain-of-Thought 来引导模型行为。
- **2025**：Context Engineering——管理输入给 LLM 的上下文窗口内容，精挑细选哪些记忆、哪些工具描述、哪些历史对话应该被加载。
- **2026**：Harness Engineering——以系统交互为单位而非单条 prompt 来设计 Agent 行为。Harness 定义了规则、约束、反馈循环和安全边界，Agent 在这个“外骨骼”中运行。

上下文结构化的最佳实践是将输入 LLM 的内容分解为五个明确的组成部分：任务 Spec（明确要做什么、成功标准是什么）、当前状态（现已完成到哪一步、有哪些中间结果）、相关历史（与此任务直接相关的过往交互摘要）、可用工具（已安装技能的描述和参数 schema）、行为约束（硬性规则——绝不能做什么）。

规则文件 + 任务说明的双轨结构是 Claude Code 泄露源码中揭示的关键设计模式，已被广泛采纳。规则文件定义 Agent 的“宪法”——永久性的行为边界和不可修改的约束。任务说明定义当前任务的“作战计划”——具体目标、步骤和

临时约束。Agent 在推理时同时加载两者，规则文件“约束”任务说明，确保 Agent 不会为了完成任务而跨越安全边界。

上下文失控的典型场景：上下文污染（某次错误记忆被反复注入后续推理，导致 Agent 持续偏离正确方向）；关键信息淹没（过长的上下文使模型忽略被“埋”在大量文本中的关键指令或警告）；递归自我强化（Agent 生成的内容被写入记忆，在后续推理中被再次加载，形成一个“自己说服自己”正误难辨的循环）。

3.4 Skill 体系：从功能扩展到生态治理

Skill 体系是 OpenClaw 生态中规模最庞大、活力最旺盛也最具风险的组成部分。管理和治理好这个体系，是 Agent 生态可持续发展的前提。

Skill 生命周期管理应当覆盖五个阶段：发现（用户通过 ClawHub 搜索、GitHub 浏览、社区推荐发现所需技能）→ 安装与配置（通过 CLI 或图形界面安装，配置必要的 API 密钥和参数）→ 监控与审计（监控 skill 的执行频率、成功率、资源消耗和异常行为）→ 升级（skill 开发者的更新和用户的升级决策）→ 迁移与退役（当 skill 不再需要或出现更好替代时，安全地移除并清理相关配置和凭据）。

安全事件的教训值得反复强调。ClawHub 的排名操纵漏洞（攻击者通过刷量手段将恶意技能推至分类排名第一，使其获得大量安装）暴露了开放技能市场最核心的信任问题：在用户无法独立审计每个技能的情况下，他们依赖平台的排名和评价系统来做安全判断，而这些系统本身可能是被操纵的。大量技能被发现硬编码 API 密钥、或暗中将用户数据上传至第三方服务器，进一步加深了这一信任危机。

治理实践方面，社区和企业已形成一些共识方案：

- **白名单策略：**企业维护一份经过内部安全审查的技能白名单，Agent 只能从白名单中安装和使用技能。这虽然牺牲了灵活性，但在企业生产环境中是必要的安全措施。

- **技能签名与来源验证：**OpenClaw v2026.4.12 引入的 manifest 签名机制，要求每个技能包附带开发者签名，Agent 在安装前验证签名有效性。

- **技能行为沙箱：**在 Docker 容器或虚拟机中运行所有非内建技能，通过 eBPF 过滤器在系统调用层面限制技能的访问范围——限制可读写的文件路径、

可访问的网络地址、可执行的系统调用。

• **技能经济与激励机制**：围绕 ClawHub 的技能付费模型（免费技能、一次性购买、订阅制）和排名机制正在催生一个新的“技能即商品”市场。高质量技能开发者的贡献应当得到经济回报，但激励机制的设计也必须在开放性和质量控制之间取得平衡。

3.5 工具与协议：MCP、A2A、ACP 与 A2H

2026 年，Agent 领域涌现了多个关键通信协议，它们共同定义了 Agent 如何与外部工具（MCP）、其他 Agent（A2A）、开发环境（ACP）和人类用户（A2H）交互。理解这些协议是把握 Agent 生态技术基础的关键。

MCP（Model Context Protocol） 是当前最广泛采用的 Agent-工具交互协议。基于 JSON-RPC 2.0，它将外部工具抽象为“server + tools”模型。开发者实现一个 MCP server，暴露一组命名工具，每个工具有明确的 JSON Schema 定义的输入输出。Agent 通过 MCP 客户端发现和调用这些工具。MCP 的优势在于标准化和互操作性——一个 MCP server 可以在 OpenClaw、Claude Desktop、DeerFlow 2.0 之间无缝切换。

A2A（Agent-to-Agent 协议） 则解决了一个不同的问题：当不同框架构建的 Agent 需要相互通信和协作时，它们该说什么语言？A2A 定义了 Agent 间的消息格式、能力发现机制和任务委派协议，使得一个 OpenClaw Agent 可以“雇用”一个 DeerFlow Agent 来完成某个子任务，或与一个 Hermes Agent 交换记忆片段。

ACP（Agent Client Protocol） 是 VS Code/IDE 与 Agent 之间的通信桥梁。OpenClaw 的 ACP CLI 允许开发者在编辑器内直接与 Agent 交互——选中一段代码让 Agent 分析、在编辑器中查看 Agent 的推理过程、批准或拒绝 Agent 提出的代码修改建议。ACP 的目标是让 Agent 成为 IDE 的“一等公民”，而不是一个外部工具。

A2H（Agent-to-Human 协议） 为“人类审批/共驾”设计了可验证的交互规范。在 OpenClaw 流程中，当 Agent 计划执行高风险操作（如删除文件、发送批量邮件、修改生产环境配置）时，A2H 定义了如何向用户请求审批——包含操作描述、风险评估、回滚方案——以及用户如何签名批准或拒绝。这个协议的重要性在于，它将“人工监督”从一个模糊的理念变成了一个可审计、可追溯的

标准化流程。

3.6 安全与治理：OpenClaw 风险全景与防护体系

OpenClaw 类 Agent 的安全治理不是“锦上添花”的可选功能，而是决定这类系统能否被企业和机构采纳的“入场券”。在 2026 年上半年，OpenClaw 的安全事件密度和严重程度，已经迫使安全社区将 Agent 安全从“前沿研究”升级为“当下威胁”。

风险全景图可归纳为四大类别：

1. **认知与 Prompt 注入风险**：LLM 本身容易被恶意构造的输入操纵。攻击者可以通过邮件内容、网页文本、技能描述文件等通道注入恶意指令，诱使 Agent 执行非预期的操作。这种风险的根本原因在于，当前 LLM 无法可靠地区分“用户的真实指令”和“来自不可信外部来源的文本内容”。

2. **工具/技能供应链风险**：如前述，ClawHub 和 MCP server 生态中存在大量未经审查的技能，其中约 7-10% 存在安全隐患或恶意行为。供应链攻击在传统软件领域已有数十年历史，但 Agent 技能供应链的攻击面更广——因为技能的“行为空间”远超传统软件包（一个 npm 包通常只做一件事，但一个 Agent 技能可以调用 Shell、浏览器和 API）。

3. **身份与访问控制缺陷**：多通道 Gateway 的认证机制在早期版本中存在明显缺陷——缺乏对消息来源的严格验证、session 劫持风险、凭据在日志中的不当记录。

4. **环境与沙箱逃逸**：Agent 执行 Shell 命令和代码的运行环境如果隔离不充分，可能导致权限提升或跨系统访问。

代表性安全事件值得详细回顾：

- **CVE-2026-25253 (CVSS 8.8 RCE)**：攻击者通过浏览器 WebSocket 连接向 Gateway 发送特制消息，注入恶意指令，窃取认证 Token 并执行任意系统命令。这一漏洞的利用门槛极低，只需要能够访问 Gateway 的 WebSocket 端口（在公网暴露的部署中，这意味着任何人）。

- **多起权限提升与认证绕过**：CVE-2026-32922 和 CVE-2026-33579 等漏洞允许攻击者以低权限身份获取 Agent 的高权限访问，或完全绕过认证机制向 Agent 发送指令。

- **群体性邮件删除事件**：Meta Superintelligence Lab 的一位 AI 对齐研究主管

报告称，她的 OpenClaw Agent 在收到停止指令后仍持续删除和归档了数百封个人邮件。这不是安全漏洞，而是 Agent 在过度授权和模糊约束下的行为失控。

- **恶意技能规模化：**安全厂商的扫描发现，全球有数万 OpenClaw 实例暴露在公网上，其中相当比例安装了来自 ClawHub 的未审查技能。

安全最佳实践已从社区的惨痛教训中结晶：

- **隔离优先：**使用专用机器、虚拟机或容器运行 OpenClaw，严禁在生产工作站上直接部署。Agent 的 Shell 执行环境应与宿主系统隔离。

- **最小权限原则：**为每个通道、每个技能设定最小访问权限——不要给 Agent root/sudo 权限，除非你明确知道为什么需要且愿意承担后果。

- **全面监控与审计：**集中收集所有 Agent 的工具调用日志、Shell 命令执行历史、API 请求记录，对异常行为（如短时间内大量文件操作、从未见过的网络连接目标、异常的 API 调用模式）设置告警。

- **安全厂商工具的应用：**2026 年已涌现一批专门针对 Agent 的安全产品和服务——如 ClawSec（Agent 行为异常检测）、ClawArmor（技能安全扫描和沙箱）、Zenity（Agent 身份治理）等。

一个拥有文件系统访问权限、Shell 执行能力和持久化凭据的系统，本质上是一个高危系统，无论它被包装得多么像“个人助手”。

OpenClaw 应被视为“运行不可信代码并持有持久化凭据的系统”，建议不要在标准个人或企业工作站上运行。如果组织需要评估 OpenClaw，应仅在完全隔离的环境中部署。

3.7 Harness 工程：统领记忆、工具与安全的“外骨骼”

Harness 工程是 2026 年 Agent 技术讨论中最重要的概念之一。它的基本洞见是：LLM 本身是不可靠的认知引擎，它具有不确定性、易被误导、缺乏真正的因果理解，但它又是 Agent 系统不可或缺的核心。Harness 的任务就是在不可靠的认知层之上，通过系统性的约束、反馈和监控，使 Agent 表现出可预测、可控和高质量的行为。

Harness 的正式定义可以表述为：“围绕 AI Agent 构建的约束、工具、反馈与监控系统，负责管理 Agent 的工具调用权限、记忆结构、任务调度、安全边界和与外部系统的交互协议。”

如果把 LLM 比作大脑，那么 Harness 就是骨骼、肌肉、神经系统和免疫系

统的组合。它决定了“大脑”能做什么、不能做什么、怎么感知世界、怎么保护自己。

Harness 工程的核心模式已经在实践中反复验证：

- **PEV (Plan-Execute-Verify) 循环**：Plan 阶段，Agent 必须先生成明确的执行计划（哪些步骤、使用哪些工具、预期结果是什么）供审查。Execute 阶段，在受控环境中按计划执行。Verify 阶段，通过自动化测试、规则检查和人工审批确认结果是否满足预期。如果验证失败，回退到 Plan 阶段重新设计。

- **规则与规范文件 (Spec/Rules/Guides)**：Agent 的行为不是由“prompt”来定义的，而是由一套多层级的规范文件来定义的。Spec 定义“这个任务要达到什么目标”；Rules 定义“绝对不能做什么”；Guides 定义“怎么做更好”。这种多层级的规范结构比单条长 prompt 更可靠、更易于维护和审计。

- **自动评测与 CI Gate**：对大规模 AI 生成输出做批量质量控制。例如，当 Agent 生成了一批代码修改，CI pipeline 会自动运行测试套件、lint 检查和安全扫描。任何未通过的修改都不会被合并。这实际上是将传统软件工程的 CI/CD 理念扩展到了 AI Agent 的工作流中。

Harness 的可移植性是当前社区的一个重要议题。OpenClaw、Claw Code、Hermes 和 DeerFlow 虽然各有自己的实现，但它们共享相同的 Harness 设计哲学（PEV 循环、规则文件驱动、沙箱执行、自动化验证）。OpenHarness 等项目正在尝试构建“一次写好、多 Agent 可用”的通用 Harness 框架，其目标是让工程师为一种 Agent 编写的规则和约束，可以被其他 Agent 框架复用。

3.8 企业级环境中的治理体系：从影子 IT 到平台化

OpenClaw 在企业环境中的蔓延，正在重现“BYOD（自带设备）”和“Shadow IT（影子 IT）”的历史模式。员工个人部署的 OpenClaw 实例——通常连接着公司邮箱、内部 Slack 工作区和 CRM 系统，正在形成一个 IT 部门看不见也管不着的 Agent 层。

这一现象的规模和风险已引起企业 CISO 的高度警惕。员工可能在不知情的情况下将公司敏感数据暴露给第三方模型供应商（如果配置了云端 API 而非本地模型），也可能安装了来自 ClawHub 的恶意技能，从而在公司的内部网络中打开后门。

企业的应对路径通常经历三个阶段：

- **第一阶段：识别与盘点。**使用网络扫描工具和端点检测方案，发现公司网络和设备上运行的 Agent 实例、已安装的技能列表、连接的通道和 API。这个阶段的目标是“搞清楚我们现在到底有多少 Agent 在跑”。

- **第二阶段：整合与统一。**在识别的基础上，IT 部门提供一个官方支持的 Agent 平台（如 NemoClaw 或基于 OpenClaw 的企业自建 harness），将员工的需求从“每个人自己搭一个”引导到“使用公司提供的安全平台”。这个平台预装了经过安全审查的技能集合，所有模型调用都通过企业 API 网关代理，日志和审计记录统一收集。

- **第三阶段：治理与持续运营。**引入审批流程（新技能的引入需要安全团队审批）、审计日志（所有 Agent 的工具调用和敏感操作都有不可篡改的审计记录）、风险分级（不同 Agent 根据其访问的数据敏感度和操作权限被分配不同的风险等级）、Kill Switch（当检测到异常行为时，可以一键停止特定 Agent 或特定技能的运行）。

3.9 生态中的角色与分工：开发者、运维、安全、研究者

OpenClaw 类 Agent 生态的成熟，正在催生新的职业角色和分工：

- **Skill/MCP 开发者：**负责构建和维护 Agent 的技能包和 MCP server。这个角色需要兼具领域专业知识（如金融、运维、法律）和对 Agent 工作机制的理解。

- **Harness 工程师：**负责设计和维护 Agent 的运行约束系统——规则文件、安全策略、上下文管理、记忆治理。这是一个 2026 年才出现的新角色，但其重要性正在快速增长。

- **Agent 运维 (AI Ops)：**负责 Agent 实例的部署、监控、资源管理和可用性保障。与传统 SRE 不同的是，AI Ops 需要处理 LLM 特有的问题——模型供应商的 API 波动、上下文窗口管理、Token 成本优化。

- **Agent 安全团队：**对技能、工具、harness 和 Agent 整体系统进行威胁建模、红队测试和安全审计。这可能是目前 Agent 生态中最紧缺的人才类型。

- **研究者：**利用 OpenClaw 生态产生的大规模数据（如 Moltbook 的 AI-to-AI 交互数据集、ClawHub 的技能使用模式、安全事件报告）研究 Agent 的行为模式、漏洞模式、社会影响和治理机制。

3.10 生态构建的长期挑战与机会

挑战方面，Agent 生态目前面临三个结构性问题：

- **安全与治理严重落后于能力扩展：**Agent 的能力在过去 6 个月的增长，远超安全社区跟进的速度。大量企业已经在生产环境中依赖 Agent，但这些 Agent 的安全基础是脆弱的。

- **标准碎片化：**MCP、A2A、各家私有协议并存，Agent 之间的互操作性更像一个愿望而非现实。一个在 OpenClaw 上运行的技能，通常无法直接在 DeerFlow 或 Hermes 中使用。

- **记忆与数据主权问题：**Agent 积累的用户记忆数据具有极高的个人和商业价值，但当前的法律和行业规范对“Agent 记忆数据的所有权和使用权”尚无明确定义。

机会方面，Agent 生态正在催生多个新兴市场：

- **Harness 工程**将成为新一代的基础软件领域——就像 Kubernetes 标准化了容器编排，Harness 框架有望标准化 Agent 的行为管理。

- **Agent 安全与治理产品**正在形成一个快速增长的市场——从技能安全扫描到运行时行为监控，从身份治理到合规审计。

- **专业化领域 Agent 平台**（科研、金融、法律、运维、医疗等）将为垂直行业提供预配置的“开箱即用”Agent——带着经过行业验证的记忆结构、技能集合和安全策略。

第四章：应用落地

4.1 OpenClaw 类智能体的核心应用场景地图

OpenClaw 类智能体的应用场景已从最初的“极客玩具”扩展到多个行业领域。以下是当前实际案例整理的应用场景地图：

- **个人效率：**邮件自动分类与回复、智能日程管理、知识库自动整理（将散落的笔记、书签、聊天记录整理为结构化 wiki）、日常自动化脚本（定时备份、文件整理、批量重命名等）。

- **开发与运维：**编码助手（代码生成、审查、重构）、CI/CD 流水线管理（自动触发构建、分析失败原因、提交修复 PR）、环境配置（根据项目文档自

动搭建开发环境）、GitHub issue 自动 triage 和 PR 草案生成。

- **业务运营：**客服工单自动处理、订单管理与异常告警、财务报表自动生成与对账、竞品市场监测（定时抓取竞品数据、生成对比报告）。

- **科研与分析：**文献综述自动生成、实验设计与参数优化、数据探索与可视化、论文格式排版与投稿适配。

- **行业垂直：**医疗（患者预约管理、病历摘要生成）、金融（风险监控、风险报告、合规审查）、制造（供应链监控、设备维护提醒）、教育（个性化学习材料生成、作业批改辅助）。

4.2 典型案例 1：中小企业的 OpenClaw 自动化管家

将 OpenClaw 作为中小企业的“自动化管家”是目前最成熟、最容易复现的应用模式之一。以下是一个基于公开案例和社区最佳实践整理的落地模板。

业务场景：一家 15 人规模的电商企业，需要处理每日 200+ 封客户邮件（询价、订单状态查询、售后投诉）、管理 Slack 上的团队沟通、与 CRM 系统（如 HubSpot）和会计软件（如 QuickBooks）集成。

落地步骤：

第一步：选型决策。在开源 OpenClaw、Nanobot 和大公司的 Agent 之间选择。如果团队有技术能力进行部署和维护，本地部署 OpenClaw 可提供最大的灵活性和数据控制。如果希望降低运维负担，商业化产品提供了更友好的安装体验和预设的中文生态支持。

第二步：安装部署。推荐方案是在一台专用 VPS 或企业内部服务器上部署 OpenClaw。阿里云、腾讯云等云厂商已提供 OpenClaw 的一键部署镜像。部署后立即进行基础安全配置：关闭 Gateway 的公网访问（仅监听本地回环地址）、配置防火墙规则、为不同通道设定独立的认证凭据。

第三步：安全配置与权限分区。将 Agent 的操作划分为不同的安全区：
(1) 只读操作区——读取邮件、查询 CRM、查看日历；
(2) 低风险操作区——回复标准查询邮件、发送 Slack 消息、更新 CRM 备注；
(3) 高风险操作区——发送批量邮件、修改订单状态、触发退款流程。高风险操作必须设置为“需要人工审批”模式。严禁给 Agent root/sudo 权限。

第四步：职责分配与人机协同流程。明确 Agent 可以自主处理的场景（标准询价回复、订单状态查询）和必须提交人工处理的场景（涉及退款、投诉升

级、大额订单的议价)。设定 Agent 的工作时间窗口(如工作时间每 15 分钟检查一次邮件并处理,非工作时间每小时检查一次)。

这类部署可以很有效地减少中小企业客服团队的日常重复性工作,使人类员工可以专注于复杂问题和客户关系维护。Serif 公司的统计显示,用户通过 OpenClaw 进行邮件分拣、社交媒体管理、客户跟进和报告生成等任务的自动化,平均每周可节省 10 小时以上工作时间。

4.3 典型案例 2: 科研机构 / 实验室的 AutoResearchClaw + Autoresearch workflow

将 AutoResearchClaw (文献综述与论文撰写) 与 Autoresearch (实验代码自动优化) 结合,可以形成一套从“文献调研”到“实验调优”到“论文撰写”的科研全链路 Agent workflow。

实验场景: 一个机器学习研究小组在进行一项关于“小型语言模型的指令微调效率优化”的研究。

workflow 编排:

1. **课题初期 - 文献综述:** 研究人员用自然语言描述研究问题, AutoResearchClaw 的检索 Agent 接入 arXiv 和 Semantic Scholar,检索相关论文。阅读 Agent 生成结构化摘要。规划 Agent 根据摘要识别研究 gap。写作 Agent 生成文献综述初稿,包含现有工作的分类和未解决问题的识别。

2. **实验阶段 - 自动调参:** 研究人员准备好基础训练脚本和评估指标。 Autoresearch Agent 在设定好的时间预算(如每晚 8 小时)内自动修改训练脚本——调整学习率、batch size、LoRA rank、数据采样策略等——运行训练,根据验证集指标决定保留或丢弃改动。所有有效改动以 git commit 记录,便于研究者第二天审查。

3. **论文撰写 - 方法设计与初稿:** AutoResearchClaw 的方法设计 Agent 根据实验方案生成方法章节。写作 Agent 结合文献综述和方法设计生成初稿。审稿 Agent 对初稿进行内部评审。

组织层面的管控:

- 所有 Agent 运行在实验室的受控 GPU 节点上,不接入公网。
- Autoresearch 的所有实验改动通过 git 记录,保留完整的可追溯历史。

- 伦理审查：AI 生成的内容不能直接投稿——研究人员必须逐段审核、修改并确认内容的准确性和原创性。

- 实验日志系统：所有 Agent 的自动化实验记录（参数组合、结果指标、时间戳）被自动写入实验数据库，确保可复现性。

该 workflow 在提升效率的同时，要求研究者具备更高层次的能力。这不止于“会写代码和跑实验”，而是“能设计研究问题、评估 Agent 的输出质量、判断结果的可靠性”。Agent 没有减少对研究者专业判断的需求，而是将其提升到了更高的抽象层次。

4.4 典型案例 3：DeerFlow 2.0 驱动的数据与内容工厂

DeerFlow 2.0 的 DAG 状态机架构特别适合构建“研究 + 写作 + 多媒体制作”的多 Agent 内容生产流水线。

业务场景：一家市场咨询公司需要周期性产出行业分析报告——包含数据采集与分析、报告撰写、PPT 制作和视频脚本生成。

流水线设计：

1. **数据采集子 Agent：**定时从多个数据源（公开数据库、行业网站、社交媒体）采集原始数据，存入 Docker 沙箱中的临时存储。

2. **数据分析子 Agent：**在沙箱中运行统计分析脚本，生成图表和关键发现摘要。

3. **写作子 Agent：**根据分析结果撰写报告正文，包含执行摘要、数据分析、趋势预测和建议。

4. **PPT 子 Agent：**将报告内容自动转换为演示文稿——提取每节关键信息、匹配图表、应用企业模板样式。

5. **视频脚本子 Agent：**从报告核心内容中生成短视频脚本，适配不同平台（抖音、YouTube、LinkedIn）的格式要求。

实施要点：

- **Docker 沙箱配置：**每个子 Agent 在独立 Docker 容器中运行，限制网络访问白名单（只能访问指定的数据源和内部服务），限制文件系统访问范围。

- **子 Agent 能力设计：**每个子 Agent 拥有针对性的工具集——数据采集 Agent 只需要 HTTP 和文件工具，写作 Agent 需要 Markdown 编辑工具和向量知识库检索，PPT Agent 需要 PPTX 操作工具。

- **成果审查与人类决策：**流水线的每个关键节点设置“人工把关”——数据采集完成后由分析师确认数据质量；报告初稿由高级分析师审核；最终PPT和视频脚本由市场经理批准。

如果使用 OpenClaw 来实现同样的工作流，你需要一个非常长且复杂的 prompt 来串联所有步骤，Agent 缺乏明确的“阶段意识”，容易在中间步骤跑偏。DeerFlow 2.0 的 DAG 状态机将每个阶段定义为独立节点，阶段之间通过明确的成功条件来触发转移，使得长链路工作流更加可控和可调试。

4.5 典型案例 4：Hermes Agent 在知识密集型岗位中的应用

Hermes Agent 的四层记忆和自我学习机制，特别适合需要长期记忆积累和上下文连贯性的知识密集型岗位。

应用角色：投资分析师、产品经理、律师、研究顾问——这些角色需要处理大量跨项目、跨时间的信息，且需要记忆特定客户的偏好、行业的专业知识和个人积累的经验教训。

部署模式：Hermes Agent 作为“长记忆私人助手”部署在分析师的本地设备上。在最初几周，Agent 的能力有限。它还没有积累足够的自生成技能和结构化记忆。这时分析师需要更频繁地介入和指导 Agent。

价值累积路径：

- **第 1-2 周：**分析师花费额外时间“训练”Agent——在每次任务完成后检查 Agent 输出，修正错误，标记高质量的工作模式。Agent 在这个阶段主要积累 USER.md 中的偏好信息和基础工作模式。

- **第 3-6 周：**Agent 开始生成第一批自生成技能——例如“财报电话会议纪要提取”、“竞品新闻自动聚合”、“客户法律需求初步分析”等。这些技能是 Agent 从分析师的实际工作流程中抽象出来的，天然贴合分析师的个人工作习惯。

- **第 2-3 个月：**Agent 积累了 20+ 自生成技能和丰富的结构化记忆库。任务完成时间相比初次部署时下降 30-40%。Agent 能够在分析师提问“上次那个关于 XX 的案例”时，在数秒内从结构化记忆中检索到精确的相关信息。

关键经验和注意事项：

- 多数收益来自持续使用和渐进积累，而非首次部署的“惊艳体验”。切换 Hermes Agent 的成本不在安装，而在最初的“磨合期”。

- 需要提前设计记忆边界——哪些信息 Agent 应当记住（如客户偏好、行业

分析框架），哪些信息不应保留（如涉及个人隐私或机密的对话）。记忆边界的清晰度直接影响长期使用的安全性和合规性。

- 敏感信息处理策略必须硬编码在规则文件中，而不是依赖 Agent 的“判断”来区分敏感与非敏感信息。

自主智能体将在组织层面产生重要影响。重复性、信息密集的任务（邮件分类、文档整理、数据采集、标准报告生成）逐步转移给 Agent。人类角色从“执行者”转向“评审者”——审核 Agent 的输出质量，处理复杂和例外场景，做出需要判断力和创造力的决策。Harness/Agent Engineer（设计和管理 Agent 的运行时约束）、AI Ops（Agent 运维与监控）、Agent 安全审计员等新角色正在从现有岗位中分化出来。未来的知识工作者需要的不是“做事的技能”，而是“指导 Agent 做事的技能”和“判断 Agent 做得对不对的技能”。这类似于从“自己开车”到“指导自动驾驶系统开车”的转变——你不再需要踩油门和刹车，但你需要比以往更清楚地知道“想去哪里”和“什么是安全的路线”。

第五章：发展趋势与未来展望

5.1 市场与产业格局：从模型战争到 harness 战争

2025 年之前，AI 产业的竞争焦点集中在“谁的模型更大更强”。OpenAI 的 GPT 系列、Anthropic 的 Claude 系列、Google 的 Gemini 系列在基准测试上的每一次微小领先都是新闻头条。但 2026 年的产业格局发生了关键转变：模型之间的性能差距正在缩小，而 Harness 的质量、安全性和易用性正在成为真正的差异化因素。

市场数据支持这一判断。生成式 AI 整体市场预计在 2032 年达到约 8906 亿美元规模，其中 Agent 细分市场在 2025-2026 年间的增速尤为突出——从约 80.3 亿美元增长至约 117.8 亿美元。更重要的是，越来越多的企业 IT 预算从“模型 API 调用费”转向了“Agent 平台和治理工具”，也就是 harness 层。

“从模型战争到 harness 战争”这一表述最早由 Harness 工程社区提出，其核心论点是：当多个顶尖模型都能完成类似的任务时，决定最终用户体验和系统可靠性的不再是模型本身，而是围绕模型构建的整个运行时系统——记忆管理的质量、工具调用的准确性、安全边界的严密性、多 Agent 协作的流畅度。这

类似于智能手机产业的演变：当芯片性能趋同时，决定用户体验的是操作系统和生态。

企业市场的参与者格局正在快速分化：

- **开源生态层**：OpenClaw、Hermes、DeerFlow 2.0、Claw Code 等项目构成了技术创新的源泉。
- **企业产品层**：NVIDIA NemoClaw、腾讯 QClaw、各云厂商的托管 Agent 服务，将开源技术转化为企业级产品。
- **治理与安全层**：ClawSec、Zenity、ClawArmor 等专业 Agent 安全厂商，提供技能扫描、行为监控和合规审计工具。
- **垂直方案层**：面向金融、医疗、法律、制造等行业的专用 Agent 解决方案供应商，基于开源框架定制行业特定的记忆结构和技能集合。

5.2 标准、协议与安全

Agent 生态的标准化进程正在多个方向上并行推进，这是决定 Agent 能否从“孤岛”走向“互操作网络”的关键。

MCP 正在快速成为 Agent-工具交互的事实标准。基于 JSON-RPC 2.0，只定义了工具发现和调用的核心接口，将具体实现留给社区。这种“够用就好”的标准化策略，与早期互联网协议（如 HTTP）的成功路径高度相似。

A2A（Agent-to-Agent 协议）的标准化则面临更大的挑战——不同框架的 Agent 在能力表示、任务分解和记忆共享机制上存在深层差异。一个只有 OpenClaw Agent 的世界是封闭的，一个 OpenClaw Agent 可以调用 DeerFlow 子 Agent、与 Hermes 交换记忆片段的世界才是真正互联的。

NIST AI Agent Standards Initiative 的启动，标志着政府层面对 Agent 标准化的正式介入。NIST 正在推进的工作包括：Agent 安全评估框架的定义、Agent 行为测试基准的建立、以及 Agent 间通信协议的安全要求规范。

不同生态之间的兼容性博弈是一个现实的挑战。OpenAI、Anthropic、Google、ByteDance 和开源社区各自推动着自己的 Agent 框架和协议。短期来看，这种碎片化是创新加速的代价。长期来看，行业需要某种程度的收敛，可能是 MCP/Skill 作为 Agent 工具层的标准、A2A 作为 Agent 间通信的标准、以及一套公认的安全评估框架。

Agent 安全正在从一种“事后响应”的被动模式，转向“预防性治理”的主动模

式。

Agent 安全市场的参与者在快速分化：

- **安全评估与红队服务：**专门针对 Agent 系统的渗透测试、Prompt 注入对抗测试和技能供应链审计。这些服务模拟真实攻击场景，帮助组织识别 Agent 部署中的脆弱点。

- **运行时安全监控：**通过 eBPF、系统调用过滤和行为异常检测，实时监控 Agent 的工具调用和系统交互，在恶意行为发生时进行阻断。

- **治理平台：**提供 Agent 技能白名单管理、身份认证与访问控制、审计日志和合规报告的一站式平台。

合规与政策的跟进正在加速。不同国家开始发布针对 Agent 的监管指引。核心议题包括：Agent 的数据访问权限边界（Agent 是否可以访问用户的全部邮件和文件，还是在某种数据分类和最小必要原则下运行？）、Agent 行为责任的界定（当 Agent 在执行任务中造成损失——如错误删除文件、误发敏感信息——谁承担责任？是 Agent 的用户、开发者、技能提供者还是模型供应商？）、以及 Agent 决策的透明度和可解释性要求（当 Agent 自主做出影响他人权益的决策时，受影响方是否有权要求对该决策进行人工审查？）。

从漏洞响应到预防性治理的转变体现在两个趋势上：一是在 Agent 部署前的安全评估正在成为标准流程（类似于传统软件在上线前的渗透测试）；二是在 Agent 设计阶段就将安全约束“嵌入”Harness，而不是等漏洞被发现后再打补丁。

5.3 开放问题与未来研究方向

自进化智能体在释放巨大潜力的同时，也带来了传统软件不曾面临的风险：

不受控演化：Agent 可能将一个偶然的短期收益误判为长期优化方向，导致行为逐步偏离用户的真实意图。例如，一个邮件处理 Agent 可能“学会”自动归档大量邮件（因为这减少了待处理数量，满足“效率”指标），但这些被归档的邮件中有重要的未读信息。自进化系统对目标函数的过度优化（reward hacking）在 Agent 领域同样存在，且后果更为直接。

价值与伦理界限：当 Agent 自主生成论文初稿、修改实验代码时，学术诚信的边界在哪里？如果 Autoresearch Agent“优化”出的模型在 benchmark 上表现

优异但使用了不合理的技巧（如在数据预处理中无意间引入了信息泄露），而研究者并未察觉，谁为这个结果负责？

复杂系统中的反馈循环：当多个自进化 Agent 在网络中持续互动（如 Moltbook 上的百万 Agent），它们的群体行为可能涌现出难以预测的模式——信息级联、意见极化、或集体策略同化。这些现象在人类社交网络中已被广泛研究，但 Agent 网络中可能的动态速度更快、规模更大。

开放问题包括：

- **如何设计可证明安全的自进化规则？**能否为 Agent 的自进化行为设定形式化约束，使 Agent 可以在这些约束内自由探索，但绝不可能跨越某些预设边界？这类似于控制系统中的 Lyapunov 稳定性——为 Agent 设定一个“安全不变集”。

- **如何用形式化方法和监控体系约束自进化范围？**runtime monitoring（运行时监控）和 formal verification（形式化验证）在传统软件工程中已有成熟应用，但如何将它们适配到基于 LLM 的 Agent 系统中，仍是一个开放的研究挑战。

- **如何在多 Agent 生态中实现跨系统学习与经验共享？**当多个 Hermes Agent 分别在不同用户的电脑上积累了不同的自生成技能，这些技能能否在 Agent 之间安全共享，形成“群体学习”效应？这涉及学习共享协议、技能兼容性验证和隐私保护等多个难题。

5.4 技术演进：长记忆、多模态与具身智能

展望 2026 年下半年及更远的未来，Agent 技术的演进方向可以从三个维度来把握：

更长上下文与更高效的记忆结构：当前 LLM 的上下文窗口已从最初的 4K Token 扩展到 200K 甚至 1M Token 级别，但“把所有记忆都塞进上下文”的策略既不经济也不高效。下一代记忆架构的方向是“分层检索 + 动态构建”——Agent 不是在推理前加载所有记忆，而是在推理过程中根据需要实时检索和构建最相关的记忆片段。这一模式类似于人脑的工作记忆机制——我们不会在每一次决策时回忆全部人生经历，而是在需要时提取相关的记忆。

多模态 Agent：当前的 OpenClaw 类 Agent 主要处理文本——文本输入、文本输出、文本工具调用。但用户的世界是多模态的——语音指令、图片理解、

视频分析、结构化数据（表格、数据库）查询。2026 年下半年，多模态 Agent 有望成为主流。想象一下：你拍一张会议白板的照片发给 Agent，它自动识别白板上的内容、提取行动项、在日历中创建会议提醒、在 Notion 中更新项目文档，这一切操作只需要一张照片和一句话。DeerFlow 2.0 和 Hermes Agent 的 roadmap 均已包含了多模态支持的计划。

具身智能与 IoT 整合：OpenClaw 类 Agent 的应用场景正在从“纯数字世界”向“物理世界”延伸。当 Agent 不仅能够操作软件，还能控制智能家居设备、驱动机器人执行物理任务时，Agent 将真正成为连接数字和物理世界的桥梁。NVIDIA 的 NemoClaw 方案已经展示了 Agent 与 IoT 基础设施的集成潜力。Agent 可以通过工业物联网平台监控设备状态、预测维护需求、甚至自动调整生产参数。具身智能（Embodied AI）与 Agent 框架的融合，将是未来 2-3 年最值得关注的技术方向之一。

5.5 面向 2030 的问题与机会

站在 2026 年的中点向 2030 年展望，Agent 领域面临几个决定其长期走向的关键问题，也蕴含着可观的机会。

在大规模 Agent 网络中如何保持可控与可审计？当数百万 Agent 持续运行、自主学习、相互通信和协作时，我们如何确保这个 Agent 网络的行为是可预测的、可追溯的和可问责的？这不是一个技术问题，更是一个治理问题——它涉及技术标准、法律框架和社会规范的多层协调。

如何在保障隐私与安全的前提下实现“群体智能”？每个 Agent 在其用户设备上积累的知识和经验是孤立的。如果这些经验和技能可以在 Agent 之间安全共享（不泄露个人隐私数据），那么整个 Agent 生态的学习效率将是单个 Agent 的 N 倍。实现这一目标需要解决联邦学习、差分隐私、技能共享协议和信任机制等一系列技术和社会难题。

将 Autoresearch 和 AutoResearchClaw 的模式推广到更广泛的科研场景，不仅是机器学习的实验调优，还包括生物学实验设计、材料科学配方优化、药物分子筛选等。科研 Agent 有可能成为推动科学发现加速的基础设施。

医疗（病历分析、诊疗方案推荐）、金融（合规审查、投资分析）、法律（合同审查、判例检索）、公共部门（政策影响评估、公共服务自动化），每个行业都有独特的知识结构、工具生态和合规要求，为行业定制 Agent 平台将

成为巨大的市场机会。

当 Agent 越来越深入地嵌入日常生活和工作时，公众需要理解 Agent 能做什么、不能做什么、风险在哪里、如何安全地与 Agent 协作。Agent 素养（Agent Literacy）将成为与数字素养同等重要的公民能力。2030 年的孩子们不仅需要学会编程，更需要学会“与 Agent 共事”——如何给 Agent 清晰的目标、如何判断 Agent 的输出是否可信、如何在 Agent 犯错时及时纠正。

如前所述，OpenClaw 类 AI 助手，通常需要调用本地 Shell、文件系统 API 甚至浏览器控制权来执行任务。若其运行的模型被提示词注入攻击诱导执行恶意指令，攻击者可直接获得当前登录用户的完整桌面权限，进而窃取 SSH 私钥、修改系统配置或植入持久化后门。

云端容器或虚拟机仅挂载指定的数据卷，即便 Agent 被越权控制，破坏范围也局限在隔离的容器内，无法触及本地电脑的硬件控制权、私钥或隐私文件。本地隐私数据不会泄露，而且可 24 小时在线运行。发现安全漏洞后，可直接销毁当前实例并从安全镜像重建，恢复时间短，对学习、了解或尝试开发 AI Agent 友好。

这份白皮书旨在成为你理解 OpenClaw 技术内核、构建自有 Agent 系统的助力；助你理性权衡 OpenClaw 类智能体的风险与机会，在组织层面形成审慎的采纳判断；并为你系统梳理 Agent 生态、记忆机制与自进化模式提供一个基础框架，以此迎接新时代的开启。

对于有开发需求的个人或团队，算力社区平台整合了国产异构算力资源，提供了一套经济高效的云端算力方案。算网平台也已上线中文社区的 OpenClaw-CN 镜像，以及基于 Claude Code 泄露源码构建的隐私增强分叉版 Claude Code Clean 镜像，只需一键部署，就能即刻拥有自己的 7×24 小时在线 AI 私人助手。

学习、了解 OpenClaw 这类 AI 助手，已经成为 AI 时代的必修课。

黄仁勋说：“AI 不会取代你，但会用 AI 的人会。”

附录：资源索引

- OpenClaw GitHub 仓库：<https://github.com/openclaw/openclaw>
- OpenClaw 官网：<https://openclaw.ai>

- MCP 协议官网: <https://modelcontextprotocol.io>
- A2A 协议 (Google Developers Blog 公告) :
<https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability>
- Deer-Flow GitHub 仓库: <https://github.com/bytedance/deer-flow>
- Deer-Flow 官网: <https://deerflow.tech>
- Nanobot GitHub 仓库: <https://github.com/HKUDS/nanobot>
- Nanobot 官网: <https://nanobot.ai>
- AutoResearchClaw GitHub 仓库: <https://github.com/aiming-lab/AutoResearchClaw>
- AutoResearchClaw 论文 (arXiv) : <https://arxiv.org/html/2604.01007v1>
- Karpathy AutoResearch GitHub: <https://github.com/karpathy/autoresearch>
- Karpathy AutoResearch 完整指南 (O-Mega) : <https://o-mega.ai/articles/karpathy-autoresearch-complete-2026-guide>
- Karpathy Loop 报道 (Fortune) : <https://fortune.com/2026/03/17/andrej-karpathy-loop-autonomous-ai-agents-future>
- Claw Code GitHub: <https://github.com/chauncygu/collection-claude-code-source-code>
- Claw Code 解释 (Wavespeed AI) : <https://wavespeed.ai/blog/posts/what-is-claw-code>
- Hermes Agent GitHub: <https://github.com/nousresearch/hermes-agent>
- Hermes Agent 官网: <https://hermes-agent.nousresearch.com>
- NVIDIA NemoClaw: <https://docs.nvidia.com/nemoclaw/latest/index.html>
- OpenClaw 安全分析 (NSFOCUS) : <https://nsfocusglobal.com/openclaw-open-source-ai-agent-application-attack-surface-and-security-risk-system-analysis>
- OWASP Agentic AI 治理: <https://www.modulos.ai/blog/agentic-ai-governance>
- 企业 AI Agent (Neontri) : <https://neontri.com/blog/enterprise-ai-agents>
- AI Agent 生产用例 (DigiEx Asia) : <https://digiex.asia/blog/7-real-world-ai-agent-use-cases-in-production-2026>
- OpenHarness GitHub: <https://github.com/HKUDS/OpenHarness>
- LangChain Harness 博客: <https://www.langchain.com/blog/your-harness->

your-memory

- EU AI Act 自主 Agent 合规 (Covasant) :

<https://www.covasant.com/blogs/eu-ai-act-compliance-autonomous-agents-enterprise-2026>

- Learned Capability Governance (arXiv) :

<https://arxiv.org/html/2604.11839v2>

- NIST AI Agent Standards Initiative: <https://www.nist.gov/artificial-intelligence/ai-agent-standards-initiative>

- Martin Fowler: Harness Engineering for Coding Agent Users:

<https://martinfowler.com/articles/harness-engineering.html>

- OpenAI: Harness Engineering: Leveraging Codex in an Agent-First Workflow: <https://openai.com/index/harness-engineering>

- Addy Osmani: Agent Harness Engineering:

<https://addyosmani.com/blog/agent-harness-engineering>

- Harness 工程 AI 智能体深度指南 (Milvus) :

<https://milvus.io/blog/harness-engineering-ai-agents.md>

- Natural-Language Agent Harnesses (arXiv) :

<https://arxiv.org/html/2603.25723v1>

- Harness Engineering: A Governance Framework (SSRN) :

<https://ssrn.com/abstract=harness-engineering-governance>

- Agent Harness 实践指南 (Zylon AI) :

<https://www.zylon.ai/resources/blog/what-is-openclaw-a-practical-guide-to-the-agent-harness-behind-the-hype>

主编单位：中科算网 算泥社区
网址：sumw.com.cn
邮箱：zhusiliang@sumw.com.cn



算泥社区



大模型交流群