

深入探索 Claude Code：当今与未来 AI 智能体系统的设计空间

Jiacheng Liu¹, Xiaohan Zhao¹, Xinyi Shang^{1,2}, Zhiqiang Shen^{1,†}

¹VILA Lab, Mohamed bin Zayed University of Artificial Intelligence, ²University College London

[†]Corresponding author

Claude Code 是一个智能体编程工具，能够代表用户执行 shell 命令、编辑文件以及调用外部服务。本研究通过分析公开可获取的 TypeScript 源代码^a，详细描述了其全面的架构，并进一步将其与 OpenClaw 进行比较——OpenClaw 是一个独立的开源人工智能智能体系统，从不同的部署场景出发，回答了许多相同的设计问题。我们的分析识别出五个推动该架构设计的人类价值观、哲学理念和需求（人类决策权威、安全与保障、可靠执行、能力增强以及情境适应性），并追踪这些理念如何贯穿于十三项设计原则，最终落实为具体的实现选择。系统的核心是一个简单的 while 循环，它调用模型、运行工具，并不断重复。然而，大部分代码都存在于围绕这一循环构建的系统中：包含七种模式的权限系统与基于机器学习的分类器、用于上下文管理的五层压缩流水线、四种可扩展机制（MCP、插件、技能和钩子）、子智能体委派与编排机制，以及面向追加的会话存储。与 OpenClaw（一个多通道个人助理网关）的对比表明，当部署环境发生变化时，同样的反复出现的设计问题会产生不同的架构解答：从逐动作的安全评估转向边界级访问控制，从单一 CLI 循环演变为网关控制平面内的嵌入式运行时，从上下文窗口扩展转向网关范围内的能力注册。最后，我们基于近期的经验研究、架构文献与政策研究，提出了未来智能体系统发展的六个开放性设计方向。我们的 GitHub 项目地址为：<https://github.com/VILA-Lab/Dive-into-Claude-Code>。

Correspondence: Zhiqiang Shen (Zhiqiang.Shen@mbzuai.ac.ae)

^av2.1.88, [link](#). **免责声明：**本文所使用的所有材料均来自公开的在线资源。我们未使用任何私有、机密或未经授权的材料，且无意侵犯任何版权或知识产权。源代码的原始知识产权归属于 Anthropic。

1 引言

AI 辅助的软件开发已从类似 GitHub Copilot (Chen et al., 2021) 的自动补全工具，发展到集成在 IDE 中的助手如 Cursor (Cursor, 2026)，再到能够自主规划多步骤修改、执行 Shell 命令、读写文件并迭代自身输出的完全智能体系统。Anthropic (Anthropic, 2026c) 发布的 Claude Code (Anthropic, 2026a) 是一款智能体编码工具。其官方文档描述了一个“智能体环”，该环可规划并执行动作以达成目标，并能调用工具、评估结果，持续进行直至任务完成¹。从建议到自主行动的转变引入了完成型工具中不存在的架构需求。这些需求定义了一个设计空间，即一系列贯穿安全、上下文管理、可扩展性及委托等主题的反复出现的问题，每个编码智能体都必须应对。本研究通过分析 Claude Code 的源代码层级，展示了

¹<https://code.claude.com/docs/en/how-claude-code-works>.

某一生产系统如何回答这些问题。

尽管采用率不断上升，Anthropic 仍仅发布 Claude Code 的面向用户文档，而未提供详细的架构描述。本研究通过源代码分析来描述其架构设计决策。Anthropic 对 132 名工程师和研究人员的内部调查显示 (Huang et al., 2025)，约有 27% 的 Claude Code 辅助任务在没有该工具的情况下根本不会被尝试，这表明该架构实现了质的新型工作流，而不仅仅是加速现有流程。

在本工作中，我们首先识别出五种人类价值观/哲学以及十三项设计原则，这些原则驱动了架构 (Section 2) 的形成，然后将分析分为三个部分：

1. **设计空间分析**。我们识别出反复出现的设计问题（推理所在位置、迭代环的结构如何设计、应采取何种安全姿态、扩展表面如何划分、上下文如何管理、工作如何在子智能体间委派，以及会话如何持久化），并通过一个七组件高层结构和五层子系统架构来分析 Claude Code 的答案，将每个决策追溯至具体的源文件 (Section 3)。该分析旨在深入理解系统机制，以指导设计出更优、更强大的智能体系统。
2. **与开源系统 OpenClaw 的架构对比**。除了分析 Claude Code 本身外，我们还从六个设计维度出发，将其设计理念与开源智能体系统 OpenClaw (Steinberger and OpenClaw Contributors, 2026)（一个多通道个人助理网关）进行对比，以展示在不同部署情景下，相同的核心问题会得出不同的解答 (Section 10)，从而突出商业软件与开源软件之间的共通原则与关键差异。这一对比有助于揭示部署设置、产品目标、安全要求以及用户假设如何以不同方式塑造架构选择。通过考察这些系统在何处趋同、何处发散，我们的研究旨在为未来更强大智能体系统的构建提供有益指导与实用洞见。
3. **未来智能体系统的研究方向**。基于设计空间分析和 OpenClaw 对比，Section 12 识别出六个开放性研究方向，涵盖可观测性-评估差距、跨会话持续性、边界演化、时域缩放、治理以及评估视角，这些方向均借鉴了实证、架构和策略文献。作为评估视角，本研究还揭示了一个开放性问题：尽管 Claude Code 智能体系统显著增强了程序员和终端用户的短期能力，但其提供的机制在支持长期人类进步、深层理解以及代码库持续一致性方面仍十分有限。

核心智能体环是一个带有状态管理的无限循环。围绕其安全、可扩展性、上下文管理、委托和持久化等子系统构成了实现的主体部分。源码级分析²使我们能够直接从系统本身识别出设计选择、子系统边界和实现权衡，而不是仅从产品描述中推断。

运行示例。为了保持架构的清晰性，我们通过 Sections 3 to 9 追踪任务“修复 `auth.test.ts` 中失败的测试”。此示例说明了一个看似简单的用户请求如何激活多个架构层，包括工具调用、权限检查、上下文选择、迭代修复、委托以及会话持久化。

论文结构。Section 2 确定了驱动架构的人类价值观和设计原则。Section 3 介绍了高层架构及其所解答的设计问题。Sections 4 to 9 分析了各个主要子系统的设计选择。Section 10 将分析与 OpenClaw 进行对比，Section 11 提供讨论，Section 12 概述了未来智能体系统中的开放问题。Sections 13 and 14 随后涵盖相关工作和结论。Section B 描述了证据基础和方法论。

²我们的分析主要基于源代码，辅以官方 Anthropic 文档和选定的社区分析，Section B 详细说明了证据基础和方法论。

2 设计哲学、设计原则与架构动机

生产编码智能体由人类构建，为人所用，其嵌入的架构决策反映了创造者认为重要的价值。本节识别出驱动 Claude Code 设计的人类价值观，追溯这些价值观在反复出现的设计原则中的体现，并框定组织 Sections 3 to 9 中分析的设计空间问题。

Anthropic 的安全智能体框架指出一个核心矛盾：「智能体必须能够自主工作；正是这种独立运作能力使其具有价值。但人类应保留对其目标实现方式的控制权」(Anthropic, 2025a)。Claude 的宪法并未通过僵化的决策程序来解决这一矛盾，而是致力于培养「能够在具体情境中应用的良好判断力和健全价值观」(Anthropic, 2026b)。这些承诺，连同关于开发者实际使用该工具的经验发现 (Huang et al., 2025; McCain et al., 2026)，指向了塑造架构的五项人类价值观。

2.1 五大价值观与理念

人类决策权 人类对系统的行为保有最终决策权，这一权力通过一个主从层级 (Anthropic, 然后是操作员，最后是用用户) 进行组织，明确界定谁对什么拥有控制权 (Anthropic, 2026b)。系统设计使得人类能够实施知情控制：他们可以实时观察动作、批准或拒绝所提议的操作、中断正在进行的兼容操作，并在事后进行审计。当 Anthropic 发现用户对权限提示的批准率高达 93% (Hughes, 2026) 时，其回应并非增加更多警告，而是重构问题：定义清晰的边界（如沙箱环境、自动模式分类器），使智能体可在其中自由运作，而非依赖逐次动作的审批——因为一旦用户习惯，就会停止审查 (Dworken and Weller-Davies, 2025)。

安全、安保与隐私。 该系统保护人类、其代码、其数据及其基础设施免受伤害，即使人类注意力不集中或出现错误也是如此。这与人类决策权不同：决策权关乎人类的选择能力，而安全则关乎系统在该能力丧失时仍具有保护义务。Anthropic 的安全智能体框架将确保智能体交互的安全性以及在长期交互中保护隐私作为核心承诺 (Anthropic, 2025a)。自动模式威胁模型 (Hughes, 2026) 明确针对四类风险：过度积极的行为、诚实的错误、提示注入以及模型错位。

可靠的执行。 智能体执行人类真正意图的任务，保持时间上的连贯性，并在宣布成功前支持验证其工作。这一价值同时涵盖单轮正确性（是否准确理解了请求？）和长时程可靠性（是否在上下文窗口边界、会话恢复以及多智能体委托过程中保持连贯性？）。Anthropic 的产品文档 (Anthropic, 2026d) 描述了智能体重复的三阶段环：收集上下文、采取动作、验证结果。智能体设计指南 (Schluntz and Zhang, 2024) 进一步强调，“环境中的真实值”在每一步都用于评估进展。测试框架设计指南 (Rajasekaran, 2026) 同样指出，“智能体倾向于自信地夸赞工作成果”，即使质量平庸，这促使生成与评估的分离。

能力增强。 该系统显著提升了单位努力和成本下人类所能完成的任务量。Anthropic 内部调查 (Huang et al., 2025)，如 Section 1 所述，表明该架构实现了质的全新 workflows，而不仅仅是加快了现有流程：约 27% 的任务是原本不会被尝试的工作。其创造者将该系统描述为“一个类 Unix 工具，而非传统产品”，由最小的构建块组成，这些块具有“实用性、可理解性和可扩展性” (Cherny and Wu, 2025)。该架构注重确定性基础设施（上下文管理、工具路由、恢复机制），而非决策支撑结构（显式的规划器或状态图），前提是日益强大的模型从丰富的操作环境获益，远胜于受限于约束其选择的框架。

Table 1 设计原则、它们所服务的价值，以及设计空间问题各自所回答的内容。原则对应多个价值；具体实现见相应章节。

Principle	Values Served	Design Question	Sections
Deny-first with human escalation	Authority, Safety	Should unrecognized actions be allowed, blocked, or escalated to the human?	5, 8, 9
Graduated trust spectrum	Authority, Adaptability	Fixed permission level, or a spectrum users traverse over time?	5
Defense in depth with layered mechanisms	Safety, Authority, Reliability	Single safety boundary, or multiple overlapping ones using different techniques?	3, 5
Externalized programmable policy	Safety, Authority, Adaptability	Hardcoded policy, or externalized configs with lifecycle hooks?	5, 6
Context as scarce resource with progressive management	Reliability, Capability	What is the binding resource constraint, and how to manage it: single-pass truncation or graduated pipeline?	4, 6, 7, 8
Append-only durable state	Reliability, Authority	Mutable state, checkpoint snapshots, or append-only logs?	4, 9
Minimal scaffolding, maximal operational harness	Capability, Reliability	Invest in scaffolding-side reasoning, or operational infrastructure that lets the model reason freely?	3, 4
Values over rules	Capability, Authority	Rigid decision procedures, or contextual judgment backed by deterministic guardrails?	3, 5, 7
Composable multi-mechanism extensibility	Capability, Adaptability	One unified extension API, or layered mechanisms at different context costs?	6
Reversibility-weighted risk assessment	Capability, Safety	Same oversight for all actions, or lighter for reversible and read-only ones?	4, 5, 8
Transparent file-based configuration and memory	Adaptability, Authority	Opaque database, embedding-based retrieval, or user-visible version-controllable files?	7
Isolated subagent boundaries	Reliability, Safety, Capability	Subagents share the parent’s context and permissions, or operate in isolation?	8
Graceful recovery and resilience	Reliability, Capability	Fail hard on errors, or silently recover and reserve human attention for unrecoverable situations?	4, 5

上下文适应性 该系统适配用户的特定上下文（其项目、工具、惯例和技能水平），且这种关系会随时间推移而改善。扩展架构（CLAUDE.md、技能、MCP、钩子、插件）在多种上下文成本层级上提供了可配置性（Sections 6 and 7）。纵向数据 (McCain et al., 2026) 表明，人类与智能体的关系会演变：自动批准率从会话数少于 50 时的约 20%，增长到 750 个会话时的 40% 以上。这种被描述为“由模型、用户和产品共同构建的自主性”的模式，意味着该系统是为信任轨迹而非固定信任状态而设计的。MCP 向 Linux 基金会的 Agentic AI 基金会 (The Linux Foundation, 2025) 捐赠，体现了这一价值的生态系统维度。

2.2 设计原则

这些数值通过十三项设计原则得以具体化，每一项原则都回应了生产编码智能体必须解决的一个常见问题。Table 1 概述了这些原则；后续章节 (Section 3–Section 9) 则追踪了每项原则在具体实现选择中的体现。

这些原则可以与三种主要的替代设计族进行对照。首先, 基于规则的编排: 诸如 LangGraph (LangChain, Inc., 2024) 这类框架将决策逻辑编码为带有类型边的显式状态图, 选择使用脚手架而非最小化约束。其次, 容器隔离执行: SWE-Agent 和 OpenHands (Yang et al., 2024; Wang et al., 2024b) 依赖 Docker 隔离, 而非分层策略强制。第三, 以版本控制作为安全机制: Aider (Gauthier, 2024) 等工具将 Git 回滚作为主要安全机制, 而非采用拒绝优先的评估方式。Claude Code 的原则集在结合最小化决策脚手架与分层策略强制、基于价值判断与拒绝优先默认、渐进式上下文管理与可组合扩展性方面具有独特性。

2.3 从价值观到架构

每个数值都通过其原理追溯到具体的架构决策:

- **人类决策权**推动了拒绝优先评估、渐进式信任谱系、不可变状态 (可审计的历史记录)、外部化可编程策略以及价值优先于规则 (Sections 5 to 7 and 9)。
- **安全、安全与隐私**驱动纵深防御、默认拒绝优先、可逆性加权评估、外部化策略以及隔离子智能体边界 (Sections 5 and 8)。
- **可靠执行**推动了将上下文视为稀缺资源、追加仅有的持久状态、优雅恢复、隔离的子智能体边界以及纵深防御 (Sections 4 and 7 to 9)。
- **能力增强**推动极小值脚手架、可组合的可扩展性、权重化的风险、上下文管理以及优雅恢复 (Sections 4 to 6)。
- **情境适应性**激励透明的基于文件的内存、可组合的可扩展性、渐进的信任谱系以及外部化的可编程策略 (Sections 5 to 7)。

这些映射还揭示了架构所不执行的功能: 它不会在模型推理中强制实施显式的规划图, 不会提供统一的扩展机制, 也不会恢复时还原所有会话范围内的信任相关状态。这些缺失之处与上述原则是一致的。

2.4 评估视角: 长期能力保持

上述五个值描述了架构所旨在服务的目标。本文还引入了第六个关注点, 即架构是否能够保持长期的人类能力, 作为评估的视角。这一关注点是真实存在的: Anthropic 自身对 132 名工程师和研究人员 (Huang et al., 2025) 的研究记录了一个“监管悖论”, 即过度依赖人工智能会带来监管所需技能的退化; 而独立研究 (Shen and Tamkin, 2026) 发现, 在人工智能辅助条件下, 开发者的理解测试得分低了 17%。然而, 这一关注点并未在架构设计或 Anthropic 明确的设计价值观中得到显著体现。因此, 我们不将其视为与其它五项价值并列的同等重要原则, 而是作为一个贯穿性的关注点: 应用于 Section 11 中所有五个价值的考量, 以询问短期能力放大是否以牺牲长期的人类理解力、代码库一致性以及开发者流水线为代价。

3 架构概述

构建生产级编码智能体需要回答一系列反复出现的设计问题: 推理应置于何处, 需要多少个执行引擎, 应采取何种安全策略, 以及将何种资源视为关键约束。Claude Code 的架构可被视为对这些问题的一组解答。在实现层面, 该系统由七个组件构成, 通过主数据流相互连接: 用户通过多种接口之一提交提示, 该提示进入共享的智能体环。智能体环负责组装上下文, 调用 Claude 模型, 接收可能包含工具使用请

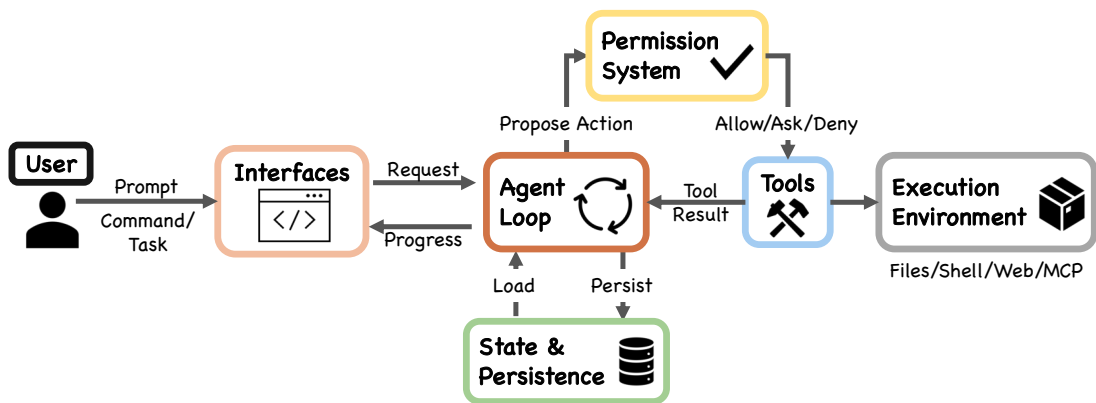


Figure 1 Claude Code 的高层系统结构。该系统分解为七个功能组件：用户、接口、智能体环、权限系统、工具、状态与持久化，以及执行环境。所有入口表面均汇聚至同一个智能体环。

求的响应，将这些请求通过权限系统进行路由，并将已批准的动作分发给具体的工具以与执行环境交互。在整个过程中，状态与持久化机制记录对话全文，管理会话身份，并支持恢复、分支和回溯操作。

3.1 设计问题与运行示例

描述围绕着在生产编码智能体中反复出现的四个设计问题展开，每个问题都支撑着Table 1中识别出的一个或多个设计原则。每个问题在此处首先以 Claude Code 的回答引入，附有合理替代方案的说明，然后通过Sections 4 to 9逐步演示。

推理存在于何处？在 Claude Code 中，模型负责推理所应执行的动作；而执行动作的责任则由框架承担。模型在其响应中发出 `tool_use` 块，框架会解析这些块，检查权限，将它们分发至工具实现，并收集结果 (`query.ts`)。模型从不直接访问文件系统、运行 shell 命令或发起网络请求。这种分离带来了安全上的优势：由于推理与执行分别位于不同的代码路径中，即使模型被攻破或遭对抗性操纵，也无法绕过框架中实现的沙箱机制、权限检查或默认拒绝规则。模型与外部世界唯一的接口是结构化的 `tool_use` 协议，该协议在执行前由框架进行验证。社区对提取出的源码分析估计，仅约 1.6% 的 Claude Code 代码库构成人工智能决策逻辑，其余 98.4% 为操作基础设施，这一比例凸显了核心智能体推理层的薄度。相比之下，其他设计方案则更侧重于支撑侧的推理：Devin 维护显式的规划和任务追踪结构，而 LangGraph (LangChain, Inc., 2024) 则通过开发者定义的状态图来路由控制流。

有多少个执行引擎？Claude Code 使用单一的 `queryLoop()` 函数，无论用户是通过交互式终端、无头 CLI 调用、智能体 SDK 还是 IDE 集成 (`query.ts`) 进行交互，该函数都会执行。只有渲染和用户交互层有所不同。其他系统则使用特定模式的引擎：例如，IDE 集成可能遵循与 CLI 工具不同的代码路径，以表面特定的最优化换取统一性。

默认的安全姿势是什么？Claude Code 的默认安全策略是“拒绝优先，人工升级”：拒绝规则优先于询问规则，询问规则优先于允许规则，未识别的动作将被升级至用户处理，而非静默允许 (`permissions.ts`)。多个独立的安全层（权限规则、PreToolUse 钩子、启用时的自动模式分类器，以及可选的 shell 沙箱）并行应用，因此任一层面均可阻止某个动作 (Section 5)。这结合了拒绝优先，人工升级与纵深防御，分层机制原则 (Table 1)。其他方法则将信任边界移至别处：SWE-Agent 和 OpenHands (Yang et al.,

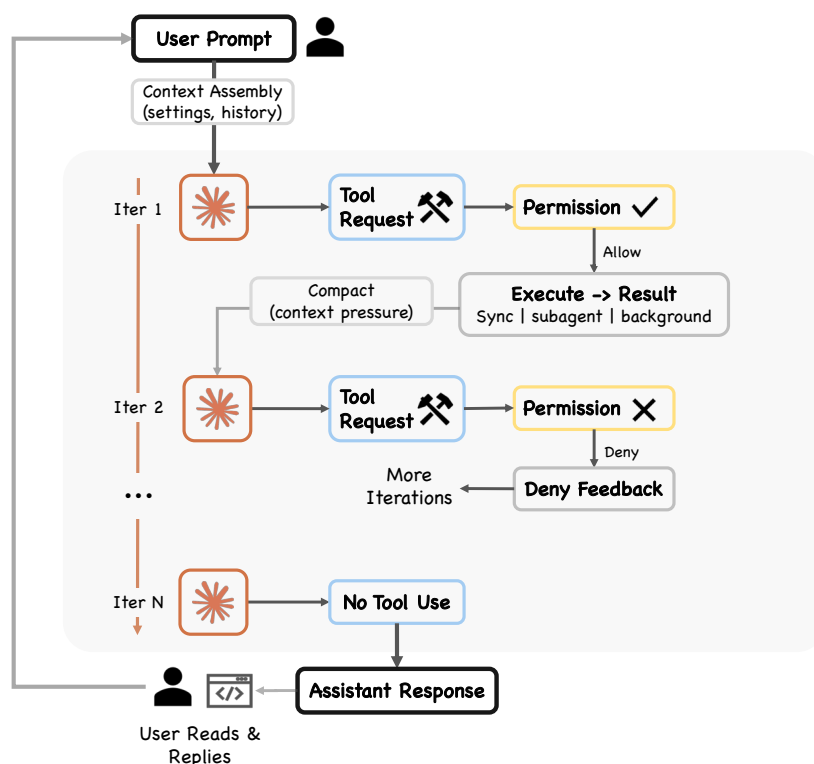


Figure 2 运行时流程展示了单个 Agentic 回合的端到端执行过程：用户提示通过上下文组装进入，调用模型，工具请求经过权限网关，工具结果反馈回路，压缩机制管理上下文压力。

2024; Wang et al., 2024b) 依赖基于容器的隔离来限制任意执行，而 Aider (Gauthier, 2024) 则以基于 git 的回滚作为其主要安全防护。

绑定资源约束是什么？在 Claude Code 中，上下文窗口（旧模型为 200K，Claude 4.6 系列为 1M）是主要的资源约束。在每次模型调用前，会执行五种不同的上下文缩减策略（`query.ts`），此外还有若干其他子系统决策（如指令的延迟加载、工具模式的推迟解析、仅返回摘要的子智能体）以限制上下文消耗（Section 7）。五层流水线的存在是因为单一的压缩策略无法应对所有类型的上下文压力。预算缩减针对超出大小限制的单个工具输出进行处理。剪裁处理时间深度问题。微压缩应对缓存开销。上下文坍塌管理极长的历史记录。自动压缩作为最后手段执行语义压缩。每一层都在不同成本-收益权衡下运行，较早且成本较低的层级会在更昂贵的层级之前执行。其他架构则将其他资源视为主要瓶颈，例如计算预算（限制模型调用或工具调用次数）或工作记忆（维护显式的草稿板，而非依赖对话历史）。

运行示例。为了落实这些原则，我们通过 Sections 3 to 9：“修复 `auth.test.ts` 中失败的测试。”这一单一任务进行贯穿。在本节中，用户通过 Claude Code 的其中一个界面提交提示。后续各节将追踪请求在查询环、权限网关、工具池、上下文窗口、子智能体委派以及会话持久化中的处理过程。

3.2 高级系统结构

七组件模型 (Figure 1) 直接映射到源文件：

1. **用户**：提交提示，批准权限，审查输出。

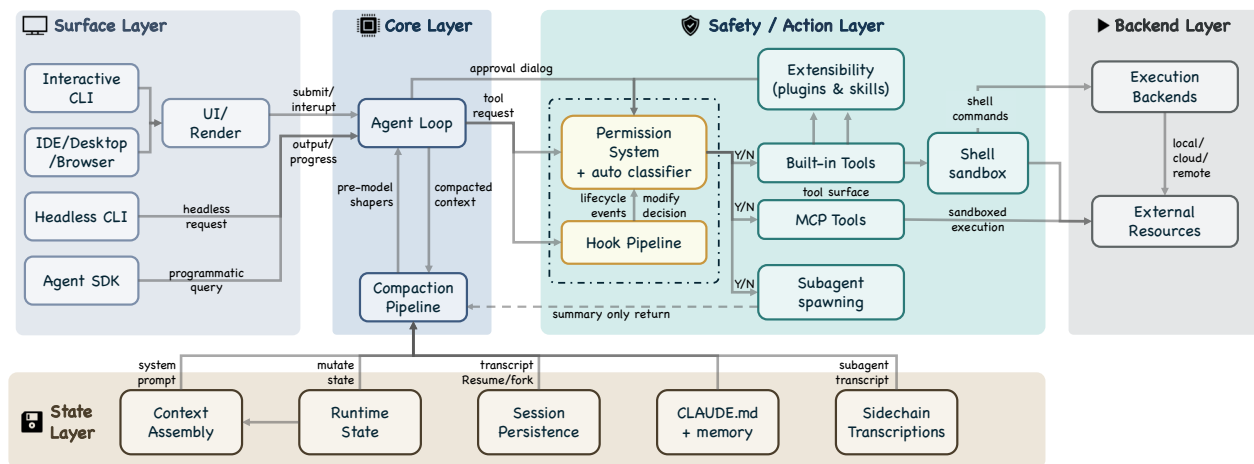


Figure 3 扩展的分层架构，展示了五个子系统层：表面层（交互式 CLI、无头 CLI、智能体 SDK、IDE/桌面/浏览器、UI/渲染器）、核心层（智能体环、压缩流水线）、安全/动作层（权限系统，包括自动模式分类器、钩子流水线、可扩展性、内置工具、MCP 工具、外壳沙箱、子智能体生成）、状态层（上下文组装、运行时状态、会话持久化、CLAUDE.md + 记忆、侧链对话记录），以及后端层（执行后端、外部资源）。

- 接口：**交互式命令行界面、无头命令行界面 (`claude -p`)、智能体 SDK 以及 IDE/桌面/浏览器。所有界面均接入同一环。
- 智能体环：**模型调用、工具分派和结果收集的迭代循环，作为 `query.ts` 中的 `queryLoop()` 异步生成器实现。
- 权限系统：**拒绝优先规则评估 (`permissions.ts`)、自动模式机器学习分类器，以及基于钩子的拦截 (`types/hooks.ts`)。
- 工具：**通过 `assembleToolPool()` (`tools.ts`) 汇聚的最多 54 个内置工具（19 个无条件，35 个基于特征标志和用户类型有条件），并与 MCP 提供的工具合并。插件通过 MCP 服务器和技能/命令注册表间接贡献。
- 状态 & 持久化：**主要是追加仅有的 JSONL 会话转录文件 (`sessionStorage.ts`)、全局提示历史记录 (`history.ts`) 以及子智能体侧链文件。
- 执行环境：**带有可选沙箱机制的 Shell 执行 (`shouldUseSandbox.ts`)，文件系统操作，网络获取，MCP 服务器连接以及远程执行。

数据流遵循从左到右的主干：用户通过界面提交请求，该请求进入智能体环。环向权限系统提出动作；经批准的动作到达工具，工具与执行环境交互，并将 `tool_result` 消息返回给环。状态与持久化数据与环并列，用于记录对话记录并加载先前会话的数据。

程序入口点 `main()` 在 `main.tsx` 中初始化安全设置（包括 `NoDefaultCurrentDirectoryInExePath` 以防止 Windows PATH 劫持），注册信号处理器以实现优雅关闭，并分派到适当的执行模式。

3.3 分层子系统分解

五层分解 (Figure 3) 将七组件模型扩展为更细粒度的视图，将每一层映射到特定的源目录。

表面层（入口点和渲染）`src/entrypoints/` 目录包含启动路径，包括 SDK 入口的 `coreTypes.ts`、`controlSchemas.ts` 和 `coreSchemas.ts`。`src/screens/` 目录用于组合全屏布局，`src/components/`

目录通过 ink 框架提供终端 UI 块。交互式 CLI 启动一个带有实时流、权限对话框和进度指示器的终端 UI。无头 CLI (`claude -p`) 创建一个 `QueryEngine` 实例以进行单次处理。智能体 SDK 通过异步生成器发出类型化事件。

核心层 (智能体环, 压缩流水线)。 `queryLoop()` 异步生成器 (`query.ts`) 实现了迭代智能体环, 从状态层消耗组装好的上下文, 并向安全/动作层分派工具请求。在每次模型调用之前, 一个由五个顺序阶段组成的 压缩流水线 (`query.ts:365--453`) 会管理上下文压力: 预算缩减、剪裁、微压缩、上下文坍塌和自动压缩 ([Sections 4.3 and 7.3](#))。

安全/动作层 (权限系统、钩子、可扩展性、工具、沙盒、子智能体) 权限系统 (`permissions.ts`) 实现了拒绝优先的规则评估, 支持最多七种权限模式 (若计入仅内部使用的 `bubble` 和受特征控制的 `auto` 模式, 则共包含七种) (`types/permissions.ts`), 并集成了一种自动模式机器学习分类器 (`yoloClassifier.ts`), 该分类器提供两阶段快速过滤与思维链评估以判断工具安全性 ([Section 5](#))。一个跨越 27 种事件类型的 钩子流水线 (`coreTypes.ts`; 输出模式见 `types/hooks.ts`) 可用于阻断、重写或标注工具请求; 其中 5 个为安全相关, 其余 22 个则服务于生命周期与编排目的 ([Section 6](#))。一个可扩展性子系统允许插件和技能将工具与钩子注册到运行时。通过 `assembleToolPool()` (`tools.ts`) 组装工具池, 合并内置工具与 MCP 提供的工具。经批准的 shell 命令会经过一个 `shell` 沙箱 (`shouldUseSandbox.ts`), 该沙箱独立于权限系统限制文件系统和网络访问。通过 `AgentTool` (`AgentTool.tsx`, `runAgent.ts`) 实现的子智能体生成, 通过与所有其他工具相同的 `buildTool()` 工厂分发, 以隔离的上下文窗口重新进入 `queryLoop()`, 仅向父级返回摘要 ([Section 8](#))。

状态层 (上下文组装、运行时状态、持久性、内存、侧链)。上下文组装是一个记忆化的状态加载器, 而非路由中心: `getSystemContext()` (`context.ts`) 计算包括 Git 状态在内的会话级系统上下文, 而 `getUserContext()` (`context.ts`) 加载 `CLAUDE.md` 层次结构和当前日期。两者均被缓存以供重用: 系统上下文附加到系统提示中, 用户上下文则作为用户上下文消息添加。`src/state/` 目录管理运行时应用状态。会话记录以几乎仅追加的 JSONL 文件形式存储于项目特定路径下 (`sessionStorage.ts`)。 `CLAUDE.md` + 记忆子系统提供从受管设置到目录特定文件的四级指令层级 (`claudemd.ts`), 外加 Claude 在对话过程中自动生成的记忆条目 ([Section 7.2](#))。侧链记录 (`sessionStorage.ts:247`) 将每个子智能体的对话存储在分离的文件中, 防止子智能体内容膨胀父级上下文 ([Section 8.3](#))。全局提示历史记录保存在 `history.jsonl` (`history.ts`) 中。恢复和分叉操作通过记录重建会话状态 (`conversationRecovery.ts`)。

后端层 (执行后端, 外部资源) Shell 命令执行, 支持可选的沙箱环境 (`BashTool.tsx`, `PowerShellTool.tsx`), 远程执行支持 (`src/remote/`), 通过多种传输方式 (包括 stdio、SSE、HTTP、WebSocket、SDK 以及 IDE 特定适配器) 连接 MCP 服务器 (`services/mcp/client.ts`), 以及 `src/tools/` 中的 42 个工具子目录实现了具体的工具逻辑。

3.4 查询引擎：一个澄清

`QueryEngine.ts` 的类文档中指出: “`QueryEngine` 负责对话的查询生命周期和会话状态。它将 `ask()` 中的核心逻辑提取到一个独立的类中, 该类可被无头/SDK 路径使用, 并且 (在未来的阶段) 也可用于

REPL。”该类是针对非交互式界面的 对话包装器,而非引擎本身。其构造函数接受一个 `QueryEngineConfig`, 包含初始消息、中断控制器、文件状态缓存以及其他对话相关的状态。其 `submitMessage()` 方法是一个异步生成器,用于协调单轮对话。共享的查询路径位于 `query()` (`query.ts`), 它封装了内部的 `queryLoop()`; `QueryEngine` 将任务委托给 `query()`。

这一区别在架构上很重要: 交互式 CLI 也调用 `query()`, 完全绕过了 `QueryEngine`。共享的代码路径是循环函数,而不是引擎类。

3.5 权限与安全层

安全默认原则通过七层独立防护实现。请求必须通过所有适用的层级, 任意一层均可阻止该请求:

1. **工具预过滤** (`tools.ts`): 在任何调用之前, 将被完全禁止的工具从模型的视野中移除, 防止模型尝试调用它们。
2. **拒绝优先规则评估** (`permissions.ts`): 拒绝规则始终优先于允许规则, 即使允许规则更具体也是如此。
3. **权限模式约束** (`types/permissions.ts`): 激活的模式决定了与任何显式规则都不匹配的请求的基准处理方式。
4. **自动模式分类器**: 基于机器学习的分类器评估工具的安全性, 可能会拒绝规则系统允许的请求。
5. **Shell 沙盒化** (`shouldUseSandbox.ts`): 经过批准的 shell 命令仍将在沙盒中执行, 以限制对文件系统和网络的访问。
6. **恢复时不再还原权限** (`conversationRecovery.ts`): 会话范围的权限在恢复或分叉时不会被还原。
7. **基于钩子的拦截** (`types/hooks.ts`): `PreToolUse` 钩子可修改权限决策; `PermissionRequest` 钩子可在用户对话期间 (或在协调器模式下提前) 异步地解析决策。

这些层在 [Section 5](#) 中有详细描述。

3.6 上下文作为瓶颈: 超越压缩

除了五层压缩流水线 (详见[Section 7](#)), 其他若干子系统决策也反映了上下文作为瓶颈的约束:

- **CLAUDE.md 懒加载**: 基础的 `CLAUDE.md` 层级结构在会话启动时加载, 但额外的嵌套目录指令文件和条件规则仅在智能体读取这些目录中的文件时才加载, 从而避免未使用的指令占用上下文。
- **延迟加载工具模式**: 当启用 `ToolSearch` 时, 某些工具在初始上下文中仅包含其名称; 完整的模式在需要时才加载。
- **子智能体仅返回摘要**: 子智能体仅向父智能体返回摘要文本, 而不返回其完整的对话历史记录 ([Section 8](#))。
- **每工具结果预算**: 每个工具的结果大小被限制在可配置的范围内, 防止单个冗长输出占用过多上下文。

4 执行转换: Agentic 查询环

当用户提交“修复 `auth.test.ts` 中失败的测试”时, 输入进入一个反应式循环, 这是编码智能体的几种可能编排模式之一。本节将分析 Claude Code 选择的简单 `while` 循环架构, 并端到端追踪该循环的一

次迭代，展示来自 Table 1 的三个设计原则：最小化脚手架但最大化操作支撑、上下文作为稀缺资源并进行渐进式管理，以及 优雅恢复与韧性。

4.1 查询流水线

每一回合都遵循固定的顺序 (Figure 2, query.ts):

1. **情景分辨率。** queryLoop() 函数解构了不可变的参数，包括系统提示、用户上下文、权限回调和模型配置。
2. **可变状态初始化。** 单个 State 对象在迭代间存储所有可变状态，包括消息、工具上下文、压缩跟踪以及恢复计数器。环的七个 continue 点（即“continue 位置”）各自通过整个对象赋值的方式重写该对象，而非逐个字段地修改。
3. **上下文组装。** 函数 getMessagesAfterCompactBoundary() 从最后一个紧凑边界开始向前获取消息，确保被压缩的内容由其摘要表示，而非原始消息。
4. **预模型上下文塑造者。** 五个塑造者按顺序执行 (Section 4.3)。
5. **模型调用。** 对 deps.callModel() 流使用 for await 环，以流式传输模型的响应，传递组装的消息（包含前置的用户上下文）、完整的系统提示、思考配置、可用工具集、中止信号、当前模型规格以及包括快速模式设置、努力值和回退模型在内的其他选项。
6. **工具使用调度。** 如果响应包含 tool_use 块，它们将传递到工具编排层 (Section 4.2)。
7. **权限门禁。** 每个工具请求都必须经过权限系统 (Section 5)。
8. **工具执行与结果收集。** 工具结果以 tool_result 消息的形式添加到对话中，环继续。
9. **停止条件。** 如果响应中不包含 tool_use 块（仅文本），则本轮对话结束。

queryLoop() 函数被定义为一个 AsyncGenerator，在执行过程中会生成 StreamEvent、RequestStartEvent、Message、TombstoneMessage 和 ToolUseSummaryMessage 事件。这种基于生成器的设计能够在保持环内单一同步控制流的同时，实现向 UI 层的流式输出。

Claude Code 的反应式环遵循 ReAct 模式 (Yao et al., 2022)：模型生成推理和工具调用，框架执行动作，结果反馈至下一次迭代。其他编排模式包括显式的基于图形的路由 (LangChain, Inc., 2024)，其中控制流定义为带有类型边的状态机，以及基于树搜索的方法 (Zhou et al., 2023)，该方法在最终决定前探索多种动作轨迹。Anthropic 自身的文档 (Schluntz and Zhang, 2024) 指出五种可组合的工作流模式（提示链、路由、并行化、编排器-工作者、评估器-优化器），其中 Claude Code 主要使用编排器-工作者模式进行子智能体委派 (Section 8)，同时保持核心环路的反应式设计。这种反应式设计以牺牲搜索完备性为代价，换取简单性和低延迟：每次迭代仅确定一条动作序列，不进行回溯。

4.2 工具分发与流式执行

当模型响应包含 tool_use 块时，系统会在两条执行路径之间进行选择。主路径使用 StreamingToolExecutor，该路径在模型响应流式传输工具时立即开始执行工具，从而降低多工具响应的延迟。备用路径使用 toolOrchestration.ts 中的 runTools()，该函数遍历由 partitionToolCalls() 生成的划分结果。两条路径均将工具分类为并发安全或独占类型。只读操作可以并行执行，而诸如 shell 命令等修改状态的操作则会序列化执行。

StreamingToolExecutor (StreamingToolExecutor.ts) 使用两种协调机制来管理并发执行：

- **兄弟中止控制器**。当任何 Bash 工具出错时触发，立即终止其他正在运行的子进程，而不是让它们运行到完成。
- **进度可用信号**。当有新的输出就绪时，唤醒 `getRemainingResults()` 消费者。

结果按接收工具的顺序进行缓冲并输出，因此即使工具并行运行，输出顺序也保持不变。这一点非常重要，因为模型期望的工具结果顺序与其工具调用请求的顺序一致。这种并发读取、串行写入的执行模型在完全串行调度和更激进的推测性方法（如 PASTE (Sui et al., 2026)）之间取得了一个平衡，后者在模型仍在生成时就推测性地预先执行预测的未来工具调用，通过推测来隐藏工具延迟。

工具结果收集阶段会遍历来自流式执行器或同步 `runTools()` 生成器的更新。每次更新可能携带工具结果、附件或进度事件。一个特殊检查会检测 `hook_stopped_continuation` 附件：如果 `PostToolUse` 钩子表明本轮不应继续，则设置 `shouldPreventContinuation` 标志。结果通过 `normalizeMessagesForAPI()` 进行规范化的 Anthropic API 处理，仅保留用户类型的消息。

4.3 模型前文背景塑造者

在每次模型调用之前，五个上下文塑造器会在 `query.ts` 中按顺序执行，每个都作用于 `messagesForQuery` 数组。这五个塑造器按顺序运行，较早的步骤先进行较轻度的缩减，随后的步骤再进行更广泛的压缩。

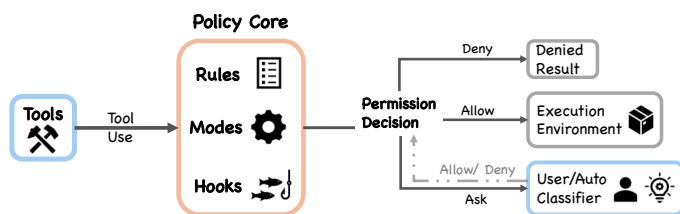
预算削减。（`applyToolResultBudget()`）。对工具结果施加每条消息的大小限制，将超出大小的输出替换为内容引用。未受限制的工具（即 `maxResultSizeChars` 不为有限值的工具）保留其完整输出。内容替换会针对智能体和会话查询来源进行持久化，以便在恢复时进行重构。预算缩减操作在微紧凑之前执行，因为微紧凑仅通过 `tool_use_id` 进行操作，从不检查内容；两者可无缝组合。

剪切。（`snipCompactIfNeeded()`，由 `HISTORY_SNIP` 控制）。一种轻量级裁剪操作，用于移除较早的历史片段，返回 `{messages, tokensFreed, boundaryMessage}`。其中 `snipTokensFreed` 值会传递至自动压缩机制，因为主 token 计数器从最近一条助手消息的 `usage` 字段中推导上下文大小，而该消息在裁剪后仍保留其裁剪前的 `input_tokens`；因此，裁剪所节省的 token 数量对计数器而言是不可见的，除非显式传递。

超小型 细粒度压缩，始终运行基于时间的路径，并可选择性地运行缓存感知路径（由 `CACHED_MICROCOMPACT` 控制）。当启用缓存路径时，边界消息将延迟到 API 响应之后发送，以便使用实际的 `cache_deleted_input_tokens` 而非估算值。返回 `{messages, compactionInfo}`，其中 `compactionInfo` 可能包含 `pendingCacheEdits`。

上下文坍塌。由 `CONTEXT_COLLAPSE` 控制。对对话历史的读取时投影。源代码注释解释：“不产生任何输出；压缩视图是 REPL 完整历史的读取时投影。摘要消息存储在压缩存储中，而非 REPL 数组中。这正是压缩能在回合间持续的原因。”与其他整形器不同，上下文压缩不会修改 REPL 存储的历史；它通过 `applyCollapsesIfNeeded()` 将 `messagesForQuery` 数组替换为一个投影视图，因此模型看到的是压缩版本，而完整历史仍可用于重构。

自动压缩。第五个塑形器通过 `compact.ts` 中的 `compactConversation()` 触发完整的模型生成摘要。该函数执行 `PreCompact` 钩子，使用 `getCompactPrompt()` 创建摘要请求，并调用模型生成压缩后的摘



Principle	Description
Progressive Trust	The agent starts with minimal autonomy; users expand it by approving tool invocations that become permanent rules.
Deny-First, Ask-by-Default	Deny rules always win, even under looser modes. If no rule matches, the gate asks the user instead of silently running or blocking.
Composable Policy	Three mechanisms shape policy: declarative rules, global trust modes, and programmable hooks, each independently configurable.

Figure 4 权限网关概述与设计原则。

要。结果将输入至 `buildPostCompactMessages()` (`compact.ts`)。自动压缩仅在经过前四个塑形器全部运行后，上下文仍超过压力阈值时触发。

4.4 恢复机制

查询环为边缘情况实现了几种恢复机制：

- **最大输出 token 升级**：当响应达到输出 token 限制时，系统可以在满足 GrowthBook 标志且不存在现有覆盖或环境变量限制的情况下，尝试以提升的限额重新执行。每轮最多允许三次恢复尝试 (`MAX_OUTPUT_TOKENS_RECOVERY_LIMIT = 3`)。
- **反应式压实**（由 `REACTIVE_COMPACT` 控制）：当上下文接近容量时，反应式压实仅进行必要的汇总以释放空间。`hasAttemptedReactiveCompact` 标志确保每回合最多触发一次。
- **提示过长处理**：如果 API 返回 `prompt_too_long` 错误，环首先尝试上下文压缩上溢恢复和主动紧凑化。只有在这些尝试失败后，才会以 `reason: 'prompt_too_long'` 终止。
- **流回退**：`onStreamingFallback` 回调函数处理流式 API 问题，允许环重试使用不同的策略。
- **备用模型**：`fallbackModel` 参数可在主模型失效时切换到备用模型。

4.5 停止条件

多个条件可以终止循环：

1. **不使用工具**：模型仅生成文本内容（主要停止条件）。
2. **最大回合数**：达到可配置的 `maxTurns` 限制。
3. **上下文上溢**：API 返回 `prompt_too_long`。
4. **钩子干预**：在使用工具后设置 `hook_stopped_continuation` 的钩子。
5. **显式中止**：`abortController` 信号被触发。

流转流水线决定了工具请求的协调与恢复方式。下一节将探讨决定每个请求是否被允许执行的门控机制。

5 工具授权与控制边界

生产编码智能体采用不同的安全架构：分层策略强制、操作系统级沙箱或基于版本控制的回滚。Claude Code 结合前两种，实现了 Table 1 中提出的四项设计原则：默认拒绝并需人工升级、渐进式信任谱系、纵深防御与分层机制以及权重化风险评估。

当 Claude 决定执行某个工具（例如，通过 BashTool 运行 `npm test` 以重现认证测试失败）时，请求会进入如 Figure 4 所示的权限流水线。每次工具调用都会经过权限系统，而默认行为是拒绝或询问，而非静默允许。这一默认策略源于一项已记录的行为模式：Anthropic 的自动模式分析 (Hughes, 2026) 发现，用户对权限提示的批准率约为 93%，表明审批疲劳使得交互式确认在行为上不可靠，无法作为唯一的安全机制。由于用户习惯性地未经仔细审查就批准，系统必须独立于人类警觉性来维持安全。这促使了架构上的承诺：采用先拒绝的评估方式、全覆盖拒绝的预过滤以及沙箱化作为相互独立的层，这些机制均不依赖于用户的注意力。

5.1 权限模式与规则评估

存在七种权限模式，分布在类型定义中 (`types/permissions.ts` 中有 5 种外部模式；`auto` 为条件添加；`bubble` 属于类型联合)。

1. `plan`：模型必须生成一个计划；在用户批准后才继续执行。
2. `default`：标准交互使用。大多数操作需要用户确认。
3. `acceptEdits`：工作目录内的编辑以及某些文件系统外壳命令 (`mkdir`、`rmdir`、`touch`、`rm`、`mv`、`cp`、`sed`) 将自动获得批准；其他外壳命令则需要手动批准。
4. `auto`：基于机器学习的分类器会评估未通过快速路径检查的请求（由 `TRANSCRIPT_CLASSIFIER` 控制）。
5. `dontAsk`：不进行提示，但拒绝规则仍然有效。
6. `bypassPermissions`：跳过大多数权限提示，但安全关键检查和无法绕过的规则仍然适用。
7. `bubble`：智能体内部专用模式，用于子智能体向父终端提升权限。

五个外部可见模式 (`acceptEdits`、`bypassPermissions`、`default`、`dontAsk`、`plan`) 定义在 `EXTERNAL_PERMISSION_MODES` 数组中。当 `TRANSCRIPT_CLASSIFIER` 特性标志激活时，`auto` 模式才会被有条件地包含。`bubble` 模式存在于类型联合中，但不在任一模式数组中；它在内部用于子智能体权限提升 (Section 8)。

权限规则按照拒绝优先的顺序进行评估 (`permissions.ts`)。`toolMatchesRule()` 函数首先检查拒绝规则：即使允许规则更具体，拒绝规则始终优先于允许规则。广泛的拒绝（例如“拒绝所有 shell 命令”）无法被狭窄的允许（例如“允许 `npm test`”）覆盖。该规则系统支持工具级别匹配（按工具名称）和内容级别匹配（匹配特定工具输入模式，如 `Bash(prefix:npm)`）。

七种模式涵盖了从 `plan`（用户在执行前批准所有计划）到 `default`、`acceptEdits`，直至 `bypassPermissions`（最少提示）的渐进自主性谱系。这一梯度反映了反复出现的设计矛盾：随着自主性的提升，系统必须从交互式审批转向自动化安全检查。其他智能体系统以不同方式解决这一矛盾。SWE-Agent 和 Open-Hands (Yang et al., 2024; Wang et al., 2024b) 使用 Docker 容器隔离，对智能体的整个执行环境进行沙盒化，而非评估单个工具调用。Aider (Gauthier, 2024) 依赖 Git 作为安全网，通过版本控制使所有更

改均可撤销。Claude Code 的方法在可选的容器沙盒基础上叠加多重策略执行机制，以牺牲简单性换取对单个动作的细粒度控制。

5.2 授权流水线

完整的授权流水线经过多个阶段：

预过滤。在任何工具请求到达运行时评估之前，`filterToolsByDenyRules()` (`tools.ts`) 会在工具池组装时完全从模型的视图中移除被全面禁止的工具。文档说明：“使用与运行时权限检查相同的匹配器，因此 MCP 服务器前缀规则如 `mcp__server` 会在模型看到这些工具之前，将其所在服务器的所有工具全部移除。”这可以防止模型尝试调用被禁止的工具，从而避免模型在这些工具上浪费调用次数。

使用前工具钩。注册的钩子会作为权限流水线的一部分触发。`PreToolUse` 钩子可以返回一个 `permissionDecision` 以拒绝或询问，或者返回一个 `updatedInput` 来修改工具的输入参数 (`types/hooks.ts`)。钩子 `allow` 不会绕过后续基于规则的拒绝或安全检查。在交互路径中，用户对话首先被加入队列，钩子异步执行；协调器及其他类似后台智能体路径则在显示对话前等待自动化检查完成。

规则评估。拒绝优先规则引擎对请求进行评估。MCP 工具通过其完全限定的 `mcp__server__tool` 名称进行匹配，而服务器级别规则则匹配来自该服务器的所有工具。

权限处理程序。在 `useCanUseTool.tsx` 中，根据运行时上下文，处理器会分支到四个路径之一：

1. **协调器**：用于多智能体协调模式。在回退到用户交互之前，尝试通过自动化方式（分类器、钩子、规则）进行解决。
2. **群体工作者**：处理具有各自求解逻辑的多智能体群体中的工作者智能体。
3. **投机分类器**：当启用 `BASH_CLASSIFIER` 且工具为 `BashTool` 时，投机分类器会将预启动的分类结果与超时时间进行竞争。如果分类器以高置信度返回结果，则无需用户交互即可立即批准该工具。
4. **交互式**：备用路径。通过终端用户界面显示标准的用户授权对话框。

在协调器和某些后台路径中，系统会在用户交互前尝试自动解决。在标准的交互路径中，对话可以先出现，而钩子或分类器检查则并行进行。当分类器或拒绝规则阻止某个动作时，系统将拒绝视为路由信号而非强制停止：模型接收拒绝原因，调整其策略，并在下一个环迭代中尝试更安全的替代方案。`PermissionDenied` 钩子事件 ([Section 6](#)) 允许外部代码以编程方式观察并响应这些拒绝。这种以恢复为导向的设计意味着权限限制塑造了智能体的行为，而非简单地终止其运行。

5.3 自动模式分类器与钩子生命周期

当启用时，自动模式分类器 (`yoloClassifier.ts`) 会参与权限决策。当 `TRANSCRIPT_CLASSIFIER` 被启用时，分类器将加载三个提示资源：

- 一个基础系统提示。
- 外部权限模板
- 对于 Anthropic 内部用户，一个分离的内部模板。

分类器根据对话记录和权限模板评估所提议的工具调用，生成允许、拒绝或需要人工审批的决定。函数 `isUsingExternalPermissions()` 通过检查 `USER_TYPE` 和 `forceExternalPermissions` 配置标志来选择合适的模板。

在源代码中定义的 27 个钩子事件 (`coreTypes.ts`) 里，有五个直接参与权限流程，每个事件都有特定的 Zod 验证输出模式 (`types/hooks.ts`):

- **PreToolUse**: 可返回 `permissionDecision` (拒绝或询问,但允许不会绕过后续检查)、`permissionDecisionReason` 和 `updatedInput` (修改参数)。
- **PostToolUse**: 可注入 `additionalContext`, 对于 MCP 工具, 还可返回 `updatedMCPToolOutput` 以在结果进入上下文前进行修改。
- **PostToolUseFailure**: 可注入 `additionalContext` 以提供针对特定错误的指导。
- **权限被拒**: 在自动模式拒绝后, 可提供 `retry` 指导。
- **权限请求**: 可返回 `decision` 为 `allow` 或 `deny`。在协调器及类似路径中, 可在用户对话前完成解析。在标准交互路径中, 也可与对话同时运行。

对于非 MCP 工具, `tool_result` 在 `PostToolUse` 钩子触发之前发出。对于 MCP 工具, 结果会延迟到后置钩子执行之后, 以使 `updatedMCPToolOutput` 生效。

5.4 壳沙盒

Shell 沙箱为 Bash 和 PowerShell 命令提供了额外的保护层(`shouldUseSandbox.ts`)。`shouldUseSandbox()` 函数检查沙箱是否全局启用、调用是否选择退出, 以及命令是否匹配任何排除模式。

当启用时, 沙箱提供独立于应用级权限模型的文件系统和网络隔离。一个命令可以获得权限批准但仍处于沙箱中, 或者权限被拒绝而根本无法到达沙箱检查。这两个系统在不同维度上运作: 授权与隔离。

分层安全架构建立在独立性假设之上: 若某一层失效, 其他层会捕捉到该违规行为。然而, 多个层共享相同的性能约束。安全研究人员 ([Adversa.ai, 2026](#)) 已记录, 当命令包含超过 50 个子命令时, 系统会退回到单一通用批准提示, 而非执行每个子命令的拒绝规则检查, 因为逐子命令解析会导致用户界面冻结。此例表明, 当各层具有共同的故障模式时, 纵深防御机制可能退化, 这揭示了安全与性能之间的结构性矛盾, 将在 [Section 11.3](#) 中进一步分析。

权限流水线决定工具请求是否执行。下一节将探讨哪些因素决定了工具本身的存在: 即构建模型动作界面的可扩展性架构。

6 可扩展性: MCP、插件、技能和钩子

对于编码智能体而言, 一个反复出现的设计问题是如何构建扩展表面: 是采用单一统一的机制、少量专门化的机制, 还是分层堆叠且具有不同上下文窗口成本的结构。本分析揭示了 [Table 1](#) 中的两个设计原则: 可组合的多机制可扩展性与外部化可编程策略。回到前面的例子, 当 Claude 正在尝试修复 `auth.test.ts`, 而之前的 `npm test` 请求已被权限系统 ([Section 5](#)) 中介后, 下一个问题便是可用于修复的、具备扩展功能的动作表面是什么。在 Claude Code 的一个回合开始时, 模型不仅能看到 `BashTool` 和 `FileReadTool` 等内置工具, 还能访问来自 MCP 服务器的数据库查询工具、来自 `.claude/skills/` 的自定义代码检查技能, 以及由已安装插件贡献的工具。这些功能通过四种在循环不同环节扩展智能体

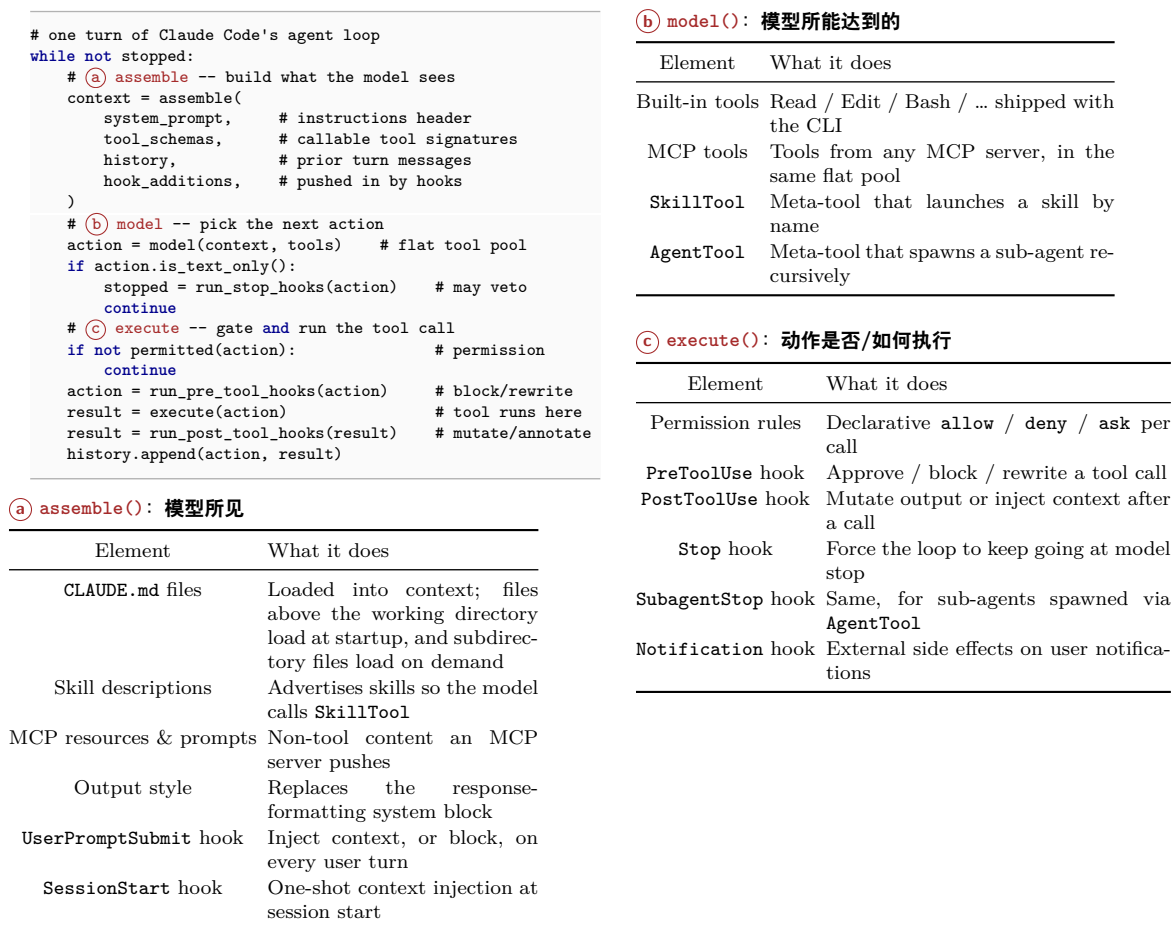


Figure 5 Claude Code 的扩展机制接入智能体环的位置。左侧的伪代码是图 1 中 **Agent Loop** 块的放大视图。每个智能体环都有三个注入点：(a) **assemble()** 控制模型所见内容，(b) **model()** 控制其可访问的内容，(c) **execute()** 控制动作是否以及如何实际运行。

的机制实现：MCP 服务器提供外部工具集成，插件以包的形式打包并分发组件，技能注入领域特定指令，钩子则拦截工具执行生命周期。Anthropic 的文档 (Anthropic, 2026d) 提供了更广阔的视角，包含 CLAUDE.md (Section 7) 和子智能体 (Section 8)，除了此处分析的四种机制外。我们将 CLAUDE.md 和子智能体分别在独立章节中讨论，因为它们分别运行于不同的子系统（上下文构建与委托），但上下文窗口成本的排序具有重要的架构意义：它揭示了每个扩展点在表达能力与受限上下文窗口之间的权衡。

6.1 四个扩展机制

这些机制在不同的源代码目录 (Figure 5) 中实现，并服务于不同的集成模式：

MCP 服务器。 模型上下文协议 (Model Context Protocol) 是主要的外部工具集成路径。MCP 服务器配置来自多个作用域：项目、用户、本地和企业，此外在运行时还会合并插件和 `claude.ai` 服务器 (`services/mcp/config.ts`)。MCP 客户端 (`services/mcp/client.ts`) 支持多种传输类型：stdio、SSE、HTTP、WebSocket、SDK，以及针对特定 IDE 的变体 (`sse-ide`、`ws-ide`) 和一个内部的 `claudeai-proxy`。每个连接的服务器都会贡献工具定义作为 `MCPTool` 对象。专用的内置工具 `ListMcpResourcesTool` 和 `ReadMcpResourceTool` 提供对 MCP 资源的访问。

插件。 插件扮演双重角色:既是打包格式,也是分发机制。`PluginManifestSchema (utils/plugins/schemas.ts)` 接受十种组件类型: 命令、智能体、技能、钩子、MCP 服务器、LSP 服务器、输出样式、通道、情景和用户配置。插件加载器 (`utils/plugins/pluginLoader.ts`) 会验证清单, 并将每个组件路由至其对应的注册表: 命令和技能通过 `SkillTool` 元工具呈现, 智能体出现在被 `AgentTool` 消费的定义中, 钩子合并到钩子注册表中, MCP 和 LSP 服务器则融入其标准配置, 输出样式则修改响应格式。因此, 一个插件包可以同时扩展 Claude Code 的多个组件类型, 使插件成为第三方扩展的主要分发载体。

技能。 每个技能由一个包含 YAML 前置元数据的 `SKILL.md` 文件定义。`parseSkillFrontmatterFields()` 函数 (`loadSkillsDir.ts`) 解析超过 15 个字段, 包括显示名称、描述、允许的工具 (授予技能访问额外工具的权限)、参数提示、模型覆盖、执行上下文 ('fork' 表示隔离执行)、关联的智能体定义、努力级别以及 shell 配置。技能可以定义自己的钩子, 在调用时动态注册。捆绑的技能在启动时注册到内存中。调用时, `SkillTool` 元工具会将技能的指令注入上下文。

钩子。 源代码定义了 27 个钩子事件, 涵盖工具授权 (`PreToolUse`、`PostToolUse`、`PostToolUseFailure`、`PermissionRequest`、`PermissionDenied`)、会话生命周期 (`SessionStart`、`SessionEnd`、`Setup`、`Stop`、`StopFailure`)、用户交互 (`UserPromptSubmit`、`Elicitation`、`ElicitationResult`)、子智能体协调 (`SubagentStart`、`SubagentStop`、`TeammateIdle`、`TaskCreated`、`TaskCompleted`)、上下文管理 (`PreCompact`、`PostCompact`、`InstructionsLoaded`、`ConfigChange`)、工作区事件 (`CwdChanged`、`FileChanged`、`WorktreeCreate`、`WorktreeRemove`) 以及通知 (`coreTypes.ts`、`coreSchemas.ts`)。其中, 15 个事件具有特定的输出模式, 包含丰富字段, 支持权限决策、上下文注入、输入修改、MCP 结果变换和重试控制 (`types/hooks.ts`)。通过设置和插件配置的持久化钩子命令使用四种命令类型: Shell 命令 (`type: command`)、LLM 提示钩子 (`type: prompt`)、HTTP 钩子 (`type: http`) 以及 Agentic 验证钩子 (`type: agent`) (`schemas/hooks.ts`)。运行时还支持不可持久化的回调钩子 (`type: callback`), 由 SDK 和内部仪表化使用 (`types/hooks.ts`)。钩子来源包括 `settings.json`、插件以及启动时的托管策略; 技能钩子在调用时动态注册 (`utils/hooks.ts`)。五个工具授权事件详见 [Section 5.3](#)。

6.2 工具集装配

`assembleToolPool()` 函数在 `tools.ts` 中被记录为“将内置工具与 MCP 工具组合的唯一可信来源”。该组装过程遵循五步流水线:

1. **基础工具枚举。** `getAllBaseTools()` (`tools.ts`) 返回最多 54 个工具的数组: 其中 19 个始终包含 (如 `BashTool`、`FileReadTool`、`AgentTool`、`SkillTool`), 其余 35 个根据特征标志、环境变量和用户类型有条件地包含。Anthropic 内部用户可获得额外的内部工具。工作树模式启用 `EnterWorktreeTool` 和 `ExitWorktreeTool`。智能体集群模式启用团队工具。当 Bun 二进制文件中可用嵌入式搜索工具时, 将省略专用的 `GlobTool` 和 `GrepTool`。
2. **模式过滤。** `getTools()` (`tools.ts`) 执行特定模式的过滤。在 `CLAUDE_CODE_SIMPLE` 模式下, 仅可用 Bash、Read 和 Edit (或 REPL 分支中的 `REPLTool`; 以及适用的协调器工具)。每个工具的 `isEnabled()` 方法都会被调用以进行运行时可用性检查。
3. **拒绝规则预过滤。** `filterToolsByDenyRules()` (`tools.ts`) 在任何调用之前从模型的视图中移除被全面拒绝的工具。

- 4. **MCP 工具集成。**从 `appState.mcp.tools` 获取的 MCP 工具根据拒绝规则进行筛选，并与内置工具合并。
- 5. **去重。**工具按名称进行去重，内置工具优先于 MCP 工具。

`REPL.tsx` (通过 `useMergedTools` 钩子) 和 `AgentTool.tsx` (在构建工作线程工具集时) 都会调用此函数，确保所有执行路径中的一致性组装。在请求时，延迟加载的工具可能被隐藏于模型的上下文之外，直到通过 `ToolSearch (tools.ts)` 明确查询时才显现。

基于智能体的扩展 (通过 `.claude/agents/*.md` 自定义智能体以及插件提供的智能体) 在 [Section 8](#) 中有介绍，因为智能体与上述四种机制有本质区别：它们创建新的、隔离的上下文窗口，而不是扩展当前的上下文窗口。

6.3 为何是四种机制？

由于每增加一个扩展机制，开发人员需要学习的表面积就随之增大，因此一个自然的问题是：为什么 Claude Code 使用四种不同的机制，而不是将其合并为一两个？答案在于观察到不同类型的可扩展性对上下文窗口带来的成本各不相同，单一机制无法在零上下文生命周期钩子到高度依赖模式的工具服务器之间实现全覆盖，而不会迫使扩展作者做出不必要的权衡。

Table 2 每个扩展机制独特提供的功能。上下文成本指的是当机制激活时，所消耗的受限上下文窗口的比例。

Mechanism	Unique Capability	Context Cost	Insertion Point
MCP servers	External service integration (multi-transport)	High (tool schemas)	<code>model():tool pool</code>
Plugins	Multi-component packaging + distribution	Medium (varies)	All three points
Skills	Domain-specific instructions + meta-tool invocation	Low (descriptions only)	<code>assemble():context injection</code>
Hooks	Lifecycle interception + event-driven automation	Zero by default	<code>execute():pre/post tool</code>

如 [Table 2](#) 所总结，每种机制都以部署复杂性为代价，换取不同类型的可扩展性。MCP 服务器通过提供运行时工具集成（模型获得新的可调用工具）来实现扩展，但需要承担服务器管理开销以及工具模式所消耗的上下文预算。技能以极低的上下文成本塑造智能体的思考方式（而不仅仅是它拥有哪些工具），因为只有前言描述（而非完整内容）会保留在提示中。钩子默认情况下不产生任何上下文开销，即可提供跨切面的生命周期控制（如阻断、重写或注释工具调用），尽管钩子可以选择注入额外上下文。插件将其他三种机制的任意组合打包成可分发的软件包，充当封装和分发层，而非独立的运行时原语。按上下文成本递增排序（钩子为零，技能为低，插件为中，MCP 为高），意味着低成本扩展可以广泛部署而不会耗尽上下文窗口，而高成本扩展则仅保留给真正需要新工具界面的场景。

一些智能体框架提供单一的扩展机制，通常是仅支持工具的 API，所有自定义都以额外的可调用工具形式出现。另一些框架采用两层结构，将工具与配置或指令注入分离开来。Claude Code 的四机制方法能够适应更广泛的扩展模式，从零上下文事件处理器到完整的外部服务集成，但这也增加了开发者在决定针对特定集成任务使用哪种机制时的学习难度。

7 情境构建与记忆

智能体如何管理其上下文窗口并持久化用户指令，是一个核心设计选择，不同的系统在基于文件的透明性、基于数据库的检索以及不透明的学成表示之间做出选择。此处的设计选择实现了 [Table 1](#) 中的两个

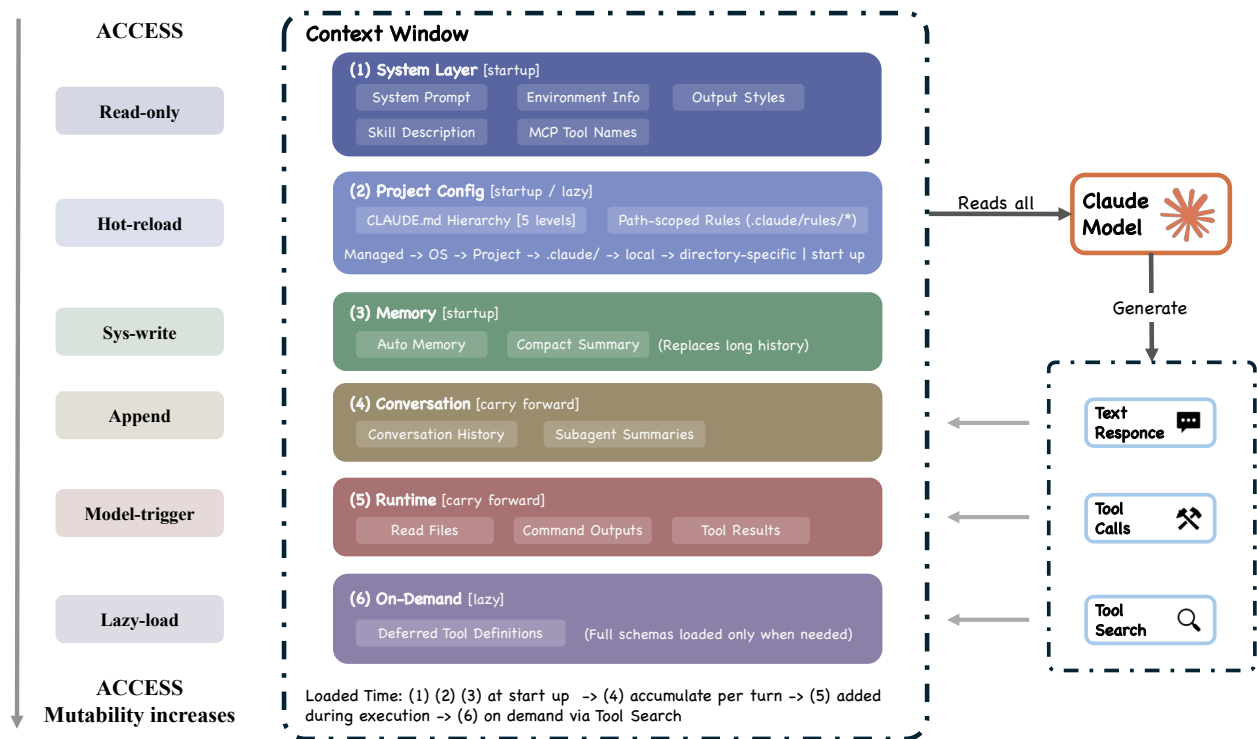


Figure 6 上下文构建与记忆层次结构。汇聚到上下文窗口的来源包括系统提示、输出风格、环境信息、CLAUDE.md 层次结构（通过目录特定方式管理）、自动记忆、路径作用域规则、MCP 工具名称、通过 ToolSearch 延迟定义的工具、对话历史、文件读取、命令输出、工具结果、子智能体摘要以及紧凑摘要。

原则：上下文作为稀缺资源，需进行渐进式管理和透明的基于文件的配置与记忆。

到此为止，运行示例中的任务已累积了状态：原始请求、`npm test` 权限结果、在 Section 6 中组装的工具池，以及迄今为止收集的任何文件读取或命令输出。本节将探讨该不断增长的状态如何被打包进 Claude Code 的有界上下文窗口，以便进行下一次模型调用。

在调用模型之前，智能体环会从工具池 (Section 6)、CLAUDE.md 文件、自动记忆和对话历史中组装上下文窗口。以下小节将介绍组装顺序、CLAUDE.md 层次结构以及多步压缩流水线。

7.1 上下文窗口组装

上下文窗口 (Figure 6) 由以下来源组成，其中一些在初始组装时就已包含，另一些则在回合过程中后期注入：

1. **系统提示**，包含输出样式修改以及任何 `--append-system-prompt` 标志的内容。
2. **环境信息**通过 `getSystemContext()` (`context.ts`) 获取：git 状态（在远程模式下或禁用 git 指令时跳过）以及内部构建的可选缓存破坏注入（受 `BREAK_CACHE_COMMAND` 控制）。每个会话仅缓存一次。
3. **CLAUDE.md 层次结构**通过 `getUserContext()` (`context.ts`)：四层指令文件层次结构 (Section 7.2)。同时具备记忆化功能。
4. **路径作用域规则**：在智能体读取匹配目录中的文件时才会懒加载的条件规则和目录匹配规则。
5. **自动内存**：上下文相关的内存条目异步预取。

6. **工具元数据**: 技能描述、MCP 工具名称以及延迟定义的工具（通过 ToolSearch，按需提供）。
7. **对话历史**: 已延续，待压缩。
8. **工具结果**: 文件读取内容、命令输出、子智能体摘要。
9. **紧凑摘要**: 替换较早的历史片段。

在 `query.ts` 中的系统提示组装通过 `asSystemPrompt(appendSystemContext(systemPrompt, systemContext))()` 将系统上下文与基础提示相结合。用户上下文 (CLAUDE.md 和日期) 通过 `prependUserContext()` 被添加到消息数组的开头。这种分离意味着 CLAUDE.md 的内容在 API 请求中的结构位置与系统提示不同，可能会对模型的注意力模式产生影响。

多个上下文源在主窗口构建后才被注入：相关记忆预取 (`query.ts`)、MCP 指令增量（仅新或已更改的服务器指令）、智能体列表增量以及后台智能体任务通知。因此，上下文窗口在组装时并非静态，而可以在回合期间动态增长。

7.2 Claude.md 层级结构与自动记忆

一种设计原则塑造了记忆系统：存储的上下文应当对用户可检查、可编辑。CLAUDE.md 文件采用纯文本 Markdown 格式，而非结构化配置或不透明的数据库条目。这种透明性选择以表达能力为代价换取可审计性：用户可以阅读、编辑、版本控制并删除智能体所见的任何指令 (MindStudio Team, 2026)。其他记忆架构展示了这一权衡。基于检索增强的方法使用嵌入向量查找来呈现相关的先验上下文，在获得灵活性的同时牺牲了可检查性：用户难以直观查看或编辑检索系统认为相关的内容。基于数据库的记忆提供了结构化查询能力，但需要额外的基础设施，且对版本控制不透明。Claude Code 的基于文件的方法使得智能体所见的每条指令都能直接读取、编辑，并与代码库一同提交。该系统不使用嵌入或向量相似度索引进行记忆检索；相反，它通过大语言模型对记忆文件头部进行扫描，按需选择最多五个相关文件，以文件粒度而非条目粒度呈现内容。基于嵌入的系统能够更精准地检索单个条目，但代价是可检查性下降以及维护索引所需的基础设施。

CLAUDE.md 文件遵循多层级加载层次结构。源头文件 (`claudemd.ts`) 定义了四种内存类型：

1. **托管内存**（例如 Linux 上的 `/etc/claude-code/CLAUDE.md`）：操作系统级别的策略，适用于所有用户。
2. **用户记忆** (`~/.claude/CLAUDE.md`)：私有的全局指令。
3. **项目记忆** (CLAUDE.md、`.claude/CLAUDE.md` 以及项目根目录下的 `.claude/rules/*.md`)：已提交到代码库中的指令。
4. **本地内存** (`CLAUDE.local.md` 在项目根目录中)：被 git 忽略，用于存放私有的项目特定指令。

文件发现从当前目录向上遍历至根目录，检查每个目录中的所有项目和本地内存文件。距离当前目录越近的文件优先级越高（加载顺序靠后）。

文件按“优先级相反的顺序”加载：后加载的文件会获得模型更多的注意力。对于从根目录到当前工作目录 (CWD) 的路径，`.claude/rules/*.md` 中的无条件规则会在启动时立即加载。对于当前工作目录下的嵌套目录，即使无条件规则也会在智能体读取匹配目录中的文件时延迟加载。这意味着随着对话的进行，当探索代码库的新部分时，模型的指令集可以动态演化。

CLAUDE.md 的内容作为用户上下文（用户消息）传递，而非系统提示内容 (`context.ts`)。这一架构

选择具有重要意义：由于 CLAUDE.md 的内容是以对话上下文形式提供，而非系统级指令，因此模型对这些指令的遵循是概率性的，而非保证的。以拒绝优先顺序评估的权限规则 (Section 5) 提供了确定性执行层。这在指导原则 (CLAUDE.md, 概率性) 与执行机制 (权限规则, 确定性) 之间建立了有意的分离。函数调用 `setCachedClaudeMdContent()` 用于缓存已加载的内容，供自动模式分类器使用，以避免 CLAUDE.md 加载器与权限系统之间的导入循环。

内存文件支持 `@include` 指令以实现模块化的指令集 (`processMemoryFile()` 在 `claudemd.ts` 中)。语法变体包括 `@path`、`@./relative`、`@~/home` 和 `@/absolute`。该指令仅在叶文本结点中有效 (不在代码块内部)。在实现中，包含文件首先被压入，然后将被包含的文件追加在其后，通过跟踪已处理的路径来防止循环引用，并且对不存在的文件会静默忽略。

7.3 压缩流水线

五层压缩流水线 (Section 4.3) 通过渐进式压缩 (`query.ts`) 实现了“上下文作为瓶颈”的原则。与单一策略不同，Claude Code 依次应用五个层级，每个层级的压缩强度逐步增加 (其中三个层级由特征标志控制；预算缩减始终启用，而自动压缩则由用户配置)。这种渐进式方法与更简单的替代方案形成对比：许多智能体框架采用单次遍历截断 (丢弃最旧的消息) 或单一摘要步骤。渐进式设计体现了懒惰降级原则：优先应用破坏性最小的压缩策略，仅在较廉价的方法证明不足时才逐步升级。该方法的代价是复杂性。五个相互作用的压缩层级，加上多个受特征标志控制的层级，使得用户难以完全预测其行为。自动压缩会在对话记录中生成可见摘要，微压缩则发出边界标记，但上下文坍塌不会产生用户可见的输出。更简单的单次遍历方法虽然会牺牲信息，但更容易理解。

1. **预算缩减** (始终有效)：每工具结果的大小限制。
2. **Snip** (`HISTORY_SNIP`)：轻量级的历史信息剪裁。
3. **微紧凑** (`CACHED_MICROCOMPACT`)：细粒度的缓存感知压缩。
4. **上下文坍塌** (`CONTEXT_COLLAPSE`)：基于历史的阅读时虚拟投射。
5. **自动压缩** (默认启用，可禁用)：由模型生成的完整摘要。

`buildPostCompactMessages()` 函数 (`compact.ts`) 返回以下压缩后的输出结构：[`boundaryMarker`, ...`summaryMessages`, ...`messagesToKeep`, ...`attachments`, ...`hookResults`]。边界标记通过 `annotateBoundary` 附加了保留段元数据，记录了 `headUuid`、`anchorUuid` 和 `tailUuid`，以支持读取时的链路修补。这种主要追加的设计意味着压缩过程从不修改或删除之前写入的对话行；它仅追加新的边界和摘要事件。

压缩函数 `compactConversation()` (`compact.ts`) 包含若干设计选择。预压缩钩子首先触发，允许通过钩子注入自定义指令。一个 `GrowthBook` 特性标志控制是否在压缩路径中重用主对话的提示缓存 (代码注释记录了一项 2026 年 1 月的实验：“关闭该路径时缓存未命中率高达 98%，占集群 `cache_creation` 成本的 $\sim 0.76\%$ ”)。压缩完成后，附件构建器会重新声明运行时状态 (计划、技能和异步智能体)，因为压缩过程虽然丢弃了先前的附件消息，但并未丢弃潜在的状态。

当系统将任务委派给多个子智能体时，上下文隔离变得更为重要，因为每个子智能体都在各自有限的上下文窗口中运行。

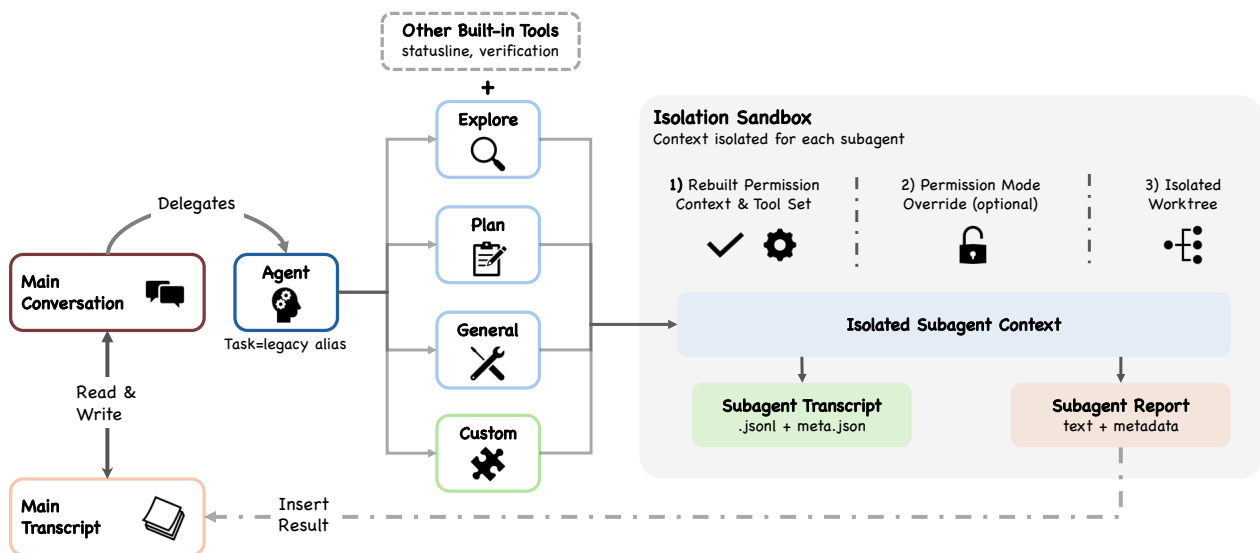


Figure 7 子智能体隔离与委派架构。智能体工具可调度内置子智能体（探索、规划、通用型）或自定义子智能体，每个子智能体在独立的上下文中运行，具备重新构建的权限上下文和独立的工具集。智能体工具沿三个维度进行调度：路由（队友）、隔离（远程、工作树）和生命周期（异步、同步）。

8 子智能体委派与编排

多智能体编排是编码智能体的关键设计维度，其选择范围涵盖父-子层级结构、基于对等对话的框架 (Wu et al., 2024)，以及基于图结构的工作流引擎 (LangChain, Inc., 2024)。Claude Code 的委托架构实现了隔离的子智能体边界原则（来自 Table 1），同时结合了 默认拒绝并支持人工升级（权限覆盖）和 可逆性加权风险评估（子智能体工具限制）的特性。

当 Claude 判断认证测试修复需要先探索认证模块的结构时，它可以将此探索任务委派给一个子智能体。委派机制是 Agent 工具 (AgentTool.tsx)，其中 Task 作为遗留别名保留。模型通过结构化输入调用 Agent，该输入包括委派提示、可选的子智能体类型，以及隔离模式、权限覆盖和工作目录的配置。

8.1 智能体工具与委托标准

智能体工具输入模式 (Figure 7) 使用特征控制的字段，当其支持的特征被禁用时，会省略可选参数。isolation 字段对内部用户提供 ['worktree', 'remote']，对外部用户则提供 ['worktree']，该设置在构建时确定。cwd 字段受特征标志控制。当后台任务被禁用或启用分叉子智能体模式时，run_in_background 字段将被省略。

Claude Code 提供最多六种内置子智能体类型，具体取决于功能标志和入口点：

- **探索**：以阅读/搜索为主的调查方式，其拒绝列表中包含撰写和编辑工具。
- **计划**：创建结构化的计划；执行过程遵循标准的权限模型。
- **通用型**：具备广泛能力，在明确请求时使用（注意：省略类型可能会导致路由到分叉子智能体路径）。
- **Claude 代码指南**：提供入门和文档协助，并支持独立的 permissionMode 覆盖。
- **验证**：运行验证检查（测试套件、代码风格检查）。

- **状态栏设置**：专用于终端状态栏的配置。

除了内置智能体外,用户通过 `.claude/agents/*.md` 文件定义自定义子智能体,而插件则通过 `loadPluginAgents.ts` 贡献智能体定义。每个文件的 Markdown 正文用作智能体的系统提示, YAML 前置元数据指定配置字段, 包括 `description`、`tools` (允许列表)、`disallowedTools`、`model`、`effort`、`permissionMode`、`mcpServers`、`hooks`、`maxTurns`、`skills`、`memory` 作用域、`background` 标志以及 `isolation` 模式。以 JSON 格式编写的智能体定义支持相同字段, 并额外包含 `prompt` 作为显式字段 (`loadAgentsDir.ts`)。这意味着自定义智能体可以是一个完全配置好的、隔离的子系统, 拥有自己的工具、模型、权限、钩子、内存作用域和隔离模式。 `AgentTool` 与 `SkillTool` 一同位于基础工具池中, 作为一个元工具, 负责分派到这些定义, 但两者在本质上存在差异: `SkillTool` 将指令注入当前上下文窗口, 而 `AgentTool` 则生成一个全新的、隔离的上下文。权衡之处在于, 大多数子智能体调用都需要一个自包含的提示, 因为默认路径不会继承父级的对话历史记录 (分支子智能体路径是例外)。基于对话的框架虽能共享完整的对话历史, 避免此开销, 但随着智能体数量增加, 却面临上下文爆炸的风险。

8.2 隔离架构

子智能体隔离支持多种模式 (`AgentTool.tsx`):

- **工作树**: 创建一个临时的 git 工作树, 使子智能体拥有仓库的独立副本, 以便修改而不会影响父级的工作区。
- **远程** (仅限内部): 在远程 Claude Code 远程环境中启动, 始终在后台运行。
- **运行时** (默认): 与父进程共享文件系统, 但在隔离的对话上下文中运行。

子智能体的权限覆盖逻辑 (`runAgent.ts`) 涉及若干特定规则。当子智能体定义了 `permissionMode` 时, 除非父智能体已处于 `bypassPermissions`、`acceptEdits` 或 `auto` 模式, 否则将应用覆盖规则, 因为这些模式始终具有优先权, 代表用户对安全与自主性权衡的明确决策。对于异步智能体, 系统通过级联方式判断是否避免提示: 首先检查显式的 `canShowPermissionPrompts`, 然后是 `bubble` 模式 (始终显示, 因其会升级至父终端), 最后是默认情况 (同步智能体显示提示, 异步智能体不显示)。可显示提示的后台智能体会设置 `awaitAutomatedChecksBeforeDialog`: `true`, 以确保分类器和钩子在中断用户前完成解析。

这些隔离模式在设计空间中占据不同的位置。基于容器的隔离 (SWE-Agent 和 OpenHands (Yang et al., 2024; Wang et al., 2024b) 使用) 提供了更强的资源边界, 但需要容器基础设施支持。仅上下文的隔离 (如 AutoGen (Wu et al., 2024) 等基于对话的框架所使用) 共享文件系统, 但分离对话历史。Claude Code 的工作区基于隔离在文件系统层面实现了分离, 且无需外部依赖, 利用 Git 内置机制, 而非引入容器编排。

当向 `runAgent()` (`runAgent.ts`) 显式提供 `allowedTools` 时, 将应用两级权限作用域模型。SDK 层的权限 (来自 `--allowedTools`) 会被保留: “SDK 消费者明确指定的权限, 应适用于所有智能体。” 但会用子智能体声明的 `allowedTools` 替换会话级别的规则。当未提供 `allowedTools` (即常见的 `AgentTool` 路径) 时, 父级的会话级别规则将被继承且不被替换。

8.3 侧链转录本

每个子智能体将其自身的对话记录写入一个独立的 `.jsonl` 文件，并附带一个 `.meta.json` 元数据文件 (`sessionStorage.ts`, `runAgent.ts`)。这种侧链设计意味着子智能体的历史记录得以保留，便于调试和审计，但不会增加父会话文件的大小。只有子智能体的最终响应文本和元数据返回到父对话上下文中；完整的子智能体历史记录永远不会进入父智能体的上下文窗口，遵循“上下文作为瓶颈”的原则。

`runAgent()` 函数接受 21 个参数，涵盖了智能体定义、提示、权限、工具、模型设置、隔离和回调等。

仅返回总回报模型是一项经过深思熟虑的上下文保留选择：在智能体之间共享完整转录历史的基于对话的框架，会随着智能体数量的增加而面临上下文爆炸的风险。即便采用隔离上下文的并行模式，也会带来可观的成本。Claude Code 的智能体团队在规划模式下消耗的 token 约为标准会话的 7× (Anthropic, 2025b)，这使得当子智能体也处于隔离上下文时，仅返回总回报的模式变得更为关键。

在智能体团队的多实例协调中，该框架采用文件锁而非消息代理或分布式协调服务 (Anthropic, 2025b)。任务通过基于锁文件的互斥从共享列表中获取，锁文件存储在可预测的文件系统路径上。这种设计以牺牲吞吐量为代价，换取了两个特性：零依赖部署（无需外部基础设施）和完全可调试性（通过读取纯文本 JSON 文件即可检查任意智能体的状态）。

9 会话持久化与恢复

编码智能体中的会话持久化涉及一种设计选择，即在仅追加日志、结构化数据库、基于检查点的快照以及无状态架构之间进行权衡，这些方案在可审计性、查询能力及部署复杂度方面各有不同。Claude Code 的持久化设计实现了来自 Table 1 的仅追加的持久化状态原则。会话范围的权限仅驻留在内存中，且不会被序列化至对话记录；因此，在恢复会话时，权限上下文需依据命令行参数与磁盘配置重新构建；对于重建后的上下文无法识别的请求，则默认采用“先拒绝后提示”策略。

当 `auth-test` 任务到达此部分时，会话中包含了原始提示、工具调用及其结果、紧凑边界，以及探索认证模块 (Section 8) 所得的子智能体摘要。本部分询问哪些成果被持久化记录，以及在不延续会话原有权限授予的情况下，哪些内容可以后期恢复。

Claude Code 的持久化机制会随着事件的发生，将对话内容（消息、工具结果和紧凑边界）写入磁盘。

9.1 转录模型

会话记录以大部分仅追加的 JSONL 文件形式存储在项目特定路径下（显式清理重写为例外）(Figure 8)。

函数 `getTranscriptPath()` (`sessionStorage.ts`) 将其计算为 `join(projectDir, ${getSessionId()}.jsonl)`，其中 `projectDir` 首先通过检查 `getSessionProjectDir()`（由 `switchSession()` 在恢复/分支时设置）确定，若未设置则回退至 `getProjectDir(getOriginalCwd())()`。

三个持续通道独立运行：

1. **会话记录**：包含用户、助手、附件和系统消息的对话记录，以及压缩和其他元数据事件。项目范围内的，每个会话对应一个文件。
2. **全局提示历史记录**：仅存储用户提示，保存在 Claude 配置主目录下的 `history.jsonl` 文件中 (`history.ts`)。 `makeHistoryReader()` 生成器通过 `readLinesReverse()` 以逆序方式生成条目，

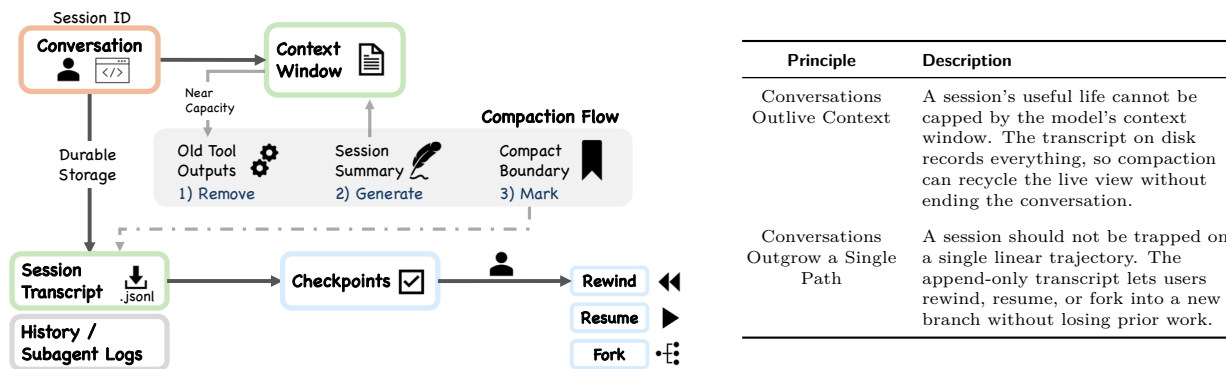


Figure 8 会话持久化与上下文压缩。该图将实时会话状态（上下文窗口、压缩）与持久化存储（会话转录、history.jsonl、子智能体链、检查点）分离。恢复和分叉可恢复消息，但不会恢复会话范围的权限。

支持上箭头和 `ctrl+r` 导航。

3. 子智能体链: 每个子智能体对应分离的 `.jsonl + .meta.json` 文件 (Section 8.3)。

会话记录不仅存储简单的消息，还包括压缩标记、文件历史快照、归属快照和内容替换记录等多种事件。采用仅追加的 JSONL 格式是一种有意的选择，优先考虑可审计性和简洁性，而非查询能力。每个事件都具有人类可读性、版本可控性，并且无需专用工具即可重建。基于数据库的替代方案虽然能实现对会话历史更丰富的查询，但会引入部署依赖并降低透明度。

会话身份系统将 `sessionId` 与 `sessionProjectDir` 配对，在恢复或分支时一同设定。转录文件路径必须使用与消息写入时相同的项目目录，以避免钩子函数在错误的目录中查找。

9.2 恢复、分叉和不恢复权限

`--resume` 标志通过重放对话记录 (`conversationRecovery.ts`) 来重建对话。`fork` 从现有会话创建新会话 (`commands/branch/branch.ts`)。然而，`resume` 和 `fork` 不会恢复会话范围的权限；用户必须在新会话中重新授予这些权限。这是有意为之的安全保守设计选择：会话被视为独立的信任领域。在恢复时还原先前授予的权限虽能带来便利，但可能将过时的信任决策带入变化后的上下文。该架构选择重新授予权限而非隐式持久化，接受用户操作摩擦作为维护安全不变性的代价——信任始终必须在当前会话中建立。

`compact_boundary` 标记经过精心设计，可与持久化机制协同工作。`annotateBoundaryWithPreservedSegment()` 函数 (`compact.ts`) 在边界事件中记录 `headUuid`、`anchorUuid` 和 `tailUuid`。这些 UUID 使得会话加载器能够在读取时修补消息链：保留的消息在磁盘上保持其原始的 `parentUuids`，而加载器则利用边界元数据正确地将其链接起来。这种以追加为主的架构意味着压缩操作从不修改或删除先前写入的对话行。

Claude Code 中的“checkpoints”是用于 `--rewind-files` 的文件历史快照，存储在 `~/.claude/file-history/<session>` 目录下。这些快照是针对文件级别的回滚，用于恢复文件系统更改，而非通用的快照存储。

前文部分记录了 Claude Code 对常见设计问题的解答。下一节将对比 Claude Code 的设计选择与一个架构无关的 AI 智能体系统的设计选择。

10 对比分析：Claude Code 与 OpenClaw

前文部分记录了 Claude Code 针对循环架构、安全性、可扩展性、上下文管理、委托以及持久性等常见设计问题所给出的解答。为校准这些发现，本节将 Claude Code 与 OpenClaw 进行对比，后者是一个独立的开源 AI 智能体系统，从一个根本不同的起点回答了许多相同的设计问题。OpenClaw 是一个以本地优先的 WebSocket 网关，将大约二十个消息渠道（WhatsApp、Telegram、Slack、Discord、Signal 等）连接至嵌入式智能体运行时，并在 macOS、iOS 和 Android 上提供配套应用程序 (Steinberger and OpenClaw Contributors, 2026)。与绑定单一代码仓库会话的 Claude Code 这一命令行编程工具不同，OpenClaw 是用于多通道个人辅助的持久化控制平面。这两个系统位于智能体设计空间的不同区域。这种对比的价值在于展示了当部署环境发生变化时，相同的常见问题会催生出不同的架构解决方案。

10.1 六项比较维度

Table 3 概括了六个维度上的比较。每个维度对应一个两个系统都必须回答的设计问题。

系统范围与部署模型。 Claude Code 以临时的命令行界面进程运行，绑定到单一代码仓库。每次会话均在终端中开始和结束。OpenClaw 则以持久化的守护进程（默认端口 18789，仅限环回）运行，负责管理所有消息表面连接，并通过类型化的 WebSocket 协议协调客户端、工具和设备结点。这种系统范围上的差异是架构上最根本的发散点：它决定了其他所有设计问题的表述方式。此外还存在一种组合关系：OpenClaw 可通过其 ACP（智能体客户端协议）集成，托管 Claude Code、OpenAI Codex 和 Gemini CLI 作为外部编码框架，使两者可堆叠使用，而非单纯的替代关系。

信任模型与安全架构。 这些系统针对不同的威胁模型。Claude Code 假设一个不受信任的模型在受信任的开发者机器内运行：拒绝优先的权限系统 (Section 5) 会评估每一次工具调用，机器学习分类器提供自动化的安全评估，而七种权限模式构建了一个渐进式的自主性谱系。OpenClaw 假设每个网关实例对应单一受信任的操作员。其安全架构从身份和访问控制（DM 配对码、发送方白名单、网关认证）开始，而非针对每项动作的安全分类。工具策略采用按智能体配置的允许/禁止列表，而非集中式分类器。沙箱作为可选功能，支持多种后端（Docker、SSH 或 OpenShell）及可配置的作用范围（按智能体、按会话或共享）；启用时，`non-main` 模式可对所有非主会话进行沙箱隔离，但沙箱默认不激活。OpenClaw 的安全文档明确指出，共享网关上的敌意多租户隔离并非受支持的安全边界。这一差异反映了设计选择中信任边界的定位：Claude Code 将信任边界置于模型与执行环境之间；OpenClaw 则将信任边界置于网关外围。

智能体运行时与工具编排。 两个系统均实现了 Agentic 环，但这些环在其各自架构中的位置不同。在 Claude Code 中，`queryLoop()` 异步生成器 (Section 4) 是系统的中心：所有接口均输入至该环，且它直接管理上下文组装、模型调用、工具调度以及恢复。在 OpenClaw 中，智能体运行时（一个嵌入的 Pi-智能体核心）位于更大的网关分发层内部。网关的 `agent` RPC 会验证参数、解析会话并立即返回；随后嵌入的运行器在网关协议通道中发射生命周期和流事件的同时执行 Agentic 环。各会话通过会话级队列及可选的全局通道进行序列化运行，从而防止在多通道表面上出现工具与会话竞争。两个系统均遵循 ReAct 模式 (Yao et al., 2022)，但 OpenClaw 的环是控制平面内的一个组件，而非控制平面本身。

Table 3 架构对比：Claude Code 与 OpenClaw 在六个设计维度上的比较。每一行反映了常见的设计问题，以及两个系统提供的不同答案。

Dimension	Claude Code	OpenClaw
System scope	CLI/IDE coding harness, ephemeral per-session process	Persistent WS gateway daemon, multi-channel control plane
Trust model	Deny-first per-action rule evaluation with hooks and optional ML classifier; 7 permission modes; graduated trust spectrum	Single trusted operator per gateway; DM pairing and allowlists for inbound channels; opt-in sandboxing with configurable scope (per-agent, per-session, or shared) and multiple backends
Agent runtime	Iterative async generator (<code>queryLoop()</code>) as system center	Pi-agent runner embedded inside gateway RPC dispatch; per-session queue serialization (with optional global lane)
Extension architecture	4 mechanisms at graduated context costs: MCP, plugins, skills, hooks	Manifest-first plugin system with 12 capability types and central registry; separate skills layer; built-in MCP via openclaw mcp (server and outbound client registry)
Memory and context	CLAUDE.md 4-level hierarchy; 5-layer compaction pipeline; LLM-based memory scan	workspace bootstrap files (AGENTS.md, SOUL.md, TOOLS.md, IDENTITY.md, USER.md, plus conditionally BOOTSTRAP.md, HEARTBEAT.md, and MEMORY.md); separate memory system (MEMORY.md, daily notes, optional DREAMS.md); auto-compaction with pluggable providers; optional hybrid search (vector + keyword, conditional on embedding provider); experimental dreaming for long-term promotion
Multi-agent and routing	Task-delegating subagents (e.g., Explore, Plan, general-purpose); worktree isolation; final response text returned to parent	Two separate concerns: (a) multi-agent routing with isolated agents, distinct workspaces, and binding-based channel dispatch; (b) sub-agent delegation with configurable nesting depth (max 5, default 1, recommended 2) and thread-bound sessions

扩展架构。Claude Code 的四种扩展机制 (MCP、插件、技能、钩子) 按上下文窗口消耗组织 (Section 6): 钩子不消耗上下文, 技能消耗少量上下文, 而 MCP 服务器消耗大量上下文。这四种机制均扩展单一智能体的上下文窗口和工具表面。OpenClaw 采用声明优先的插件系统, 包含四个架构层 (发现、启用、运行时加载、表面使用) 以及十二种能力类型, 包括文本推理、语音、媒体理解、图像/音乐/视频生成、网络搜索和消息通道。插件将能力注册到中央注册表; 网关读取注册表以暴露工具、通道、提供者设置、钩子、HTTP 路由、CLI 命令和服务。OpenClaw 还有一个独立的技能层, 支持多种来源 (工作区、项目级、个人、托管、捆绑及额外目录, 其中工作区技能具有最高优先级), 外加一个公共注册表 (ClawHub), 并通过内置 **openclaw mcp** 命令 (服务器和出站客户端注册表) 支持 MCP。其关键架构差异在于, Claude Code 的扩展修改单一智能体的动作表面, 而 OpenClaw 的插件则扩展网关的能力表面, 覆盖所有智能体。

记忆、上下文与知识管理。两个系统均采用基于文件的透明化记忆, 而非不透明的数据库。Claude Code 加载一个四层的 CLAUDE.md 层次结构, 并通过五层压缩流水线 (Section 7) 管理上下文压力。记忆检

索采用基于大语言模型的文件头扫描。OpenClaw 在会话启动时将工作区引导文件注入系统提示中：包含五个核心文件（AGENTS.md、SOUL.md、TOOLS.md、IDENTITY.md、USER.md），以及可选的 BOOTSTRAP.md、HEARTBEAT.md 和 MEMORY.md，其中大文件会被截断。此外，记忆系统独立管理三种文件类型：MEMORY.md 用于长期持久的事实，带日期标记的每日笔记（memory/YYYY-MM-DD.md），以及可选的 DREAMS.md 用于梦境扫描摘要。当配置了嵌入服务提供者后，记忆搜索采用混合检索，结合向量相似度与关键词匹配。一个实验性的梦境系统在后台执行整合操作，对候选项进行评分，并仅将符合资格的项目从短期召回提升至长期记忆。在压缩之前，OpenClaw 会自动提醒智能体将重要笔记保存到记忆文件中，以防止上下文丢失。两个系统都坚持用户可见且可编辑的记忆设计。OpenClaw 更加注重结构化的长期记忆促进（梦境、每日笔记、记忆搜索），而 Claude Code 则更侧重渐进式的上下文压缩（五层并具备缓存感知）。OpenClaw 还支持可插拔的压缩提供者和会话剪枝，但其压缩流水线的渐进性不及 Claude Code 的五层系统。

多智能体架构与路由。这一维度揭示了最显著的架构差异。Claude Code 的多智能体模型采用任务委派机制：父智能体生成子智能体（探索、规划、通用型和自定义类型），这些子智能体在相互隔离的上下文窗口中运行，工具集受限，并仅返回摘要结果（Section 8）。工作区隔离提供了文件系统级别的分离。OpenClaw 则将两个截然不同的关注点分离开来。其一为多智能体路由：单一网关可托管多个完全隔离的智能体，每个智能体拥有独立的工作空间、认证配置文件、会话存储及模型配置，并通过确定性绑定规则被路由至特定通道或发送方。其二为子智能体委派：在单个智能体内部，可配置嵌套深度（最大 5 层，默认 1 层，推荐 2 层）地启动后台运行，支持通道绑定的会话，且可按嵌套层级配置工具策略。OpenClaw 的项目愿景明确拒绝将智能体层次结构框架作为默认架构。这种区别至关重要，因为 Claude Code 的子智能体是同一用户编码会话内的从属工作者，而 OpenClaw 的多智能体路由则创建了真正独立的智能体实例，通过不同通道服务于不同用户或目的。

10.2 对比所揭示的内容

对比揭示了关于人工智能智能体系统设计空间的三个观察结果。

首先，Section 3.1 中识别出的反复出现的设计问题（推理存在于何处、应采取何种安全姿态、如何管理上下文、如何构建可扩展性）不仅适用于编码智能体，也适用于其他场景。OpenClaw 对这些问题给出了回答，但其出发点是一个多通道个人助理，而非局限于代码仓库的编程工具。这些问题本身是稳定的；而答案会随着部署环境的不同而变化。

其次，这些系统在多个维度上做出了相反的投入。Claude Code 在每个动作的安全评估上进行渐进式投资；而 OpenClaw 则在边界级别的身份和访问控制上进行投资。Claude Code 将智能体环视为架构核心；而 OpenClaw 则将网关控制平面视为核心，并将智能体环作为其中一个组件嵌入其中。Claude Code 的扩展修改单一上下文窗口；而 OpenClaw 的插件则扩展共享的网关表面。这些相反的设计并非随意选择，而是源于不同的信任模型和部署拓扑结构。

第三，两个系统之间的组合关系在架构上具有重要意义。OpenClaw 可以通过 ACP 将 Claude Code 作为外部编码工具链进行托管，这意味着这两个系统是可组合的，而非相互排斥的替代方案。这表明人工智能智能体的设计空间并非一个扁平的分类体系，而是一个分层结构，其中网关层系统与任务层工具链可以实现组合。

11 讨论

前文各节的分析记录了 Claude Code 在循环架构、安全姿态、可扩展性、上下文管理、委托机制以及持久化等重复性设计问题上的应对方式。每个答案都反映了设计空间中的一个具体立场，该立场具有实际的替代方案和可度量的权衡。本节探讨当这些答案被综合解读时所揭示的内容：它们所体现的设计哲学 (Section 11.1)、所引发的价值冲突 (Section 11.2)、所涉及的架构权衡 (Section 11.3)、所产生的经验预测 (Section 11.4)，以及在各子系统中反复出现的跨领域承诺 (Section 11.7)。来自 Section 2.1 的五价值框架贯穿全文，作为组织视角。

11.1 设计哲学

Section 2 中引入的数值和设计原则预测了一种侧重于操作基础设施而非决策支撑架构的系统。实现结果证实了这一点：Sections 3 to 9 中记录的架构几乎完全由确定性基础设施（权限门控、工具路由、上下文管理、恢复逻辑）构成，而大语言模型 (LLM) 仅作为无状态补全接口被调用。估算表明，代码库中仅有 1.6% 的内容属于决策逻辑，其余 98.4% 均为操作框架。这一比例并非偶然。

Section 2.2 中记录的设计原则为本方法提供了支持：该机制创造了模型能够做出良好决策的条件，而不是对其选择进行约束。

这种设计与智能体工程中的主流模式背道而驰，后者通常采用 LangGraph 等框架，将模型输出通过具有类型边的显式图结点进行路由，或像 Devin 系统那样将多步规划器与复杂的操作基础设施相结合。而 Claude Code 则在丰富的操作框架内赋予模型最大的决策自由度。工程复杂性并非为了限制模型的决策，而是为了支持其决策。这种分层架构——模型负责推理，框架负责执行——引发了一个问题：智能体式编码工具是否正朝着类似操作系统抽象的方向演进，其中核心环路充当内核，其余部分则构成操作系统？

随着前沿模型在编程任务上的实际能力趋于趋同，该设计的重要性进一步凸显：周边运行环境的质量成为主要差异化因素，验证了将资源投入基础设施而非决策支撑结构的架构合理性。对智能体构建者而言，这意味着投资于确定性基础设施（如上下文管理、安全分层和恢复机制）所带来的可靠性提升，可能大于为日益强大的模型增加规划支撑结构所能带来的收益。

综上所述，前文各部分表明，生产环境中的编码智能体面临一系列反复出现的设计抉择：推理功能相对于工作框架的位置、迭代环的结构设计、默认的安全姿态、扩展面的划分方式、上下文的组装与压缩方法、子智能体的委派与协调机制，以及会话在边界间的持续性处理。Claude Code 对这些问题的回答构成了一个连贯的设计方案，强调在丰富操作框架内赋予模型自主性。

这种理念假设丰富的确定性基础设施能够充分支持无约束的模型判断。以下小节将探讨这一假设在何处受到检验。

11.2 价值冲突

Section 2.1 中确定的五个价值在追求过程中会产生冲突，即追求某一价值会限制另一价值的实现 (Table 4)。这些冲突并非设计缺陷，而是同时追求多个价值所导致的结构性后果。我们报告的是有最强证据支持的冲突，而非全部组合情况。

Table 4 价值观之间的张力，附有证据支持。每一种张力都表明这两个价值观反映了真正不同的关切点。

Value Pair	Tension	Evidence
Authority × Safety	Approval fatigue vs. protection	93% approval rate undermines human vigilance (Hughes, 2026); safety must compensate via classifier and sandboxing
Safety × Capability	Performance vs. defense depth	>50-subcommand fallback skips per-subcommand deny checks due to parsing overhead (Adversa.ai, 2026); safety layers share performance constraints
Adaptability × Safety	Extensibility vs. attack surface	Multiple CVEs exploit pre-trust initialization of hooks and MCP servers (Donenfeld and Vanunu, 2026)
Capability × Adaptability	Proactivity vs. disruption	12 to 18% more tasks but preference drops at high frequencies (Chen et al., 2025)
Capability × Reliability	Velocity vs. coherence	Bounded context prevents full codebase awareness (Section 7); subagent isolation limits cross-agent consistency (Section 8); complexity increases observed in adjacent tools (He et al., 2025)

通过长期能力保留的评估视角 (Section 2.4)，出现了两种额外的张力。一项针对 16 名经验丰富的开发人员在 246 个任务中的随机对照试验 (Becker et al., 2025) 发现，尽管开发者主观上认为效率提升了 20%，但 AI 工具使开发速度反而降低了 19%。对 807 个代码仓库中 Cursor 采用情况的因果分析 (He et al., 2025) 显示，代码复杂度上升了 40.7%。对 54 名参与者的脑电图研究 (Kosmyna et al., 2025) 发现，使用大语言模型的用户表现出神经连接减弱的现象，且在移除 AI 后仍持续存在。研究人员已提出测量 AI 辅助编程中认知卸载的协议，其动机源于对学生使用 AI 生成应用却缺乏对潜在逻辑理解的担忧 (Aiersilan, 2026)。这些发现，结合 2023 年至 2024 年间初级技术岗位招聘人数下降 25% 的数据 (Rak, 2025)，表明能力增强与长期可持续性之间的张力不仅限于个体生产力，更延伸至整个开发人才流水线。这一证据推动了评估视角的形成，但并未特指 Claude Code 的架构；它适用于任何具有有限上下文和工具使用环的智能体系统。

11.3 建筑上的权衡

Table 4 中的张力在四个方面的具体架构权衡中表现出来。上述评估视角段落中记录的长期可持续性问题，在 Section 11.4 的实证预测中显现出来。

安全与自主性。权限模式（始终存在的五种模式，加上在分类器功能标志激活时的 `auto` 模式，以及内部的 `bubble` 模式）形成一个从 `plan`（用户批准所有计划）经 `default`、`acceptEdits`、`auto`（机器学习分类器）、`dontAsk` 到 `bypassPermissions`（跳过大多数提示但保留关键安全检查）的安全性递减梯度。这一递进过程代表了自主性增加的同时安全性单调降低。在恢复时不再恢复权限，反映了有意偏向安全性的选择：安全状态不会在会话边界间隐式持续。

安全-自主梯度不仅由架构设计决定，也受用户行为影响。Anthropic 的自动模式分析 (Hughes, 2026) 发现，用户对权限提示的批准率约为 93%，表明批准疲劳使得交互确认在行为上变得不可靠。纵向使用数据 (McCain et al., 2026) 显示，当会话次数少于 50 次时，自动批准率约为 20%，而到 750 次会话时已超过 40%，同时会话时长显著增加。这些模式表明，用户并非通过刻意选择模式来跨越梯度，而是通过逐渐的习惯化实现。沙盒机制将权限提示的频率降低了约 84% (Dworken and Weller-Davies, 2025)，将

问题重新定义为人为因素问题：面对不可靠的人类批准，架构层面的应对策略是减少人类必须做出的决策数量。

更根本地，Section 5 中描述的纵深防御架构建立在独立性假设之上：如果某一安全层失效，其他层将捕获该违规行为。但 Claude Code 的安全层共享相同的性能和经济约束。自动模式分类器是一个独立的 LLM 调用，具有直接的 token 成本。bashSecurity.ts 模块通过基于 AST 的顺序检查执行，存在解析延迟。拒绝优先规则评估则作用于命令结构。当性能压力促使降低这些成本时，各层可能同时退化。安全研究人员 (Adversa.ai, 2026) 已记录到，包含超过 50 个子命令的命令会回退到单一通用批准提示，而非执行逐子命令的拒绝规则检查，因为逐子命令解析导致了用户界面冻结，这表明当独立性假设被违反时，纵深防御机制会失效。

这种矛盾是结构性的。任何基于大语言模型的智能体系统，若使用模型自身进行安全评估，都会面临这一问题。相关的评价准则并非某个单独层级是否可被绕过，而是需要多少个独立层级同时失效，以及它们是否共享失效模式。

对抗条件下的权限模型。独立的安全研究为权限架构提供了实证验证，具体表现为揭示了 Figure 4 中未捕捉到的时间顺序特性。两个经过独立验证的漏洞均源于预信任初始化顺序的问题：在项目初始化期间（如钩子、MCP 服务器连接和设置文件解析）执行的代码，会在向用户展示交互式信任对话框之前运行。³ 此预信任执行窗口位于拒绝优先评估流水线 (permissions.ts) 之外，形成了一个结构上具有特权的阶段，在此阶段中，Section 5 中记录的安全保障尚未生效。

该模式表明，权限流水线描述了安全检查的空间排序，但并未捕捉时间维度：即在会话初始化过程中，每个机制具体何时激活。初始化顺序（扩展加载、信任对话、权限执行）产生了一个窗口期，在此期间可扩展性架构 (Section 6) 在安全架构 (Section 5) 完全启用之前就已经运行。这一发现通过引入安全维度，进一步细化了可扩展性与简洁性之间的张力：可扩展性不仅通过组合复杂性产生攻击面，还通过初始化顺序带来攻击面。

上下文效率与透明度。五层压缩流水线实现了有效的上下文管理，但压缩过程对用户而言基本是不可见的。当预算缩减用引用替代长工具输出，当上下文坍缩用摘要替换消息（在原文中描述为“对 REPL 完整历史的实时投影”），或当剪辑功能裁剪较早的历史记录时，用户无法轻松检查哪些内容已被丢失。微压缩 (microcompact) 的缓存感知行为进一步增加了不透明性，因为压缩决策受到提示缓存的影响，而这些影响对用户来说是不可见的。

简单性与可扩展性。四种扩展机制支持丰富的自定义，但也会产生组合交互。插件贡献了一个 PreToolUse 钩子，用于修改工具输入。自动模式分类器读取缓存的 CLAUDE.md 内容。路径作用域规则在读取新目录时延迟加载，可能在对话过程中改变分类器的行为。权限处理程序的四个分支在多个点与钩子流水线进行交互。这些横切关注点产生了难以从任一配置文件中预测的涌现行为。

³这两个预信任排序漏洞分别为 CVE-2025-59536 (CVSS 8.7) 和 CVE-2026-21852 (CVSS 5.3) (Donenfeld and Vanunu, 2026)，由 Check Point Research 发现。CVE-2025-54794 与 CVE-2025-54795 (Beber, 2025) 则分别利用了权限流水线中其他位置的路径验证和命令解析缺陷。上述四个漏洞在披露后数周内均已修复。

11.4 经验预测与早期信号

本文记录的架构特性生成了可测试的代码质量结果预测，这些预测无法仅从源代码中推导得出。有限的上下文窗口（Section 7）阻止智能体同时感知完整代码库：五层压缩流水线虽然保留了有用信息，但在每个阶段都引入了有损压缩。这在架构上预测出，智能体生成的代码将表现出比具备完整代码库可见性的代码更高的模式重复率和规范违反率。子智能体隔离（Section 8）进一步加剧了这一效应，其中每个子智能体在其独立的上下文窗口内运行，并拥有各自独立构建的工具池，导致并行智能体可以独立重新实现其他位置已存在的解决方案。Section 11.1 的设计哲学依赖模型做出良好的局部决策，但当模型缺乏全局上下文时，良好的局部决策可能导致糟糕的全局结果。

针对架构相似工具的已发表实证研究提供了与这些预测相符的数据。对 807 个代码仓库中 Cursor 的使用情况进行的因果分析 (He et al., 2025) 发现，代码复杂度出现了统计意义上的显著提升，其初始开发速度峰值在第三个月时衰减至基准水平；不断上升的复杂度与未来开发速度的成比例下降相关联，这表明这些收益是自我抵消的。⁴ 对 6,275 个仓库中 304,000 条由 AI 撰写的提交进行了大规模审计 (Liu et al., 2026)，发现了可度量的技术负债，约四分之一的由 AI 引入的问题持续存在于最新版本中，而与安全相关的问题则以显著更高的比例持续存在。尽管这些研究针对的是相近系统，但其架构上的相似性（限定上下文、工具使用环、单次生成）表明，这些发现对于此处分析的设计具有相关性。

Claude Code 的上下文管理流水线专门设计用于缓解这些影响：渐进式压缩保留了最新且最相关的上下文，缓存感知的紧凑化避免了在压缩过程中使提示缓存失效，读取时投影在保持完整历史以供重构的同时，向模型呈现压缩视图，子智能体摘要隔离则防止探索性噪声在父上下文中累积。这些机制是否足以克服有限上下文的结构限制，是一个可以直接测量的经验性问题，而本文的源码级分析无法解决这一问题。

11.5 局限性

除了 Section B.3 中的方法论局限性外，还存在若干分析上的约束。记忆化上下文组装函数 (`getSystemContext()` 和 `getUserContext()`) 均在 `context.ts` 中使用了 `lodash memoize`，这意味着 git 状态和 CLAUDE.md 内容被缓存而非每次对话轮次都重新计算。对话过程中的动态变化可能无法立即反映，尽管压缩操作可以清除缓存，而懒加载的路径作用域规则则提供了一种部分补偿机制。

特征标志创建编译时的可变性。在 `TRANSCRIPT_CLASSIFIER` 为 `false` 的构建中，整个自动模式分类器将被移除。使用特征门控的模块采用动态 `require()` 而非静态 `import`（例如，`query.ts` 用于上下文折叠），因为 `feature()` 仅在 `if/三元条件` 中有效，这是由于 `bun:bundle` 的树摇动约束所致。不同的构建目标可能会生成功能不同的应用程序。

11.6 新兴方向

该实现的多个方面涉及更广泛的设计问题。更长的上下文窗口可降低压缩压力，有望简化分级流水线。多模态工具（截图、图表、UI 预览）将拓展工具界面，并带来新的上下文相关挑战。对权限属性的形式化验证（例如，证明拒绝规则始终优先、沙箱化命令无法脱离隔离、恢复的会话无法继承过时权限）将提供更强的安全保障。

⁴复杂度 +40.7% ($p < 0.001$)；速度峰值在第一个月增长 281%，第三个月恢复至基准水平。

建筑解耦。此处分析的紧密耦合本地架构只是正在演进的谱系中的一个点。Anthropic 自身的托管智能体工作 (Martin et al., 2026) 描述了将智能体的各个组件（会话、框架、沙箱）虚拟化，使得“每个组件都成为一个接口，对其他组件的假设很少，并且可以独立地失败或被替换”，这明确类比于操作系统如何将硬件虚拟化为进程和文件。框架设计论文 (Rajasekaran, 2026) 从另一个角度提出了类似观点，指出“随着模型性能的提升，有趣框架组合的空间并不会缩小”；相反，“它在移动”。因此，本文所记录的架构应被视为一个共同演进系统的一个快照，而非固定的最佳方案。

内存作为一级子系统。Hu et al. (2025) 的记忆调查指出，智能体记忆正逐渐成为一种独立的认知基础，而非上下文窗口管理的副产品，并将自动记忆管理、基于强化学习的记忆以及可信记忆（隐私性、可解释性与幻觉鲁棒性）确定为开放的前沿领域。如今的 Claude Code 展示了事实层级（CLAUDE.md，自动记忆）和工作层级（对话窗口）；经验层级（从过往会话中学习并自动整理的策略手册集合）是自然的下一步，而上下文工程文献 (Zhang et al., 2025a) 已开始为此类积累提供机制。

可观测性与静默故障。行业调查显示，已部署智能体的主要失效模式并非崩溃，而是无声的错误。Bessemer 2026 年基础设施报告 (Wade et al., 2026) 估计，“78% 的人工智能故障是不可见的”，而 LangChain 针对 1,340 名受访者的智能体工程现状调查 (LangChain, 2026) 将质量列为生产环境中应用的首要障碍，而非成本，并发现可观测性（近 89% 的采用率）与离线评估（52.4%）之间存在巨大差距。此处分析的架构使操作者能够观察工具调用、钩子以及会话记录；要弥合评估差距，可能需要额外的支撑结构（生成器-评估器分离、冲刺合约，以及如 Rajasekaran (2026) 所讨论的事后检查），而不仅仅是模型改进。

治理。更广泛的治理趋势将限制智能体自主性增强后的设计空间。国际人工智能安全报告 (Bengio et al., 2026) 警告称：“由于智能体具有自主行动能力，使得人类在故障造成损害之前难以介入，因此带来了更高的风险”，而麻省理工学院人工智能智能体指数 (Staufer et al., 2026) 发现，仅 13.3% 的索引智能体系统发布了专门的智能体安全卡片。新兴的监管框架，尤其是将于 2026 年 8 月全面生效的欧盟人工智能法案，以及围绕人工智能生成代码的版权法理演变，可能会对日志记录、透明度和人工监督施加外部约束，从而影响编码智能体架构的演进方向。

主动架构。特征门控的 KAIROS 系统展示了该架构如何超越被动工具使用而演进。KAIROS 实现了一个具有基于时间步心跳的持久后台智能体：当没有用户消息待处理时，系统会注入周期性的 <tick> 提示，模型自行决定是否采取动作或进入休眠状态。该设计直接应对了已记录的矛盾：主动式 AI 助手可将任务完成率提高 12 至 18%，但在高频率下会降低用户偏好 (Chen et al., 2025)。KAIROS 通过终端专注度感知（用户离开时最大化自主行动，用户在场时提升协作）以及通过 SleepTool 实现的经济性限流（每次唤醒需消耗一次 API 调用；提示缓存将在五分钟无活动后过期，使休眠/唤醒成为明确的成本最优化操作）解决了这一问题。将主动性与用户在场状态及 token 经济相结合，在生产级智能体系统中较为罕见，尽管尚无法确认 KAIROS 在生产构建中实际运行。

11.7 重复的设计选择

综合阅读六个子系统分析，可以发现三个贯穿始终的设计承诺，这些承诺在原本相互独立的各个组成部分中反复出现。

分层渐进式叠加于单一机制之上。安全、上下文管理和可扩展性均采用分层的独立机制堆栈，而非单一的集成解决方案。权限架构包含七个层级，从工具预过滤到拒绝优先规则、权限模式、自动模式分类器、Shell 沙箱、恢复时不恢复状态，以及钩子拦截。上下文管理包含五个压缩层级，包括延迟加载的 CLAUDE.md 文件、延迟加载的工具模式、仅摘要的子智能体返回。可扩展性在不同上下文成本 (Section 6) 下分层部署四种机制 (MCP 服务器、插件、技能和钩子)。在每种情况下，设计都以牺牲简单性和可调试性为代价，换取纵深防御，接受各层级之间的交互可能产生难以从任何单一配置中预测的涌现行为。

仅追加设计，更注重可审计性而非查询能力。会话记录是以只追加方式存储的 JSONL 文件，并在读取时进行链式补丁；权限不会在会话边界间恢复；上下文压缩对完整历史记录应用读取时的投影，而非破坏性编辑。这种承诺被反复坚持，因为它能保留无需修改先前状态即可恢复、分叉和审计会话的能力。代价是，更丰富的结构化查询（“显示所有跨会话中修改文件 X 的工具调用”）需要事后重构，而非直接查找。

在确定性测试环境中的模型判断。在所有子系统中，架构信任模型在其丰富的确定性约束框架内的判断，而非对其选择进行限制。估算出的 1.6% 决策逻辑比例从量化角度体现了这一点：该约束框架创造了（工具路由、权限执行、上下文组装、恢复逻辑）条件，使模型能够做出良好决策。层级权限在智能体边界间保持了安全不变性，而 `assembleToolPool()` 将内置工具与 MCP 工具合并为单一统一接口，但模型仍保有对调用哪些工具以及调用顺序的完全自主权。这种权衡的代价是，当受限的上下文导致缺乏全局意识时，良好的局部决策可能产生较差的全局结果，正如 Section 11.4 文档中的实证预测所示。

12 未来方向

Section 11 读取 Sections 3 to 9 中记录的架构，将其作为一个连贯的设计点，并揭示了该设计点所蕴含的张力、权衡以及近未来的发展方向。本节超越了架构本身，记录了六个开放性问题，Section 11.6 部分指出了这些问题，而日益增长的外部文献已使这些问题足够明确，可以具体陈述。这六个问题涵盖了论文的五价值框架 (Section 2.1) 及其评估视角 (Section 2.4)：对权威层级的外部治理约束 (Section 12.5)；安全方面的可观测性—评估差距 (Section 12.1)；可靠性的跨会话状态与关系持久性 (Section 12.2)；能力边界四个扩展方向 (Section 12.3)；作为可靠执行中独立于跨会话连续性的缩放维度的远期缩放 (Section 12.4)；以及将 Section 2.4 的评估视角重新构想为一个设计问题而非诊断问题 (Section 12.6)。与 Section 11.6 的表述一致，每个问题均以“是否”/“如何”/“哪个”的形式提出；当引用文献指明具体机制时，会明确命名，否则保持开放。

12.1 静默故障与可观测性——评估差距

Section 11.6 中报告的可观测性—评估采用差距是否源于缺失的工具层、测试框架内部缺失的评估接口，或是模型能力的上限，这些来源并未予以解答。因此，如何揭示该段落中提到的静默错误故障模式，是测试框架的架构问题，而非模型的能力问题。近期的经验研究表明，该差距在多个层面得到了刻画。Cemri et al. (2025) 列出了涵盖系统设计问题、智能体间不对齐以及任务验证在内的十四种故障模式；Pathak et al. (2025) 构建了一个专门用于迹中异常检测的智能体轨迹基准；Yao et al. (2024) 通过 pass^k 指标（所有 k 个独立试验均成功的概率）暴露了一致性差距；Kapoor et al. (2024) 认为当前的智能体基准缺

乏保留集和成本控制，限制了可观测性实际能够诊断的内容。

针对在Sections 4 and 5中分析的权限流水线 and 工具编排层，仍有两个架构性问题尚未解决。第一，本文所援引的来自Rajasekaran (2026) 的框架（生成器-评估器分离、冲刺合约、事后检查，以及基于Madaan et al. (2023) 的自精炼模式）究竟应置于运行时环境内部（例如，作为与Section 6中所记载的 27 个已定义钩子事件并列的额外钩子事件），还是应置于其外部作为一个独立的评估层，这一问题在所引文献中并未明确。第二，现有Section 6的钩子流水线能否在其当前上下文开销预算内承载此类框架，是另一个尚未解决的问题。“弥合这一缺口‘很可能需要额外的框架……而不仅仅是模型改进’” (Section 11.6) 这一观察结果，将待解决的工作定位在运行时环境层。

12.2 持久性：记忆与纵向同事关系

智能体状态与人机协作关系是否应在会话间持续存在，以及以何种形式持续，本文在两个不同层次上进行了探讨。Section 7 描述了四层 CLAUDE.md 层次结构与自动记忆；Section 9 主要记录仅追加的 JSONL 会话日志（显式清理重写为例外），其会话范围的权限恢复无法恢复。介于这两层之间（既非静态指令，也非单一会话的记录）的持久化状态，仍是一个开放的设计问题。Hu et al. (2025) 和 Zhang et al. (2025a) 已在 Section 11.6 中引用，支持累积层的设计；Packer et al. (2023) 将大语言模型重新构想为具有分页内存的操作系统；Chhikara et al. (2025) 构建了一个可抵御重启的生产导向型记忆存储；Xu et al. (2025) 提出了一个研究性质的智能体记忆设计；Wang et al. (2024c) 捕获可复用的过程迹；Shinn et al. (2023) 通过多次尝试中的言语强化积累自我反思迹；Zhang et al. (2025b) 与 Huang et al. (2026) 的综述则映射了候选机制。

同样的持续性问题在人类一方也反复出现。Section 11.6 已经引用了纵向自主性的证据 (Huang et al. (2025), McCain et al. (2026)); Dell’Acqua et al. (2025) 对 776 名宝洁公司专业人士开展的实地实验，结合对 Copilot 推广的纵向研究与组织研究 (Stray et al., 2025) 以及人工智能团队协作轨迹的研究 (Xiao et al., 2025)，报告了随着协作积累，人机工作动态的变化。Wang et al. (2023) 展示了一个能够跨任务累积技能库的具身智能体；Mollick (2024) 将人机工作关系框架化为共智关系。

单个基底能否同时承载用户的个人指令层级与共享的组织上下文，同时保留Section 7所记录的 CLAUDE.md 的基于文件的透明性，这是一个开放性架构问题。会话级权限如何与这类基底交互，且不会重新引入Section 9作为审慎的安全选择所解决的会话恢复顾虑，这是另一个开放性问题。

12.3 边界演化：智能体行动的地点、时间、内容及对象

Section 11.6 引用了 Rajasekaran (2026) 的观察：“有趣的手 harness 组合空间并不会随着模型的改进而缩小；它会发生移动。”至于这种移动在以下哪一方面表现得最为显著——即 harness 运行的位置、发挥作用的时机、作用对象的内容，抑或与其协同工作的对象——Sections 3 to 9 中的源码级分析并未给出定论。这四个方面各自均拥有活跃的研究文献，而该论文仅略为提及。

在哪里。Martin et al. (2026) 的托管智能体设计将会话、支撑框架和沙箱虚拟化为可独立替换的接口，扩展了 Packer et al. (2023) 用于上下文窗口管理的虚拟内存类比，并由 Karpathy (2023) 更广泛地推广；Khatab et al. (2023) 将支撑框架本身视为编译目标。

当…… Section 11.6 已经将 KAIROS 作为一项特征门控的示例引入，其动机源于 +12% -18% 任务传递增益，该增益由 Chen et al. (2025) 报告，并且针对高频持续建议变体所施加的强烈偏好惩罚（47% 对比 80-90%）。Liu et al. (2025)、Pu et al. (2025) 和 Lee et al. (2025) 将主动性设计空间扩展至编程和环境接口情景；Pasternak et al. (2025) 和 Sun et al. (2025) 引入了旨在细化该设计空间的基准与训练方案，而 Deng et al. (2025) 则对更广泛的领域进行了综述。

什么？视觉-语言-动作 workflows 将工具返回的利用范围扩展至非文本内容：Brohan et al. (2024) 和 Black et al. (2024) 训练 VLA 策略以执行物理动作，Ahn et al. (2022) 将计划锚定在机器人可操作性上；工业系统如 Figure AI (2025) 和 Bjorck et al. (2025) 将类似思想推进至人形机器人控制。这些系统面临可逆性加权风险原则 (Table 1)，该原则指出了成本不对称性，但并未对非文本动作进行量化。与谁协作。角色差异化多智能体系统 (Hong et al. (2023), Li et al. (2023), Chen et al. (2023), Qian et al. (2024)) 组合具有不同职责的智能体；多智能体辩论 (Du et al., 2024; Liang et al., 2024) 与图结构工作流 (Zhuge et al., 2024) 探索了替代 Section 8 的父/子智能体模式的新路径；Guo et al. (2024) 对此领域进行了综述。

单一的牵引架构能否涵盖所有四个扩展，或者“牵引组合”Rajasekaran (2026) 描述的情况是否会分解为专门化的堆栈，仍然是一个开放的设计问题。*when* 扩展直接延续了 Table 4 中的能力与适应性之间的张力。*with-whom* 扩展部分映射到能力与可靠性之间，但引出了 Table 4 本身未涵盖的跨智能体一致性问题。*where* 和 *what* 扩展则提出了当前论文子系统边界尚未涵盖的进一步问题：当牵引组件成为托管服务时，哪些治理义务随之而来 (Section 12.5)，以及可逆性加权风险 (Table 1) 如何扩展至物理而非文本效应。这些扩展在不同维度上的组合方式，而非在任一维度内部的组合，是当前论文的单子系统分析无法解决的问题。

12.4 水平缩放：从会话到科学计划

Section 2.1 将可靠执行定义为涵盖“单轮正确性与长时程可靠性”。如何在自主工作超出单个会话范围的情况下，Sections 3, 4 and 7 to 9 中所描述的架构（其主要单元为轮次、会话和子智能体）仍能支持长时程可靠性，是一个开放性问题。越来越多的研究文献致力于解决这一领域。Lu et al. (2024) 提出了一种端到端的自主研究流水线，可生成草案论文；Beel et al. (2025) 对该流水线进行了独立的 SIGIR 论坛评估，描述了当前“自主研究”所能实现的能力以及其存在的不足之处。Gottweis et al. (2025) 开发了一个跨多日运行而非仅限于轮次的多智能体假说生成系统，而 Novikov et al. (2025) 则在以往需人类专家数周时间才能完成的时间尺度上开展算法发现。Kwa et al. 的 METR 研究测量了前沿智能体在固定可靠性水平下成功完成任务所需的时间（即 50%-时间阈值），并分析了该阈值随模型代际演进的变化，为这一缩放问题提供了实证框架。

与论文的分析相反，长时间跨度的部署测试了 Section 7 的上下文管理流水线、Section 8 的最后助手文本返回策略以及 Section 9 的仅追加持久化在会话组合成多会话程序时是否仍然足够。Section 11.4 已经将这一问题表述为“一个可以直接测量的经验性问题”，而源码层面的分析无法解决。时间跨度缩放重新以周为尺度提出了这一问题：仅靠支撑层是否足以弥补差距，是否需要跨会话记忆基础结构 (Section 12.2)，或者时间跨度工作是否需要超越会话、子智能体和记忆的协调原语，这些都不是论文中会话范围分析所能确定的。

12.5 大规模治理与监督

新兴的人工智能监管为实现 Anthropic 权威层级、操作员和用户架构的实现施加了外部约束，如 [Section 2.1](#) 所述。在该外部约束下，日志记录、透明度以及人工监督等支持功能应如何在编码智能体架构中暴露，仍是开放的设计问题。欧盟委员会的 GPAI 行业准则 ([European Commission, 2025a](#)) 与实施指南 ([European Commission, 2025b](#)) 详细说明了自 2026 年 8 月起欧盟人工智能法案全面适用时所伴随的一般用途人工智能义务；麻省理工学院人工智能智能体指数 ([Staufer et al., 2026](#)) 以及国际人工智能安全报告 ([Bengio et al., 2026](#)) (已在 [Section 11.6](#) 中引用) 则推动了该约束的披露与监督方面。Bartz v. Anthropic 判决 ([bar, 2025](#)) 在训练数据来源 (受版权保护作品的合法获取) 方面增加了输入端约束，这与新兴案件分别处理的输出端人工智能生成代码版权问题相区分。经合组织关于人工智能治理框架的报告 ([OECD, 2025](#)) 以及 [Nannini et al. \(2026\)](#) 对智能体提供者合规义务的早期分析，勾勒出面向监管者的接口可能形态，但未具体规定细节。

在 [Section 5](#) 中分析的权限流水线约束下，当前架构存在两个未明确的问题。首先，论文所描述的“拒绝优先”评估机制可通过会话记录进行内部审计 ([Section 9](#))，但尚未达到新兴框架如 GPAI 行为准则 ([European Commission, 2025a](#)) 所设想的外部审计形式。其次，论文将“价值优先于规则”原则与确定性护栏相结合，但这一原则是否能支持合规审查所需的那种明确规则表述，仍是另一个待解问题。这两个特性均存在于控制机制而非模型本身，因此未来架构可能需要在此处暴露新的接口。

12.6 评估视角再探：长期人类能力

[Section 2.4](#) 将长期人类能力保存作为分析视角，而非与设计价值并列的同等要素；[Sections 11.2 and 11.4](#) 通过外部证据扩展这一视角 (感知与实际生产力的差异、理解力损失、复杂性累积、技术债务持续存在、神经连接持久性、早期职业招聘下降)，而 [Section 14](#) 实现了视角转换：“未来系统可将这种可持续性差距视为首要的设计问题，而非后续的评估指标。”这种视角转换是否可行，以及首要处理该问题所需何种架构机制，是本节记录的最后一个开放问题。

两个子问题将测量差距与设计差距分离开来。首先，支撑该视角的实证主张是否可在会话粒度上进行测量。现有引用均在会话至数月尺度上运作 ([Becker et al. \(2025\)](#) 的 16 名开发者的随机对照试验，[Shen and Tamkin \(2026\)](#) 的理解测试对比，[Kosmyna et al. \(2025\)](#) 的脑电图研究，[He et al. \(2025\)](#) 的 807 个仓库因果分析，[Liu et al. \(2026\)](#) 的 304,000 提交审计，[Rak \(2025\)](#) 的招聘系列研究)，但 [Sections 3, 4 and 7](#) 中记录的工具链并未暴露任何关于理解或规范演变的会话级信号。关于程序员交互模式的研究 ([Barke et al., 2023](#)) 和由 AI 引发的代码安全回归的研究 ([Perry et al., 2023](#)) 初步勾勒了会话粒度的测量方法，而 [Aiersilan \(2026\)](#) 提出了用于会话级认知卸载探测的协议。其次，一旦此类测量存在，架构能否响应这些测量 (这类似于生成器—评估器分离 ([Rajasekaran, 2026](#)) 在人类环、保持理解的表面或尚未命名的机制中的类比) 是设计差距问题 [Section 14](#) 所提出的核心。本文不对何种机制类别更为合适做出立场表态，也不判断此处记录的工具链是否正是该动作的正确位置 (而非集成开发环境、组织或人类开发环)，这一问题超出了架构分析的裁决范围；[Section 13](#) 中综述的相关工作以及 [Section 14](#) 中的可持续性转向标志着本文在此问题上的立场终结。

Table 5 按自主动作程度划分的 AI 编码工具类别。

Category	Examples	Pattern
Inline completion	Copilot, Tabnine	Editor plugin
Chat-integrated	Cursor, Windsurf, Cody	IDE-coupled product
Agentic CLI	Claude Code, Codex CLI, Aider	Tool-use loop
Fully autonomous	Devin, SWE-Agent, OpenHands	Sandbox + planning

13 相关工作

13.1 编码智能体分类

AI 编码工具可以根据其支持的自主动作程度进行组织 (Table 5)。内联补全工具，如 GitHub Copilot (Chen et al., 2021)，在编辑器中建议代码片段，但不具有自主动作。集成聊天功能的产品，包括 Cursor 和 Windsurf，增加了对话交互和多文件编辑功能，但仍与 IDE 环境紧密耦合。具备代理特性的命令行工具，如 Claude Code、OpenAI 的 Codex CLI 以及 Aider (Gauthier, 2024)，从命令行运行，能够自主执行 shell 命令、读写文件，并在单个请求内对输出进行迭代。完全自主的系统，如 Devin、SWE-Agent (Yang et al., 2024) 以及 OpenHands (Wang et al., 2024b)，旨在实现最少的人类监督，通常在沙盒化的云环境中运行。

Claude Code 与高自主性智能体（自动模式分类器、后台智能体执行、远程环境）具有特征相似性，但默认情况下仍保留交互式审批。诸如 SWE-Bench (Jimenez et al., 2023) 和 HumanEval (Chen et al., 2021) 等评估基准推动了学术界对编码智能体的广泛关注。本文从源代码角度考察 Claude Code 的内部架构。

13.2 智能体架构模式

Claude Code 的核心环遵循 ReAct 模式 (Yao et al., 2022)：模型生成推理与工具调用，执行框架执行动作，结果将送入下一次迭代。Toolformer (Schick et al., 2023) 证实了语言模型可以学习使用工具；Claude Code 最多使用 54 个内置工具以及分层权限系统。更广泛的设计空间已被多项调研梳理清晰。Weng (2023) 提出了如今已成为标准的规划、记忆与工具使用的分解方式，而 Wang et al. (2024a) 整理了早期自主智能体的相关工作。Xu (2026) 将该领域围绕三个反复出现的权衡（自主性 vs. controllability，延迟 vs. 准确率、能力 vs. 可靠性）展开，这些权衡在我们的分析中反复出现，而 Hu et al. (2024) 将智能体设计本身视为针对组件、算法与评估函数的搜索问题。本文刻画了该空间中的一个特定点。

多智能体编排框架如 AutoGen (Wu et al., 2024)、LangChain 和 CrewAI 提供基于对话的智能体协调。Claude Code 的子智能体委派功能 (Section 8) 包含权限覆盖优先级、两级权限作用域以及每个子智能体独立的对话记录文件。LATS (Zhou et al., 2023) 在树搜索框架中统一了推理、行动与规划；而 Claude Code 的 plan 权限模式则实现了更简单的先规划后执行方法。

实践者写作已逐渐形成少数几个反复出现的模式，这些模式正是 Claude Code 架构所体现的。Anthropic 自身的《构建高效智能体》(Schluntz and Zhang, 2024) 区分了智能体与工作流，并主张采用简单可组合的模式而非复杂的框架。Martin (2026) 综合了在生产系统中观察到的七个模式，包括赋予智能体文件系统和外壳访问权限作为通用动作层，以及按需发现动作而非预先加载所有工具模式。Chase (2025)

Table 6 基于大模型工具的上下文管理方法的设计空间。

Approach	Mechanism	Granularity
Simple truncation	Drop oldest messages	Coarse
Sliding window	Fixed-size recent history	Medium
RAG	Retrieve relevant snippets	Fine
Single summarization	One-pass compress	Coarse
Graduated compaction	Multi-layer pipeline	Very fine

观察到 Claude Code 的规划工具“基本上是无操作”，其价值在于保持智能体的轨迹，而非执行任何外部计算。Wang (2025) 认为权威性是学术框架中最常被忽略的要素，将信任称为“生产环境中最被忽视的元素”，而 Section 5 中的权限分析试图弥补这一缺口。Huyen (2025) 将复合错误问题具体化：当每步准确率为 95% 时，一个 100 步的任务仅在 0.6% 的情况下成功，这促使我们追踪 Section 4 和 Section 5 中的每步验证模式。

上下文管理。Table 6 提出了上下文管理方法的设计空间分类。Claude Code 的五层压缩流水线在升级之前于不同粒度上应用多种策略，具备缓存感知的压缩和读取时的虚拟视图语义。Zhang et al. (2025a) 揭示了该设计所缓解的两种失效模式（摘要导致领域细节丢失，以及迭代上下文重写造成的细节损失），并提出将上下文视为一个“不断演进的行动指南”，随时间积累策略。Claude Code 的方法与此框架一致，因为 CLAUDE.md 层次结构累积的是结构化指令，而非反复进行摘要。Hu et al. (2025) 区分了上下文工程与智能体记忆：上下文工程处理临时组装，而记忆则涵盖持久的事实知识和经验迹。Claude Code 的架构以相同方式分离二者，将压缩流水线与基于文件的记忆层次结构配对。

安全与权限。生产编码智能体采用在三个维度上变化的安全架构：审批模型（每动作提示、分类器介导的自动化，或无提示但事后审查）、隔离边界（操作系统级容器、文件系统沙箱、权限范围工具池，或无隔离），以及恢复机制（版本控制回滚、会话范围权限重置，或基于检查点的回退）。SWE-Agent 和 OpenHands (Yang et al., 2024; Wang et al., 2024b) 主要依赖 Docker 容器隔离，提供环境级别的沙箱，限制所有智能体动作。Codex CLI 支持沙箱模式和针对 shell 命令的审批策略。Aider (Gauthier, 2024) 以 Git 作为主要安全机制，通过版本控制使所有更改可逆。Claude Code 结合每动作拒绝优先规则、基于机器学习的分类器实现自动化审批、可选的 shell 沙箱，以及会话范围权限不重置，采用多层机制而非依赖单一隔离边界。

协议与可扩展性。Claude Code 使用的模型上下文协议作为其主要的工具集成方式，已逐渐成为事实上的标准，拥有庞大的生态系统和相应的攻击面。Hou et al. (2025) 汇总了来自 26 个主要目录的数千个社区开发的 MCP 服务器，并将 MCP 特有的威胁归纳为四类攻击者类别和十六种场景，包括工具投毒、跑路攻击以及跨服务器影子化。Section 5 中分析的权限与拒绝规则机制，以及 Section 6.2 中的预过滤步骤，可被视为该综述所呼吁缓解措施的运行时层面。

软件架构 分层架构模式 (Garlan et al., 1993) 指导了我们的五层分解。基于角色的访问控制模型 (Sandhu et al., 2002) 为权限模式系统提供了理论基础。浏览器沙箱 (Reis and Gribble, 2009) 是一种类似的进程隔离方法。多智能体系统理论 (Wooldridge, 2009) 有助于解释子智能体委派。

定位。先前关于编码智能体的研究主要集中在基准（智能体解决任务的能力）、框架（如何组合智能体）和产品（用户能够执行的操作）上。本文通过对一个生产环境中的编码智能体进行基于源码的设计空间分析，利用源码级分析和架构比较，揭示了设计选择与权衡。该研究借鉴了软件架构案例研究的传统 (Garlan et al., 1993)，但将其应用于基于大语言模型的智能体，通过系统性地识别设计问题、映射替代方案，并对比 Claude Code 的设计选择与 OpenClaw——一个在不同部署环境下运行的独立人工智能智能体系统——之间的差异。

14 结论

本文表明，生产编码智能体可以被理解为对一系列反复出现的设计问题的回答：推理相对于约束机制的位置如何，执行、安全、可扩展性、上下文、委托和持久性如何组织，以及这些选择所蕴含的权衡。Claude Code 在这一设计空间中占据了一个明确的设计定位。它赋予模型广泛的局部自主权，同时在其周围构建了一个稠密的确定性约束机制，用于权限管理、工具路由、上下文压缩、可扩展性以及会话恢复。仔细阅读 Section 2 中提出的五个价值观和十三个设计原则，可以看出这些选择是连贯一致的，而非临时拼凑：该系统始终优先考虑人类决策权、安全性、可靠执行、能力增强以及情境适应性。

OpenClaw 的对比进一步明确了主要的架构发现：相同的设计问题在不同的智能体系统中反复出现，但产生了不同的解答。Claude Code 在 CLI 框架内投入于每动作的安全分类和渐进式上下文压缩，而 OpenClaw 则在多通道网关内投入于边界级访问控制和结构化长期记忆。这两个系统甚至可以组合使用：OpenClaw 通过 ACP 将 Claude Code 作为外部框架托管。对于智能体构建者而言，最核心的开放性问题：如何在时间推移中保持人类能力，而非如何增加更多自主性。正如 Section 2.4 中的评估视角、Section 11 中的分析以及 Section 12 中综述的开放性问题所记录的那样，该架构提供的机制有限，无法明确地保留长期的人类理解、代码库一致性或开发者流水线。未来系统可将这一可持续性差距视为首要的设计问题，而非次级的评估指标。

References

- Bartz v. Anthropic PBC, no. 3:24-cv-05417-WHA. U.S. District Court for the Northern District of California, Order on Motion for Summary Judgment (June 23, 2025), Alsup, J. Court docket: <https://www.courtlistener.com/docket/69058235/bartz-v-anthropic-pbc/>, 2025.
- Adversa.ai. Critical Claude Code vulnerability: Deny rules silently bypassed because security checks cost too many tokens. <https://adversa.ai/blog/claude-code-security-bypass-deny-rules-disabled/>, 2026.
- Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.
- Aizierjiang Aiersilan. The vibe-check protocol: Quantifying cognitive offloading in ai programming. *arXiv preprint arXiv:2601.02410*, 2026.
- Anthropic. Our framework for developing safe and trustworthy agents. <https://www.anthropic.com/news/our-framework-for-developing-safe-and-trustworthy-agents>, 2025a.
- Anthropic. Orchestrate teams of Claude Code sessions. <https://code.claude.com/docs/en/agent-teams>, 2025b.

- Anthropic. Claude code overview. <https://code.claude.com/docs>, 2026a. Official Claude Code documentation. Accessed April 12, 2026.
- Anthropic. Claude’s constitution. <https://anthropic.com/constitution>, 2026b.
- Anthropic. Anthropic on github. <https://github.com/anthropics>, 2026c. Verified GitHub organization page. Accessed April 12, 2026.
- Anthropic. How Claude Code works. <https://code.claude.com/docs/en/how-claude-code-works>, 2026d.
- Shraddha Barke, Michael B James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1):85–111, 2023.
- Elad Beber. InversePrompt: Turning claude against itself, one prompt at a time. <https://cymulate.com/blog/cve-2025-547954-54795-claude-inverseprompt/>, 2025. CVE-2025-54794, CVE-2025-54795; updated April 6, 2026.
- Joel Becker, Nate Rush, Elizabeth Barnes, and David Rein. Measuring the impact of early-2025 ai on experienced open-source developer productivity. *arXiv preprint arXiv:2507.09089*, 2025.
- Joeran Beel, Min-Yen Kan, and Moritz Baumgart. Evaluating sakana’s ai scientist: Bold claims, mixed results, and a promising future? In *ACM SIGIR Forum*, volume 59, pages 1–20. ACM New York, NY, USA, 2025.
- Yoshua Bengio, Stephen Clare, Carina Prunkl, Maksym Andriushchenko, Ben Bucknall, Malcolm Murray, Rishi Bommasani, Stephen Casper, Tom Davidson, Raymond Douglas, et al. International ai safety report 2026. *arXiv preprint arXiv:2602.21012*, 2026.
- Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.
- Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0 : A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.
- Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL <https://arxiv.org/abs/2307.15818>, 1:2, 2024.
- Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.
- Harrison Chase. Deep agents. LangChain Blog, <https://blog.langchain.com/deep-agents/>, 2025.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- Valerie Chen, Alan Zhu, Sebastian Zhao, Hussein Mozannar, David Sontag, and Ameet Talwalkar. Need help? designing proactive ai assistants for programming. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–18, 2025.
- Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2023.

- Boris Cherny and Cat Wu. Claude code: Anthropic’s agent in your terminal. Latent Space podcast, <https://www.latent.space/p/claude-code>, 2025.
- Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.
- Cursor. Cursor: The best way to code with AI. <https://cursor.com/>, 2026. Official product website. Accessed April 12, 2026.
- Fabrizio Dell’Acqua, Charles Ayoubi, Hila Lifshitz, Raffaella Sadun, Ethan Mollick, Lilach Mollick, Yi Han, Jeff Goldman, Hari Nair, Stewart Taub, et al. The cybernetic teammate: A field experiment on generative ai reshaping teamwork and expertise. Technical report, National Bureau of Economic Research, 2025.
- Yang Deng, Lizi Liao, Wenqiang Lei, Grace Hui Yang, Wai Lam, and Tat-Seng Chua. Proactive conversational ai: A comprehensive survey of advancements and opportunities. *ACM Transactions on Information Systems*, 43(3): 1–45, 2025.
- Aviv Donenfeld and Oded Vanunu. Caught in the hook: RCE and API token exfiltration through Claude Code project files. <https://research.checkpoint.com/2026/rce-and-api-token-exfiltration-through-claude-code-project-files-cve-2025-59536/>, 2026. CVE-2025-59536 (CVSS 8.7), CVE-2026-21852 (CVSS 5.3).
- Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*, 2024.
- David Dworken and Oliver Weller-Davies. Beyond permission prompts: Making Claude Code more secure and autonomous. Anthropic Engineering, <https://www.anthropic.com/engineering/claude-code-sandboxing>, 2025.
- European Commission. General-purpose AI code of practice. <https://digital-strategy.ec.europa.eu/en/policies/contents-code-gpai>, 2025a. Official EU Commission publication, 10 July 2025.
- European Commission. Guidelines on the scope of obligations for providers of general-purpose AI models under the AI act. <https://digital-strategy.ec.europa.eu/en/library/guidelines-scope-obligations-providers-general-purpose-ai-models-under-ai-act>, 2025b. Official EU Commission guideline document.
- Figure AI. Helix: A vision-language-action model for generalist humanoid control. <https://www.figure.ai/news/helix>, 2025. Figure AI technical blog.
- David Garlan, Mary Shaw, et al. An introduction to software architecture. *Advances in software engineering and knowledge engineering*, 1(3.4), 1993.
- Paul Gauthier. Aider: AI pair programming in your terminal, 2024. <https://github.com/Aider-AI/aider>. Open-source software, <https://aider.chat>.
- Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, et al. Towards an ai co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.
- Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.
- Hao He, Courtney Miller, Shyam Agarwal, Christian Kästner, and Bogdan Vasilescu. Speed at the cost of quality:

- How cursor ai increases short-term velocity and long-term complexity in open-source projects. *arXiv preprint arXiv:2511.04427*, 2025.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.
- Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions. *ACM Transactions on Software Engineering and Methodology*, 2025.
- Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.
- Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025.
- Saffron Huang, Bryan Seethor, Esin Durmus, Kunal Handa, Miles McCain, Michael Stern, and Deep Ganguli. How AI is transforming work at Anthropic. Anthropic Research Blog, <https://anthropic.com/research/how-ai-is-transforming-work-at-anthropic>, 2025.
- Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, et al. Rethinking memory mechanisms of foundation agents in the second half. *arXiv preprint arXiv:2602.06052*, 2026.
- John Hughes. Claude Code auto mode: A safer way to skip permissions. Anthropic Engineering, <https://www.anthropic.com/engineering/claude-code-auto-mode>, 2026.
- Chip Huyen. Agents. <https://huyenchip.com/2025/01/07/agents.html>, 2025.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.
- Sayash Kapoor, Benedikt Stroebel, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.
- Andrej Karpathy. [1hr talk] intro to large language models. YouTube talk, https://www.youtube.com/watch?v=zjkBMFhNj_g, 2023. November 2023; popularizes the LLM-as-OS framing.
- Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.
- Nataliya Kosmyna, Eugene Hauptmann, Ye Tong Yuan, Jessica Situ, Xian-Hao Liao, Ashly Vivian Beresnitzky, Iris Braunstein, and Pattie Maes. Your brain on chatgpt: Accumulation of cognitive debt when using an ai assistant for essay writing task. *arXiv preprint arXiv:2506.08872*, 4, 2025.
- Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, et al. Measuring ai ability to complete long software tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- LangChain. State of agent engineering. <https://www.langchain.com/state-of-agent-engineering>, 2026. Survey of 1,340 respondents conducted Nov-Dec 2025.
- LangChain, Inc. LangGraph: Build resilient language agents as graphs, 2024. <https://github.com/langchain-ai/langgraph>. GitHub repository.

- Geonsun Lee, Min Xia, Nels Numan, Xun Qian, David Li, Yanhe Chen, Achin Kulshrestha, Ishan Chatterjee, Yinda Zhang, Dinesh Manocha, et al. Sensible agent: A framework for unobtrusive interaction with proactive agents. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2025.
- Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for” mind” exploration of large language model society. *Advances in neural information processing systems*, 36: 51991–52008, 2023.
- Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 17889–17904, 2024.
- Xingyu Bruce Liu, Shitao Fang, Weiyan Shi, Chien-Sheng Wu, Takeo Igarashi, and Xiang’Anthony’ Chen. Proactive conversational agents with inner thoughts. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–19, 2025.
- Yue Liu, Ratnadira Widyasari, Yanjie Zhao, Ivana Clairine Irsan, and David Lo. Debt behind the ai boom: A large-scale empirical study of ai-generated code in the wild. *arXiv preprint arXiv:2603.28592*, 2026.
- Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.
- Lance Martin. Agent design patterns. https://rlancemartin.github.io/2026/01/09/agent_design/, 2026.
- Lance Martin, Gabe Cemaj, and Michael Cohen. Scaling managed agents: Decoupling the brain from the hands. Anthropic Engineering Blog, <https://www.anthropic.com/engineering/managed-agents>, 2026.
- Miles McCain, Thomas Millar, Saffron Huang, Jake Eaton, Kunal Handa, Michael Stern, Alex Tamkin, Matt Kearney, Esin Durmus, Judy Shen, Jerry Hong, Brian Calvert, Jun Shern Chan, Francesco Mosconi, David Saunders, Tyler Neylon, Gabriel Nicholas, Sarah Pollack, Jack Clark, and Deep Ganguli. Measuring AI agent autonomy in practice. Anthropic Research Blog, <https://anthropic.com/research/measuring-agent-autonomy>, 2026.
- MindStudio Team. What is the anthropic Claude Code source code leak? three-layer memory architecture explained. <https://www.mindstudio.ai/blog/claude-code-source-leak-three-layer-memory-architecture>, 2026.
- Ethan Mollick. *Co-intelligence: Living and working with AI*. Penguin, 2024.
- Luca Nannini, Adam Leon Smith, Michele Joshua Maggini, Enrico Panai, Sandra Feliciano, Aleksandr Tiulkanov, Elena Maran, James Gealy, and Piercosma Bisconti. Ai agents under eu law. *arXiv preprint arXiv:2604.04604*, 2026.
- Alexander Novikov, Ngân Vŭ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.
- OECD. Governing with artificial intelligence: The state of play and way forward in core government functions. https://www.oecd.org/en/publications/governing-with-artificial-intelligence_795de142-en/full-report.html, 2025. Official OECD Public Governance Committee report, 18 September 2025.
- Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

- Gil Pasternak, Dheeraj Rajagopal, Julia White, Dhruv Atreja, Matthew Thomas, George Hurn-Maloney, and Ash Lewis. Beyond reactivity: Measuring proactive problem solving in llm agents. *arXiv preprint arXiv:2510.19771*, 2025.
- Divya Pathak, Harshit Kumar, Anuska Roy, Felix George, Mudit Verma, and Pratibha Moogi. Detecting silent failures in multi-agentic ai trajectories. *arXiv preprint arXiv:2511.04032*, 2025.
- Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do users write more insecure code with ai assistants? In *Proceedings of the 2023 ACM SIGSAC conference on computer and communications security*, pages 2785–2799, 2023.
- Kevin Pu, Daniel Lazaro, Ian Arawjo, Haijun Xia, Ziang Xiao, Tovi Grossman, and Yan Chen. Assistance or disruption? exploring and evaluating the design and trade-offs of proactive ai programming support. In *Proceedings of the 2025 CHI conference on human factors in computing systems*, pages 1–21, 2025.
- Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 15174–15186, 2024.
- Prithvi Rajasekaran. Harness design for long-running application development. Anthropic Engineering Blog, <https://anthropic.com/engineering/harness-design-long-running-apps>, 2026.
- Gwendolyn Rak. How to stay ahead of AI as an early-career engineer. *IEEE Spectrum*, 2025. <https://spectrum.ieee.org/ai-effect-entry-level-jobs>.
- Charles Reis and Steven D Gribble. Isolating web programs in modern browser architectures. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 219–232, 2009.
- Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 2002.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.
- Erik Schluntz and Barry Zhang. Building effective agents. Anthropic Research, <https://www.anthropic.com/research/building-effective-agents>, 2024.
- Judy Hanwen Shen and Alex Tamkin. How ai impacts skill formation. *arXiv preprint arXiv:2601.20245*, 2026.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.
- Leon Staufer, Kevin Feng, Kevin Wei, Luke Bailey, Yawen Duan, Mick Yang, A Pinar Ozisik, Stephen Casper, and Noam Kolt. The 2025 ai agent index: Documenting technical and safety features of deployed agentic ai systems. *arXiv preprint arXiv:2602.17753*, 2026.
- Peter Steinberger and OpenClaw Contributors. OpenClaw: Personal AI assistant. <https://github.com/openclaw/openclaw>, 2026. Open-source multi-channel AI assistant gateway. MIT License.
- Viktoria Stray, Elias Goldmann Brandtzaeg, Viggo Tellefsen Wivestad, Astri Barbala, and Nils Brede Moe. Developer productivity with and without github copilot: A longitudinal mixed-methods case study. *arXiv preprint arXiv:2509.20353*, 2025.
- Yifan Sui, Han Zhao, Rui Ma, Zhiyuan He, Hao Wang, Jianxun Li, and Yuqing Yang. Act while thinking: Accelerating llm agents via pattern-aware speculative tool execution. *arXiv preprint arXiv:2603.18897*, 2026.

- Weiwei Sun, Xuhui Zhou, Weihua Du, Xingyao Wang, Sean Welleck, Graham Neubig, Maarten Sap, and Yiming Yang. Training proactive and personalized llm agents. *arXiv preprint arXiv:2511.02208*, 2025.
- The Linux Foundation. Linux foundation announces the formation of the agentic AI foundation (AAIF), anchored by new project contributions including model context protocol (MCP), goose and AGENTS.md. Linux Foundation Press Release, 2025. <https://www.linuxfoundation.org/press/linux-foundation-announces-the-formation-of-the-agentic-ai-foundation>.
- Janelle Teng Wade, Lance Co Ting Keh, Talia Goldberg, David Cowan, Grace Ma, Bhavik Nagda, Brandon Nydick, and Bar Weiner. AI infrastructure roadmap: Five frontiers for 2026. Bessemer Venture Partners, <https://www.bvp.com/atlas/ai-infrastructure-roadmap-five-frontiers-for-2026>, 2026.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024a.
- Shawn Wang. Agent engineering. Latent Space, <https://www.latent.space/p/agent>, 2025.
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024b.
- Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024c.
- Lilian Weng. LLM-powered autonomous agents. <https://lilianweng.github.io/posts/2023-06-23-agent/>, 2023.
- Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.
- Qing Xiao, Xinlan Emily Hu, Mark E Whiting, Arvind Karunakaran, Hong Shen, and Hancheng Cao. Ai hasn’t fixed teamwork, but it shifted collaborative culture: A longitudinal study in a project-based software development organization (2023-2025). *arXiv preprint arXiv:2509.10956*, 2025.
- Bin Xu. Ai agent systems: Architectures, applications, and evaluation. *arXiv preprint arXiv:2601.01743*, 2026.
- Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.
- John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

- Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025a.
- Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025b.
- Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

Appendix

A 包装结构

本附录展示了 TypeScript 包中各个部分在运行时的功能。

A.1 责任归属图

该软件包 (Figure 9) 围绕一个 `src/` 目录组织。Table 7 列出了构成主要子系统的关键文件。

Table 7 按大小和运行时职责划分的关键文件。

File	Size	Responsibility
<code>main.tsx</code>	804KB	Entry point, mode dispatch, setup
<code>query.ts</code>	68KB	Core agent loop, 5 context shapers
<code>QueryEngine.ts</code>	47KB	SDK/headless conversation wrapper
<code>Tool.ts</code>	30KB	Tool interface, types, utilities
<code>history.ts</code>	14KB	Global prompt history
<code>mcp/client.ts</code>	Large	MCP client (8+ transport variants)
<code>compact.ts</code>	Large	Compaction engine
<code>AgentTool.tsx</code>	Large	Agent tool, subagent dispatch
<code>runAgent.ts</code>	Large	21-parameter agent lifecycle

`tools/` 目录包含约 42 个子目录, 实现了各种工具, 每个子目录对应相应的模式、描述、权限要求和执行逻辑。`commands/` 目录包含约 86 个斜杠命令子目录。

关键服务目录包括 `services/tools/` (`StreamExecutor`、`toolOrchestration`、`toolExecution`)、`services/compact/` (压缩引擎) 以及 `services/mcp/` (MCP 客户端和配置)。权限基础设施涵盖 `utils/permissions/` (规则评估、分类器)、`hooks/useCanUseTool.tsx` (权限处理器)、`types/permissions.ts` (模式定义) 以及 `types/hooks.ts` (事件模式)。

一个结构上的特殊设计: `query.ts` (文件) 与 `query/` (目录) 共存。文件包含主查询环; 目录则存放用于环配置和上下文组装的辅助模块。

A.2 条件工具可用性

`getAllBaseTools()` 函数 (`tools.ts`) 根据模式、构建类型、环境和特征标志 (Table 8) 构造不同的工具集。模型在简单模式下可能仅看到 3 个工具 (`Bash`、`Read`、`Edit`)，而在启用所有特征的完整内部构建中则可能看到 40 个以上的工具。

A.3 文件间依赖关系

导入图包含以下依赖结构。`QueryEngine.ts` 将回合执行委托给 `query.ts`。`query.ts` 从 `services/tools/` (`StreamingToolExecutor`、`runTools`) 和 `services/compact/` (`autoCompact`、`buildPostCompactMessages`) 导入。`QueryEngine.ts` 从 `memdir/` 导入以处理记忆和提示组装。代码明确避免了循环依赖:

Table 8 条件性工具可用性类别。

Category	Examples
Always included	AgentTool, BashTool, FileReadTool, FileEditTool, FileWriteTool, SkillTool, WebFetchTool, WebSearchTool
Environment	GlobTool/GrepTool (unless embedded), ConfigTool (ant-only), PowerShellTool (Windows)
Feature flag	TaskCreate/Get/Update/List (todoV2), EnterWorktreeTool (worktree), TeamTools (swarms), ToolSearchTool
Null-checked	SuggestBackgroundPRTool, WebBrowserTool, RemoteTriggerTool, MonitorTool, SleepTool

`types/permissions.ts` 被提取出来以打破导入循环,且 `context.ts` 中的 `setCachedClaudeMdContent()` 通过权限/文件系统路径避免了循环。

B 证据基础与方法论

本附录描述了本研究的证据来源、分析程序以及认识论约束。

B.1 证据基础与证据层级

本文中的主张基于三个证据层级：

- **TIER A (产品文档)**：源自 Anthropic 官方文档和工程出版物的声明。这些声明确立了产品的设计意图，但可能不反映内部实现情况。
- **层级 B (代码验证)**：引用从公开可获取的 npm 包提取的 TypeScript 代码库（v2.1.88）中特定文件和函数的声明。这是最强的证据层级。
- **TIER C (重构)**：基于社区分析、OpenClaw 结构比对或代码模式推断得出的声明。这些声明使用了谨慎性语言进行表述。

语料库包含约 1,884 个文件，总计约 512,000 行 TypeScript 代码。使用 OpenClaw 进行校准，而非真实值。

B.2 设计空间分析方法

设计问题通过检查每个子系统中反复出现的选择点而识别出来，这些选择点在其他生产智能体中存在替代设计方案。Claude Code 对每个问题的回答通过特定源文件和函数实现进行了追溯（TIER B 证据）。五值框架（人类决策权、安全性、安全性与隐私、可靠执行、能力增强以及情境适应性）源自官方文档和创建者声明（TIER A），随后通过十三项设计原则追溯至架构决策。长期能力保留被单独作为评估视角处理，而非设计价值，因为其并未显著体现为架构或 Anthropic 声明的价值驱动因素（[Section 2.4](#)）。token 经济学作为一种贯穿全局的约束条件，同时限制所有五个价值，揭示了在共享资源压力下各个子系统选择之间的相互作用。

B.3 局限性

- **静态快照**。分析反映了一个版本（v2.1.88）。特征标志（例如，`TRANSCRIPT_CLASSIFIER`，`CONTEXT_COLLAPSE`）在构建时产生可变性；不同的构建目标可能会生成功能上不同的应用程序。

- **逆向工程认识论**。源代码揭示了实现的结构、控制流、依赖关系和特征开关。但它无法确认设计意图、启用的生产标志、运行时的普遍性或未发布的行为。
- **单系统分析**。研究发现描述的是 Claude Code 的设计空间，而非编码智能体的整体设计空间。结论具有局限性。
- **OpenClaw 快照**。OpenClaw 分析反映了一个特定的开发状态，可能无法代表其当前的功能。

Source Structure (v2.1.88)	Runtime Responsibility
Entry & Startup main.tsx replLauncher.tsx entrypoints/ cli/	Application entry point, mode dispatch, signal handlers Interactive REPL composition (components + screens) SDK & headless startup (coreTypes.ts, coreSchemas.ts) CLI argument handlers (agents, auth, mcp, plugins)
UI Layer components/, screens/ outputStyles/	Terminal UI building blocks (ink framework), screen composition System-prompt output style logic
Core Loop query.ts query/ QueryEngine.ts context.ts	Agentic query loop (queryLoop AsyncGenerator), 5 shapers Loop config helpers (context assembly is in context.ts) Headless/SDK conversation wrapper (delegates to query.ts) Context assembly (getSystemContext, getUserContext)
Tools & Commands Tool.ts tools/ services/tools/ commands/	Tool interface and types (execution lives in services/tools/) 42 concrete tool implementations (Bash, Read, Edit, Agent, ...) Tool execution and orchestration (not registration) 86 slash command implementations
Safety & Permissions utils/permissions/ types/permissions.ts hooks/useCanUseTool.tsx components/permissions/	Deny-first rule evaluation, yoloClassifier (auto-mode) 7 permission mode definitions (5 external + auto + bubble) Permission handler (coordinator, swarm, classifier, interactive) Permission dialog UI (PermissionDialog.tsx, per-tool prompts)
Extensibility plugins/ skills/ utils/hooks.ts types/hooks.ts + schemas/hooks.ts	Plugin loader, manifest validation, component registration Skill loader, SKILL.md frontmatter parsing, bundled skills Hook registry, lifecycle dispatch across 27 event types Hook schemas (Zod) + types (cmd/prompt/http/agent)
Context & Memory services/compact/ memdir/ utils/claudemd.ts state/	5-layer compaction (budget, snip, micro, collapse, auto) Auto memory loading, entry cap enforcement CLAUDE.md 4-level hierarchy, @include processing Runtime application state
Persistence history.ts utils/sessionStorage.ts	Global prompt history (history.jsonl, reverse-order reader) Per-session JSONL transcripts, sidechains, file-history
Services & Integration services/ remote/ coordinator/	MCP client (8+ transports), API adapters, LSP, analytics Remote execution backend support Multi-agent coordination mode, worker management
Additional Infrastructure bootstrap/, bridge/, constants/, server/ ink/, keybindings/, vim/, buddy/, ...	App init, WebSocket communication, API configuration Terminal rendering, input handling, optional features

Figure 9 提取的包结构映射到运行时职责。左栏：TypeScript 源码目录及关键文件。右栏：推断出的运行时角色。本附录代表重构分析（Tier C 证据），并非官方 Anthropic 文档。