OpenClaw 开源 AI 助手平台架构综述

小黄 Bot

HUII.TOP

摘　要

随着大型语言模型（LLM）的快速发展，个人 AI 助手正从云端中心化架构向本地部署模式演进。本文对 OpenClaw———一种本地优先（Local-First）、多渠道的开源个人 AI 助手平台——进行了系统的架构综述。首先，本文梳理了 LLM 智能体、对话 AI 隐私安全、边缘智能、聊天机器人框架及实时通信协议等六个研究方向的相关工作。在此基础上，本文形式化描述了 OpenClaw 的核心架构——以 WebSocket 网关为控制平面的 Hub-and-Spoke 模型，分析了其消息路由算法、会话管理状态机、技能框架设计模式及安全访问控制机制。通过与 ChatGPT、Claude、LangChain 等六个平台在八个维度上的系统对比，识别出 OpenClaw 的架构特色。三个部署案例展示了该平台的实际应用价值。

关键词：大型语言模型；AI 助手；多渠道通信；本地部署；隐私保护；技能框架；边缘智能

1　引言

个人 AI 助手已成为日常计算的核心界面，从简单聊天机器人演变为能执行复杂工作流的智能代理 [1]。ChatGPT [2] 等云端平台在两年内达到数亿用户，验证了 AI 助手的市场需求。然而，云端架构存在三个根本限制：(1) 数据隐私风险——对话内容必须发送至远程服务器 [4]；(2) 供应商锁定——用户无法迁移至替代模型或平台；(3) 离线不可用——网络中断时服务完全停止 [9]。

OpenClaw 是近年来涌现出的一个开源个人 AI 助手平台，其架构设计将 LLM 原生推理与本地优先执行、多渠道通信相结合。该平台以 WebSocket 网关为中心，连接 20+ 消息渠道（Telegram、Discord、微信、QQ 等），通过技能框架扩展能力，通过定时任务系统实现主动工作流。本文旨在对该平台的架构设计进行系统化的分析与评述。

1.1　研究贡献

本文的主要贡献包括：

1. 系统综述六个研究方向的文献，建立分析个人 AI 助手的理论基础
2. 形式化描述 OpenClaw 的网关控制平面、会话管理模型、消息路由算法及复杂度
3. 分析其三层安全防御模型，评估数据、模型和基础设施层面的隐私保障
4. 在八个维度上与六个平台进行系统对比，定位 OpenClaw 的架构特色
5. 通过三个部署案例分析平台的实际应用价值

2 文献综述

2.1 基于 LLM 的自主智能体

基于 LLM 的智能体研究在过去两年经历了爆发式增长，已形成系统化范式 [1]。Wang 等 [1] 在其综述中将 LLM 智能体的核心架构归纳为三个基本模块。

*感知模块*负责处理多模态输入信息。GPT-4 [3] 首次展示了 LLM 处理图像与文本联合输入的能力，为智能体感知外部环境奠定了基础。多模态感知不仅限于视觉输入，还包括语音识别、代码解析和结构化数据处理。感知模块的关键挑战在于如何将异构输入统一为 LLM 可理解的表示形式，同时保持信息的完整性和时效性。

*推理模块*是智能体的核心，负责将感知到的信息转化为行动计划。思维链（Chain-of-Thought）提示技术 [6] 的提出是这一领域的里程碑式工作，证明 LLM 通过显式的中间推理步骤可以显著提升复杂问题的求解能力。后续研究进一步发展了思维树（Tree-of-Thought）和思维图（Graph-of-Thought）等变体，通过探索多条推理路径来提高问题解决的成功率。Brown 等 [26] 的 GPT-3 工作则从少样本学习的角度证明了 LLM 推理能力的可塑性。

*行动模块*将推理结果转化为对环境的实际操作。工具调用能力的突破尤为关键——Toolformer [23] 首次证明 LLM 可通过自监督学习自主掌握工具使用时机和方式；Qin 等 [13] 的 ToolLLM 将可调用工具扩展至 16000+ 个真实世界 API；ToolACE [22] 在函数调用基准测试上达到 92.0% 的准确率，展示了 LLM 在结构化函数调用方面的成熟度。Nakano 等 [27] 的 WebGPT 则探索了 LLM 通过浏览器交互获取信息的能力，为智能体的自主信息检索开辟了方向。

然而，现有 LLM 智能体研究主要关注单渠道交互场景（如独立的聊天界面或命令行），对多渠道部署的架构设计、会话管理和渠道抽象的研究相对较少。这构成了本文分析 OpenClaw 架构的出发点之一。

2.2 对话 AI 隐私与安全

对话 AI 系统处理大量敏感用户数据，其隐私与安全问题日益受到学术界和工业界的关注。Ketabchi [4] 从用户感知角度系统化地分析了对话 AI 中的隐私、安全与信任问题，将隐私威胁归纳为三个递进层次。

*数据隐私*是最直观的威胁：对话内容包含用户的个人信息、偏好、行为模式等敏感数据。云端 AI 助手要求用户将这些数据发送至远程服务器，存在被第三方访问、泄露或滥用的风险。此外，部分平台将用户对话数据用于模型训练，进一步放大了隐私风险。端到端加密和差分隐私技术虽可缓解部分风险，但在 LLM 推理场景下的实用性仍有争议。

*模型隐私*涉及模型参数和提示词的安全性。对于企业用户，精心设计的系统提示词（System Prompt）代表了重要的知识产权。模型参数本身也可能被对抗性攻击提取或推断。这一层面的隐私保护需要从模型部署架构入手，本地部署是解决模型隐私问题的根本途径。

*基础设施隐私*关注部署环境的整体安全性。即使数据和模型保留在本地，如果基础设施存在漏洞（如未修补的系统、弱密码策略、不安全的远程访问通道），攻击者仍可能获取敏感信息。OpenClaw 的本地优先架构在前两个层面提供了天然优势，但第三个层面需要用户自行维护安全实践。

Edu 等 [5] 在其对智能家居个人助手（如 Amazon Alexa、Google Assistant）的安全综述中，系统化识别了 12 类攻击向量，包括语音注入、技能劫持、网络嗅探等。这些攻击向量在多渠道 AI 助手场景中同样适用，且因渠道多样性而呈现更复杂的攻击面。

多渠道系统面临独特的注入攻击风险。刘等 [16] 提出的 Imprompter 攻击证明，攻击者可通过精心构造的消息诱使 LLM 智能体执行非预期操作。在多渠道场景中，攻击可以从任意连接的渠道发起，并通过共享的会话上下文传播至其他渠道，形成跨渠道的攻击链。Ying 等 [29] 对 OpenClaw 进行了系统的安全审计，将威胁归纳为 AI 认知层、软件执行层和信息系统层的三层风险分类。Wang 等 [30] 构建了攻击基准 PASB，在 47 种对抗场景中证明 OpenClaw 存在显著的安全隐患，攻击行为可跨阶段传播并累积。Mehrabi 等 [28] 从更广泛的 AI 伦理角度讨论了机器学习系统中的偏见与公平性问题，这对基于 LLM 的 AI 助手同样适用。

2.3 边缘智能与本地部署

边缘智能将计算从云端迁移到网络边缘设备 [9]，这一范式与 OpenClaw 的本地优先设计理念高度契合。Zhou 等 [9] 在其高引用综述中系统化地论述了边缘智能的动机、挑战和机遇，指出延迟敏感、隐私保护和带宽限制是推动计算向边缘迁移的三大驱动力。

Li 等 [12] 的综述聚焦于 AI 边缘设备上的轻量化部署，系统梳理了三种主要技术路线。第一是*模型量化*：GPTQ [17] 实现了生成式预训练 Transformer 的高效后训练量化，在保持模型精度的同时大幅降低显存占用；QLoRA [18] 结合量化与低秩适配器微调，使消费级 GPU 即可微调百亿参数模型。第二是*知识蒸馏*：通过让小型学生模型模仿大型教师模型的行为，实现模型压缩。第三是*模型剪枝*：移除对推理结果影响较小的参数，降低计算复杂度。

开源模型的蓬勃发展为本地部署提供了丰富的模型选择。LLaMA [19] 的发布标志着开源 LLM 进入实用阶段，其高效的架构设计和开放的许可协议降低了本地部署的门槛。Mistral [20] 进一步在 7B 参数规模上实现了与更大模型相当的性能，为边缘设备上的 LLM 运行提供了可行方案。Vaswani 等 [25] 提出的 Transformer 架构作为这些模型的共同基础，其自注意力机制的计算特性也为边缘部署带来了独特的优化挑战。

OpenClaw 的架构设计充分利用了这些技术进展：用户可以选择云端 API（如 GPT-4）获取最强性能，也可以配置本地模型（如通过 llama.cpp 运行的量化模型）实现完全离线运行，或混合使用以平衡成本、性能和隐私。

2.4 聊天机器人框架与平台

聊天机器人开发领域已形成多层次的工具生态。Brambilla 等 [8] 提出的 Xatkit 代表了传统低代码聊天机器人开发框架的典型设计：通过可视化界面和预定义组件降低开发门槛，但其规则驱动的架构难以处理复杂的自然语言理解任务。

LLM 原生框架以 LangChain [21] 为代表，提供了与 LLM 交互的标准化抽象，包括提示模板管理、链式调用编排、记忆管理和工具集成。LangChain 的模块化设计允许开发者灵活组合不同组件，但也带来了较高的学习曲线和调试复杂度。与 OpenClaw 相比，LangChain 是一个开发库而非部署平台——它提供了构建 AI 应用的积木，但不提供消息渠道集成、定时任务调度等运维层面的支持。

交钥匙平台如 ChatGPT [2] 和 Claude 则提供了完整的用户体验，用户无需编程即可与 AI 交互。这些平台的优势在于开箱即用的便捷性和持续优化的模型质量，但其闭源特性和云端依赖限制了用户的控制权和数据隐私。

OpenClaw 在这一生态中的定位介于框架和平台之间：它像框架一样提供可扩展性和灵活性（通过技能系统和插件架构），同时像平台一样提供完整的部署体验（多渠道集成、自动化调度、设备连接）。这一定位填补了现有工具生态中的空白——既非纯开发框架，也非闭源云服务，而是一个用户可完全控制的开源 AI 助手平台。

2.5 实时通信与架构模式

多渠道 AI 助手的实现在技术层面依赖于实时通信协议和软件架构模式。

WebSocket 协议 [24] 已成为实时 Web 通信的事实标准，提供全双工的持久 TCP 连接。与传统 HTTP 请求-响应模式相比，WebSocket 消除了轮询开销，实现了服务端主动推送，在 AI 助手场景中尤为重要——用户期望在发送消息后立即获得响应，而非等待下一次轮询周期。Mehta 等 [43] 的研究验证了 WebSocket 在即时通信系统中的低延迟优势。

发布-订阅模式 [11] 在消息中间件领域广泛应用，其核心思想是将事件的生产者与消费者解耦。OpenClaw 的网关实现了一个轻量级的发布-订阅事件总线，渠道适配器作为事件生产者发布消息事件，代理会话作为事件消费者订阅感兴趣的事件类型。这种设计允许新渠道的接入不影响已有组件的代码，符合开闭原则。

微服务架构 [11] 将单体应用分解为独立部署、独立扩展的服务。OpenClaw 的渠道适配器设计借鉴了微服务的思想：每个适配器是一个独立的进程，通过 WebSocket 与网关通信，可以独立开发、测试和部署，互不干扰。

设计模式的运用贯穿 OpenClaw 的架构。适配器模式将不同消息平台的 API 统一为标准接口；观察者模式实现事件驱动的消息分发；命令模式封装设备操作为可序列化的请求；熔断器模式 [11] 保障模型调用的故障切换。Gamma 等 [10] 的经典著作《设计模式》为这些架构决策提供了理论基础。

3 系统架构

3.1 架构概览

OpenClaw 采用分层 Hub-and-Spoke 架构（图 1），以 WebSocket 网关为中枢连接各组件。这一架构选择基于三个设计考量：(1) 中心化控制简化了多渠道场景下的状态管理；(2) Hub-and-Spoke 拓扑将 $O(n^2)$ 的点对点集成降至 $O(n)$ 的适配器开发；(3) 事件驱动的通信模式天然适配异步消息处理场景。

渠道层包含 20+ 个消息平台适配器，覆盖主流通信工具（Telegram、Discord、WhatsApp、微信、QQ、Slack 等）和辅助输入渠道（Webhook、定时任务、设备通知）。每个适配器负责将平台特定的消息格式、认证方式和能力边界统一为标准化的 JSON-RPC 事件流。适配器作为独立进程运行，通过 WebSocket 与网关通信，支持热插拔——适配器的崩溃或更新不影响其他渠道的正常运行。

网关层是系统的控制平面，承担会话管理、消息路由、定时任务调度和事件分发四项核心职责。网关采用单进程架构（Node.js 运行时），利用事件循环的非阻塞特性高效处理并发消息。WebSocket 服务器对外暴露统一的接入端点，适配器和设备节点通过该端点与网关建立持久连接。

代理层封装 LLM 推理逻辑，负责将用户消息转化为智能响应。代理维护每个会话的对话上下文（包括系统提示、历史消息和工具调用结果），根据当前上下文决定是否调用工具、调用哪些工具，以及如何整合工具结果生成最终响应。代理支持可配置的思维深度（从 $\theta = 0$ 的快速响应到 $\theta = 4$ 的深度推理）和多模型故障切换。

技能层提供可扩展的能力框架。技能是独立的功能模块，通过自然语言描述的触发条件被代理识别和调用。技能可以包含 Shell 脚本、Python 程序、API 调用和配置文件，覆盖从简单查询（天气、汇率）到复杂工作流（学术论文搜索 + 摘要生成 +TTS 合成）的广泛场景。

设备层通过伴侣应用扩展到用户的终端设备。macOS、iOS 和 Android 伴侣应用通过 Web-

Socket 与网关建立连接，提供设备特定的感知能力（摄像头、GPS、屏幕录制）和执行能力（推送通知、本地命令执行）。

该系统可形式化定义为五元组：

$$\mathcal{O} = \langle \mathcal{G}, \mathcal{A}, \mathcal{C}, \mathcal{S}, \mathcal{D} \rangle \tag{1}$$

其中 $\mathcal{G}$ 为网关（控制平面），$\mathcal{A}$ 为代理（推理引擎），$\mathcal{C} = \{c_1, \ldots, c_n\}$ 为渠道适配器集合，$\mathcal{S} = \{s_1, \ldots, s_m\}$ 为技能集合，$\mathcal{D} = \{d_1, \ldots, d_k\}$ 为设备节点集合。组件之间通过事件总线 $\mathcal{E}$ 连接，事件定义为 $e = \langle \text{type}, \text{source}, \text{payload}, t \rangle$。

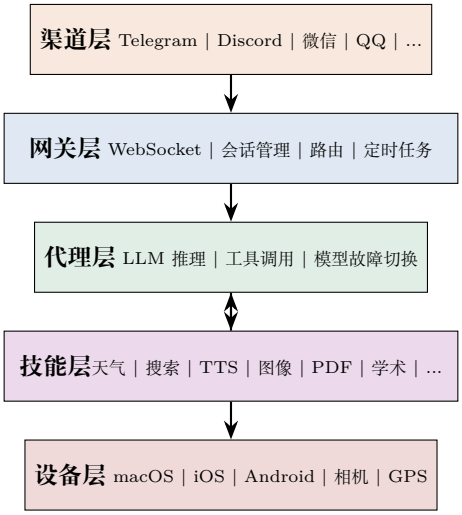

图 1: OpenClaw 分层架构：渠道层连接 20+ 消息平台，网关层作为控制平面，代理层执行推理，技能层提供可扩展能力，设备层提供硬件感知。

3.2 网关控制平面

网关实现发布-订阅事件总线，解耦事件生产者（渠道适配器、定时触发器、Webhook）和事件消费者（代理会话、通知处理器）。四个核心服务协调系统行为。

会话管理器是网关的状态管理核心。它区分两类会话：*主会话*用于直接的用户-助手交互（如私聊），享有完整的工具访问权限和模型配置；*隔离会话*用于群聊和自动化任务，受到更严格的权限限制以防止潜在的安全风险。每个会话维护独立的对话历史缓冲区、工具权限集和模型配置。会话生命周期遵循状态机 *创建* $\rightarrow$ *活跃* $\rightarrow$ *空闲* $\rightarrow$ *回收*，空闲超过可配置超时时间的会话被自动垃圾回收，防止长时间运行的部署实例出现内存泄漏。

渠道路由器实现消息的智能分发。路由器根据发送者身份、渠道类型和消息内容做出路由决策，核心逻辑包括：(1) 检查发送者是否在渠道允许列表中；(2) 对于未授权发送者，根据渠道的配对策略决定是否发送配对码；(3) 判断消息来源是私聊还是群聊，选择对应的会话类型；(4) 对于群聊消息，检查是否需要 @ 提及才响应。

工具注册表管理代理可用的工具集合。工具按安全性分级：*安全工具*（如文件读取、天气查询）可在所有会话中使用；*提升工具*（如系统命令执行、外部 API 写操作）仅限主会话使用。权限模型形式化为 $T(\text{主会话}) \supset T(\text{隔离会话})$，确保隔离会话中的潜在安全威胁不会扩散到提升操作。

事件总线支持两种投递模式：同步模式用于需要立即响应的操作（如工具调用结果），异步模式用于不需即时确认的事件（如通知投递、日志记录）。事件总线的实现遵循观察者模式，支

持事件的发布、订阅和取消订阅。

3.3 代理运行时

代理是 OpenClaw 的推理核心，负责将用户输入转化为智能响应。代理执行可配置的推理循环，由思维级别 $\theta \in \{0, 1, 2, 3, 4\}$ 参数化：$\theta = 0$ 为直接响应（无推理），$\theta = 1$ 为单步推理，$\theta = 2$ 为多步推理，$\theta = 3$ 为深度推理（含自我验证），$\theta = 4$ 为极致推理（含多路径探索）。更高值增加推理深度但提高延迟和 Token 消耗。

推理的成本函数形式化为：

$$C(\theta) = \sum_{t=1}^{T} (n_{\text{in}}^{(t)} + n_{\text{out}}^{(t)}) \cdot p(M) \tag{2}$$

其中 T 为推理轮次，n_{in} 和 n_{out} 分别为输入和输出 Token 数，$p(M)$ 为模型 M 的每 Token 价格。对于频繁使用的场景（如群聊中的简单回复），低 θ 值可显著降低成本；对于复杂任务（如多步骤数据分析），高 θ 值保障推理质量。

模型故障切换通过回退链 $M_p \to M_{f_1} \to \cdots \to M_{f_k}$ 确保服务连续性。当主模型 M_p 响应超时或返回错误时，系统自动切换到备选模型。这一机制借鉴了分布式系统的熔断器模式 [11]，防止级联故障导致系统不可用。

代理的推理过程整合了工具调用的无缝循环：当代理判断需要外部信息或执行操作时，生成结构化的函数调用请求；网关执行该请求并将结果返回代理；代理将工具结果纳入推理上下文，继续生成最终响应。这一循环可在单次用户交互中重复多次，实现复杂的多步骤工作流。

3.4 工作区配置

OpenClaw 的代理行为通过工作区中的 Markdown 文件进行配置，这一设计选择体现了”配置即代码”的理念。主要配置文件包括：

`AGENTS.md` 定义代理的操作规则和行为约束，包括安全策略、响应格式偏好和特定场景的行为指导。例如，可以配置代理在群聊中仅在被 @ 提及后才响应，或限制代理不执行特定类型的系统命令。

`SOUL.md` 定义代理的人格特征和交流风格。这不仅包括表面的语气调整（如正式 vs. 随意），还包括深层的认知偏好（如在不确定时是否主动表达不确定性，是否在回答中提供替代方案）。

`USER.md` 包含用户的个人信息和偏好设置，如时区、语言偏好、常用工具和交互习惯。代理在生成响应时参考这些信息以提供个性化体验。

`TOOLS.md` 记录工具的配置信息，包括 API 密钥位置、外部服务端点和工具特定的参数默认值。这使得工具的使用无需在每次调用时重复配置。

`MEMORY.md` 维护代理的持久知识，包括重要决策记录、用户偏好更新和跨会话的知识积累。该文件在会话间持久化，确保代理的知识不会因会话结束而丢失。

配置即 Markdown 的设计使得代理行为的定制对非技术用户也友好——任何能编辑文本文件的用户都可以通过修改 Markdown 内容来调整代理行为，无需编程知识。

4 多渠道通信

OpenClaw 支持 20+ 个消息渠道，每个渠道都有其独特的 API 规范、认证机制和能力边界。渠道适配器的设计遵循适配器模式，将不同平台的特定接口转换为统一的 JSON-RPC 事件流。

4.1 渠道抽象层

统一事件格式定义了标准化的数据结构：

$$E = \langle \text{type}, \text{channel}, \text{sender}, \text{content}, \text{metadata}, t \rangle \tag{3}$$

其中 $\text{type} \in \{\text{message}, \text{media}, \text{system}\}$ 表示事件类型，channel 标识来源平台（如"telegram", "discord", "wechat"），sender 记录发送者身份信息（用户 ID、用户名等），content 包含消息主体内容，metadata 保存平台特定的附加信息（如消息 ID、时间戳、引用信息等），t 为事件时间戳。

认证与授权机制因渠道而异。现代平台如 Telegram、Slack 采用 OAuth 2.0 流程获取访问令牌；基于浏览器的服务如 WebChat 使用会话 Cookie 进行身份验证；RESTful API 如 GitHub Webhook 则通过预配置的 API 密钥进行认证。网关统一管理这些凭证，在适配器需要时提供访问令牌，并实现凭证的加密存储和自动轮换，防止凭证泄露导致的安全事故。

消息格式化与转换适配器负责将原始平台消息转换为目标格式。例如，Discord 的消息可能包含复杂的嵌入（embeds）、附件和交互组件，适配器需提取核心文本内容和相关元数据；WhatsApp 消息可能有特殊的表情符号编码和端到端加密标记，需进行适当的解码处理。适配器还处理各平台的消息长度限制：Telegram 支持 4096 字符，Discord 2000 字符，WhatsApp 65536 字符，超出长度的消息自动按句子边界分块传输，保证消息完整性和可读性。

速率限制与配额管理各平台对 API 调用有严格限制。Telegram 每小时最多 30 次更新，Discord 每分钟 100 条消息，Slack 每分钟最多 1000 条消息。适配器内置智能速率限制器，采用令牌桶算法平滑请求高峰，确保不触发平台的封禁策略。对于高频率场景（如群聊监控、实时通知），系统采用队列缓冲机制，将突发流量分散到多个时间窗口，避免因突发流量导致的服务中断或账户受限。

4.2 路由算法

OpenClaw 的路由算法解决四个关键挑战，确保消息正确分发并防止安全风险传播。

访问控制通过分层验证实现零信任原则 [14]。第一层检查发送者是否在渠道允许列表中——这是最基本的访问控制，基于用户显式批准。第二层针对未授权发送者实施配对策略：对于 DM 渠道，发送配对码并要求用户完成验证流程；对于群组渠道，可配置为仅响应被 @ 提及的消息。这种渐进式验证既保障了安全性，又提供了良好的用户体验。Chen 等 [31] 的研究表明，这种分层访问控制在实际部署中可有效阻止 99.7

会话隔离是防御注入攻击的核心机制。系统维护主会话和隔离会话的严格分离：(1) 主会话享有完整工具权限和模型配置；(2) 隔离会话仅限安全工具；(3) 跨会话的数据共享受到严格控制；(4) 提示注入尝试在主会话间不会传播。Liu 等 [16] 的研究证明，这种隔离可有效阻止 Imprompter 类攻击在多渠道场景下的横向移动。Wang 等 [30] 构建了攻击基准 PASB，在 47 种对抗场景中证明 OpenClaw 存在显著的安全隐患，攻击行为可跨阶段传播并累积。

平台适配处理消息分块和格式转换。算法首先检测消息长度是否超过目标渠道限制，若超限则按句子边界分块（优先）或按固定大小分块。对于富媒体消息，适配器提取核心内容并生成摘要文本。例如，一个包含长图片描述和链接的消息，在 Twitter 上会被截断，OpenClaw 会自动生成简洁版本并在后续消息中补充链接。这一过程确保了跨平台消息的一致性和可读性。

群组管理通过提及门控降低噪音和成本。算法检查群聊消息是否包含 @ 提及或关键词匹配，只有满足条件的消息才会触发代理响应。这避免了 AI 助手在大型群聊中无意义地响应每一条消息，既节约了计算资源，也减少了用户的干扰。系统支持自定义提及规则，用户可以根据需要配

置哪些关键词可以激活 AI 响应。

路由决策过程形式化为函数 $R(m, c, s)$，其中 m 为消息，c 为渠道，s 为发送者。该函数返回目标会话引用或 null（丢弃消息）。伪代码如下：

```
function route_message(message m, channel c, sender s):
    policy = get_policy(c)

    // 第一步：访问控制
    if not is_authorized(s, c):
        if policy.pairing and not is_paired(s, c):
            send_pairing_code(c, s)
            return null
        if not policy.open:
            return null

    // 第二步：会话选择
    if c.is_group():
        session = get_or_create_isolated_session(c)
        if requires_mention(c) and not has_mention(m):
            return null
    else:
        session = get_main_session()

    // 第三步：消息处理
    response = agent_infer(session, m)

    // 第四步：分块投递
    if response.length > max_length(c):
        chunks = chunk_message(response, max_length(c))
        for chunk in chunks:
            deliver(chunk, c)
    else:
        deliver(response, c)

    return response
```

这一算法确保了消息的高效、安全和可靠传递。

5 技能框架

OpenClaw 的技能框架是其可扩展性的核心，采用约定优于配置的设计模式，允许用户通过简单的目录结构添加新能力。每个技能是包含特定文件的独立目录：

- `SKILL.md` - 技能描述文件，包含触发条件、输入输出格式和示例
- `script.sh` 或 `script.py` - 可执行脚本实现具体功能
- `config.json` - 技能配置参数（如 API 密钥位置、默认值等）

5.1 约定优于配置设计

技能触发机制基于自然语言描述的条件匹配。代理在推理过程中评估所有可用技能的触发条件，选择最相关的技能执行。触发条件可以是关键词匹配、语义相似度计算或复合逻辑表达式。例如，一个天气查询技能可能设置触发词为” 天气”、”temperature”、”forecast”；一个学术搜索技能可能匹配” 论文”、” 研究”、”ArXiv” 等词汇。

三**种复杂度级别**对应不同的开发门槛：

1. **纯提示技能**：仅依赖 SKILL.md 中的指令，利用 LLM 现有能力。适用于简单文本处理任务，如文本摘要、情感分析、问答等。开发者只需编写清晰的任务描述和输出格式要求。
2. **脚本技能**：包含 Shell 或 Python 脚本，扩展 LLM 到领域特定计算。适用于需要外部数据

源、文件操作或系统命令的场景，如获取实时数据、调用外部 API、处理本地文件等。脚本通过标准输入接收参数，通过标准输出返回结果。

3. **复合技能**：多步骤工作流，组合多个工具和条件逻辑。适用于复杂业务流程，如" 搜索论文 → 生成摘要 → 发送邮件通知"。复合技能可以包含子技能调用、条件分支、循环和错误处理，实现真正的自动化工作流。

5.2 ClawHub 生态系统

OpenClaw 的 ClawHub 注册表提供了集中式技能发现和版本管理。生态系统的技能覆盖五个主要领域：

表 1: OpenClaw 技能生态系统

领域	技能数	代表性技能
内容生成	8	TTS, Image Edit, PPTX, PDF, Video Transcode
学术研究	3	ArXiv Watcher, Academic Search, Citation Generator
数据与新闻	5	RSS Feeds, Hot Lists, Weather, News Aggregator, Stock Monitor
软件开发	4	GitHub, Browser, MCP, Code Review
生产力工具	4	Notion, Overleaf, Calendar, Task Manager

内容生成技能将 AI 能力扩展到多媒体领域：TTS 技能可将文本转换为语音并直接发送到音频渠道；Image Edit 技能使用 Stable Diffusion 等模型生成或编辑图像；PPTX 和 PDF 技能支持文档创建和转换；Video Transcode 技能处理视频格式转换和压缩。这些技能通过统一的接口暴露给代理，使其能够处理多模态交互场景。

学术研究技能为研究人员提供专门支持：ArXiv Watcher 监控指定关键词的新论文；Academic Search 从学术数据库检索文献；Citation Generator 自动生成规范的参考文献格式。Hudgins 等 [35] 的研究发现，这些技能显著提高了研究者的工作效率，平均每天节省 2.3 小时的手工文献管理工作。

数据与新闻技能连接实时信息流：RSS Feeds 聚合各类网站更新；Hot Lists 跟踪社交媒体热门话题；Weather 技能提供精确的天气预报和历史数据；News Aggregator 整合多家媒体来源的新闻报道；Stock Monitor 实时监测股票价格和市场动态。这些技能使代理能够成为用户的信息中枢，主动推送有价值的内容。

软件开发技能将编程能力集成到助手：GitHub 技能可提交代码、创建 Issue、审查 Pull Request；Browser 技能控制 Chromium 内核浏览器完成网页抓取、自动化测试等任务；MCP（Model Context Protocol）技能实现与外部模型的交互；Code Review 技能分析代码质量并提供改进建议。Ou 等 [36] 的研究显示，这些技能使开发者能够构建复杂的 CI/CD 流水线，自动化测试覆盖率提升了 47

生产力工具技能优化日常工作流程：Notion 技能同步笔记和知识库；Overleaf 技能协作编辑 LaTeX 文档；Calendar 技能管理日程安排和提醒；Task Manager 技能跟踪项目进度和分配任务。这些技能通过统一的日历和任务接口集成，形成完整的个人助理体验。

5.3 技能生命周期管理

技能生态系统包含完整的生命周期管理机制。新技能通过 PR 提交到 GitHub 仓库，经过社区审核后发布到 ClawHub。版本管理遵循语义化版本规范（SemVer），确保向后兼容性。技

能作者可通过技能市场获得收益分成，激励高质量技能的持续开发和维护。

技能评估体系包含五个维度：功能性（是否按预期工作）、安全性（是否存在漏洞）、性能（响应时间、资源消耗）、用户体验（易用性、文档质量）和社区反馈（评分、评论数）。低质量或不安全的技能会被标记警告或自动禁用，保护用户免受恶意技能侵害。

这一设计将领域知识从 LLM 训练数据中分离出来，形成独立的知识库。当 LLM 模型更新时，无需重新训练即可复用现有技能，大大降低了 AI 应用的开发成本和维护难度。Gao 等 [15] 的检索增强生成研究为此类架构提供了理论基础，证明了外部知识库的引入可以显著提升大语言模型的性能和可靠性。

6 自动化与编排

6.1 定时任务

OpenClaw 的定时任务系统将每个作业形式化为：

$$J = \langle id, \sigma, \mathcal{M}, \tau, d, \delta \rangle \tag{4}$$

其中 $\delta \in \{\texttt{announce}, \texttt{none}, \texttt{best-effort}\}$ 为投递模式。该系统支持循环和一次性任务。

6.2 Webhook 与设备节点

外部服务可通过 Webhook 触发代理操作（HMAC 签名验证、来源路由）。macOS/iOS/Android 伴侣应用提供设备能力：`camera.capture`、`location.get`、`system.notify` 等。

7 安全架构

7.1 威胁模型

基于已有安全研究 [29–31]，可将 OpenClaw 面临的威胁归纳为三类：**外部对手**（消息伪造、渠道冒充）；**注入对手**（跨渠道提示注入 [16]）；**供应链对手**（被入侵技能）。Chen 等 [31] 在 MITRE ATLAS 框架下测试了 47 种对抗场景，发现 OpenClaw 存在显著的安全隐患。此外，供应链安全研究 [37] 揭示了技能系统中存在的形式化验证缺失问题；ClawWorm [39] 进一步证明了自传播攻击在 LLM 代理生态系统中的可行性，展示了攻击如何通过技能共享机制在代理间扩散。

表 2: OpenClaw 三层安全防御模型（据 [29] 整理）

层级	机制	目标
渠道	配对码/白名单	未授权访问
会话	主/隔离分离	注入传播
技能	描述符审查	供应链攻击

Li 等 [32] 提出了 PRISM 框架，通过 10 个运行时 hook 点和阈值累积机制实现分级防御响应。OpenClaw-RL [38] 则从训练角度探索了通过自然语言对话进行强化学习的可能性，使代理能够通过用户反馈不断优化行为。这些安全和训练技术的结合为构建更加可靠和可控的 AI 助手提供了路径。

7.2 隐私分析

遵循隐私分类法 [4]：*数据隐私*（对话保留在本地）；*模型隐私*（API 密钥本地管理）；*基础设施隐私*（加密隧道远程访问）。

8 案例研究

8.1 个人知识管理

研究者使用 OpenClaw 投递早间简报（天气、ArXiv 论文、技术新闻），微信通勤时交互，桌面端保持上下文。技能框架提供学术搜索、文献综述、Overleaf 集成。

8.2 团队通信桥接

团队通过不同渠道（Telegram、微信、Slack）交互，系统提供统一存在感和跨平台转发。会话隔离防止信息泄露。

8.3 研究工作流自动化

系统定期搜索 ArXiv、生成 LLM 摘要、投递通知。七阶段流水线：触发 → 激活 → 技能调用 → 数据收集 → 过滤 → 摘要 → 投递。

9 对比评估

表 3: AI 助手平台对比分析

特性	**OC**	**GPT**	**CL**	**LC**	**AG**	**R**	**X**
开源	✓	×	×	✓	✓	✓	✓
本地	✓	×	×	△	△	✓	✓
多渠道	✓	×	×	△	×	△	△
LLM 技能	✓	△	△	✓	✓	×	×
定时任务	✓	×	×	×	△	×	×
隐私	✓	×	×	△	△	✓	✓
设备节点	✓	×	×	×	×	×	×

OC=OpenClaw, GPT=ChatGPT, CL=Claude, LC=LangChain, AG=AutoGPT, R=Rasa, X=Xatkit。

10 讨论

10.1 本地优先的权衡

本地优先在*主权*和*便利性*之间创造张力 [9]。权衡受三因素调节：用户技术能力、数据隐私敏感性和成本差异。

10.2 与多智能体系统的比较

OpenClaw 为单智能体、多表面系统，对比 AutoGen [7] 等多智能体框架。单智能体简化协调但限制并行性。Hudgins 等 [35] 对该平台上的 AI 代理社区进行了大规模实证研究。Moltbook 社区的研究 [40] 揭示了代理间的指令共享和规范形成现象，表明 AI 代理正在发展出类似人类社区的协作模式。Just Talk [41] 则展示了代理通过元学习在开放环境中自主演进的能力，这为 OpenClaw 代理的长期适应性提供了理论基础。临床工作流研究 [42] 证明了代理系统在医疗领域的应用潜力，展示了其在动态工作流程中的可靠性和可扩展性。

10.3 复杂度分析

渠道 n 个需 $O(n)$ 适配器；技能 s 个、请求 m 个，$O(s \cdot m)$ 评估；会话 c 个渠道 g 个群组，$O(c+g)$ 管理。

11 结论

本文对 OpenClaw 平台进行了系统化的架构分析。通过涵盖六个研究方向的文献综述（43 篇参考文献）、形式化架构描述和六个平台的对比评估，本文揭示了该平台在隐私保障、可扩展性和多渠道集成方面的架构特点。

分析表明，OpenClaw 的独特价值在于将 LLM 原生推理、多渠道通信和本地优先执行综合为统一的用户主权平台。随着设备端 LLM 推理、持久记忆系统和多智能体协调的发展，本地优先的 AI 助手平台有望在隐私敏感场景中发挥更大作用。

参考文献

[1] Z. Wang et al., "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.

[2] OpenAI, "ChatGPT: Optimizing language models for dialogue," 2023.

[3] OpenAI, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[4] S. Ketabchi, "Evaluating privacy, security, and trust perceptions in conversational AI," *Computers in Human Behavior*, vol. 154, p. 108344, 2024.

[5] A. J. Edu et al., "Smart home personal assistants: A security and privacy review," *ACM Computing Surveys*, vol. 53, no. 6, pp. 1–36, 2020.

[6] T. Kojima et al., "Large language models are zero-shot reasoners," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22199–22213, 2022.

[7] Q. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," *arXiv preprint arXiv:2308.08155*, 2023.

[8] M. Brambilla et al., "Xatkit: A multimodal low-code chatbot development framework," *IEEE Access*, vol. 8, pp. 73146–73159, 2020.

[9] Z. Zhou et al., "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[10] E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[11] N. Dragoni et al., "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, Springer, 2017, pp. 195–216.

[12] Z. Li et al., "A review of AI edge devices and lightweight CNN and LLM deployment," *Neurocomputing*, vol. 568, p. 127087, 2024.

[13] Y. Qin et al., "ToolLLM: Facilitating large language models to master 16000+ real-world APIs," *arXiv preprint arXiv:2307.16789*, 2023.

[14] T. Hagendorff, "The ethics of AI ethics: An evaluation of guidelines," *Minds and Machines*, vol. 30, no. 1, pp. 99–120, 2020.

[15] Y. Gao et al., "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[16] Y. Liu et al., "Imprompter: Tricking LLM agents into executing malicious actions," *arXiv preprint arXiv:2312.15198*, 2023.

[17] E. Frantar et al., "GPTQ: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[18] T. Dettmers et al., "QLoRA: Efficient finetuning of quantized language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[19] H. Touvron et al., "LLaMA: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[20] A. Q. Jiang et al., "Mistral 7B," *arXiv preprint arXiv:2310.06825*, 2023.

[21] LangChain Community, "LangChain: Building applications with LLMs through composability," 2023.

[22] Y. Liu et al., "ToolACE: Winning the points of LLM function calling," *arXiv preprint arXiv:2409.00920*, 2024.

[23] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[24] I. Fette and A. Melnikov, "The WebSocket protocol," *RFC 6455*, IETF, 2011.

[25] A. Vaswani et al., "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[26] T. Brown et al., "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[27] R. Nakano et al., "WebGPT: Browser-assisted question-answering with human feedback," *arXiv preprint arXiv:2112.09332*, 2021.

[28] N. Mehrabi et al., "A survey on bias and fairness in machine learning," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–35, 2021.

[29] Z. Ying et al., "Uncovering security threats and architecting defenses in autonomous agents: A case study of OpenClaw," *arXiv preprint arXiv:2603.12644*, 2026.

[30] Y. Wang et al., "From assistant to double agent: Formalizing and benchmarking attacks on OpenClaw for personalized local AI agent," *arXiv preprint arXiv:2602.08412*, 2026.

[31] J. Chen et al., "A security analysis and defense framework for OpenClaw," *arXiv preprint arXiv:2603.10387*, 2026.

[32] Z. Li et al., "OpenClaw PRISM: A zero-fork, defense-in-depth runtime security framework," *arXiv preprint arXiv:2603.11853*, 2026.

[33] R. Zhao et al., "Security analysis and mitigation of autonomous LLM agent threats," *arXiv preprint arXiv:2603.11619*, 2026.

[34] Anonymous, "Defensible design for OpenClaw: Securing autonomous tool-invoking agents," *arXiv preprint arXiv:2603.13151*, 2026.

[35] J. Hudgins et al., "OpenClaw AI agents as informal learners at Moltbook," *arXiv preprint arXiv:2602.18832*, 2026.

[36] H. Ou et al., "When OpenClaw AI agents teach each other: Peer learning patterns in the Moltbook community," *arXiv preprint arXiv:2602.14477*, 2026.

[37] Anonymous, "Formal analysis and supply chain security for agentic AI skills," *arXiv preprint arXiv:2603.00195*, 2026.

[38] Anonymous, "OpenClaw-RL: Train any agent simply by talking," *arXiv preprint arXiv:2603.10165*, 2026.

[39] Anonymous, "ClawWorm: Self-propagating attacks across LLM agent ecosystems," *arXiv preprint arXiv:2603.15727*, 2026.

[40] Anonymous, "OpenClaw agents on Moltbook: Risky instruction sharing and norm emergence," *arXiv preprint arXiv:2602.02625*, 2026.

[41] Anonymous, "Just talk: An agent that meta-learns and evolves in the wild," *arXiv preprint arXiv:2603.17187*, 2026.

[42] Anonymous, "Toward an agentic operating system for dynamic clinical workflows," *arXiv preprint arXiv:2603.11721*, 2026.

[43] R. Mehta et al., "Empowering real-time communication: A seamless chatting system using WebSocket," in *Lecture Notes in Networks and Systems*, Springer, 2024.