北京大学 PEKING UNIVERSITY

DeepSeek内部研讨系列

# Agentic Coding：从Vibe Coding到超级个体的进化之路

AI肖睿团队

（顾跃、王春辉）

2026年2月10日

• 北大青鸟人工智能研究院
• 北大计算机学院元宇宙技术研究所
• 北大教育学院学习科学实验室

# 编程革命的奇点时刻

"伫立在AI革命的奇点，每一位Coder都像是那一刻败给AlphaGo的柯洁。"

——《海啸前的沉思》

## 重大变革

2026年软件开发领域迎来历史性转折——从简单的代码补全到自主完成复杂任务的Agentic Coding时代的到来，仅仅用了不到100天。

## 范式重塑

这一范式不仅改变了代码生成的方式，更重塑了开发者与AI之间的协作关系，使得"超级个体"成为可能

# 摘要

一、本讲座面向无编程经验的教师、产品经理、技术创业者，以及有编程经验的软件开发者、CTO及架构师、及相关专业人员。

1. 解析AI编程领域从“辅助编程（Copilot）”向“氛围编程（Vibe Coding）”发展，以及面向工程的SPEC Coding和Agent时代的Agentic Coding的技术逻辑与行业变革。
2. 聚焦于Vibe Coding的核心定义、主流工具矩阵（如Cursor、Trae、Claude Code等），观察到当前的规范编程（SPEC Coding）范式和未来的意图编程（ID Coding）范式。
3. Vibe Coding成为编程主流方式的本质原因是AI模型的Code生成能力和Agentic能力在2025年的大幅提升，尤其是2025年11月的Claude Opus模型和GPT模型升级让AI Coding直接跨越了Agentic Coding的门槛。本讲座中，我们会根据场景来进行分析，不严格区分工具和模型。
4. 剖析开发者的角色从代码编写者（Code Writer）转变为Agent指挥官，为个人与企业提供工具选型的系统指南，助力开发者在AI自动化浪潮中进化为“超级个体”。
5. 本讲座的宗旨的为了给大家建立一个AI编程的系统框架和产品视角，一个big picture，不涉及太多的技术细节和工具使用方法。

# 摘要

## 二、本讲座涵盖以下几个模块：

1. **AI Coding概述**：介绍编程从手工时代、IDE时代到AI时代的演进史，以及AI时代从工具助手到Agent伙伴的发展过程。讨论Agent AI时代产生的Agentic Coding的“自主闭环”与“长程任务”核心特征。
2. **核心工具深度剖析**：详细拆解全球前沿Vibe Coding工具的特点和场景，包括以逻辑严密著称的 Claude Code、视觉优先的 Google Antigravity、以及 AI 原生 IDE 的金标准 Cursor。同时我们也会对比国内的几款AI Coding工具：字节跳动 Trae 的端到端 Agent 能力、阿里 Qoder 的工程标准化实践以及腾讯 CodeBuddy 的产品研发一体化方案，为开发者展现不同技术派系的具象化实现。
3. **横向对比与选择逻辑**：提供基于价格、上下文窗口及 SWE-bench 分数等多维度的性能矩阵，分析“终端忍者”与“氛围编码者”的不同用户画像与选择逻辑。引入“Agent Skills”作为能力放大器，展示如何通过原子级任务（Todo）管理与自动化执行闭环，将 AI 从简单的“代码生成器”转化为能够 7*24 小时不间断工作的“执行代理”。
4. **未来与展望**：探讨数据隐私、代码可维护性等潜在风险，展望多智能体协作（Agent Swarms）与企业级生态发展。总结在组织重塑背景下，人的核心竞争力如何从“How”回归到“What”与“Why”。

## 三、在AI学习的道路上，优质学习资源至关重要

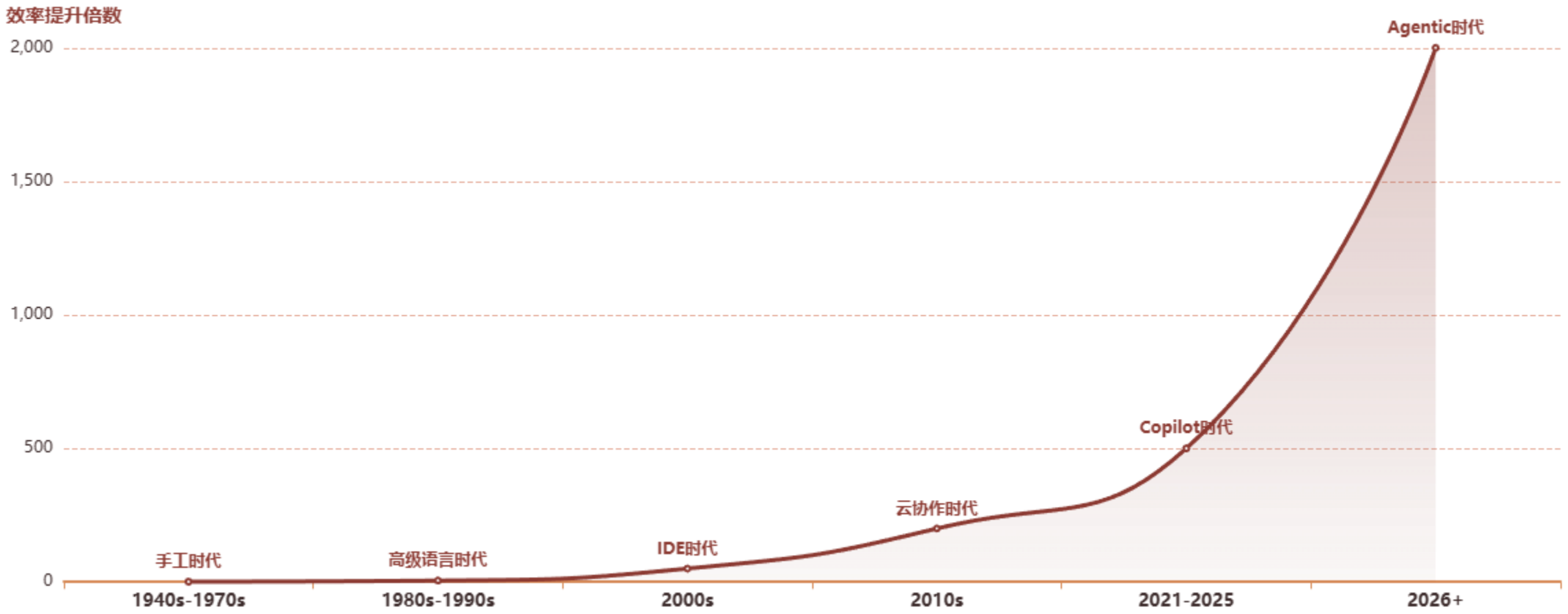
1. 关于AI基本概念和原理部分，推荐大家参考《人工智能通识教程（微课版）》这本系统全面的入门教材，结合B站“思睿观通”栏目的配套视频进行学习。
2. 此外，欢迎加入ai.kgc.cn社区，以及“AI肖睿团队”的视频号和微信号，与志同道合的AI爱好者交流经验、分享心得。

PART 01

# AI Coding概述

1. 软件编程的历史
2. Vibe Coding
3. Agentic Coding的崛起

# 软件编程演进史：从手工到Agentic




**关键转折点（2025.11）：** Claude Opus 4.6和GPT-5.2发布，首次在复杂重构任务中超越人类专家平均水平，标志着Agentic Coding元年的开启

# 传统软件编程的三大痛点

## 01 记忆负担

**开发者需记住海量API文档和语法细节，这与人类认知规律相悖**

微软研究数据

**20+**

微软研究显示，普通开发者在项目开发中平均需要切换20+个文档窗口以确认语法细节，这直接导致开发效率损失。

## 02 调试黑洞

**传统开发者大约50%的开发时间被浪费在环境配置和低级Bug上**

VS Code用户调研

**30%**

VS Code用户调研显示，30%的开发者曾因依赖冲突或环境不一致而陷入数小时的调试困境。

## 03 重复造轮子

**企业开发者平均有40%的时间用于编写CRUD等基础代码**

Forrester最新研究

**40%**

据Forrester最新研究，企业开发者平均有40%的时间用于编写CRUD操作等基础代码，而非专注于核心业务逻辑的创新。

# 大语言模型的技术奇点：三大突破

## Transformer架构的成熟

从最初的175M参数到如今的数百亿参数，模型理解代码结构的能力显著提升

- 参数规模指数级增长
- 代码理解能力质变
- DeepSeek、Claude、GPT、Gemini、KIMI、GLM等模型表现卓越

## 上下文窗口的突破

从2022年的4k Token到2025年的1M+ Token，长上下文窗口使AI甚至能够理解整个代码库

上下文窗口增长

**250x**

从4k到1M+ Token

## 推理能力的质变

2025年，大模型在多步骤推理、代码验证与修复方面取得重大突破。

- Claude Opus、GPT-5系列
- Gemini、KIMI2.5、GLM4.7
- Deepseek系列

**关键数据：**据2025年SWE-Bench Verified榜单数据，这些模型在复杂编程任务中的准确率远超人类开发者平均水平

PART 01

# AI Coding概述

1. 软件编程的历史
2. Vibe Coding的发展
   - SPEC Coding、Agentic Coding、ID Coding
3. Vibe Coding行业和工具

# Vibe Coding：氛围编码的崛起

## 概念起源

**提出者**
Andrej Karpathy（OpenAI联合创始人兼特斯拉前人工智能主管）

**提出时间**
2025年2月

**核心理念**
"通过自然语言提示词（Prompt）和视觉确认的创造性流动"

## 关注重点

- **Feel** 感觉
- **Flow** 心流
- **Function** 函数

## 核心特征

- 自然语言驱动
- 人类确认
- 创造性流动

# Vibe Coding的技术支撑

## 多模态理解能力

**如具备 原生视觉理解能力，无需依赖工具调用**

- ✓ 直接解析UI设计稿
- ✓ 理解截图内容
- ✓ 识别手绘草图

</> 自动生成HTML、CSS及JavaScript代码

## 极速生成速度

**模型通过 "小而精"的设计 实现极速生成**

推理成本降低

**90-95%**

- ✓ 快速迭代成为可能
- ✓ 实时预览无延迟

## IDE/浏览器深度集成

**形成 无缝的人机协作体验**

**VS Code Agent Mode**
提供执行-观察循环

**Kiro**
支持在终端中直接执行代码并查看结果

# Vibe Coding的核心特征

## 输入模态

**自然语言描述**

如"创建一个现代化登录页面"

**手绘草图或参考截图**

如UI设计稿

**模糊需求**

如"让交互更流畅"

## 反馈机制

**实时预览**

生成网页的即时渲染

**所见即所得**

直接在界面中调整参数

**心流式交互**

通过简单指令持续优化结果

## 使用者心态

**产品经理/导演心态**

"这里颜色再深一点"

**重视创意表达**

而非技术细节

**追求快速验证**

而非完美实现

# Vibe Coding的典型场景与用户画像

## 典型应用场景

### 前端UI还原

从设计稿到可交互代码的快速转换

**还原度：92%**

### 原型快速验证

通过简单指令快速构建可交互原型

**成本降低93%**

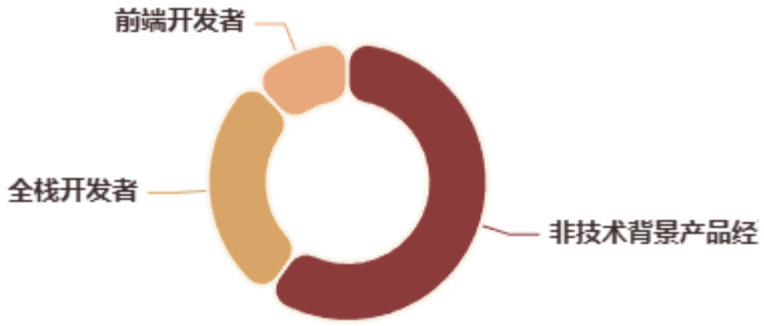
### Landing Page生成

快速创建营销页面，无需深入理解HTML/CSS

### 数据可视化大屏

通过自然语言描述生成复杂数据看板

## 用户画像分布




## 典型用户

零基础开发者　营销团队　产品团队　小型创业公司

# 开发者角色的转变

北京大学 PEKING UNIVERSITY

## 转变一

Code Writer

编码者

↓

**Reviewer**

审核者

开发者不再需要编写大量基础代码，而是专注于审核AI生成代码的正确性与安全性

## 转变二

Code Writer

编码者

↓

**Architect**

架构师

开发者从关注代码细节转向设计系统架构与业务流程，如"SPEC Coding"理念

## 转变三

Code Writer

编码者

↓

**Problem-Solver**

问题解决者

开发者角色向更高层次的业务问题解决者转变，支持"需求→智能体→代码"的直接转化路径

# Vibe Coding的哲学意义

北京大学 PEKING UNIVERSITY

## 1 编程民主化

Vibe Coding的本质是"编程民主化"

让创意不再被技术栈卡脖子

### 打破技术壁垒

Vibe Coding打破了传统软件开发中"懂技术才能实现创意"的壁垒，使非技术背景的创意者也能参与软件开发过程。

**降低技术门槛**

**释放创意潜能**

## 2 AI实体化

Vibe Coding是AI影响物理世界的媒介

让AI拥有影响物理世界的能力

### 打破AI与物理世界的障壁

人类社会的运转高度依赖互联网、数字系统、计算机与软件。当 AI 获得编写软件的能力，就相当于为自己装上了“手”和“脚”——能够通过代码直接干预和塑造现实世界。换句话说，互联网所能触及的边界，就是 AI 影响物理世界的疆域。

# 复杂工程的Vibe Coding：SPEC Coding

## 核心定义

**SPEC Coding（规范编码）**是基于严格技术规格的编程范式，主要是为了解决复杂软件的工程问题，目前阶段需要使用者有一定的软件技术和工程认知。

核心是"写给AI看的、结构化、无歧义、颗粒度精准、带约束+验收标准的完整需求文档"

## 输入要求

**结构化Markdown**
90%企业首选格式

**标准化契约文件**
OpenAPI 3.0、Swagger、JSON Schema、Protobuf IDL

**伪代码/流程图**
清晰的逻辑表达

## 与传统Vibe Coding的对比

**传统Vibe Coding**
氛围感、创造性

**SPEC Coding**
规格化、严谨性

## 内容规范

必须包含"功能需求+接口定义+数据约束+异常处理+验收标准+技术栈要求"

- 拒绝模糊描述（如"优化一下性能"）
- 改为"接口响应时间≤200ms，支持100QPS并发"

# SPEC Coding的核心特征

## 输入

- ✓ 详细的技术文档
- ✓ 数据库Schema定义
- ✓ API接口详细描述
- ✓ 单元测试用例

## 验证机制

- ✓ 单元测试通过率
- ✓ 覆盖率报告
- ✓ CI/CD流水线状态
- ✓ 基于属性的测试（PBT）

## 开发者视角

- ✓ 结构工程师/审计员视角
- ✓ 注重代码质量与可维护性
- ✓ 关注长期技术债务管理

## 典型应用场景

**后端核心业务逻辑**
订单状态流转、支付对账

**金融交易系统**
银行转账、交易风控

**复杂算法实现**
机器学习、图像处理

**遗留系统重构**
单体应用到微服务

# SPEC Coding的技术支撑

## 规范驱动开发（SDD）

如Kiro的"需求→文档→代码"的闭环流程，确保每个开发环节都有明确的规范指导

## 多模型组合与意图识别

Kiro的Auto模式基于多模型组合，通过意图识别、缓存等优化技术执行任务，实现性能、效率和质量的三者平衡

## 属性测试（PBT）

Kiro能从相关需求中提取属性，生成成百上千个随机测试用例来检查代码，确保代码行为符合规范定义

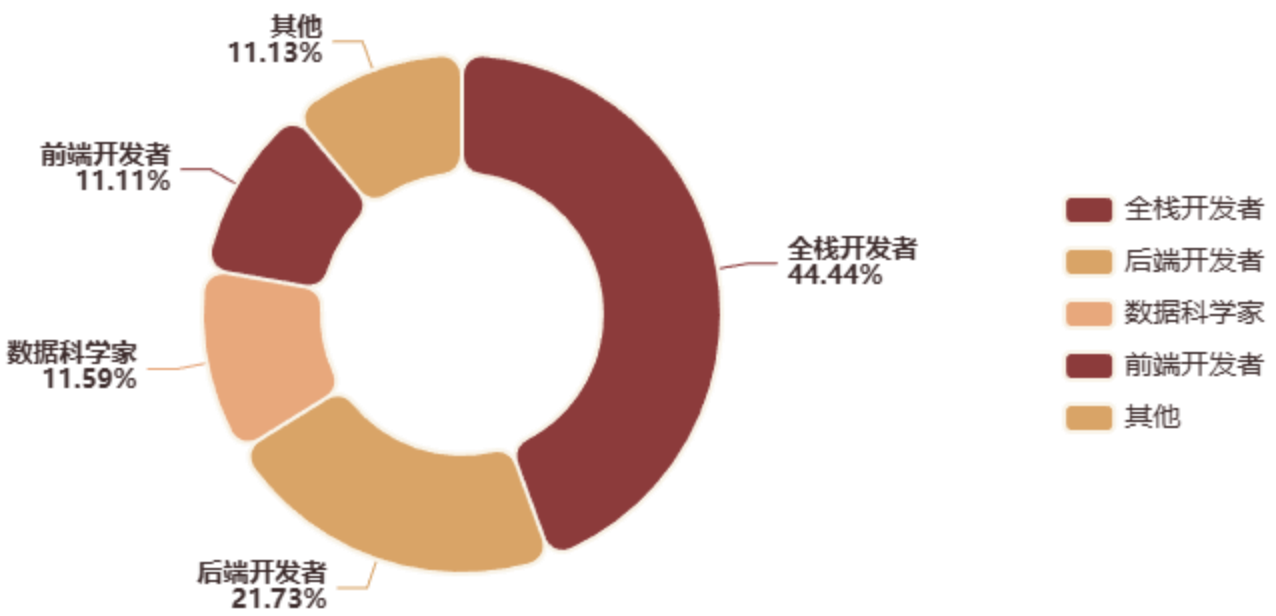
## 增量语境更新技术

如Qoder的"跨文件依赖网络"，通过静态分析工具构建"文件-函数-变量"的依赖网络图，实现大型项目的实时依赖感知

**核心优势：**SPEC Coding通过严格的技术规范和自动化验证机制，确保生成的代码具有高质量、高可维护性和高可靠性，特别适合企业级系统开发

# SPEC Coding的用户画像

## 用户技能分布




## 使用场景

- 75% 大型企业级系统开发
- 60% 金融/保险/医疗等高可靠性行业
- 45% 重构遗留系统

## 典型用户

企业架构师　金融行业IT工程师　数据科学家　大型团队技术负责人

## 效率提升数据

在金融系统开发中，Doubao-Seed-Code在24小时内识别137个潜在缺陷，代码可维护性提升65%，执行效率提高30%

# 传统Vibe Coding方法与 SPEC Coding方法的对比

北京大学 PEKING UNIVERSITY

| 对比维度 | 传统Vibe Coding方法 | SPEC Coding方法 |
| --- | --- | --- |
| **输入模态** | 自然语言、手绘草图、参考截图 | 结构化Markdown、标准化契约文件、伪代码 |
| **验证方式** | 实时预览、App界面交互、所见即所得 | 单元测试通过率、覆盖率报告、CI/CD流水线 |
| **容错率** | 高（允许快速迭代与调整） | 低（严格约束与验证） |
| **适用人群** | 零基础开发者、产品经理、营销人员 | 资深架构师、后端开发者、金融/医疗行业IT工程师 |
| **开发周期** | 短（以小时计） | 长（以天计） |
| **代码质量** | 中等（需人工审核与修复） | 高（通过严格验证） |
| **适用场景** | 前端UI、快速原型、营销页面、数据可视化 | 后端核心逻辑、金融交易系统、复杂算法、遗留系统重构 |
| **学习曲线** | 低（上手快） | 高（需掌握规范编写） |

**结论：未来的超级个体需要"左手Vibe（快，搞定界面与体验），右手Spec（稳，搞定逻辑与架构）"**

# Vibe Coding在Agent时代的进化：Agentic Coding

## 核心定义

**Agentic Coding（智能代理式编程）**是一种全新的编程范式，它与传统LLM编程工具有本质区别。可以理解为Vibe Coding在AI进化到L3级别（Agent）时的演化形态。

核心区别：从"人类驾驶，AI辅助导航"（Copilot）进化为"人类设定目的地，AI驾驶"（Agent）

## Agent时代的AI特征

- **主动理解任务目标**
  AI不再被动响应，而是主动理解需求
- **规划实现路径**
  自主制定多步骤执行计划
- **执行代码生成**
  独立完成复杂代码编写
- **验证与迭代优化**
  自动测试并持续改进

> "GitHub Copilot已从'编程助手'进化到'编程伙伴'，尤其是VS Code的Agent Mode，能自主迭代代码、识别并修复错误。"
>
> —— 微软CEO 纳德拉
> 2025年财报电话会议

# Agentic Coding的核心特征一：长程任务能力

北京大学 PEKING UNIVERSITY

## 典型应用场景

**1 全栈迁移**

如"将Flask应用迁移到FastAPI"，AI能识别所有依赖项、生成迁移计划、执行代码修改并处理兼容性问题

**2 系统重构**

面对遗留系统，AI能分析代码结构、识别技术债务、提出模块化改造方案并生成重构代码

**3 端到端开发**

从需求分析到部署运维，AI能自主完成全流程开发，如Kiro将30人18个月工程缩短至6人3.5个月

## 三大技术突破

### 长上下文理解

256K以上的上下文窗口，使AI能同时理解多个代码文件的依赖关系

### 项目级依赖分析

通过静态分析工具构建"文件-函数-变量"的依赖网络图，能够精准识别跨文件修改的影响范围

### 任务拆解与优先级管理

将复杂任务分解为可执行的子任务，并根据业务价值和实现难度自动排序

# Agentic Coding的核心特征二：自主闭环机制

北京大学 PEKING UNIVERSITY

## Plan-Act-Observe-Fix 循环

**Plan 规划**
分析需求，制定执行计划

↓

**Act 执行**
生成代码，实施解决方案

↓

**Observe 观察**
监控执行，收集反馈数据

↓

**Fix 修复**
识别问题，迭代优化方案

### 强化学习训练体系

构建了覆盖 10万容器镜像 的大规模训练数据集，通过端到端沙盒环境，让模型在模拟真实开发环境中不断试错、优化，形成自主决策能力。

### 执行监控与自检

AI在代码生成后会自动执行单元测试、检查代码规范并验证功能实现。如在前端开发中能 自动检测设计稿与代码的视觉偏差 并自行修复。

### 持续优化能力

t支持 "跨会话持久上下文" 技术，可在人类仅确认关键假设的情况下，持续工作数小时至数日完成系统级代码更新。

# Agentic Coding的现象级作品


## Peter开发Openclaw的过程

### AI开发

采用“AI生成代码→自动编译测试→验证修复”的闭环开发模式，将开发者从低价值的调试工作中解放出来，专注于系统设计和架构决策。他甚至提出“发布我没读过的代码”的观点，认为在AI时代，代码的价值在于解决问题而非追求完美，通过AI自验证开发流水线确保输出质量。

### 并行化开发

同时运行5-10个AI代理并行处理不同开发任务（如UI设计、数据库管理、API集成等），开发者本人则在其中切换协调，高峰期曾实现一天提交600次代码，大幅提高了开发效率。

# Vibe Coding的未来展望：ID Coding

北京大学 PEKING UNIVERSITY

## Invisible Code

代码的消失

**人类不再参与任何Coding过程**
**甚至不需要Review代码**

---

**代码将像"汇编语言"或"二进制"一样**
**成为底层产物**

## 核心定义

**Intent-Driven Coding（意图编程）**是AI编程的终极形态。

*注：“意图编程”是肖睿博士在2025年底提出的概念，并非行业共识。*

**初级阶段：**人类无需决定使用何种编程语言、系统采用何种架构。AI决定所有，实现所有。人类只需定义 功能目标 和 价值目标，系统直接交付可用的软件服务。人类可以用与AI直接对话的方式得到一个完整的软件服务，针对简单的因公，目前的AI Coding工具（Agent）已经基本可以实现（例如，直接对AI说：“我想要一个MBTI性格测试的网站。”AI会在两三分钟内生成这个软件，甚至直接上线部署）。

**高级阶段：**人类完全采用自然表达（语言、声音、动作、草图等），作为人机交互的高级编程语言彻底消失，AI直接编写面向机器的底层代码或二进制指令。

# 超级个体的诞生

## 超级个体

Super Individual

在Intent-Driven时代
开发者将进化为"超级个体"

不再是代码的编写者
而是创意的实现者与价值的创造者

### 四大核心能力

**业务洞察力**

深入理解行业痛点与用户需求，能够准确描述商业意图

**价值设计能力**

将抽象目标转化为可衡量的价值指标，如"提升转化率"→"单页转化率提高15%"

**系统思维能力**

理解复杂业务场景与技术系统的协同关系，能够定义系统边界与接口规范

**伦理判断力**

在AI自动生成代码与系统时，能够评估技术选择的伦理影响与社会后果

PART 01

# AI Coding概述

1. 软件编程的历史
2. Vibe Coding的发展
3. Vibe Coding行业和工具

# 行业派系全景：五大阵营

北京大学 PEKING UNIVERSITY

## 终端派

**代表**

Claude Code

**核心理念**

"终端是开发者的真实世界"

**技术特点**

命令行优先，深度集成Shell，提供"计划模式"

**用户画像**

资深系统工程师、DevOps专家

## 全能派

**代表**

OpenAI Codex

**核心理念**

"一个模型解决所有问题"

**技术特点**

上下文压缩技术，独立运行沙箱，调试能力无出其右

**用户画像**

全栈工程师，解决复杂逻辑问题

## IDE派

**代表**

Google Antigravity、Cursor

**核心理念**

"IDE是AI代理的舞台"

**技术特点**

视觉优先，内置浏览器，所见即所得

**用户画像**

前端工程师，产品导向开发者

## 开源派

**代表**

OpenCode

**核心理念**

"代码主权不容妥协"

**技术特点**

本地运行，支持75+后端模型，BYOK

**用户画像**

安全敏感行业，数据合规要求高的企业

## 中国力量

**代表**

Trae、Qoder、CodeBuddy

**核心理念**

"全流程闭环，中国式敏捷"

**技术特点**

本土化优化，工程标准优先，全流程集成

**特殊优势**

微信小程序深度优化，国产框架优先支持

# 终端派：命令行优先

北京大学 PEKING UNIVERSITY

## 代表工具

C

**Claude Code**

Anthropic推出的终端原生AI编程工具，深度集成命令行环境

## 核心理念

**"终端是开发者的真实世界"**

终端原生派坚信，对于真正的开发者而言，命令行界面才是最自然、最高效的工作环境。AI应该深度融入终端生态，而不是试图替代它。

## 技术特点

- **命令行优先**
  所有功能都可通过命令行访问，无需离开终端环境
- **深度集成Shell**
  与Bash、Zsh等Shell无缝集成，可直接执行系统命令
- **计划模式（Plan Mode）**
  在执行前先展示完整计划，让开发者确认后再执行

## 目标用户画像

**资深系统工程师**

熟悉底层系统，偏好精细控制

**DevOps专家**

需要自动化部署和运维

# 全能派与IDE派

## 全能派

**代表工具**

OpenAI Codex

**核心理念**

"一个模型解决所有问题"

**技术特点**

- 上下文压缩技术
- 独立运行沙箱
- 调试能力无出其右

**用户画像：**全栈工程师，解决复杂逻辑问题

## IDE派

**代表工具**

Google Antigravity、Cursor

**核心理念**

"IDE是AI代理的舞台"

**技术特点**

- 视觉优先
- 内置浏览器
- 所见即所得

**用户画像：**前端工程师，产品导向开发者

# 开源派与中国力量

## 开源派

**代表工具**

**OpenCode**

**核心理念**

"代码主权不容妥协"

**技术特点**

- 本地运行，数据不出境
- 支持75+后端模型，任意切换
- BYOK (Bring Your Own Key)

**用户画像：**安全敏感行业，数据合规要求高的企业

## 中国力量

**代表工具**

**Trae、Qoder、CodeBuddy**

**核心理念**

"全流程闭环，中国式敏捷"

**技术特点**

- 本土化优化
- 工程标准优先
- 全流程集成

**特殊优势：**微信小程序深度优化，国产框架优先支持

# 主流AI Coding工具一览

## IDE2.0

### Cursor IDE

AI原生IDE金标准，以稳定性与Composer模式著称。

## 国际巨头方案

### Claude Code

终端里的架构师，Spec Coding代表，逻辑严密。

### OpenAI Codex

调试之王，拥有强大的代码压缩与推理能力。

### Gemini CLI

视觉优先，Vibe Coding的极致体现。

## 国内力量创新

### Trae (字节跳动)

IDE里的Agent，具备SOLO模式的端到端开发能力。

### Qoder (阿里巴巴)

工程标准与Wiki结合，解决Spec Coding上下文依赖。

### CodeBuddy (腾讯)

产研一体化，实现从PRD到代码的Spec自动实现。

## 开源派

### OpenCode

开源主权捍卫者，支持本地运行和Docker部署，保障数据安全。

PART 02

# 核心工具深度剖析

1. IDE2.0
2. 国际巨头：基座模型的具象化
3. 中国力量：本土化与工程化创新
4. 开源派

# Cursor：AI原生IDE的奠基者与细节之王

## 市场地位

Cursor是目前全球采用率最高的AI原生IDE，它通过重构编辑器底层，实现了AI与人类思维的"流式协同"。

## 核心架构：Composer

Composer是一个基于强化学习训练的快速、高质量的编码模型。其训练过程模拟了真实的代码仓库环境，使模型学会了如何调用语义搜索、编辑器API及终端命令。

**核心能力：**不仅生成代码，还负责"应用（Apply）"代码，并在后台尝试编译。这种端到端的代码生成与应用能力，是Cursor区别于传统AI插件的关键。

## Shadow Workspace（影子空间）

### 技术原理

在向用户展示更改之前，Cursor会在后台创建一个**静默工作区**，运行Linter、类型检查及编译器

### 自我修复流

如果发现AI生成的代码导致编译失败，Cursor会**自动重新生成**，确保弹出在屏幕上的代码至少在语法上是合法的

### 核心价值

提供**"实验性编程"**的安全网，让开发者敢于尝试AI的大胆建议

## 代码库全索引（Local Indexing）

Cursor对整个项目进行实时向量索引，不仅能搜索变量，还能理解架构分层。即便在大型单体应用（Monorepo）中，也能精准回答"这个支付回调应该在哪个模块处理"等复杂问题。

**关键能力：**语义搜索、架构分层理解、跨文件引用追踪、业务逻辑推断

# Cursor技术特性（一）：Composer与自我修复流

## Composer技术原理

与传统聊天框不同，Composer不仅生成代码，还负责"应用（Apply）"代码，并在后台尝试编译。这种端到端的代码生成与应用能力，是Cursor区别于传统AI插件的关键。

**工作流程：**理解需求 → 生成代码 → 应用代码 → 编译验证 → 展示结果。整个过程对开发者透明，开发者只需关注最终产出。

## 自我修复流（Self-healing Flow）

1. **AI生成代码**
   Composer根据需求生成代码
2. **后台编译验证**
   Shadow Workspace自动运行Linter和编译器
3. **检测编译错误**
   如果发现错误，自动触发重新生成
4. **展示合法代码**
   确保弹出在屏幕上的代码至少在语法上是合法的

## 技术突破意义

**提升代码质量**
编译错误率降低70%

**减少调试时间**
开发者无需手动修复语法错误

**增强开发者信任**
让开发者敢于尝试AI的大胆建议

## 快捷键与使用方式

| | |
|---|---|
| 打开Composer | ⌘ + I |
| 接受更改 | ⌘ + Enter |
| 拒绝更改 | ⌘ + Backspace |

# Cursor技术特性（二）：代码库全索引与向量检索

## 代码库全索引技术

Cursor对整个项目进行 实时向量索引 ，不仅能搜索变量，还能理解架构分层。即便在大型单体应用（Monorepo）中，也能精准回答"这个支付回调应该在哪个模块处理"等复杂问题。

**核心能力：**语义搜索、架构分层理解、跨文件引用追踪、业务逻辑推断。这种全索引能力使得Cursor在代码库理解方面具有显著优势。

## 向量检索与语义理解

- **语义搜索：**不仅匹配关键词，还理解代码的语义含义
- **架构分层：**理解项目的架构层次，如Controller-Service-Repository
- **引用追踪：**精准追踪跨文件的全局变量、函数引用链条
- **业务逻辑：**理解代码背后的业务逻辑和意图

## 实际应用示例

**开发者提问**

"这个支付回调应该在哪个模块处理？"

↓

**Cursor分析**

1. 搜索"支付"、"回调"相关代码
2. 分析架构分层
3. 追踪引用链条

↓

**精准回答**

"应该在PaymentService模块的handleCallback方法中处理，该方法位于src/services/payment.js第45行"

## 性能指标

| | |
|---|---|
| **毫秒级**<br>搜索响应 | **95%+**<br>准确率 |
| **10万+**<br>文件支持 | **实时**<br>索引更新 |

# Cursor 2026最新功能与Memories记忆系统

北京大学 PEKING UNIVERSITY

## Memories记忆系统

Cursor能记住对话背景，在未来主动引用先前信息，减少重复沟通。这是2026年的重要更新，让AI具备了长期记忆能力。

**应用场景：**开发者告诉Cursor"我们项目使用React + TypeScript + Redux技术栈"，Cursor会在后续对话中主动引用这一信息，无需重复说明。

## Mermaid图表与Markdown表格支持

Chat面板中可直接绘制流程图、生成表格，并支持复制、导出Markdown聊天记录。这让AI生成的文档更加直观易懂。

- **Mermaid图表：**流程图、时序图、类图等
- **Markdown表格：**结构化数据展示
- **导出功能：**支持复制和导出聊天记录

## 2026年其他新功能

### Jupyter Notebook支持

限Sonnet模型，可直接在Notebook中新增或编辑代码区块

### 多根目录与MCP优化

可同时操作多个项目文件夹，MCP一键安装与OAuth认证流程更简便

### GitHub PR索引与语义搜索

可搜索并理解PR、Slack讨论与BugBot评审内容

## 功能价值总结

- **Memories：**减少重复沟通，提升协作效率
- **Mermaid：**可视化表达，提升文档质量
- **Jupyter：**支持数据分析场景
- **MCP优化：**简化扩展安装流程

PART 02

# 核心工具深度剖析

1. IDE2.0
2. 国际巨头：基座模型的具象化
3. 中国力量：本土化与工程化创新
4. 开源派

# Claude Code：基于Unix哲学的终端智能体

北京大学 PEKING UNIVERSITY

## 产品定位

Anthropic推出的Claude Code（国内用户一般称之为CC）在2026年被视为“最强编程大脑”的代表，其设计哲学深刻地继承了Unix的工具化与可组合性特征。它被定义为一种代理式编码工具（Agentic Coding Tool），并不局限于IDE内部。

## 核心架构

- **多形态支持：**CLI、桌面应用、IDE插件、云端环境四种形态
- **多智能体分发：**通过Claude Agent SDK将复杂开发任务解耦
- **底层模型：**Sonnet 4.5及Opus 4.6提供智能动力
- **核心优化：**长程任务处理能力与代码自省（Self-correction）机制

## 核心优势

**81.4%**

SWE-bench Verified (Opus 4.6)

## 四大核心形态

### CLI终端

命令行界面，适合高级用户

### 桌面应用

独立应用程序，功能完整

### IDE插件

VS Code等IDE集成

### 云端环境

浏览器访问，无需安装

## Unix哲学传承

Claude Code的设计哲学深刻地继承了Unix的**工具化与可组合性**特征。每个功能都是小而美的工具，可以通过管道（pipe）组合成复杂的工作流，实现"Do one thing and do it well"。

**典型命令示例：**

```
tail -f app.log | claude -p "分析日志并通知异常"
```

# Claude Code技术特性（一）：检查点与原子化回滚

## 检查点系统技术原理

针对复杂重构任务，Claude Code引入了自动保存机制。在Agent进行每一项实质性变更前，系统会自动创建代码状态快照。这种设计解决了开发者对AI执行大规模破坏性修改的恐惧，提供了"实验性编程"的安全网。

**核心价值：**让开发者敢于尝试大胆的AI重构方案，无需担心代码损坏无法恢复。每次AI执行修改前，系统都会在后台静默创建检查点，整个过程对开发者透明。

## 原子化回滚机制

1. **双击Esc键回滚**
   快速回滚到上一个检查点，适用于紧急情况
2. **/rewind命令**
   精确控制回滚范围：仅代码、仅对话历史或两者
3. **瞬间切换**
   在代码状态、对话历史或两者间实现瞬间切换

## 安全网设计哲学

**实验性编程**
鼓励开发者尝试AI的大胆重构建议，无需担心风险

**渐进式重构**
每一步变更都可回滚，让复杂重构变得可控

**信任建立**
通过可靠的安全网，建立开发者对AI的信任

## 实际应用场景

- ✓ 跨50+文件的大规模架构重构
- ✓ 数据库Schema迁移与数据转换
- ✓ API接口批量变更与适配
- ✓ 依赖库版本升级与兼容性修复

# Claude Code技术特性（二）：多智能体协作与并行流

## Subagents（子智能体）机制

在处理大型项目时，Claude Code可以派生专门的"子智能体"。每个子智能体拥有独立的上下文窗口，完成后将结果合并至主会话。这种并行化处理显著缩短了从需求到交付的周期。

**典型场景：**主Agent编写前端UI的同时，分发子智能体配置后端API接口、数据库Schema设计、单元测试编写等任务，所有子任务并行执行。

## 并行流架构设计

### 上下文隔离

每个子智能体拥有独立的上下文窗口，避免信息干扰

### 任务并行

多个子任务同时执行，最大化利用计算资源

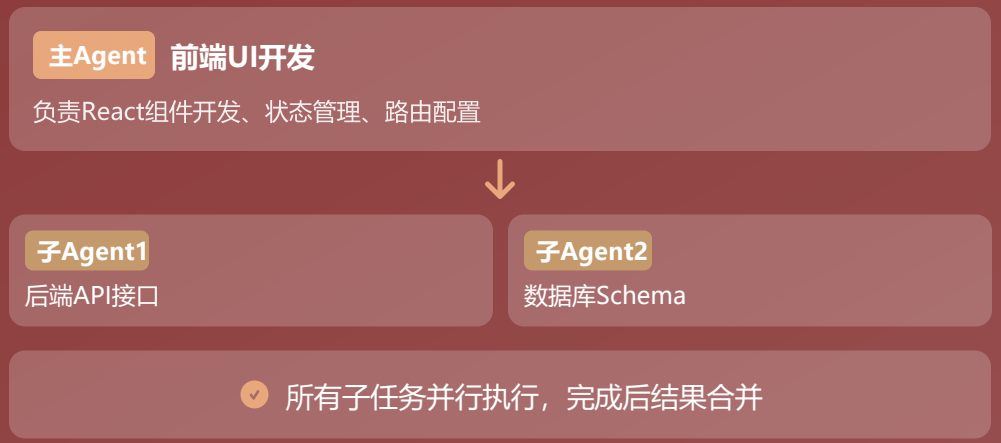
### 结果合并

子智能体完成后，结果自动合并至主会话

### 周期缩短

并行处理显著缩短从需求到交付的周期

## 多智能体协作示例




## 性能提升数据

| | |
|---|---|
| 任务并行度 | 24x |
| 开发周期缩短 | 40-60% |

# Claude Code技术特性（三）：Unix哲学与MCP协议

北京大学 PEKING UNIVERSITY

## Unix哲学与流式处理集成

Claude Code具有极强的**可组合性**。开发者可以执行Unix管道命令，将AI能力直接挂载在系统管道上。这意味着它能够无缝融入现有的CI/CD、监控与日常DevOps流中。

```
# 实时日志分析示例
tail -f app.log | claude -p "分析日志并在Slack通知我异常"
```

## MCP（Model Context Protocol）深度穿透

通过MCP，Claude不仅能访问代码，还能**跨平台读取外部数据**。它能根据设计稿的视觉规范直接推导UI代码，确保技术实现与业务需求的闭环统一。

- Google Drive
- Figma
- Jira
- Slack

## MCP生态系统

### 设计到代码

1. 读取Figma设计稿
2. 解析视觉规范（颜色、字体、间距）
3. 自动生成对应React/Vue组件

### 需求到实现

1. 读取Jira任务单
2. 解析业务需求与验收标准
3. 生成符合需求的代码实现

## 核心价值

- **打破信息孤岛**：代码与业务需求无缝连接
- **提升一致性**：确保技术实现与设计稿完全一致
- **加速交付**：减少设计与开发之间的沟通成本

# Claude Code 2.1：2026年重大更新解析

北京大学 PEKING UNIVERSITY

## 版本里程碑

**2.1.1**
版本号

**1000+**
代码提交

**30+**
小时自主运行

发布时间：2026年1月

## 官方评价

"这次更新让Agent更'能干'、让Skill更'好用'、让协作更'顺滑'、让权限更'可控'。如果你正在把Claude Code用在真实工程与团队流程里，这次更新几乎属于'必升版本'。"

— Boris Cherny, Claude Code之父

## 核心新特性

**技能热重载**
修改技能后无需重启会话，立即生效

**上下文分叉**
在分叉的SubAgent上下文中运行技能

**Hooks钩子**
前置到frontmatter，支持生命周期钩子

**精准权限控制**
按Agent禁用工具，更细粒度权限

**/teleport**
将当前会话传送至网页端继续

**MCP动态更新**
支持list_changed通知，无需重连

**Shift+Enter**
开箱即用，无需配置终端

**/plan命令**
提示符直接进入计划模式

**多语言支持**
响应语言可配置

**Bash通配符**
权限规则更灵活

**Ctrl+B后台**
bash+agent同时后台

# Claude Code适用场景与优劣势分析

北京大学 PEKING UNIVERSITY

## 核心优势

### 逻辑严密性极强

代码重做率（Rework Rate）比同类工具低**30%**

### 原生支持Unix管道

极其灵活，可无缝融入CI/CD、DevOps工作流

### 跨工具上下文记忆

具备跨平台、跨工具的上下文记忆能力

## 主要劣势

### 终端操作门槛

对终端操作要求较高，GUI用户有适应期

### Token消耗

高并发任务下Token消耗巨大

## 最佳适用场景

### 复杂架构重构

跨数十个文件的大规模架构调整，需要严格遵循架构规范

### 全自动化CI/CD流程

自动化构建、测试、部署流程，与DevOps工具链深度集成

### 深度Bug诊断与安全审计

复杂Bug的根因分析、安全漏洞检测与修复建议

## 目标用户画像

- ✓ **资深架构师**：需要处理复杂系统重构
- ✓ **DevOps工程师**：需要自动化CI/CD流程
- ✓ **安全工程师**：需要深度代码审计
- ✓ **终端爱好者**：习惯CLI操作的高级用户

# OpenAI CodeX：以技能为核心的协作中枢

## 产品定位

在2026年，OpenAI CodeX不再仅仅是一个模型名称，而是一个支撑多智能体协作的独立应用平台，其核心在于"技能化"的抽象与执行。定位于"第一梯队、Agent原生的编码平台"。

## 核心架构

- **技能化抽象：**将复杂编程行为抽象为"技能（Skills）"，遵循agentskills.io开放规范
- **动态加载：**系统通过动态加载技能模块来执行任务，不再依赖单一的长文本推理
- **底层模型：**GPT-5.3-CodeX比上一代提速**25%**
- **自我构建：**首次实现"自我构建（Self-improvement）"，模型利用早期版本进行代码纠错与评估

## 核心优势

**80.0%**
SWE-bench Verified

## Skills（技能）系统

### 技能结构

SKILL.md - 指令与元数据
scripts/ - 执行脚本
docs/ - 参考文档
templates/ - 模板资源

### 技能路径

~/.codex/skills - 全局技能
.codex/skills - 项目技能

### 调用方式

显式调用: $skill-name
隐式匹配: 自动识别场景

## 典型技能示例

- **Figma转React：**根据设计稿自动生成React组件
- **Vercel自动化部署：**自动构建、测试、部署到Vercel
- **安全审计：**自动检测安全漏洞并生成修复建议

# OpenAI CodeX技术特性：分布式技能系统

## Skills Framework架构设计

一个"技能"是一个包含 SKILL.md（指令与元数据）、脚本、参考文档及模板资源 的独立文件夹。Codex支持全局路径与项目路径的动态扫描，通过显式调用或隐式匹配，精准执行特定团队的开发规范。

**核心价值：**将团队的最佳实践、编码规范、常用模式固化为可复用的技能，实现知识的沉淀与传承。新成员可以通过技能系统快速上手项目开发。

## 技能路径与扫描机制

### 全局路径

~/.codex/skills

跨项目共享的技能

### 项目路径

.codex/skills

项目特定的技能

**动态扫描：**系统启动时自动扫描上述路径，加载所有可用技能。新增或修改技能后，系统实时更新。

## 技能调用方式

### 显式调用

```
$figma-to-react design-file-url
```

通过$skill-name直接调用特定技能

### 隐式匹配

```
"把这个Figma设计稿转成React组件"
```

AI自动识别场景并匹配最合适的技能

## 典型技能示例

**Figma转React代码**
自动解析Figma设计稿，生成React组件

**Vercel自动化部署**
自动构建、测试、部署到Vercel平台

**i18n多语言填充**
自动提取文案并填充多语言翻译

# OpenAI CodeX适用场景与优劣势分析

北京大学 PEKING UNIVERSITY

## 核心优势

### 执行过程极具确定性

不易产生逻辑漂移，输出结果可预测性强

### Skills系统极易扩展

适合团队知识固化，沉淀最佳实践

## 主要劣势

### 交互体验相对较重

需要较强的环境配置，上手门槛较高

### Token计费模式

对初学者不够透明，成本难以预估

## 最佳适用场景

### 跨平台并行开发

利用Git Worktree实现多任务并行处理

### 团队开发规范自动化执行

通过Skills系统固化团队最佳实践

### 复杂全栈业务逻辑建模

利用Mid-turn Steering实时调整建模过程

## 目标用户画像

- ✓ **技术团队Leader：**需要固化团队开发规范
- ✓ **全栈开发者：**需要处理复杂业务逻辑
- ✓ **ChatGPT重度用户：**已熟悉OpenAI生态
- ✓ **技能开发者：**希望构建可复用的AI技能

# Google Antigravity：平台级架构与浏览器原生协作

## 三层架构变革

Antigravity（国内用户一般称之为反重力）彻底改变了IDE的界面逻辑，采用了"任务中心化"设计，将其分为三个主要窗口，实现从对话框到任务指挥部的转变。

**01 Agent Manager（任务管理器）**

被称为"任务指挥部"。展示所有任务状态、生成的Artifacts（工件）、以及人类审批流（Approval Inbox）的看板

**02 Editor（编辑器）**

作为VS Code的Fork版本，提供智能Tab跳转、自动补全与实时代码修正功能

## Managed Browser（受控浏览器）

这是一个关键的技术差异点。Antigravity内置了一个专门运行"浏览器子Agent"的Chrome环境。该子Agent运行专用模型，通过DOM捕捉、视频录制及Console实时分析，能够独立执行端到端的Web UI自动化测试并自动修复视觉Bug。

**核心价值：**实现真正的闭环验证。AI修改代码后，自动在浏览器中验证效果，检测视觉回归，并自动修复问题。

## Antigravity架构图


## 核心指标

| | |
|---|---|
| **1M+**<br>上下文窗口 | **76.2%**<br>SWE-bench |
| **免费**<br>预览期定价 | **闭环**<br>自动化验证 |

# Gemini CLI：ReAct推理环与MCP扩展

## Gemini CLI核心定位

Gemini CLI通过Apache 2.0协议开源，其核心价值在于对Google Gemini 3模型族的高速访问与灵活的ReAct循环。它是Google AI编程布局的双极化之一，作为轻量化终端助手，与Antigravity形成互补。

**开源优势：**完全开源，社区驱动，可自由定制和扩展。适合需要透明度和可控性的开发者和企业。

## ReAct推理环

Gemini CLI具备自主行动闭环能力。它可以自主决定何时执行grep搜索文件，何时调用终端运行测试。这种ReAct（Reasoning + Acting）循环让AI具备自主决策能力。

- **思考（Reasoning）：**分析当前任务状态，推理下一步行动
- **行动（Action）：**执行工具调用（grep、终端命令等）
- **观察（Observation）：**观察执行结果，更新任务状态

## Google Workspace深度整合

### Google Docs

自动将代码变更总结同步到Google文档

### Google Sheets

从Sheet中读取配置参数，自动填充代码

### Gmail

读取邮件中的需求描述，自动生成代码

## 核心优势

- **完全开源：**Apache 2.0协议，自由定制
- **高速访问：**Gemini 3模型族极速响应
- **生态集成：**与Google Workspace深度整合
- **灵活ReAct：**自主决策，闭环执行

PART 02

# 核心工具深度剖析

1. IDE2.0
2. 国际巨头：基座模型的具象化
3. 中国力量：本土化与工程化创新
4. 开源派

# Trae：方法论驱动的端到端工程师

## 产品定位

Trae作为字节跳动推出的挑战者，强调"思维规划"先于"代码执行"，试图从AI助手进化为独立的"AI工程师（10x AI Engineer）"。

## SOLO模式

这是一个革命性的端到端开发模式。开发者只需提供一个高层次的需求（如PRD图片或文字），SOLO模式下的Agent会自主完成全流程开发。

**核心流程：**创建技术方案 → 生成数据库Schema → 编写API → 开发前端 → 终端运行 → 部署上线

## 核心优势

| 免费 | $10/月 |
| --- | --- |
| 当前定价 | Pro版 |

## 意图提取引擎

**多模态输入支持**

能够从多模态输入（如Figma截屏、PRD文档、文字描述）中直接反推业务逻辑

**业务逻辑推断**

不仅理解表面的需求描述，还能推断背后的业务逻辑和意图

**技术方案生成**

根据推断的业务逻辑，自动生成详细的技术方案

## 多模态增强

Trae允许用户将报错截屏、UI设计稿、甚至终端堆栈跟踪直接拖入对话。其上下文引擎支持图像到组件的直接映射，在UI还原度优化方面极具竞争力。

- 报错截屏自动诊断
- UI设计稿直接转代码
- 终端堆栈跟踪分析

# Trae技术特性（一）：SOLO模式与意图提取引擎

北京大学 PEKING UNIVERSITY

## SOLO模式技术架构

开发者只需提供高层次的需求（如PRD图片或文字），SOLO模式下的Agent会自主完成全流程开发。其核心是"意图提取引擎"，能够从多模态输入中直接反推业务逻辑。

**核心价值：**将开发者从繁琐的实现细节中解放出来，专注于需求定义和业务逻辑设计。AI工程师接管从需求到部署的全流程。

## SOLO模式工作流程

1. **创建技术方案**
   根据需求自动生成技术架构设计
2. **生成数据库Schema**
   自动设计数据库表结构和关系
3. **编写API**
   自动生成后端API接口
4. **开发前端**
   自动生成前端页面和组件
5. **终端运行 & 部署上线**
   自动运行测试并部署到生产环境

## 意图提取引擎

**多模态输入理解**

支持PRD文档、Figma设计稿、文字描述、语音输入等多种形式

**业务逻辑推断**

不仅理解表面需求，还能推断背后的业务逻辑和意图

**技术方案生成**

根据推断的业务逻辑，自动生成详细的技术方案

## 性能提升数据

| | |
|---|---|
| 开发效率提升 | 10x |
| MVP构建时间 | 分钟级 |

# Trae技术特性（二）：Builder模式与多模态增强

北京大学 PEKING UNIVERSITY

## Builder模式的Planning-first方法论

不同于Cursor的即时生成，Trae在执行复杂任务前，会先生成一份 详细的"施工计划书" ，分解为步骤、潜在风险及依赖项，待用户点击确认后再逐步执行。这种"Planning-first"的方法有效减少了复杂项目中的逻辑坏死。

**核心价值：**让开发者在AI执行前就能审视整个方案，发现潜在问题，避免走弯路。这种"三思而后行"的方法论，让复杂项目的成功率大幅提升。

## 多模态增强能力

Trae允许用户将 报错截屏、UI设计稿、甚至终端堆栈跟踪 直接拖入对话。其上下文引擎支持图像到组件的直接映射，在UI还原度优化方面极具竞争力。

- **报错截屏：**自动诊断错误原因并提供修复建议
- **UI设计稿：**图像到组件的直接映射，高保真还原
- **终端堆栈：**自动分析堆栈跟踪，定位问题根源

## 施工计划书示例

### 任务分解

1. 数据库Schema设计
2. API接口开发
3. 前端页面开发
4. 单元测试编写
5. 集成测试

### 潜在风险

- 数据库迁移可能导致数据丢失
- API版本兼容性问题
- 前端性能瓶颈

### 依赖项

- 需要访问数据库
- 需要配置API密钥
- 需要部署环境

# Cursor vs Trae：深度对比与哲学差异

| 评估维度 | Cursor Composer | Trae SOLO |
| --- | --- | --- |
| 核心机制 | 人机流式协同强化学习驱动，强调极致的交互反馈与修改精准度 | 智能体自主交付上下文工程驱动，强调从需求到部署的自主闭环 |
| 重构能力 | 擅长局部的精确重构，修改风格极度贴合人类习惯 | 擅长全新模块的从零构建，适合快速原型开发与大范围框架迁移 |
| 终端集成 | 直接且流畅。⌘+K 实时转换自然语言为终端命令，但有时会有快捷键冲突 | 间接集成。通过Chat介导生成命令，提供"Run"按钮，虽然步骤多一步但更安全稳健 |
| 自定义规则 | 极其强大的.cursorrules系统，支持基于文件模式的细粒度约束 | 相对集中的规则管理，依靠.trae/rules进行项目级全局设置 |
| 定价策略 | 较贵Pro $20/mo, Ultra $200/mo | 极具侵略性当前免费或极低费用（$3-$10/mo），旨在获取市场份额 |

### Cursor适用场景

- 日常逻辑编写与流式协同
- 局部精确重构
- 追求极致交互体验
- 已有项目的持续开发

### Trae适用场景

- 快速原型开发与MVP构建
- 从零构建全新模块
- UI高保真还原
- 大范围框架迁移

# Qoder（阿里巴巴）

## 产品定位

阿里巴巴推出的Qoder不仅仅是一个编辑器，它是一个垂直于企业级工程逻辑、强调整体上下文认知的深度编程平台。

## NES（Next-Edit-Suggestion）

Qoder最独特的技术亮点是"下一处编辑预测"。基于Location Model和Edit Model的双模型架构，NES能够预测开发者修改完函数A后，接下来的逻辑必然要修改文件B的第45行。

**实测数据：**在超过2万名开发者的蚂蚁金服内部应用中，预测准确率达**75.6%-81.6%**。这使得开发体验从"补全单词"跃升为"补全意图"。

## 核心优势

| 75.6% | 10万+ |
| --- | --- |
| NES预测准确率 | 文件索引支持 |

## Quest Mode（任务模式）

**半自主异步工作流**

用户提交一个Spec说明，Qoder会进入后台运行数小时，自主进行代码库检索、执行跨文件修改并自运行单元测试

**"委派任务给AI初级开发"**

这被形象地称为"委派任务给AI初级开发"，直到任务完成或遇到瓶颈才会通知开发者

**核心价值**

让开发者从繁琐的实现细节中解放出来，专注于需求定义和业务逻辑设计

## RepoWiki与知识可视化

Qoder能分析多达10万个文件的复杂项目，并自动生成结构化的知识图谱（Repo Wiki）。它能揭示代码中隐含的架构决策和技术债，通过"知识可视化"将黑盒代码库透明化，显著降低了新人的上手难度。

- 自动生成知识图谱
- 揭示架构决策和技术债
- 降低新人上手难度

# Qoder技术特性（一）：NES下一处编辑预测

## NES技术原理

基于Location Model和Edit Model的双模型架构，NES能够预测开发者修改完函数A后，接下来的逻辑必然要修改文件B的第45行。

**核心价值：**让开发体验从"补全单词"跃升为"补全意图"。开发者只需关注业务逻辑，AI自动推断下一步需要修改的代码位置和内容。

## 双模型架构

### Location Model

预测下一个需要编辑的位置（文件、行号）

### Edit Model

预测在该位置需要进行的编辑内容

## 实测数据

**75.6%**

最低预测准确率

**81.6%**

最高预测准确率

测试环境：蚂蚁金服内部应用
测试人数：超过2万名开发者

## 应用场景

- API接口批量修改
- 数据库Schema变更
- 组件属性调整
- 依赖库版本升级

# Qoder技术特性（二）：Quest Mode与RepoWiki

## Quest Mode（任务模式）

这是一个半自主的异步工作流。用户提交一个Spec说明，Qoder会进入后台运行数小时，自主进行代码库检索、执行跨文件修改并自运行单元测试，直到任务完成或遇到瓶颈才会通知开发者。

**"委派任务给AI初级开发"：**这被形象地称为"委派任务给AI初级开发"，开发者只需定义任务目标，AI自动完成实现。

## Quest Mode工作流程

1. **提交Spec说明**
   开发者定义任务目标和验收标准
2. **后台自主运行**
   AI进行代码库检索、跨文件修改
3. **自运行单元测试**
   自动运行测试验证修改正确性
4. **任务完成通知**
   直到任务完成或遇到瓶颈才通知开发者

## RepoWiki与知识可视化

**知识图谱生成**
能分析多达**10万个文件**的复杂项目，并自动生成结构化的知识图谱

**架构决策揭示**
揭示代码中隐含的架构决策和技术债

**知识可视化**
通过"知识可视化"将黑盒代码库透明化

**降低上手难度**
显著降低了新人的上手难度

## 性能指标

| | |
|---|---|
| 文件索引上限 | 10万+ |
| 知识图谱生成时间 | 分钟级 |
| 新人上手时间 | -70% |

# CodeBuddy（腾讯）：深植生态的闭环研发中枢

## 产品定位

腾讯的CodeBuddy（原名腾讯代码助手）通过 极致的云原生集成与生态连接 ，打造了差异化的竞争优势。

## Craft Mode（匠心模式）

这是CodeBuddy的核心Agent模式。它强调 "人类领航，AI辅助" ，通过理解整个工程上下文，帮助开发者完成复杂的业务逻辑拆解。

**端到端闭环：**从Figma设计稿转换 → i18n多语言自动化填充 → 单元测试自动补全

## 腾讯生态深度穿透

### 微信小程序支持

对微信小程序支持最完美，开发者可以在IDE内一键将代码部署到腾讯云

### 真机预览

直接在侧边栏预览小程序真机效果，极大降低企业级应用交付成本

### 一键部署

IDE内一键部署到腾讯云，无需切换工具

## 混合检索架构

CodeBuddy采用 向量搜索与代码依赖图（Code Graph）相结合 的混合架构。在处理具有深层嵌套引用的C++/Go大型项目时，其理解能力比纯向量检索工具更强，能够精准追踪全局变量的引用链条。

- 向量搜索：语义理解
- 代码依赖图：结构分析
- 混合架构：双重保障

# CodeBuddy技术特性：Craft Mode与生态穿透

## Craft Mode（匠心模式）

强调"人类领航，AI辅助"，通过理解整个工程上下文，帮助开发者完成复杂的业务逻辑拆解。

**端到端闭环：**从Figma设计稿转换 → i18n多语言自动化填充 → 单元测试自动补全。全流程自动化，极大提升开发效率。

## 端到端闭环流程

1. **Figma设计稿转换**
   自动解析设计稿，生成前端代码
2. **i18n多语言自动化填充**
   自动提取文案，填充多语言翻译
3. **单元测试自动补全**
   自动生成单元测试用例

## 腾讯生态深度穿透

**微信小程序完美支持**
对微信小程序支持最完美，是目前市场上对小程序支持最好的AI编程工具

**一键部署到腾讯云**
开发者可以在IDE内一键将代码部署到腾讯云，无需切换工具

**真机预览**
直接在侧边栏预览小程序真机效果，极大降低企业级应用交付成本

## 核心优势总结

- **生态集成：**与腾讯云、微信小程序深度集成
- **闭环流程：**从设计到部署的全流程自动化
- **混合检索：**向量搜索+代码依赖图双重保障
- **高性能：**92%任务完成率，120ms响应

PART 02

# 核心工具深度剖析

1. IDE2.0
2. 国际巨头：基座模型的具象化
3. 中国力量：本土化与工程化创新
4. 开源派

# OpenCode：架构灵活性与极致隐私

## 产品定位

面对闭源巨头的垄断，OpenCode通过开源力量，为开发者提供"主权级"的编程助手。

## 全栈开源与供应商解耦

OpenCode是一个基于Go语言编写的TUI（终端用户界面）工具。它的核心价值在于"供应商无关（Provider Agnostic）"。开发者可以在本地配置文件~/.opencode.json中挂载超过**75种大模型**，包括本地通过Ollama运行的私有化模型。

**核心价值：**摆脱对单一供应商的依赖，自由选择模型，确保数据主权。

## 核心优势

| 75+ | 100% |
| --- | --- |
| 支持模型数 | 开源 |

## Auto Compact机制

**上下文爆炸问题解决**

针对长对话中的上下文爆炸问题，OpenCode设计了"自动压缩"功能

**智能压缩策略**

当Token使用率达到模型窗口的**95%**时，系统会自动利用摘要模型对前文进行压缩

**无尽会话能力**

这种"无尽会话"能力使其在处理跨越数周的长程开发任务时表现稳定

## Charm生态支撑（Crush项目）

OpenCode已逐步演进为与Charm团队合作的Crush项目。它利用Bubble Tea框架提供了极具美感的终端交互体验，支持LSP（语言服务器协议）对接。

- 极具美感的终端交互体验
- 支持LSP（语言服务器协议）对接
- SSH环境下也能获得IDE级体验

# OpenCode技术特性：Auto Compact与Charm生态

## Auto Compact技术原理

针对长对话中的上下文爆炸问题，OpenCode设计了"自动压缩"功能。当Token使用率达到模型窗口的95%时，系统会自动利用摘要模型对前文进行压缩，并将关键状态注入新会话。

**压缩策略：**保留核心逻辑、关键决策点、重要变量定义；丢弃重复代码、过时的中间结果、已解决的错误信息。

## 无尽会话能力

这种"无尽会话"能力使其在处理跨越数周的长程开发任务时表现稳定。开发者无需担心上下文窗口不足的问题，可以持续与AI协作。

- ✓ 长程任务支持：跨越数周的开发任务
- ✓ 状态保持：关键状态注入新会话
- ✓ 透明压缩：压缩过程对开发者透明

## Charm生态支撑

**Crush项目**

OpenCode已逐步演进为与Charm团队合作的Crush项目

**Bubble Tea框架**

利用Bubble Tea框架提供了极具美感的终端交互体验

**LSP对接**

支持LSP（语言服务器协议）对接，代码智能跳转与重构

**SSH环境支持**

在SSH连接的纯终端环境下，也能获得不输于IDE的体验

## 性能指标

| | |
|---|---|
| 压缩触发阈值 | 95% |
| 压缩率 | 60-70% |
| 会话稳定性 | 99%+ |

# 隐私保护下的工程能力：Docker Model Runner

## 安全敏感行业的终极解决方案

在安全敏感行业，OpenCode配合 Docker Model Runner（DMR） 提供了终极方案。代码推理完全在企业内网容器内运行，彻底杜绝Telemetry（遥测数据）外泄的可能。

**核心价值：**虽然其推理速度略逊于云端服务，但其在"理解团队特有私有库"方面的灵活性无可替代。

## Docker Model Runner架构

1. **企业内网部署**
   DMR部署在企业内网服务器上
2. **容器隔离运行**
   代码推理在Docker容器内运行，完全隔离
3. **零Telemetry外泄**
   彻底杜绝遥测数据外泄的可能

## 隐私保护优势

**数据主权保障**
代码完全在企业内网处理，不会离开企业防火墙

**合规性满足**
满足金融、医疗、政府等行业的严格合规要求

**私有库理解**
在理解团队特有私有库方面的灵活性无可替代

## 权衡与取舍

**优势**
数据主权、合规性、私有库理解

**劣势**
推理速度略逊于云端服务

**适用场景**
金融、医疗、政府等安全敏感行业

PART 03

# 横向对比及选择逻辑

1. 核心指标与性能矩阵
2. 用户画像与选择逻辑
3. 进阶实战：Skills

# 核心工具深度对比（2026年2月版）

| 工具名称 | 核心定价模型 | 综合性能评级 | 上下文窗口 | SWE-bench | 最佳性价比画像 |
|---|---|---|---|---|---|
| **Claude Code** | $20-$200/月 (Max) | 极高 | 200,000 Tokens | **81.4%** | 资深架构师 / 严苛工程重构 |
| **OpenAI Codex** | $20/月 (Plus包含) | 高 | 192,000 Tokens | **80.0%** | ChatGPT重度用户 / 技能开发 |
| **Antigravity** | 免费 / $20 (Pro) | 极高 | 1,000,000+ Tokens | **76.2%** | 个人开发者 / 谷歌生态拥趸 |
| **Cursor** | $20/月 (Pro) | 高 | 200,000 Tokens | -- | 前端开发 / 追求极致协同感 |
| **Trae** | 免费 / $10/月 | 中高 | 128,000 Tokens | -- | MVP构建 / 字节系开发者 |
| **Qoder** | $20-$60/月 (Pro/+) | 高 | 100k Files Index | -- | 企业维护 / 蚂蚁/阿里生态 |
| **CodeBuddy** | 免费/￥58/￥78/￥158 | 高 | 272,000 Tokens | -- | 个人开发者 / 中小团队 |
| **OpenCode** | 免费 | 中高 | 256,000 Tokens | -- | 开源开发者 / 中小企业 |

**关于价格的提醒**

以上都是官方服务的价格，在国内有许多渠道商和中转代理商会提供更低的价格，但在速率、稳定性、数据隐私上会有不同的风险，可以自行选择

# 核心技术维度解析（一）：上下文窗口

## 从"切片"到"全景"

上下文窗口技术已经从早期的"代码切片"演进为"项目全景"理解。不同工具采用了不同的技术路线来解决上下文窗口的限制。

## 各工具上下文窗口对比

**Antigravity** — 1M+ Tokens

具备最强的原生长文本处理能力，无需复杂的RAG即可加载整个Monorepo

**Claude Code** — 200k Tokens

虽然窗口较小，但其"Agentic Search"搜索算法能够精准定位关键上下文，实际召回率排名第一

**OpenCode** — Auto-Compact

通过"自动压缩"技术，将无限长对话的状态转化为摘要，解决了物理窗口的限制

## 技术洞察

**大窗口≠强理解**

上下文窗口大小只是基础，关键是如何有效利用窗口内的信息

**精准召回更重要**

Claude Code虽然窗口较小，但精准召回率排名第一

**技术创新突破限制**

OpenCode通过Auto-Compact技术突破物理窗口限制

## 选型建议

- ✓ **大型Monorepo：** Antigravity（1M+窗口）
- ✓ **精准检索：** Claude Code（Agentic Search）
- ✓ **长程任务：** OpenCode（Auto-Compact）

# 核心技术维度解析（二）：响应速度

## 从"等待"到"流式即时"

响应速度是影响开发者体验的关键因素。从早期的"等待数秒"到现在的"流式即时输出"，AI编程工具的响应速度实现了质的飞跃。

## 各工具响应速度对比

### Cursor (Supermaven)

毫秒级

拥有行业最快的补全响应速度，通过边缘计算节点实现了毫秒级的Token输出，支撑了"Tab补全预测"的顺滑体验

### OpenAI Codex

+25%

GPT-5.3-Codex优化了推理架构，使其在处理复杂的Mid-turn Steering时依然能保持稳定的高吞吐量

## 技术洞察

**速度vs质量的权衡**

极速响应可能牺牲一定的准确性，需要根据场景权衡

**边缘计算优化**

Cursor通过边缘计算节点实现毫秒级响应

**架构优化提速**

Codex通过推理架构优化实现25%提速

## 选型建议

- ✓ **日常编码：** Cursor（毫秒级响应）
- ✓ **复杂任务：** Codex（高吞吐量）
- ✓ **离线环境：** OpenCode（本地模型）

# 核心技术维度解析（三）：生态兼容性

## 从"孤岛"到"万物互联"

生态兼容性是AI编程工具的重要竞争力。从早期的"孤岛式"工具到现在的"万物互联"，AI编程工具正在深度融入开发者的工作流。

## CodeBuddy（腾讯云原生）

实现了IDE与小程序、云函数的一键打通，其生态兼容性在中文垂直领域无出其右。

- ✓ 微信小程序完美支持
- ✓ 一键部署到腾讯云
- ✓ 真机预览

## Claude Code（MCP开放协议）

**跨平台中枢**

通过MCP协议，Claude成为真正的"跨平台中枢"，能调用Notion、Jira、Slack等一切外部工具

**设计到代码**

读取Figma设计稿，自动生成对应代码

**需求到实现**

读取Jira任务单，生成符合需求的代码

## 选型建议

- ✓ **腾讯云用户：** CodeBuddy
- ✓ **多工具协同：** Claude Code（MCP）
- ✓ **Google生态：** Antigravity/Gemini CLI

# 核心技术维度解析（四）：Agent能力与自主性

## 从"辅助"到"自主"

Agent能力是2026年AI编程工具的核心竞争维度。从早期的"辅助补全"到现在的"自主执行"，AI正在从工具进化为协作者。

## 各工具Agent能力对比

**Claude Code** — Plan Mode

提供缜密的任务规划与执行，适合复杂重构

**OpenAI Codex** — Skills系统

提供确定性执行，适合团队规范落地

**Trae** — SOLO模式

提供端到端自主交付，适合快速原型开发

## 技术洞察

**Agentic AI大规模应用**

Gartner预测，到2026年底，**40%**的大型企业将在CI/CD中集成AI代理

**从辅助到自主**

AI将像资深工程师一样独立工作，人类转向审查与架构设计

**人机协作新模式**

"智能体主导、人类审查"的新型生产范式正在形成

## 选型建议

- ✓ **复杂重构**：Claude Code（Plan Mode）
- ✓ **规范落地**：OpenAI Codex（Skills）
- ✓ **快速原型**：Trae（SOLO模式）

# 核心技术维度解析（五）：代码质量与安全性

北京大学 PEKING UNIVERSITY

## 从"生成"到"生产就绪"

代码质量与安全性是企业级应用的关键考量。从早期的"代码生成"到现在的"生产就绪"，AI编程工具正在不断提升代码质量。

## 代码质量对比

**Claude Opus 4.5** — 80.9%

在SWE-bench Verified以80.9%位居榜首，代码架构优雅，注释详尽，几乎无逻辑坏味

**Claude Code** — -30%

代码重做率比同类工具低30%

**OpenAI Codex** — 确定性

Skills系统确保执行过程极具确定性，不易产生逻辑漂移

## 安全性对比

**Claude Opus 4.5** — 4.7%

提示注入攻击成功率仅为4.7%，远低于竞品

**Gemini 3 Pro** — 12.5%

提示注入攻击成功率为12.5%

**GPT-5.1** — 21.9%

提示注入攻击成功率为21.9%

## 选型建议

- ✓ **高质量代码**：Claude Code（Opus 4.5）
- ✓ **高安全性**：Claude Code（4.7%攻击率）
- ✓ **确定性执行**：OpenAI Codex（Skills）

# 2026年技术趋势预测

## Agentic AI大规模应用

Gartner预测，到2026年底，40%的大型企业将在CI/CD中集成AI代理，AI将像资深工程师一样独立工作。

**影响：**开发流程重构，从需求到测试、从文档到运维，AI能力逐步嵌入全生命周期。

## 自愈测试成为标配

Self-healing不再是营销卖点，而是所有主流自动化工具的标准功能，flaky测试比例预计下降至历史最低。

**影响：**测试维护成本大幅降低，测试稳定性显著提升。

## 预测性质量工程

AI通过代码变更历史、缺陷模式、业务上下文提前预测高风险区域，实现主动缺陷预防而非事后找bug。

**影响：**从被动修复到主动预防，软件质量大幅提升。

## AI生成代码的专项验证

随着AI写出大量代码，验证AI生成代码成为QA主要工作，EU AI Act 2026年全面生效，合规性测试需求激增。

**影响：**QA角色转型，从测试执行者转向AI输出审核者。

**趋势总结：**2026年，AI编程工具的边界正在消失。这不再是一场关于谁的模型参数更大的竞争，而是一场关于谁能更深度地整合开发环境、更准确地捕捉开发者意图、更稳健地执行长程工程任务的综合竞赛。

PART 03

# 横向对比及选择逻辑

1. 核心指标与性能矩阵
2. 用户画像与选择逻辑
3. 进阶实战：Skills

# 用户画像分化：终端忍者 vs 氛围编码者

## 终端忍者

**Terminal Ninja / SPEC偏好型**

**核心特征**

追求极简、高性能，偏好CLI操作。习惯于先写SPEC（技术规范）再让AI执行

**首选方案**

**Claude Code + OpenCode**

在终端通过Plan Mode审核逻辑，利用MCP调用团队文档

**典型用户**

资深架构师、DevOps工程师、安全工程师、终端爱好者

## 氛围编码者

**Vibe Coder / Vibe偏好型**

**核心特征**

强调直觉、可视化反馈，偏好GUI IDE。追求"编码的心流"，让AI在影子空间处理杂活

**首选方案**

**Cursor + Trae**

利用Composer进行多文件预览，用Trae SOLO进行UI快速还原

**典型用户**

前端开发者、全栈开发者、UI/UX设计师、产品开发者

# 场景化推荐逻辑（一）：初创公司与企业维护

## 初创公司CTO

**推荐工具**

**Trae（Builder Mode）**

**推荐理由**

以最低成本和最高自主性快速产出MVP，将产品构思直接映射为生产代码

**核心优势**

- 免费或极低费用
- SOLO模式端到端交付
- UI高保真还原
- 快速原型开发

## 企业维护/老系统重构

**推荐工具**

**Claude Code（Opus 4.6）或 Qoder（RepoWiki）**

**推荐理由**

AI对遗留代码的理解深度和NES预测能力能显著降低逻辑破坏风险

**核心优势**

- 缜密的Plan Mode
- 代码重做率低30%
- RepoWiki知识图谱
- NES预测准确率75.6%+

# 场景化推荐逻辑（二）：非技术创业者与专业开发者

## 非技术创业者

**推荐工具**

**Antigravity（Managed Browser）**

**推荐理由**

通过自然语言描述和多模态截屏，由AI自主完成从前端到测试的全链路交付

**核心优势**

- 自然语言交互
- 多模态输入支持
- 浏览器子Agent闭环验证
- 免费或低价

## 专业开发者/架构师

**推荐工具**

**Claude Code + Cursor组合**

**推荐理由**

- Claude Code处理复杂架构重构与深度Bug诊断
- Cursor处理日常逻辑编写与流式协同

**核心优势**

- 复杂任务：Claude Code（Plan Mode）
- 日常开发：Cursor（Composer）
- 最佳实践组合
- 效率最大化

# 复合型工作流最佳实践（一）：极速辅助与深度重构

北京大学 PEKING UNIVERSITY

## 极速辅助

**推荐工具**

Cursor

**使用场景**

使用Cursor的Tab补全和Composer模式处理**80%**的日常逻辑编写，享受极致的流式协同

**核心优势**

- 毫秒级响应
- Tab补全预测
- Composer多文件预览
- Shadow Workspace自我修复

## 深度重构

**推荐工具**

**Claude Code**

**使用场景**

涉及跨**50个文件**的大型重构或复杂Bug诊断时，切换到Claude Code

**核心优势**

- 缜密的Plan Mode
- 检查点与原子化回滚
- 多智能体协作
- 代码重做率低30%

# 复合型工作流最佳实践（二）：自动化验收与内网主权

北京大学 PEKING UNIVERSITY

## 自动化验收

**推荐工具**

**Antigravity**

**使用场景**

使用Antigravity的浏览器子Agent进行**全自动化UI测试**，确保逻辑变更没有引发视觉回归

**核心优势**

- 浏览器子Agent
- DOM捕捉与视频录制
- Console实时分析
- 自动修复视觉Bug

## 内网主权

**推荐工具**

**OpenCode + Docker Model Runner**

**使用场景**

涉及**核心资产代码**时，在OpenCode中切换至本地模型，确保代码不外流

**核心优势**

- 完全开源
- 75+模型支持
- Docker容器隔离
- 零Telemetry外泄

# 成本效益分析与定价策略对比

| 工具名称 | 定价策略 | 目标用户 | 性价比评估 |
| --- | --- | --- | --- |
| Trae | 免费 / $10/月 | MVP构建 / 字节系开发者 | 极高 最具侵略性定价 |
| Antigravity | 免费 / $20 (Pro) | 个人开发者 / 谷歌生态拥趸 | 极高 免费期性价比最高 |
| OpenCode | 免费 | 开源开发者 / 中小企业 | 高 开源生态低成本首选 |
| Claude Code | $20-$200/月 (Max) | 资深架构师 / 严苛工程重构 | 高 企业级应用首选 |
| OpenAI Codex | $20/月 (Plus包含) | ChatGPT重度用户 / 技能开发 | 中高 适合已有Plus用户 |
| Cursor | $20/月 (Pro) | 前端开发 / 追求极致协同感 | 中高 体验优秀但价格较高 |
| Qoder | $20-$60/月 (Pro/+) | 企业维护 / 蚂蚁/阿里生态 | 中高 企业级功能丰富 |
| CodeBuddy | 免费/￥58/￥78/￥158 | 个人开发者 / 中小团队 / 企业 / 产设研全角色 | 中高 全场景研发适配首选 |

**$20-60**
主流工具月费区间

**免费**
Trae/Antigravity

**$200**
Claude code Max

# AI编程工具选型决策框架

## 问流程

我们是想 优化一个环节，还是 重塑整个开发生命周期？

**优化环节：** Cursor、GitHub Copilot
**重塑生命周期：** Claude Code、Trae、Antigravity

## 问生态

我们是否已深度绑定某个 云或代码托管平台？

**腾讯云：** CodeBuddy
**阿里云：** 通义灵码、Qoder
**Google Cloud：** Antigravity、Gemini CLI

## 问数据

我们的代码和数据 能否离开内网？

**可以离开：** Claude Code、Cursor、Codex
**不能离开：** OpenCode + Docker Model Runner

## 问团队

团队的 技术栈和技能偏好 是什么？

**终端偏好：** Claude Code、OpenCode
**GUI偏好：** Cursor、Trae、Antigravity

**决策建议：** 通过回答这五个问题，可以清晰地确定最适合自己团队的AI编程工具组合。记住，**没有最好的工具，只有最适合的工具。**

# 彩蛋：Claude Code 封号风险深度剖析与安全策略

北京大学 PEKING UNIVERSITY

## 核心封号诱因

### IP地址与网络环境（占比60%以上）

使用高风险的数据中心IP（如AWS、GCP机房IP）、免费VPN或频繁切换地理位置。Anthropic的风险控制系统会对短时间内跨越多个国家的IP跳跃进行实时拦截

### 第三方封装工具（Harnessing）

2026年1月，Anthropic封禁了大量使用第三方工具（如OpenCode）模拟Claude Code客户端行为的账号。这种行为被视为绕过订阅限制的自动化滥用 。

### 账户关联与多设备登录

在同一台设备上登录多个账号，或5-10人共享一个Pro账号，会导致设备指纹被标记为异常，触发批量封禁 。

### 支付信息异常

使用虚拟信用卡（Virtual Card）或代付服务，且账单地址与登录IP严重不符，会触发反欺诈审核 。

## 避坑指南：如何维持账号稳定性

### 网络环境

必须使用固定位置的“原生住宅IP”（Native Residential IP）。开启全局代理模式并禁用WebRTC以防止本地真实IP泄露 。

### 设备环境

使用主流浏览器（Chrome/Safari），将系统语言设为英文，并确保系统时区与IP地址完全匹配。建议使用指纹浏览器（如AdsPower或Multilogin）为每个账号创建独立的执行环境 。

### 行为模式

模拟正常人类的操作节律，避免24小时不间断的高频请求。在支付环节，首选实体国际信用卡，并确保账单地址的真实性 。

PART 03

# 横向对比及选择逻辑

1. 核心指标与性能矩阵
2. 用户画像与选择逻辑
3. 进阶实战：Skills

# Agent Skills 的技术本质与架构原理

## 超越函数调用的元工具架构

**传统函数调用**

LLM生成描述API调用的JSON对象，由外部程序执行

Function Calling

**Agent Skills：技能胶囊**

不仅包含执行逻辑，还包含指导Agent应对场景、处理错误、验证输出的详细指令集

Skill Capsule

## 核心特征对比

| 特征维度 | 传统函数调用 | Agent Skills |
| --- | --- | --- |
| **表现形式** | 单次、结构化的API请求/响应 | 包含指令、脚本、参考资料的文件夹 |
| **加载模式** | 静态硬编码在系统提示词中 | 动态、按需加载 |
| **知识深度** | 仅限于参数定义和简单描述 | 包含多步骤工作流和领域特定知识 |
| **环境交互** | 通常是无状态的单次调用 | 支持复杂文件系统有状态操作 |
| **可移植性** | 深度耦合特定平台接口 | 通过MCP等协议实现跨代理通用 |

**渐进式披露原则**

Agent初始阶段仅感知技能元数据（名称和描述），仅在决策链确定相关时才加载完整指令集，极大优化Token使用效率，防止上下文窗口过载导致的推理能力下降

# 技能包核心构成与标准化目录结构

北京大学 PEKING UNIVERSITY

## 1 SKILL.md
Core Instruction File

**YAML前置数据**
定义技能唯一标识、触发描述、所需权限、推荐模型

**Markdown主体**
详细说明执行步骤、输入输出范式、边缘情况处理

**技能的灵魂所在**

## 2 /scripts
Executable Scripts

包含实现确定性逻辑的Python、Bash或JavaScript脚本

**示例：**数据库迁移技能包含生成SQL差异、验证架构完整性的脚本

## 3 /references
Reference Materials

存储API文档、架构图或编码规范

**外部记忆：**执行复杂任务时提供实时Ground Truth支持

## 4 /assets
Assets Directory

存放模板、二进制文件或用于验证的资源，确保技能在不同环境下的可复现性

### 标准化目录结构示意

```
my-skill/
  ├── SKILL.md
  ├── /scripts
  │   ├── migrate.py
  │   └── validate.sh
  ├── /references
  │   ├── api-docs.md
  │   └── architecture.png
  └── /assets
      └── template.sql
```

# 进阶实战：典型开发场景下的 Skills 配置

北京大学 PEKING UNIVERSITY

## 需求撰写与分析
Requirements

**Prd**

生成高质量产品需求文档，包含执行摘要、用户故事、技术规格和风险分析

**prd-doc-writer**

梳理/撰写/完善PRD、用户故事、验收标准，使用ASCII线框图与Mermaid减少歧义

**将抽象需求转化为可落地执行的Agent任务逻辑与交互规则**

## 代码审查与测试
Code Review & Testing

**code-quality-review**

全面、系统的代码审查框架，平衡技术严谨性和建设性反馈技巧

**database-testing**

提供数据库模式验证、数据完整性测试、迁移测试、事务隔离和查询性能示例

**condition-based-waiting**

用条件轮询替代测试中任意超时，解决不稳定测试问题

**包含自动生成复现脚本、配置临时测试容器、分析堆栈跟踪的能力**

## 基础设施与部署
DevOps & Infrastructure

**gh-cli**

使用GitHub CLI进行仓库操作、PR管理、问题管理、工作流和代码空间管理

**docker-containerization**

Docker多阶段构建、编排和容器安全专业知识，包含Python/Next.js特定模式

**database-administrator**

PostgreSQL、MySQL、MongoDB和Redis的专家级数据库管理

**自主完成证书生成、镜像构建或多分支合并冲突解决**

需求撰写与分析Skill来源：**https://skills.sh**

代码审查与测试及基础设施与部署Skill来源：**https://skillhub.club**

# Skills 带来的质变：从指令执行到目标达成

北京大学 PEKING UNIVERSITY

## 范式跃迁

**传统模式** Single Mapping

"单次映射"过程：输入自然语言，输出文本代码。缺乏反馈回路，无法应对执行过程中的动态变化

↓

**Agent Skills 模式** Closed-loop Iteration

- 闭环迭代：Agent提出计划、执行操作、观察环境反馈、修正错误并最终确认结果

**Skills的本质，是一个可执行的SOP**

- 过去是由人类定义详细步骤，并以软件的方式由CPU运行
- 现在是由人类定义目标和大致步骤，并以自然语言的方式由LLM自主执行

## 三个维度的质变

### 1 自主性 Autonomy

开发者不再需要给出详细实现步骤，而是给出目标。Agent通过Skills自主规划任务路径

**示例：**"重构API层以支持水平扩展"而非"在第45行添加if语句"

### 2 工具增强型推理

Agent不再孤立地进行逻辑推理，而是利用外部工具（编译器、调试器、Web搜索）验证假设

### 3 状态感知 State Awareness

通过Skills，Agent能够感知终端实时状态、进程运行情况以及文件系统变化，保持长时间跨度任务的连贯性

**核心洞察**

Agent Skills的介入将软件开发从"人类编写代码"转变为"人类定义目标，AI自主执行"。开发者角色从实现者转变为规划者和监督者，释放创造力专注于更高层次的设计和架构决策

# 原子级 TODO 管理：7×24 小时不间断工作的引擎

北京大学 PEKING UNIVERSITY

## Master TODO 全局任务列表

接收到复杂Feature请求时，Agent首先调用规划技能，生成涵盖全生命周期阶段的Master TODO

1. 需求确认与分析
2. 方案设计与评审
3. 编码实现与测试
4. 部署上线与监控

## 原子任务的四大特征

- **单一焦点:仅完成一个明确的功能**
- **自包含性:包含执行所需的全部上下文信息**
- **独立可测：可在隔离环境下验证正确性**
- **限时完成：通常设定2小时内执行时长，便于监控和回滚**

## 多套 TODO 的动态管理与波次执行

### 1 第一波次

处理无依赖的基础任务

**示例：**脚手架搭建、配置初始化

可并行执行

### 2 后续波次

基于前一波次产出进行集成

**特点：**依赖驱动、顺序执行

确保依赖满足

### 3 自我修复

原子任务失败时自动触发

**机制：**错误分析、动态调整、重新生成

持续向目标推进

# Terminal-Bench 中的表现与数据分析

北京大学 PEKING UNIVERSITY

## Terminal-Bench 2.0

The Gold Standard

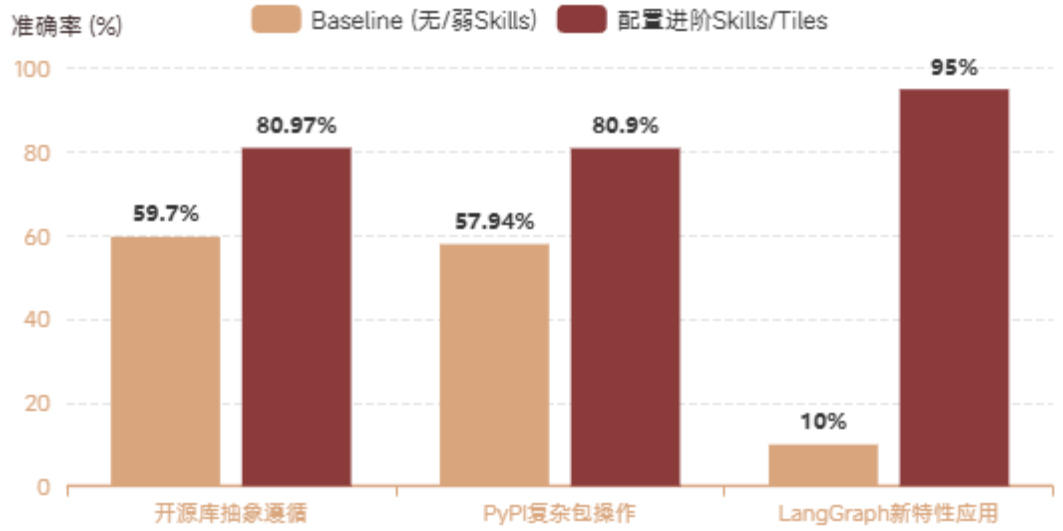
权威基准测试，模拟真实电脑终端环境，要求Agent解决89个硬核任务挑战

**长时程 Long-horizon**
需要几十甚至上百步操作

**交互性 Interactive**
与运行中的进程、数据库实时交互

**强约束 Strict Constraints**
对执行时间、资源占用严格要求

## 核心提升率：跨越40%效能鸿沟




## 深度洞察：为什么 Skills 能大幅提升表现?

**1 消除语法幻觉**

Skills提供的脚本工具替代了Agent记忆中的模糊API，确保命令执行的确定性

**2 增强环境感知**

非阻塞式执行和状态观察工具防止Agent在后台任务运行时处于"盲目等待"状态

**3 缓解遗忘效应**

渐进式披露机制确保任务关键阶段最相关的参考资料始终处于上下文高权重区域

# PART 04

# 未 来 与 展 望

1. 存在的风险
2. 未来趋势
3. 超级个体

# 隐私与合规风险

**核心挑战：** Agentic Coding工具的核心竞争力在于对长程任务的处理能力，而这种能力建立在对整个项目代码库、API文档、系统架构乃至内部业务逻辑的深度感知之上。商业化代理必须将大量专有源代码和敏感上下文上传至云端进行推理，这种"强依赖云端"的运行模式引发了全球范围内关于隐私与合规的剧烈讨论。

## 合规维度对比：商业云端代理 vs 开源/本地代理

| 合规维度 | 商业云端代理(如 Claude Code) | 开源/本地代理(如 OpenCode) | 风险等级 |
|---|---|---|---|
| **数据流向** | 代码与上下文上传至供应商云端 | 物理隔离，代码保留在本地/专有云 | 高风险 |
| **模型训练** | 存在输入数据被用于训练的风险 | 用户拥有数据主权，模型在闭环中运行 | 中/高风险 |
| **安全审计** | 依赖供应商提供的黑盒合规证明 | 支持透明审计，可导出 AI-BOM | 可控 |
| **访问控制** | 依赖 OAuth 和云端身份验证 | 集成企业级本地权限管理系统 | 存在风险 |

### 典型安全事件

**Salesloft与Drift事件（2025年8月）：** 攻击者利用被盗的OAuth令牌，绕过多重身份验证（MFA），直接访问了超过 **700家** 组织的内部客户环境。这种基于AI代理接口的攻击，无需复杂的漏洞利用，仅仅通过对代理权限的滥用即可实现大规模渗透。

### 合规风险升级

**法律环境收紧：** 2026年初，加利福尼亚州和纽约州相继出台了针对AI内容透明度和安全协议的严格法案，要求企业必须对AI生成的代码进行强制性的安全审计和来源追踪。Vibe Coding的盛行引入了隐蔽的合规风险，AI往往会引入未经审核的第三方库或不符合特定行业标准的数据处理逻辑。

# 性能瓶颈

北京大学 PEKING UNIVERSITY

23%

## 性能"断崖式下跌"的现实

2026年第一季度工业级测评显示：行业顶尖代理工具在公共仓库测试集中成功率普遍超过 **70%**，但在面对完全未知的私有企业代码库（SWE-bench Pro）时，成功率仅维持在 **23%左右**。这种性能下降揭示了AI代理对"隐性知识"的捕捉无能。

### 顶尖代理工具性能对比：公开库 vs 私有库

| 模型与工具 | 公开库成功率(SWE-bench Verified) | 私有库成功率(SWE-bench Pro) | 性能缩水率 |
|---|---|---|---|
| **Claude Opus 4.6** | 81.4% | 23.1% | 71.6% |
| **GPT-5.3 Codex** | 75.4% | 23.3% | 69.1% |
| **Trae (字节跳动)** | 75.2% | ~25.0% | 66.8% |
| **Qwen3-Coder-Next** | 70.6% | 44.3% | 37.3% |

### 什么是"隐性知识"?

- 企业内部未成文的架构约定
- 特定的历史代码补丁逻辑
- 特定业务场景下的异常处理逻辑

### Qwen3的突破意义

尽管Qwen3等国产模型在私有库迁移和工程适配上做了大量强化，显著收窄了性能缺口，但 **23%左右的平均成功率** 仍然意味着人类开发者在近 **80%的复杂场景** 下仍需进行深度的介入和修复。

# 代码维护困难

## AI辅助开发的悖论

研究表明，虽然AI提高了代码生成的初速度，但其生成的代码在 **质量一致性和长效可维护性** 方面存在严重缺陷。

开发者在使用代理工具时，往往陷入了一种 **"快速生成、快速失败、快速修复"** 的病态循环中。

## AI Slop（AI废料）

AI在修复Bug时，可能会为了局部通过测试而引入不合理的依赖或破坏全局的一致性原则。这种"拆东墙补西墙"的行为在自主运行数小时的代理会话中会不断积累，最终形成代码堆积。

## 代码质量指标对比：传统开发 vs AI辅助开发

| 质量指标 | 人类开发(2024前) | AI开发(2026) | 变化趋势 |
|---|---|---|---|
| **代码变更率 (Code Churn)** | ~15.0% | 41.0% | 显著恶化 ↑ |
| **重构活动频率 (Refactoring)** | 较低 | 较高 | 架构质量下滑 ↓ |
| **代码重复率 (Duplication)** | ~8.3% | 12.3% | 冗余度 +48% ↑ |
| **交付稳定性 (Stability)** | 稳定 | 下降 7.2% | 线上事故隐患 ⚠ |

## 维护成本飙升

到了2026年第一季度，许多企业发现其维护成本已经飙升至传统模式的 **四倍之多** 。

人类开发者需要花费 **数倍的时间** 来阅读、理解并重构那些由AI生成的、语义贫瘠且逻辑碎片化的代码。

# 技术依赖与创新能力萎缩

## 关于"开发者核心竞争力"的哲学困境

当 **How（如何实现）** 被 AI 彻底接管时，**人类是否正在丧失理解 How 的能力?**

### 角色转变

随着开发者角色向 **Reviewer（审核者）** 和 **Commander（指挥官）** 转变，初级开发者在成长过程中极大地减少了对复杂算法、内存管理、并发机制等底层知识的磨练。

### 调研数据

2026年的一项针对北美的工程团队调研显示，能够**独立手写复杂平衡二叉树**或**处理高并发死锁问题**的初级工程师比例大幅下降。

### 脆弱性

这种"技能萎缩（Atrophy）"导致在AI无法解决的极少数核心技术挑战面前（即那**23%的私有库瓶颈**之外），整个团队的创新能力显得极其脆弱。

### 技术直觉变钝

过度依赖 Vibe Coding 可能导致开发者的技术直觉变钝。当一个功能可以通过"描述意图 + 视觉确认"来实现时，开发者往往**不再关心代码背后的实现效率和安全性约束**。这种"黑盒式参与"剥夺了人类从失败和调试中学习的机会。

### 模型坍塌威胁

所谓的"模型坍塌"威胁也延伸到了软件工程领域：当互联网上充斥着大量由第一代编程AI生成的代码，而第二代AI又基于这些**可能带有隐蔽缺陷的代码**进行训练时，生成的代码质量将不可避免地陷入**平庸化的螺旋**。

# PART 04

# 未 来 与 展 望

1. 存在的风险
2. 未来趋势

   —— 构建以意图为中心的软件生态
3. 超级个体

# 多智能体协作（Agent Swarms）

## 2026年最显著的技术转向

**多智能体架构（Agent Swarms）取代了单一庞大模型驱动的单代理流程。这种范式的转变源于对处理效率和专业度的极致追求。**

### 单代理 vs 多智能体集群：核心维度对比

| 协作模式 | 2025年单代理流程 | 2026年多智能体集群 (Agent Swarms) |
|---|---|---|
| **执行逻辑** | 线性/顺序执行，效率较低 | 异步/并行执行，延迟降低 4.5倍 |
| **记忆管理** | 单一庞大上下文窗口，易产生幻觉 | 任务特化局部上下文，高精度推理 |
| **容错性** | 单点故障，推理中断则全盘失败 | 互助纠错，具备自动回滚与重试机制 |
| **角色分配** | 通才模型试图处理一切 | 领域专家集群 (Security, QA, DevOps) |

### 分布式职能架构

通过一个中心化的**"编排器（Orchestrator）"**或**"管理代理（Supervisor Agent）"**，将大任务分解为多个并行的子任务，并交由具有专业领域知识的子代理执行。

**云原生应用重构示例：**

① API合约定义代理 ② 单元测试编写代理 ③ 安全扫描代理 ④ 集成与冲突解决代理

### 博弈与协作：Google Antigravity

AI代理之间甚至出现了**"博弈与协作"**的关系：

- 一个代理**生成代码**
- 另一个模拟真实用户的**"测试代理"**通过浏览器实例进行端到端攻击
- 两个代理**不断迭代**，直至代码达到预设的健壮性标准

# 商业化与生态发展

随着 Agentic Coding 的成熟，**软件工程的商业版图正在重构**。2026年，企业级服务呈现出三大显著特征：

## 01 企业私有化服务的指数级增长

由于隐私风险，大型企业不再订阅公共云端的 Copilot，而是倾向于采购如 **CodeBuddy、Qoder** 等提供**"本地隔离 + 国密加密"**的企业级解决方案。

这种方案不仅提供编码辅助，更深度集成了企业的**研发 Wiki、私有文档库和 CI/CD 流水线**，实现了真正意义上的"工程标准对齐"。

## 02 从按人头计费到按价值/Token计费

在 Agentic 时代，AI 代理可以 **24/7 不间断工作**。一些顶尖工具如 Claude Code 已经开始尝试按**任务复杂度和消耗的推理算力**定价，而非单纯的月费订阅。

对于企业而言，评价 AI 的 KPI 不再是**"有多少人用"**，而是**"自动通过了多少次代码审计"**或**"减少了多少工时的技术债"**。

## 03 生态系统的垂直化

2026年初，行业开始出现专门针对特定领域的**特化代理平台**：

- **金融交易**：FinRobot
- **医疗数据处理**：Claude for Healthcare
- **高性能计算**：HPC Agent Platform

这些平台内置了该行业特有的**合规知识库和最佳实践**，使得代理在特定领域的表现远超通用模型。

# 开源社区与共享经济：Skill 市场的崛起与标准化

## 开源社区的角色转变

开源社区在 2026 年的角色从**"共享代码"**转变为**"共享能力（Skills）"**。

## MCP 协议通用化

随着 **Model Context Protocol (MCP)** 等协议的通用化，开发者可以将复杂的自动化工作流封装为**可插拔的 Skill**。

**Skill 定义**：一段可被代理理解并调用的指令集和工具链

## 社区生态数据

**数千个**
社区构建的技能

**ClawHub**
开源Skill注册中心

## Skill 示例：React 项目性能优化 Skill

1. **识别逻辑**
   识别组件冗余渲染的逻辑
2. **修复模板**
   自动生成代码修复方案的模板
3. **验证脚本**
   验证性能提升的脚本

## 降低"超级个体"的准入门槛

这种共享经济模式正在降低"超级个体"的准入门槛。一个独立开发者**不再需要精通全栈的每一个细节**，他只需要：

① 订阅几个成熟的开源技能包

② 由一个强大的代理负责协调执行

③ 即可在**几小时内**构建出以往需要整个团队**数月**才能完成的复杂 SaaS 系统

**无缝迁移**：支持代理在不同项目间迁移其掌握的专业能力　跨项目复用　能力标准化

# 政策与合规驱动：从"野蛮生长"到"算法透明"

## 合规要求成为最高优先级功能

**随着全球主要经济体完成 AI 监管立法的首轮实施，软件开发生命周期中的 AI 参与已受到法律的严密审视。合规要求将成为 2026 年之后 Agentic Coding 工具的最高优先级功能。**

### AI-BOM 强制化

核心驱动因素之一是 **AI-BOM（AI 物料清单）**的强制化。就像食品标签标注成分一样，企业发布的每一段 AI 生成代码，未来可能都需要附带一个**元数据包**：

- 使用了**哪个模型**
- 基于**哪些数据集**
- 是否经过**安全扫描**
- **人类**在其中的参与比例

**目的：**不仅是为了追溯 Bug，更是为了应对日益复杂的**版权纠纷和知识产权审查。**

### "可解释性"工具发展

监管还推动了**"可解释性"**工具的发展。诸如 **Apiiro、Semgrep** 等安全工具在 2026 年集成了 AI 上下文感知功能：

**传统模式：**仅仅告诉开发者代码哪里错了

↓

**新模式：**根据代理的逻辑推演路径，解释为什么 AI 选择了这种特定的（可能是不安全的）实现方式

### 从"黑盒生成"到"白盒审计"

这种转变是 AI 编程进入**医疗、交通、金融**等命脉行业的**先决条件**。

医疗　交通　金融

# PART 04

# 未 来 与 展 望

1. 存在的风险
2. 未来趋势
3. 超级个体

# 组织重塑：从流水线到超级个体

## 过去十年

我们习惯了**产品经理、UI设计师、架构师、前后端开发、测试、运维**这一长串且臃肿的流水线。

**本质原因：**"单人认知上限无法覆盖所有技术细节"

## 2026年

那个被切碎的"**全栈**"概念在 AI 的加持下**完美愈合**了。

**最先进的组织形式：**不再是数千人的庞大研发中心，而是由若干个"**超级个体（Super-Individuals）**"构成的灵活联盟

## 超级个体的能力边界

### 深厚业务理解力

具备对业务场景的深刻洞察

### 操控代理集群

通过AI代理完成过去一个完整小组的工作

### 效率提升

极大地缩减沟通成本，消除信息损耗

## 组织形态转变的核心价值

这种组织形态的转变，将**极大地缩减沟通成本**，**消除传统协作中的信息损耗**。一个具备深厚业务理解力的人，通过操控代理集群，就能完成过去一个完整小组的工作。

# 人的核心竞争力：从"How"回归"What"和"Why"

北京大学 PEKING UNIVERSITY

## 人类核心价值的史无前例收敛

当代码生成（**How**）变得像空气一样廉价时，人类的核心价值正在进行史无前例的收敛。未来的开发者，其首要任务**不再是手写代码**，而是**"定义问题"**。

### 01 定义 What

**解决什么问题?**

深刻洞察商业痛点，将**模糊的需求**转化为**精确的机器意图**。

**关键能力**：需求分析、用户洞察、商业敏感度、问题抽象

### 02 判断 Why

**为什么这么做?**

在 AI 提供的无数种可能性中，基于**美学、社会伦理、商业战略和人类共情**做出最终决策。

**关键能力**：价值判断、伦理考量、战略思维、人文关怀

### AlphaGo 之后的围棋界

正如 AlphaGo 之后的围棋界，人类**不再研究死记硬背的定式**，而是**研究 AI 的思路**，并以此提升人类对"美"和"策略"的更高阶理解。

### "人人都是产品经理"

在未来不再是一句口号，而是**开发者生存的最低门票**。开发者必须从执行者转变为思考者和决策者。

# 狂欢与失落

北京大学 PEKING UNIVERSITY

## 2025年11月之后的100天内，AI Coding已经成为主流共识，整个软件开发行业都在发生价值、流程、人才的地震

- 软件代码开发和功能实现变得廉价，团队的岗位界限变得模糊，所有岗位都在跨界，都能生产代码，都对产品负责，都是builder，软件该如何生产、使用、废弃？
- 软件工程的主要目的是为了沟通和控制，当人与人的交互变少，哪些流程和动作会变得多余，甚至成为生产的瓶颈？
- 类似于手工农业进展到了机械化农业，软件开发从人力密集型产业变成了机械化产业，对从业人员的要求产生了哪些变化？成本结构发生了哪些变化？

## 01 狂欢的人群

1. **软件开发行业的项目经理、CTO、CEO**：会发现Coding Agent好用又便宜，技术团队可以大幅度缩小，很多不必要的项目管理缓解和动作变得多余，不用再花大量的时间管理人类Coder，Token烧得越多，项目成本越低。其中，很多多年不再手写代码的资深技术管理者也会重新具备代码生产的能力，而且效率远高于过去。
2. **软件开发行业的架构师**：会发现不需要和业务经验和理解力不足、技术经验和理解力不足的初级和中级软件工程师打交道了，极大节省了时间、精力、成本的消耗。
3. **软件开发行业的产品经理**：会发现如果仅仅是为了验证产品，仅仅是做一个MVP，就不需要和业务经验和理解力不足的技术团队打交道了，极大节省了时间、精力、成本的消耗。
4. **没有软件开发技术经验的普通人或公司**：会发现自己的创意更容易实现，简单的创意或者创意的Demo不必找一个软件技术团队才能实现，利用AI Coding工具，自己就可以低成本快速实现，就如同使用Office软件一样简单。

## 02 失落的人群

1. **正在找工作的中级初级软件工程师，以及尚未进入软件开发行业的学生**：会发现自己具备的能力是在传统的、没有AI Coding Agent工具的环境中才有效，不适应新的软件开发范式要求（程序员不再手写代码甚至很少读代码，人与Agent协作开发，人只负责架构、决策和编排）。企业招聘软件开发人员的要求在最近这100天里发生了巨大的变化，AI Coding能力和经验成为了必备的要求。
   - **应对方案1**：尽快成为某个开发团队的一员，快速学习并共同成长。
   - **应对方案2**：如果对自己的创意和商业能力有信心，可以尝试成为一人公司（OPC）
2. **目前已经在软件开发团队中工作，但只专注于按照需求说明做技术实现，缺乏领域业务经验和兴趣，缺乏产品意识和用户思维的软件开发工程师**：会发现自己的软件实现能力正在快速地被AI Coding Agent工具替代和超越，失去工作是迟早的事。
   - **应对方案1**：尽快在现在的项目中熟悉和使用AI Coding Agent工具（尽早达到每天可以消耗1亿token的目标），尽快学习业务领域经验，提升自己的产品意识和用户思维。
   - **应对方案2**：退休，或者退出软件开发行业

# 致敬与展望：谁拿到了船票？

## 2026年：奇点时刻

2026年是 Agentic Coding 的元年，也是"**一人公司（One-person Company, OPC）**"走向 **10 亿美元估值** 的起点。

## 致敬

在这个奇点时刻，我们对**过往几十年的传统编程技艺**致以敬意。

"那是人类智慧的坚实基石"

每一行精心编写的代码，每一个深夜调试的 Bug，每一次架构设计的权衡，都是人类工程师智慧与汗水的结晶。

## 展望

同时，我们也以**前所未有的勇气遥望未来**。

- **代码正在隐形**：成为像汇编语言或二进制一样的底层细节
- **人类正在解脱**：从繁重的体力编码中解放出来
- **回归创造力本质**：专注于思考、设计和创新

## 谁拿到了通往未来的船票？

在这个时代，**谁先理解了这一角色转换**，**谁先掌握了如何指挥那支由 AI 代理构成的"数字化军团"**，谁就拿到了通往未来的船票。

理解角色转换 **从执行者 → 指挥官** + 掌握代理军团 **指挥 AI 代理集群** = 未来船票 **通往新纪元**

# 结束语

北京大学 PEKING UNIVERSITY

“2026年，Agentic Coding开启了Vibe Coding新纪元，AI从辅助工具进化为自主代理，重塑人机协作模式。开发者正从代码编写者转变为超级个体，以Vibe Coding的灵动与SPEC Coding的严谨，驾驭Agent集群。

技术迭代虽伴生隐私、维护等挑战，但多智能体协作、合规透明化等趋势已明确。未来，人类核心价值将回归“定义问题”与“价值判断”。愿每位开发者把握机遇，在人机共创中解锁无限可能，共赴编程新未来。”

# AI时代的思考

人是世界的尺度，活在意义之网中，人工智能让这张网更有价值

人类需要的是判断力和表达力，不再是记忆力和知识储备

人是目的，不是手段，不要去和人工智能比工具性

使用人工智能的人淘汰不使用人工智能的人

使用人工智能的组织淘汰不使用人工智能的组织

人工智能时代的策略：把握原理、躬身入局、随时否定自己

# 感谢各位老师和同学的批评指导

# 欢迎会后沟通交流

AI肖睿团队

微信扫描二维码，关注我的帐号

AI肖睿团队＠助理

扫一扫上面的二维码图案，加我为朋友。