



基于大语言模型智能体的代码生成综述

汇报人：董益宏

单位：北京大学计算机学院

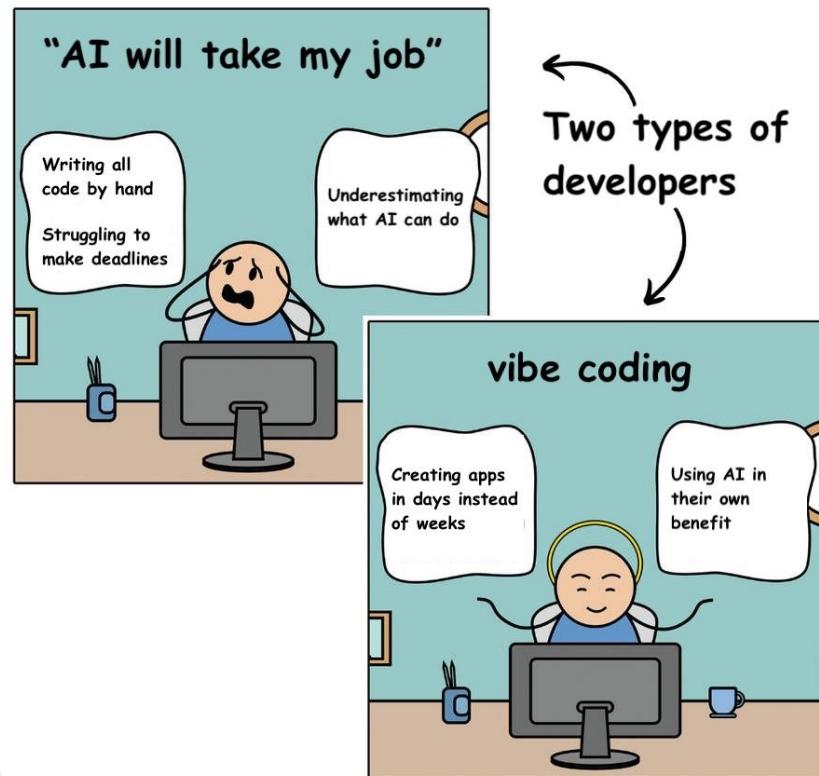
邮箱：dongyh@stu.pku.edu.cn



研究背景

➤ 氛围编程 (Vibe Coding)

- 由 OpenAI 联合创始人 Andrej Karpathy 于 2025 年 2 月在社交平台 X 中提出

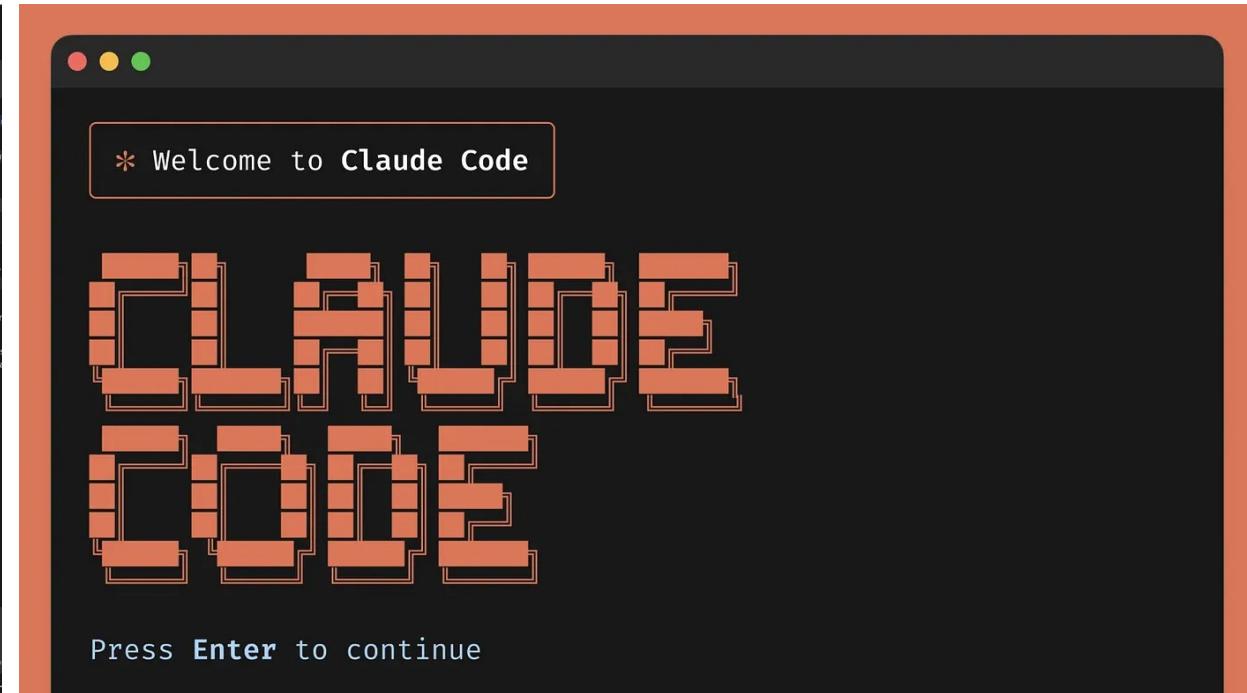


我把一种新的编码方式称为“氛围编码”，也就是说，让人完全沉浸于氛围，拥抱指数级增长，甚至忘记代码的存在。这可能是因为LLM（例如 **Cursor Composer w sonnet**）正变得越来越强大.....代码越来越长，超出了我通常的理解能力范围，我不得不花一段时间仔细阅读。有时，LLM 无法修复某个错误，我就需要设法解决或要求随机更改，直到它消失为止.....对于一次性的周末项目来说，这还不算太糟糕，但仍然相当有趣。

——Andrej Karpathy

研究背景

A screenshot of a code editor interface, likely VS Code, showing a file named `TS routes > webhook U`. The code is related to a Stripe checkout session. A large watermark 'Cursor AI' is overlaid on the bottom left. The right side of the screen shows a terminal window with a chat interface from Claude Code, displaying a message about debugging code.



核心

AI编程智能体 (Agent)
又称：代码生成智能体

➤ 代码生成智能体的三大特征

- 自主性：独立执行从任务分解、编码到调试的完整工作流
- 任务范围扩展：从生成代码片段扩展至覆盖软件开发全生命周期
- 工程实践性：重心从算法创新转向流程管理、系统可靠性与工具集成等工程挑战

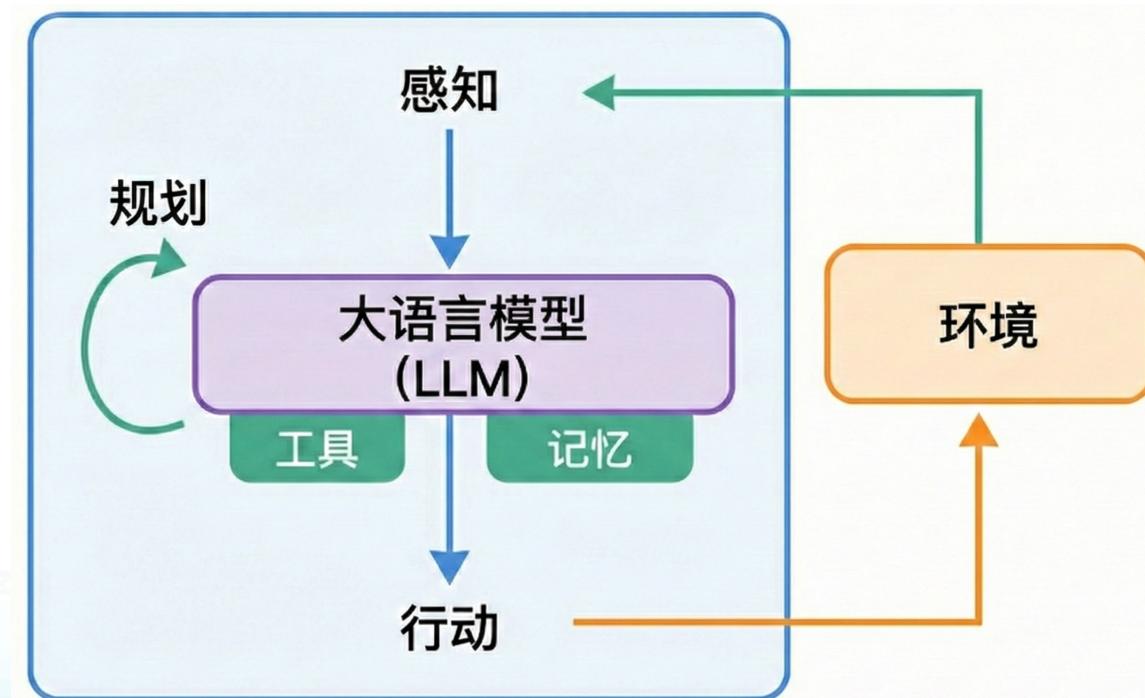
➤ 代码生成智能体的三大基本能力

- 规划
- 工具
- 记忆

智能体中最重要的能力是什么？



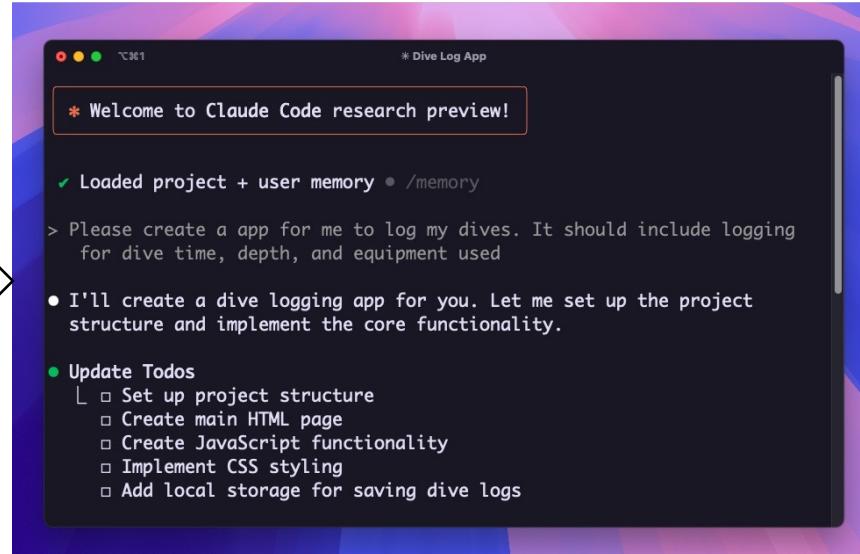
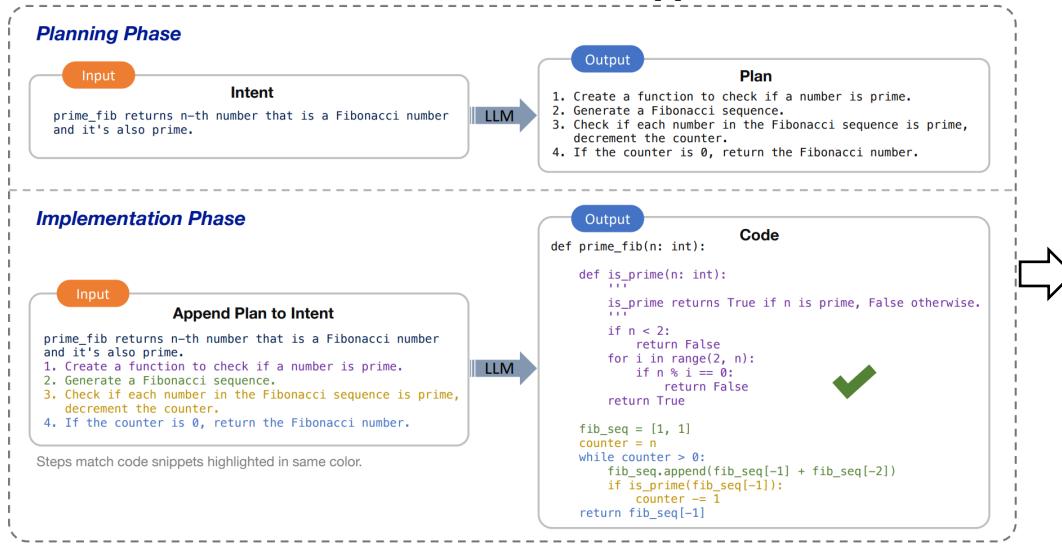
规划，因为普通代码也能调用工具、读写记忆，但那不是智能体。智能体之所以被称为“智能”，是因为它具备针对不同问题**自规划**的能力，这才是它的“大脑”所在。



核心概念定义

➤ 规划：把复杂任务分解为子任务，并规划任务的执行流程

Self-planning Code Generation [1]



Math Word Problems (free response)

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Math Word Problems (multiple choice)

Q: How many keystrokes are needed to type the numbers from 1 to 500? Answer Choices: (a) 1156 (b) 1392 (c) 1480 (d) 1562 (e) 1788

A: There are 9 one-digit numbers from 1 to 9. There are 90 two-digit numbers from 10 to 99. There are 401 three-digit numbers from 100 to 500. $9 + 90(2) + 401(3) = 1392$. The answer is (b).

StrategyQA

Q: Yes or no: Would a pear sink in water?

A: The density of a pear is about 0.6 g/cm³, which is less than water. Thus, a pear would float. So the answer is no.

Date Understanding

Q: The concert was scheduled to be on 06/01/1943, but was delayed by one day to today. What is the date 10 days ago in MM/DD/YYYY?

A: One day after 06/01/1943 is 06/02/1943, so today is 06/02/1943. 10 days before today is 05/23/1943. So the answer is 05/23/1943.

Original CoT

Thought for 29 seconds ▾

Okay, so the user is asking, "How does the inability of AI to feel impact its intelligence?" Let me try to unpack this. First, I need to understand what they mean by "feel." Are they referring to emotions, physical sensations, or something else? Given the context of AI, it's probably about emotions. So, the question is about how the lack of emotional capacity affects AI's intelligence.

CoT in DeepSeek-R1

Define a function to check if a number is prime. Check if the number is divisible by any number between 2 and n-1. If the number is not divisible by any number between 2 and n-1, it's prime. Initialize a list "fib_seq" with the first two Fibonacci numbers 1, 1. Initialize a counter "counter" with the value of n. While the counter is greater than 0, extend the list "fib_seq" with the next Fibonacci number. Check if the last number in the list "fib_seq" is prime. If it's prime, decrement the counter. Return the last number in the list "fib_seq".

Code CoT (defined in work [1])

规划≠CoT



规划体现的是分解问题的能力，而CoT体现的是单次解决问题的能力，两者是正交的

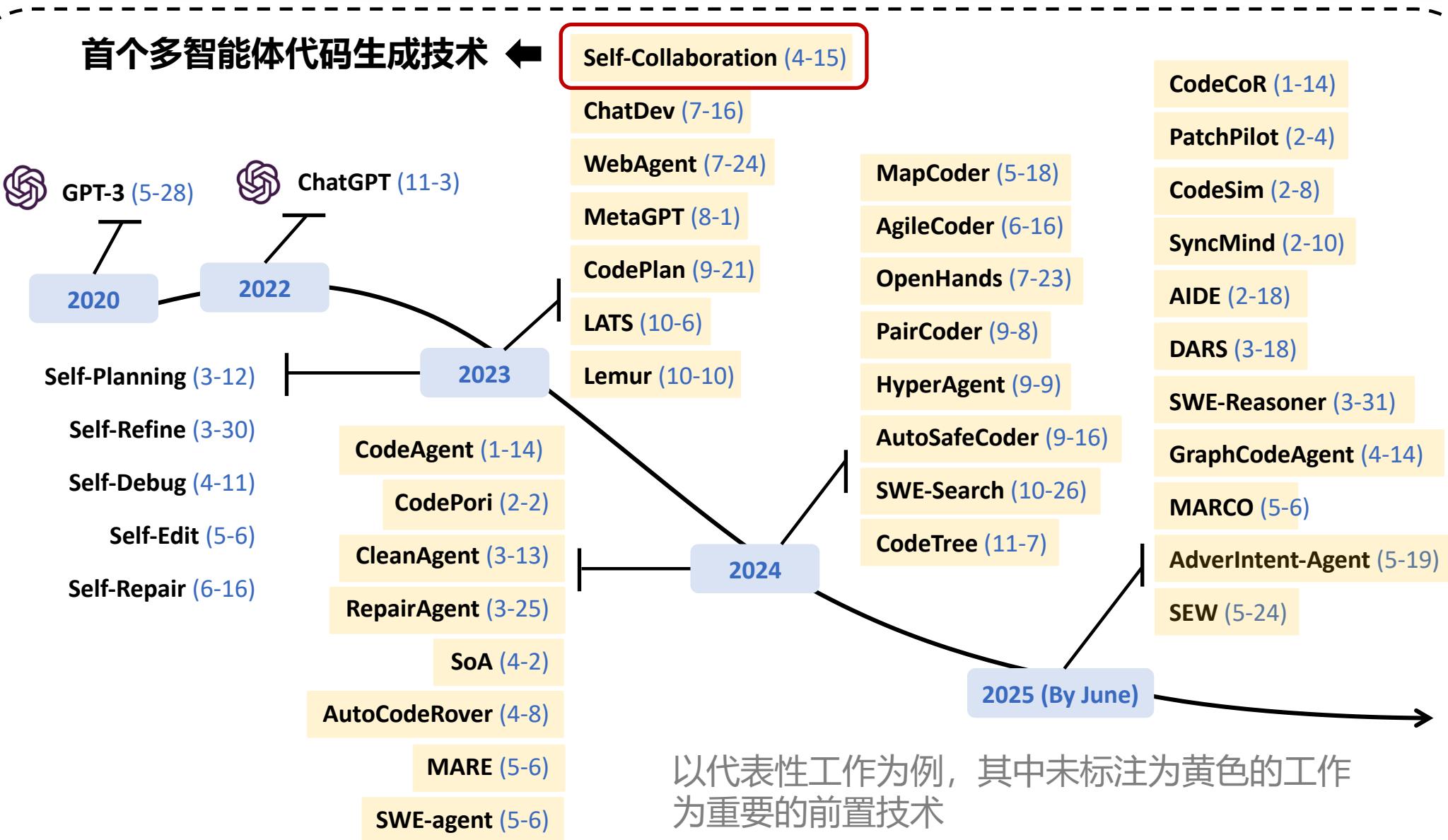
➤ 工具：使用如计算器、搜索工具、代码执行器、数据库查询等工具API

- 熟练的使用更多数量的专业工具
- 复用 -> 创造工具

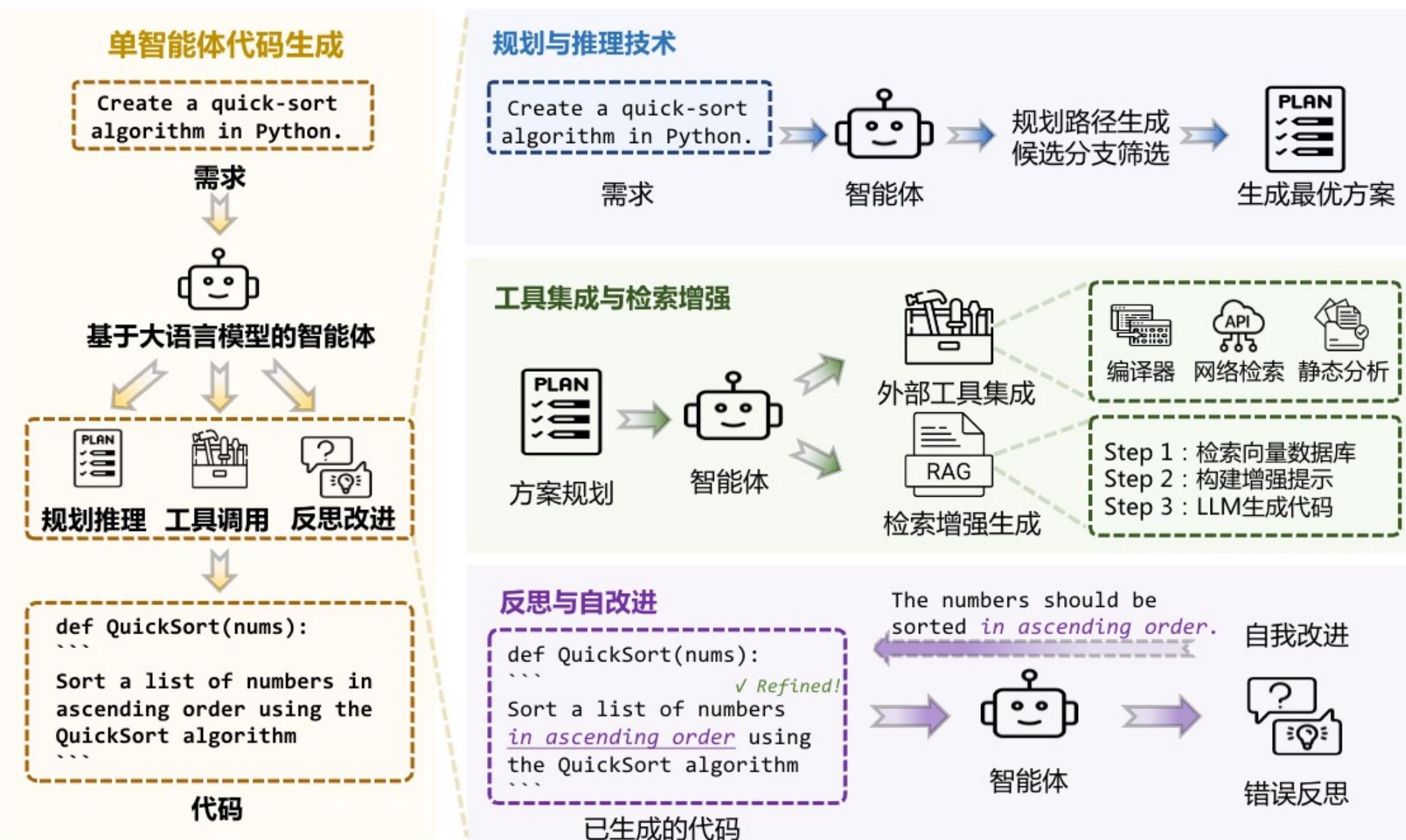
➤ 记忆：从过去的经验中学习，并保持对话或任务的连续性。

- 短期记忆：上下文信息，会在任务的执行过程产生和暂存，在任务完结后被清空。
- 长期记忆：长时间保留的信息，一般是指外部知识库，通常用向量数据库来存储和检索。
- 智能体的自进化

基于大语言模型的代码生成智能体技术演进



单智能体代码生成方法



多智能体代码生成的必然性

随着需求的不断变复杂和实现代码量的增大，
基于单智能体的代码生成也会遇到软件危机

20世纪60年代，随着软件系统规模和复杂性的
增长，计算机软件行业面临着软件危机

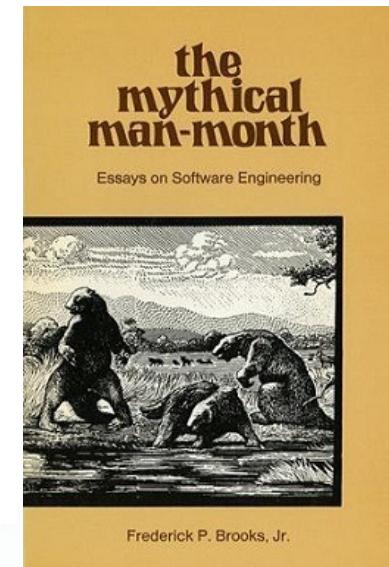
应该如何设计多智能体的交互？简单增加多智能体的数量？

$$1 \text{ 人} \times 10 \text{ 个月} = 10 \text{ 人} \times 1 \text{ 个月?}$$

布鲁克斯定律：向一个已经延期的
软件项目增加人手只会让它更延期

核心思想

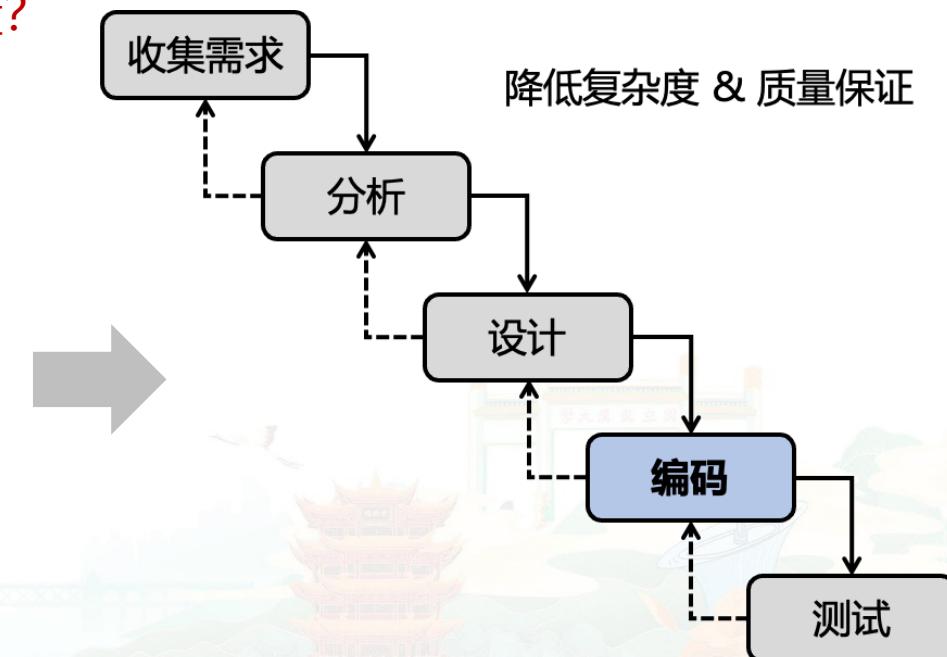
- 软件项目的复杂性和沟通成本会
随着团队规模的扩大而增加
- 简单地向项目中增加人手并不能
相应地缩短项目周期



复杂软件开发（团队合作）

单人	单智能体
时间、精力有限	输入/输出上下文窗口有限

基于多智能体的软件开发是必然的趋势



Self-collaboration Code Generation (TOSEM'24) Spec Coding

- 首次提出基于大语言模型的多智能体协同软件开发框架
- 该工作引入软件开发方法论中的瀑布模型组织多智能体进行高效的协同合作，并展示了角色扮演在唤起多智能体潜在专家知识方面的有效性

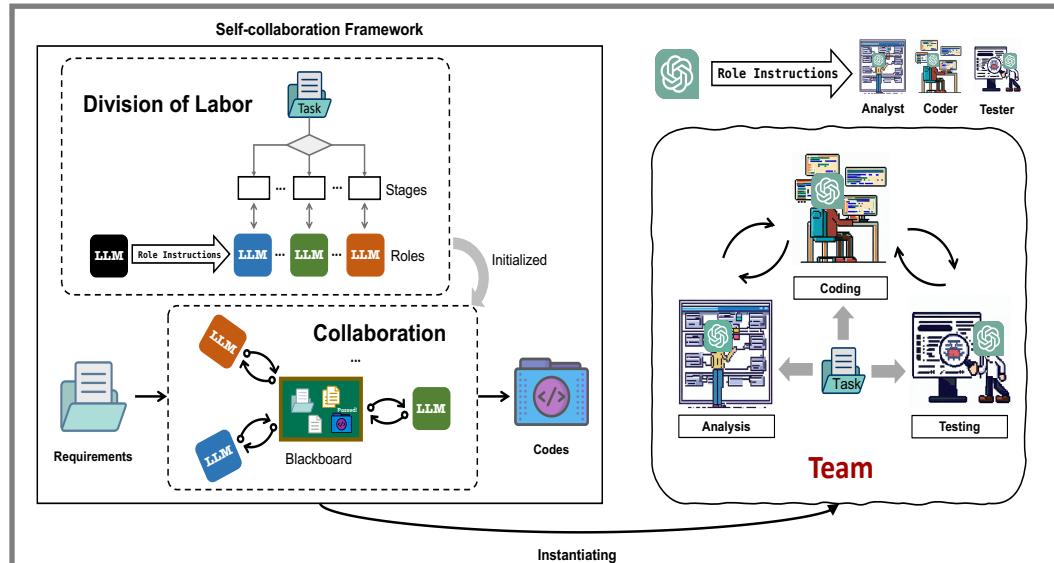
469 Citations

高引
论文

Highly Influential Citations ⓘ 32

Background Citations 98

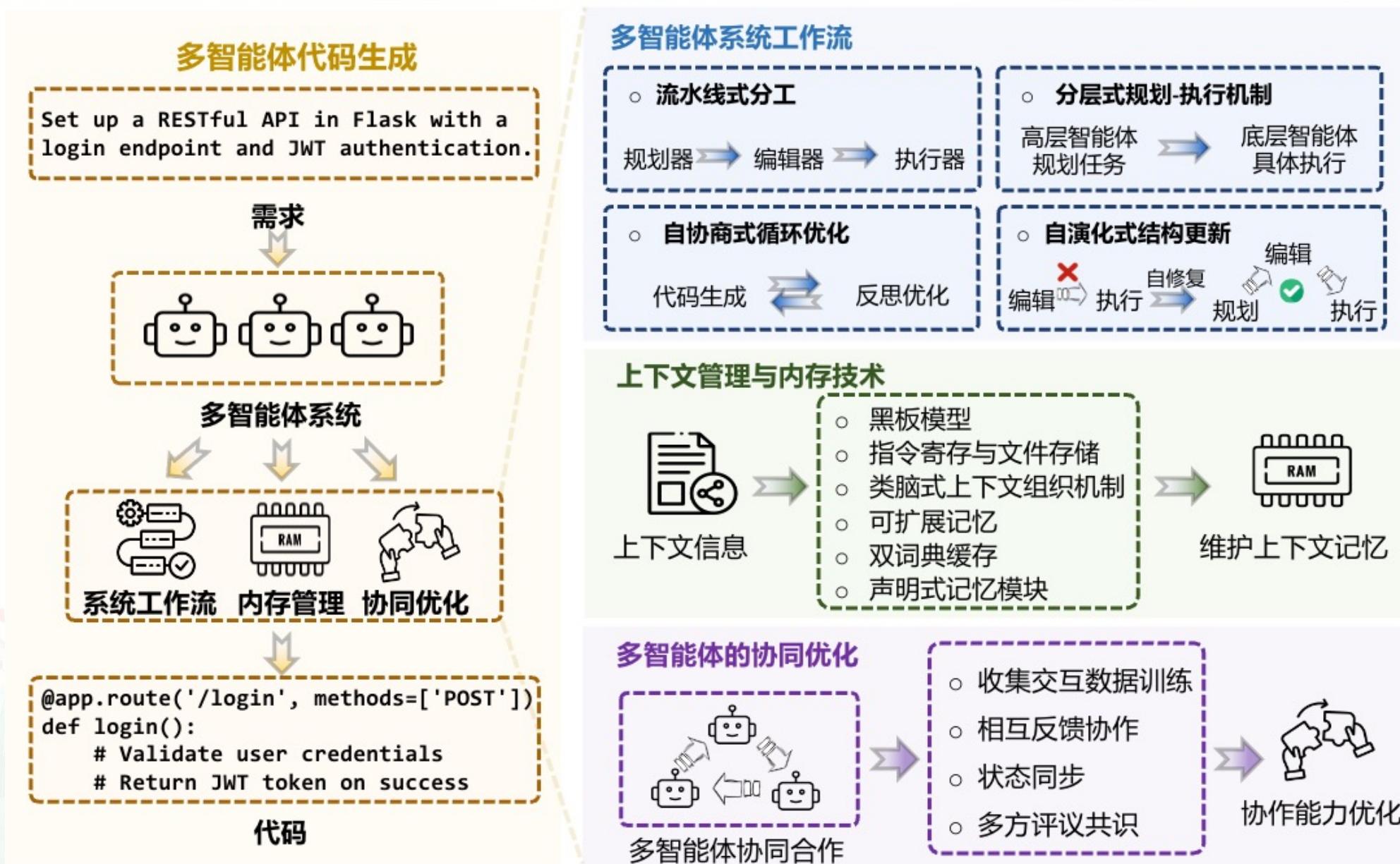
Methods Citations 55



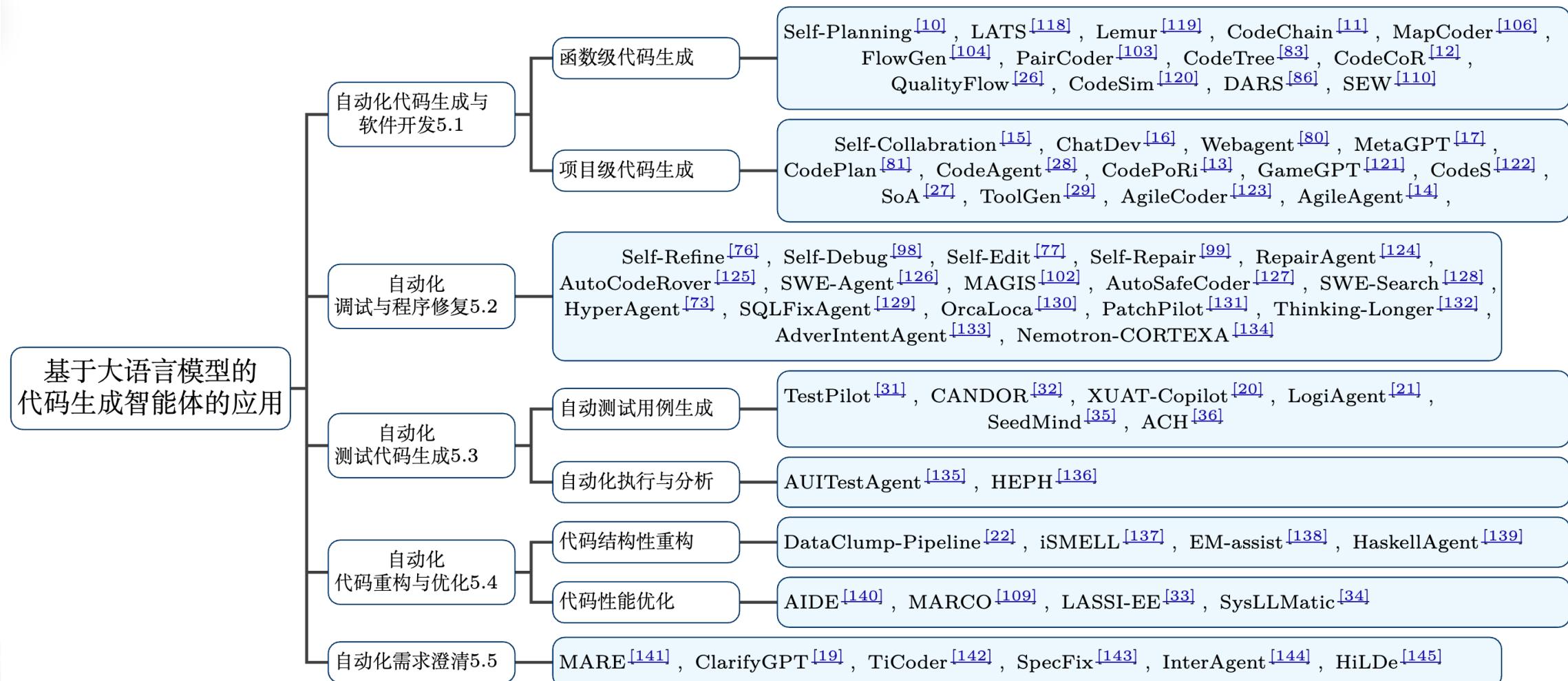
Roles	HumanEval	HumanEval-ET
Ablation of Roles		
Coder	45.1	35.3
+ Analyst	54.9 (↑ 21.8%)	45.2 (↑ 28.1%)
+ Tester	56.8 (↑ 26.0%)	45.2 (↑ 28.1%)
+ Analyst + Tester	63.5 (↑ 40.8%)	51.9 (↑ 47.1%)
Different Team Configurations		
Analyst + Coder + Tester + Compiler	64.6	50.0
Driver + Observer (Pair Programming)	57.4	44.6
The Role of Role-playing		
Few-shot Prompting (4-shot)	58.1	47.5
Instruction (zero-shot)	60.0	47.5
Role Instruction (Ours)	64.4	51.9

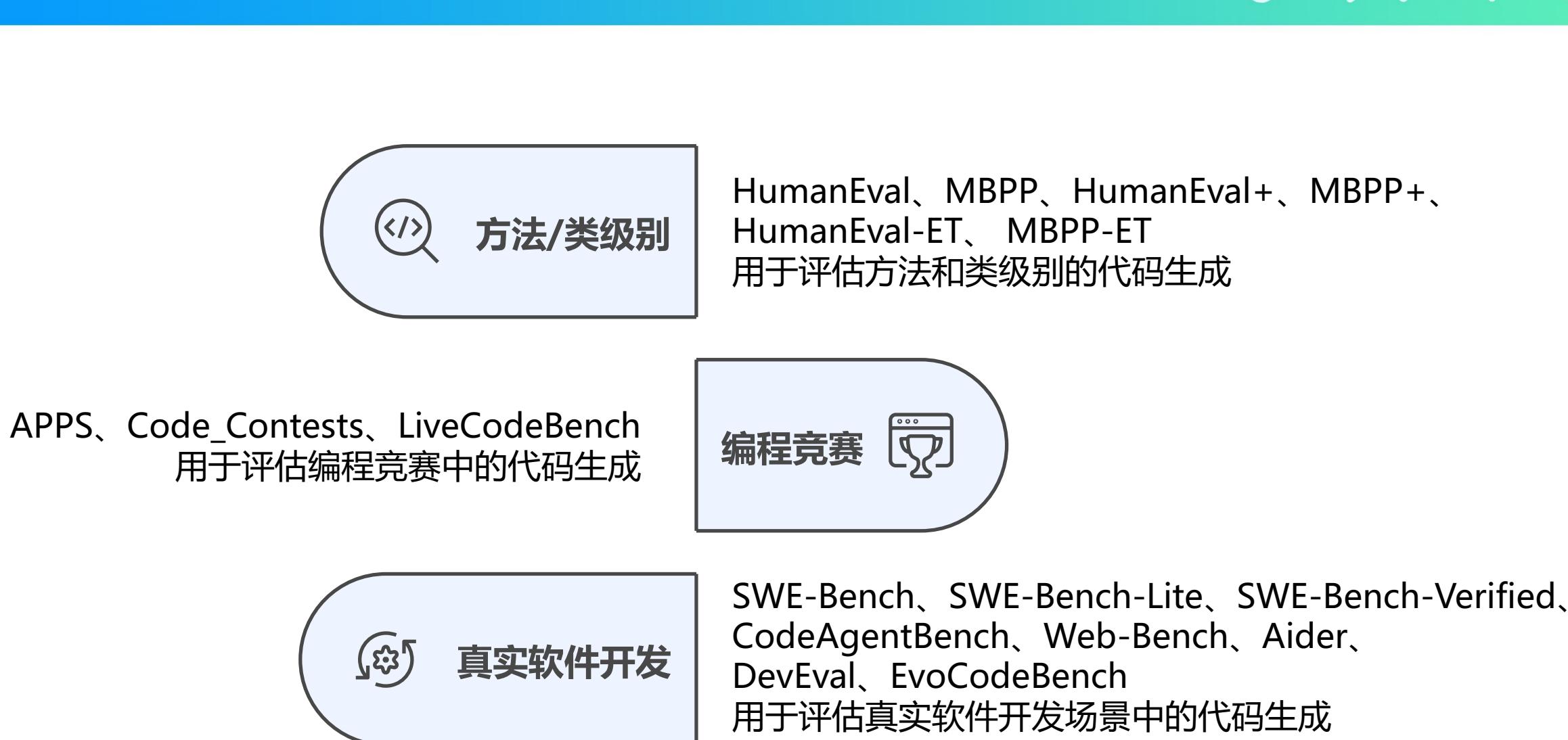
工具调用的影响
不同软件开发方法论的影响

多智能体代码生成系统



- 代码生成智能体在软件开发全生命周期中的五大核心应用场景





➤ 功能正确性指标— $pass@k$

$$pass@k = \mathbb{E}_{problems} \left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right]$$

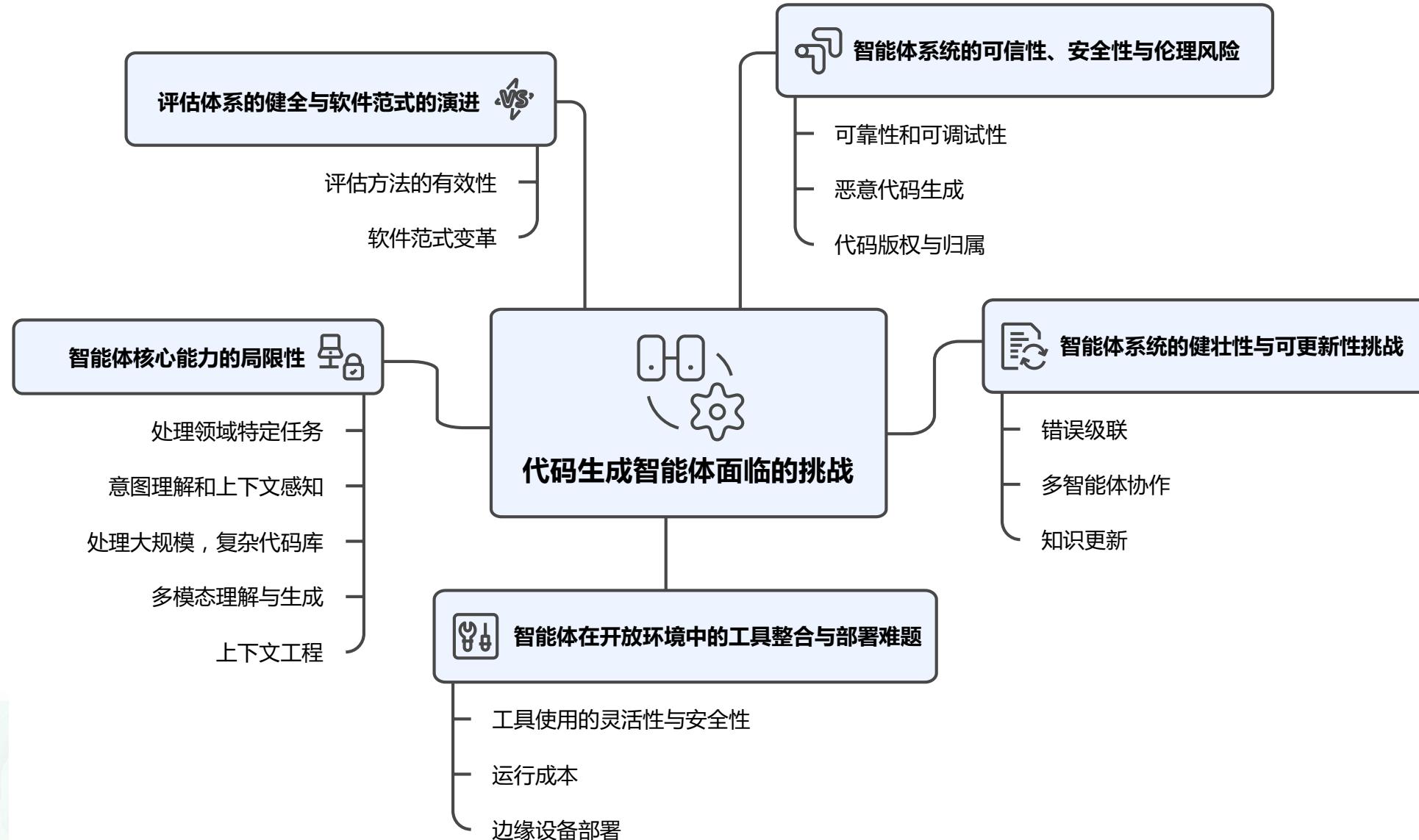
➤ 非功能性软件质量指标：

- 安全性——漏洞数量等
- 可读性与复杂性——代码注释和计算复杂度等
- 可维护性——模块间耦合度等

➤ 针对代码生成智能体的评估指标

- 结果——生成的代码是否成功解决了对应的问题，是否通过所有相关测试用例
- 过程——完成任务所采取的步骤的数量；能否正确选择工具并传递有效参数
- 效率——从接收任务到产出方案的时间延迟
- 成本——完成任务所需的API调用成本和Token消耗量；本地部署模型所需的计算资源消耗

工具名称	工具定位	主要集成方式	上下文引擎	最佳应用场景	主要优势	已知局限性
GitHub Copilot	编程助手	IDE扩展插件	检索增强	代码补全	普及率广、擅长常规任务	需人工监督，可能忽略全局上下文
Devin	自主开发团队	独立在线平台	沙箱内的知识库	端到端代码生成	高度自主，能处理任务全生命周期	真实世界可靠性低、结果不可预测、易陷入循环
Cursor	协同伙伴	AI原生IDE	向量语义索引	代码补全、代码开发、代码库问答等	深度代码库理解，卓越的交互体验	自主代码智能体能力较弱
通义灵码	协同伙伴	企业平台集成	仓库知识图谱	代码开发	对解决方案空间进行稳健、系统性的探索	更侧重企业级应用
Claude Code	自主开发团队	命令行接口	超大上下文窗 + 智能体搜索	端到端代码生成、代码库问答等	强大的代码智能体，全局代码库理解	潜在成本高



➤ 智能体核心能力的局限性

- **领域特定任务处理不足**: 缺乏深度领域知识, 易在专业任务中理解偏差或产生幻觉
- **意图理解与上下文感知薄弱**: 对模糊指令易误解, 难以准确对齐用户真实需求
- **难以处理大型复杂项目**: 在跨文件依赖与整体架构理解上能力不足, 限制项目级应用
- **多模态理解能力不足**: 无法有效利用图像等非文本信息, 导致设计与生成结果不一致
- **上下文工程不成熟**: 上下文容易出现中毒(错误或虚构信息污染后续推理)、分心(冗余信息淹没关键信号)、混淆(格式混乱导致理解偏差)、上下文冲突(相互矛盾的信息导致决策失误), 严重影响生成质量

➤ 智能体系统的健壮性与可更新性挑战

- **错误级联效应显著**: 上游错误会被不断放大, 最终导致系统性失败
- **多智能体协作难以协调**: 随着规模扩大, 交互复杂度爆炸, 易出现沟通混乱与目标漂移
- **持续学习能力不足**: 智能体难以有效吸收新知识, 同时避免遗忘旧知识

➤ 工具整合与部署难题

- **工具使用缺乏灵活性与安全性**: 无法动态发现与整合工具，且需防止工具滥用风险
- **运行成本高昂**: 多轮协作带来巨大计算与时间开销，限制大规模应用
- **缺乏轻量化部署能力**: 难以在边缘设备上实现高效运行，需要模型压缩与加速技术

➤ 可信性、安全性与伦理风险

- **可靠性与可调试性不足**: 幻觉与复杂内部决策机制使系统难以被信任且不易调试
- **恶意代码生成风险**: 可能绕过安全策略生成攻击脚本，需要严格生成前后审查机制
- **版权归属与法律风险不明确**: 生成内容可能侵犯版权，亟需追踪机制与伦理规范

➤ 评估体系与软件范式演进

- **评估体系不全面**: 现有指标忽略人类干预成本，难以准确反映系统真实效用
- **软件开发范式将发生变革**: 未来可能从“人+智能体开发软件”转向“人描述意图→智能体直接完成任务”的新模式



谢 谢 观 看

Q&A

董益宏 dongyh@stu.pku.edu.cn



论文链接



Github仓库



个人主页