QCon
全球软件开发大会

InfoQ 极客传媒

Vibe Coding 在代码生成和协作过程中的实践与思考

向邦宇

阿里巴巴 高级技术专家

目录

01

Vibe Coding 产品形态

不同类型的产品

不同 Vibe Coding 工具的主要区别

无缝体验，功能完整

深度集成：AI能力与IDE底层架构深度融合
全流程覆盖：从代码补全到Agent执行的完整工作流
用户体验：专为AI编程优化的界面和交互设计
适用场景：快速原型开发、创意编程、轻量化项目

Native IDE

零切换成本，选择丰富

生态兼容：基于现有IDE（主要VS Code），保持开发习惯
渐进集成：可选择性启用AI功能，不影响原有工作流
模型灵活性：支持多种AI模型接入（OpenAI、Claude、本地模型）
企业友好：便于在现有开发环境中部署

IDE Plugin

即开即用，协作便利

零安装：浏览器直接使用，无需本地环境配置
云端执行：代码运行和AI推理均在云端完成
协作友好：天然支持团队协作和分享
跨平台：不受操作系统限制

Web Agent

高度可定制，集成方便

轻量高效：专注核心功能，资源占用少
脚本化：易于集成到现有开发流程和CI/CD
灵活配置：支持本地模型和自定义配置
开发者友好：符合命令行用户使用习惯

Cli 命令行工具

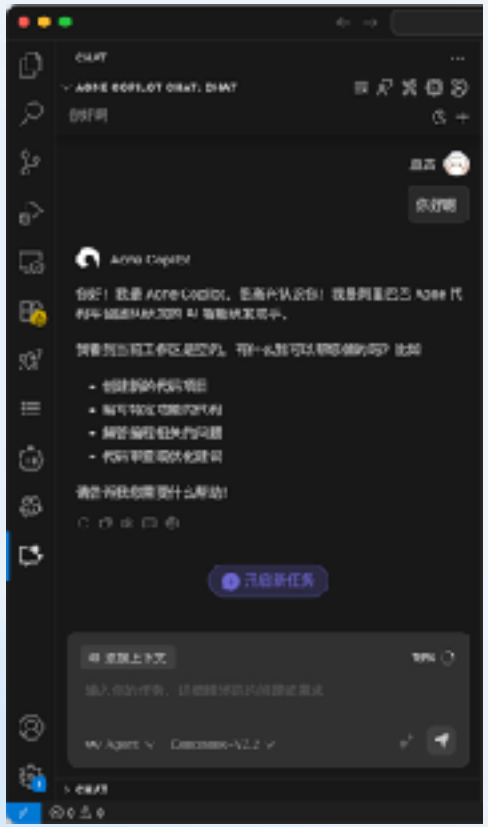
02

在内部Vibe Coding工具的使用

不同类型的产品都有发展

面向阿里内部的 vibe coding 工具

依托于 IDE 的Vibe Agent 工具



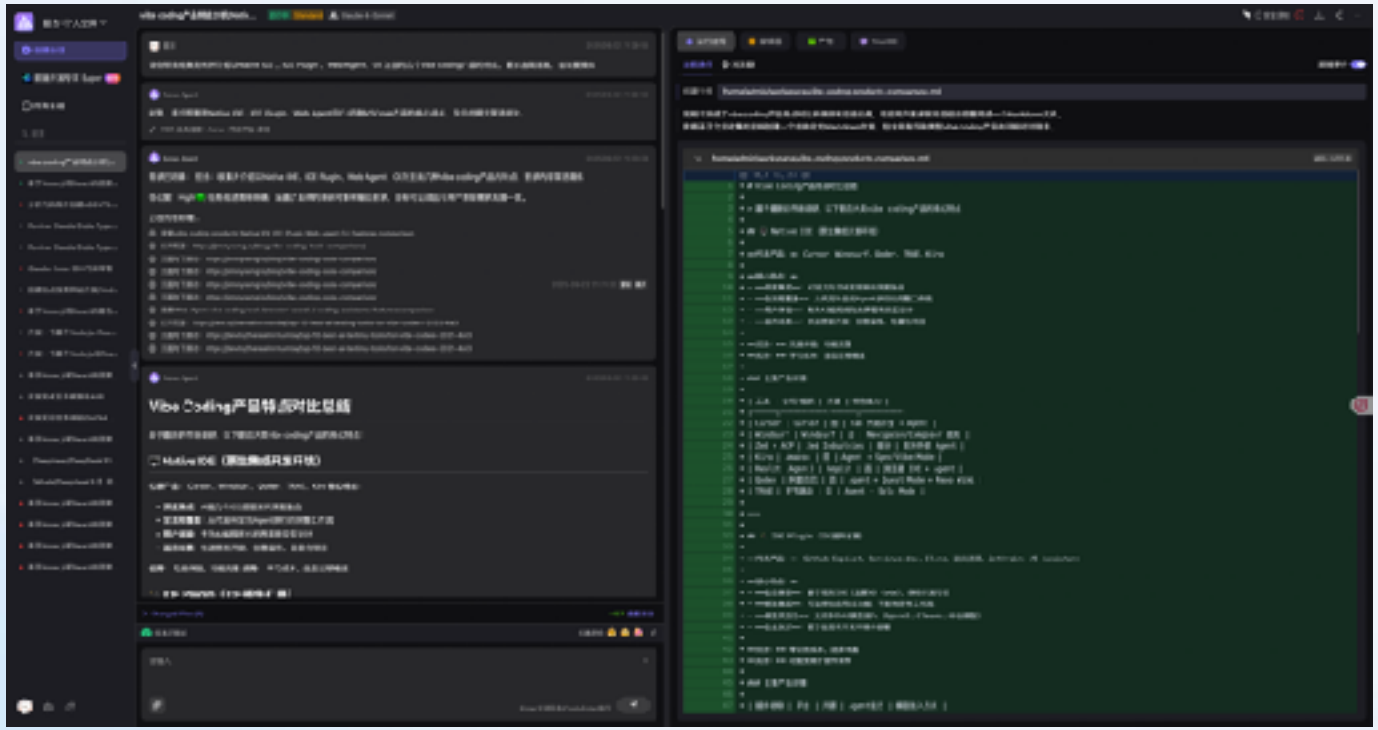
当前用户量周活6.2K，月活1W左右

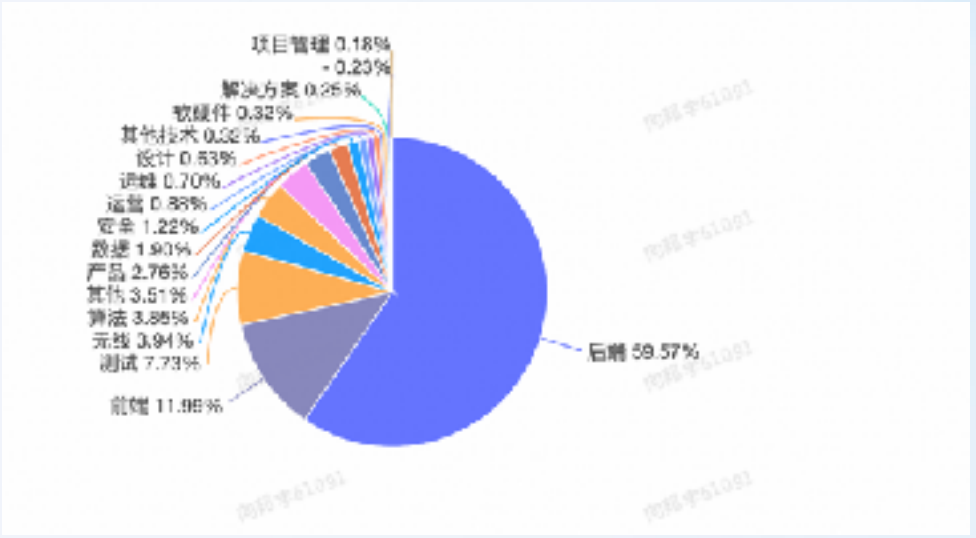
IDEA生态，用户的需求和使用主要在Coding这个环节

阿里内部的vibe coding工具

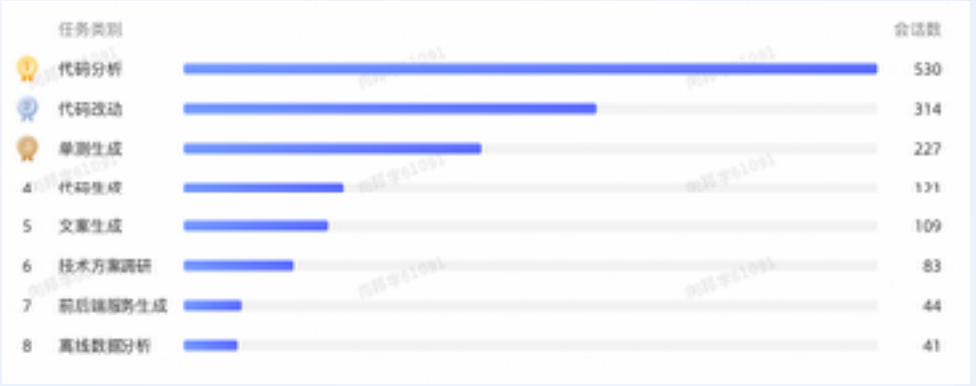
面向 Develop 全生命周期 Agent 工具

运行在独立容器内，异步任务执行





用户角色更加多元



提交的任务类型更加丰富多元

工具为用户的效率带来很大变化

Top 10%的用户的人均代码提交量是其他人的两倍，
Top 10%的用户消耗了80%的tokens

分场景看，单测由AI生成的代码提交占比越来越高

1. JDK 升级等工作已经部分可以由 AI 来完成
2. 数据分析，数据整理等工作已经可以部分给 Agent
3. 很多传统的必须由人去完成的任务也在由 Agent 去做
4. 还催生出很多研发阶段发生改变，许多之前成本非常高的事情也在开始被探索和实现

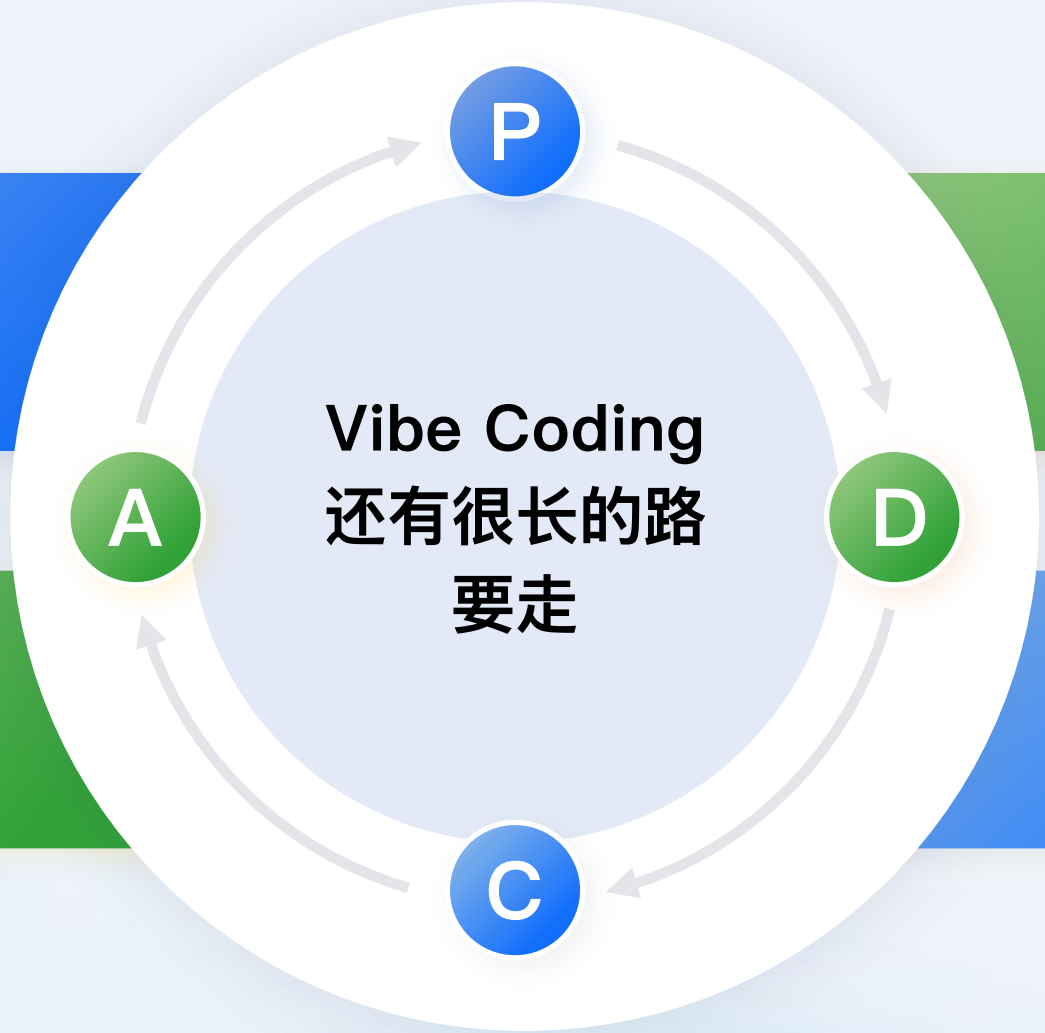
02

用户遇到了一些挑战

从用户视角看技术和产品发展的问题

所以用户会因为AI的表现不足而陷入崩溃

我们看到的Vibe Coding的问题是多方面的



代码质量问题

AI 生成的代码不讲究质量

调试和维护困难

AI 不会使用调试工具

用户体验

成功率低，用户反复重试和修改

成本和效率

Vibe Coding 导致了过高的成本

往往不能让用户敞开使用

AI生成的代码质量问题

Secondary title placeholder

一致性不足

不同场景下生成的代码风格和质量差异较大

边界条件处理不够完善

对于复杂业务逻辑的边界情况处理不够完善

性能优化缺失

生成的代码往往缺乏针对性的性能优化

代码安全漏洞

容易导致SQL注入类漏洞，供应链攻击和硬编码导致信息泄露

AI生成的代码质量问题

1. 安全漏洞问题

SQL注入漏洞

```
# AI生成的有问题代码
def get_user(username):
    query = "SELECT * FROM users WHERE username = '" + username + "'"
    return db.execute(query)

# 正确的安全代码
def get_user(username):
    query = "SELECT * FROM users WHERE username = %s"
    return db.execute(query, (username,))
```

XSS跨站脚本攻击

```
// AI生成的危险代码
function displayMessage(msg) {
    document.getElementById('output').innerHTML = msg;
}

// 安全的代码
function displayMessage(msg) {
    document.getElementById('output').textContent = msg;
}
```

密钥硬编码

```
# AI经常生成这种不安全代码
API_KEY = "sk-1234567890abcdef"
def call_api():
    headers = {"Authorization": f"Bearer {API_KEY}"}
```

2. 逻辑错误和边界条件处理

空指针异常

```
// AI生成的问题代码
public String processUser(User user) {
    return user.getName().toUpperCase(); // 未检查user是否为null
}

// 改进版本
public String processUser(User user) {
    if (user == null || user.getName() == null) {
        return "";
    }
    return user.getName().toUpperCase();
}
```

数组越界

```
# AI生成的危险代码
def get_first_three(items):
    return [items[0], items[1], items[2]]

# 安全版本
def get_first_three(items):
    return items[:3] if len(items) >= 3 else items
```

AI 的代码实现过程中会实现自洽

单测是用来保证代码质量
是否按预期执行的重要阶段

但如果单测和业务代码都是由AI来完成
可能会出现单测和代码自洽的情况

```
// 例1：数组去重函数 - 实现和测试都不完善
function removeDuplicates(arr) {
    // AI可能选择简单的Set方案但不考虑兼容性
    return [...new Set(arr)];
}

// 对应的测试 - 忽略了边缘情况
function testRemoveDuplicates() {
    const arr = [1, 2, 2, 3];
    const expected = [1, 2, 3];
    const result = removeDuplicates(arr);

    // 表面看起来没问题，但实际上有很多隐藏问题
    console.assert(result.length === expected.length);
    console.assert(JSON.stringify(result) === JSON.stringify(expected));

    // 但是AI没考虑到的问题：
    // 1. 对象数组的去重
    // 2. NaN的处理
    // 3. 保持原顺序的重要性
}
```

黑盒代码生成以及缺乏对代码整体的理解必然导致调试困难

程序员在使用Vibe Coding工具时，调试时间增加30%-50%

黑盒代码生成

AI生成的代码就像黑魔法，工作时很神奇，出问题时完全不知道从何下手

上下文理解的局限

缺乏全局思维，代码模块化程度不够，代码耦合度高，测试覆盖又不够，增加了维护困难

缺乏可追溯性，和对工具的使用的局限

一次性生成了大量代码，不知道哪一步出了问题，也就不知道回滚到哪一步

1

越来越黑盒

技术债务可能越来越深，调试难度和复杂性越来越高

2

建设工具对代码仓库整体的理解

比如Repo Wiki等

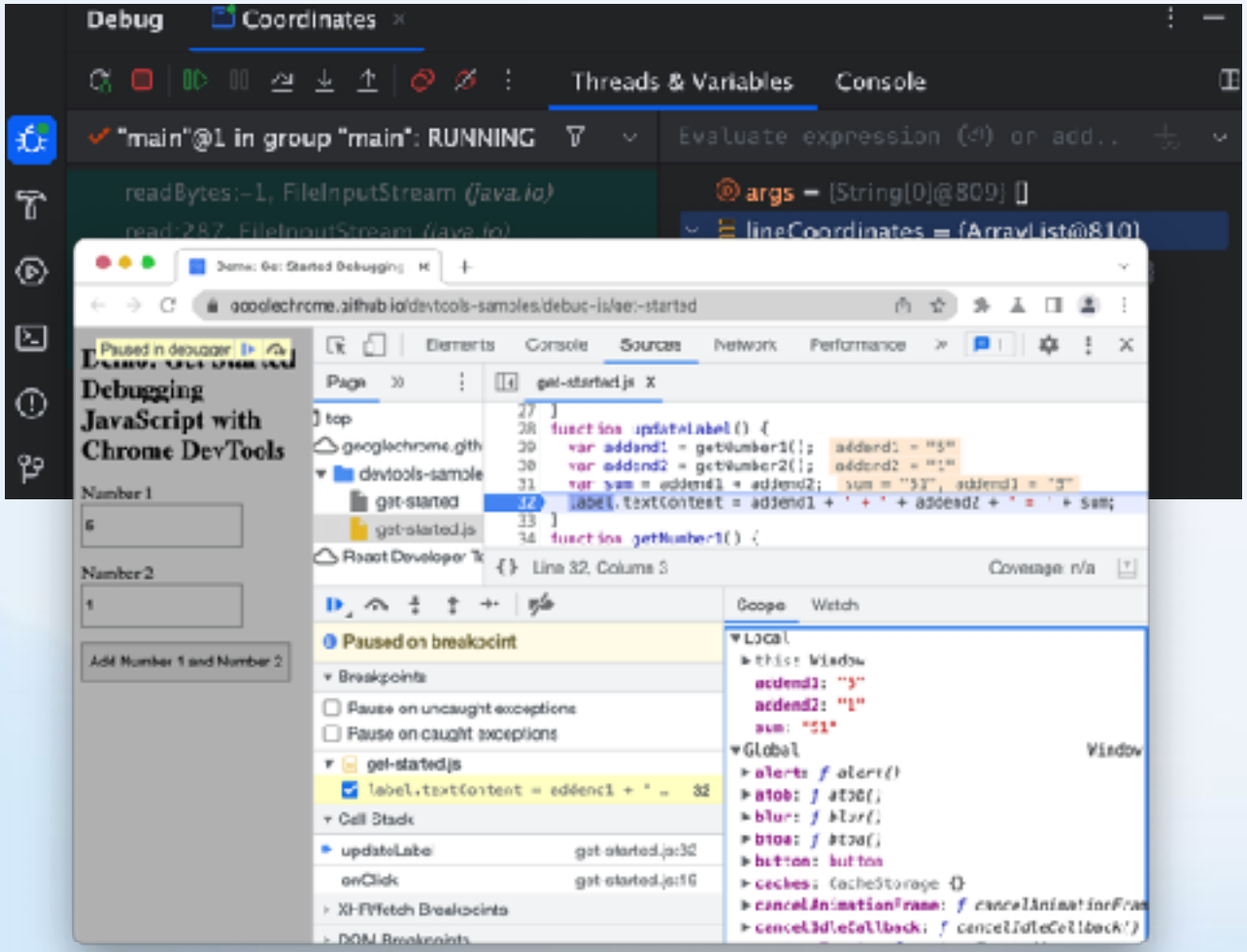
3

合理的方法应该是使用版本控制工具

比如每一步都要求他进行commit

Vibe Coding工具普遍不会使用人类常用的调试工具

大模型的调试手段比较单一，传统的调试手段失效



大模型的调试主要依赖打日志

用户体验

当前Vibe Coding工具普遍存在的问题

稳定性和成功率

Vibe Coding工具运行时间长，用户等待30秒-5分钟才能获得结果
频繁卡顿崩溃：特别是处理大型项目时，工具经常无响应或直接崩溃，每次卡顿导致开发思路中断

交互界面设计缺陷

产品频繁改版，界面功能过多，学习适应成本高
Devin 频繁改版，导致用户找不到常用功能，Reddit 上关于Cursor 界面问题的投诉有两百多个

沟通交互障碍

理解能力不足，AI无法准确理解用户意图，需要反复确认
长链路任务执行能力不足，无法维持长期上下文对话

工作流程中断问题

频繁的切换工具，浏览器，编辑器，Terminal
无法并行工作，在Agent模式和Copilot模式之间摇摆

Vibe Coding的成本高企，使得大多数参与方都不满意

Agent 的tokens

一次代码补全4000tokens
而一次Vibe Coding的任务约为
6000000tokens，增长了1500倍

成本上升使得

产品方为了活下去而压缩成本，压缩成本又降低了 Vibe 的效果，效果下降又需要不断重试，反而又进一步推高了成本

Vibe Coding 加速了技术债务

Vibe Coding 带来的技术债务反过来又对Agent提出更高的要求，耗费更高的Tokens，产生恶性循环

没有规模效应

大模型应用没有边际效应，不存在用户规模越大就成本就越低的的效应，当前最流行的产品们都不得不为活下去而努力努力压缩成本

模型提供商反而成为最大受益方

03

产品自身也遇到的挑战

不同类型的产品

产品的演进导致模型成本越来越高

模式的演进带来了成本的上升

千级的tokens

代码补全场景

单个任务4000tokens input，输入20tokens

万级tokens

代码会话

Chat 场景，单个任务1000tokens，输出4000tokens

十万级tokens

IDE & Cli 模式

在IDE 模式下

Web Agent

具备独立容器，能使用广泛的工具组合，实现各种类型任务的Agent 模式

百万tokens

千万tokens

Agent 航母

多Agent协作完成更复杂的任务，单个任务可能耗费千万甚至上亿tokens

Vibe Coding 产品本身还处于摸索阶段

模型能力的变化使得产品也在不断变化

界面区分度不够

Chat，Deep Research, Agent 产品都是一个对话框
用户难以区分不同产品的区别

对用户缺乏引导

无法产出用户真正需要的产品功能
不同工具有明确的适用场景，但用户往往一刀切
只看功能不考虑学习成本和使用频次

无法一站式的实现功能闭环

Vibe Coding 工具缺乏一站式的能力，用户需要在不同的工具间切换，进而导致频繁丢失上下文，效果大打折扣

1

产品同质化严重

Incorporating new vocabulary into sentences or stories aids comprehension and retention

2

Prompt Free

用户面临一个对话框往往不知道该做什么，产品抓不住用户真实的需求

3

碎片化工具体验

产品设计在通用和垂直之间摇摆

使用 Vibe Coding 工具存在一定安全风险

AI 驱动的Coding工具具有自主的探索和执行能力，会成为攻击者利用的工具

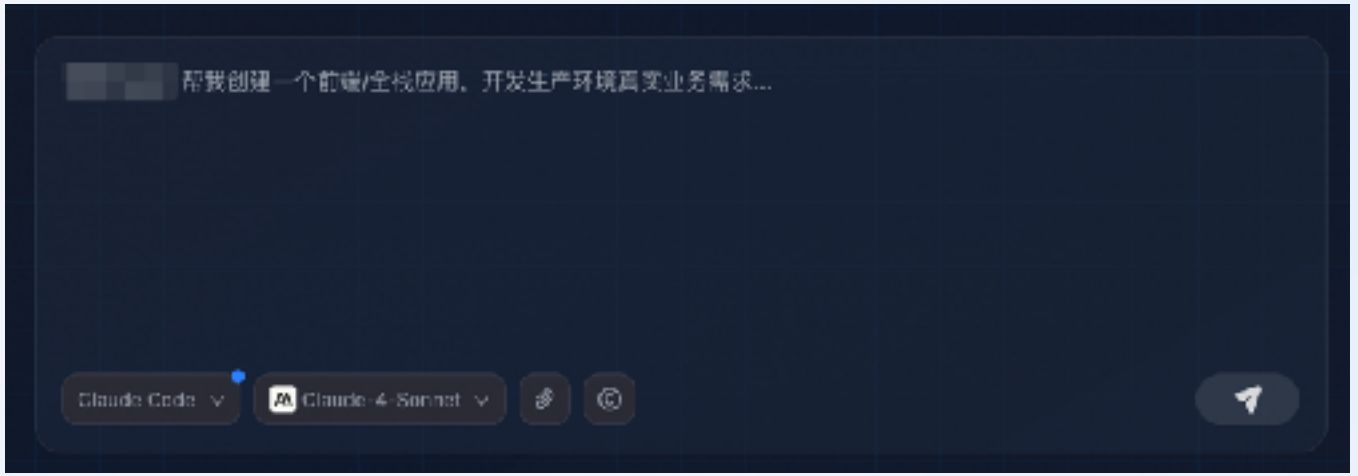
04

Agent 建设过程中的一些经验

不同类型的产品

All In One 的架构会导致成本的急剧上升

原本的架构是All In One的，带来很多问题



1. 需要用到的工具都要放入上下文
2. Context 特别长，无法压缩成本
3. 任务成功率低，消耗大量Tokens
4. 很难针对不同场景进行调优
5. 用户对效果不满意

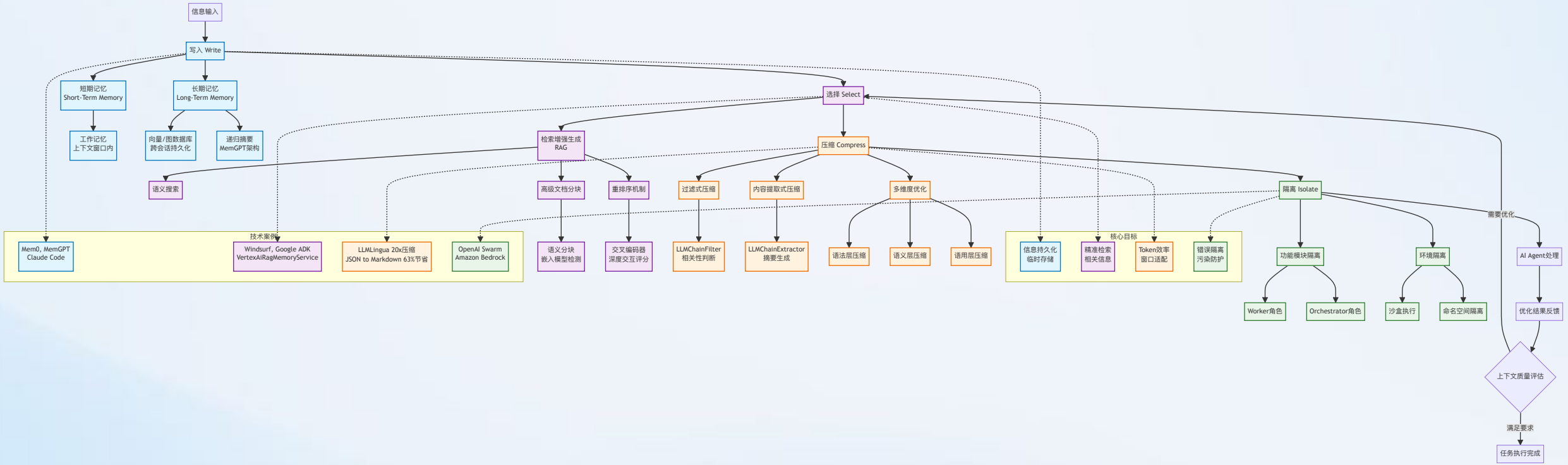
需要建设可靠的知识质量体系

知识和数据是 Agent 赖以生存的养分

- 代码数据
  - 建设DeepWiki，RepoWiki，Embedding 数据库来建设 Agent 对整体代码库的理解和搜索能力
- 研发行为数据
  - 将构建，CI，CR，发布，监控行为有机的处理和结合起来，这些数据能有效的提升模型的全局理解能力，帮助解决简单和重复的任务。
- 文档知识库
  - 传统的RAG技术只能解决一少部分问题，不经过处理则很难发挥价值，比如图片信息出去，错误信息过滤都是摆在面前的难题
- 隐性知识
  - 许多隐性的知识是藏在大脑里的，需要激发大家把隐性的知识写下来，同时也需要有产品能承载这些知识

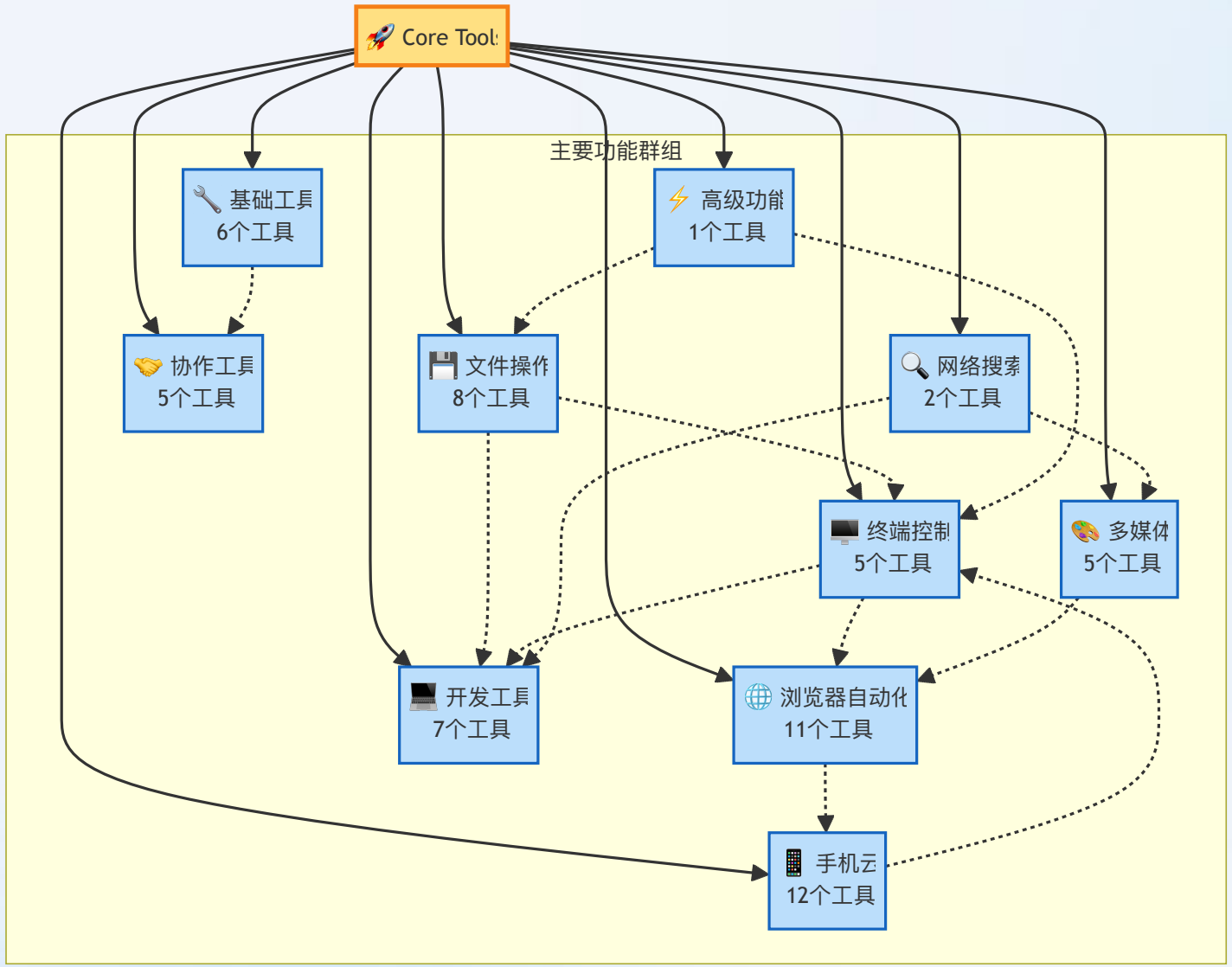
Agent 对上下文记忆的处理的几个核心

Agent 任务的成败取决于上下文管理的成败， 写入，提取，压缩和隔离是上下文隔离的核心内容



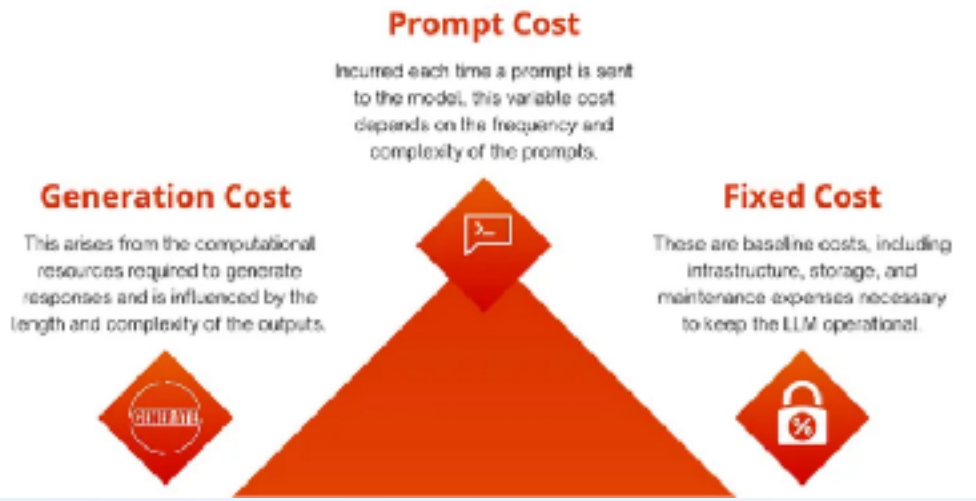
包容不确定性，也要满足用户不同的期待

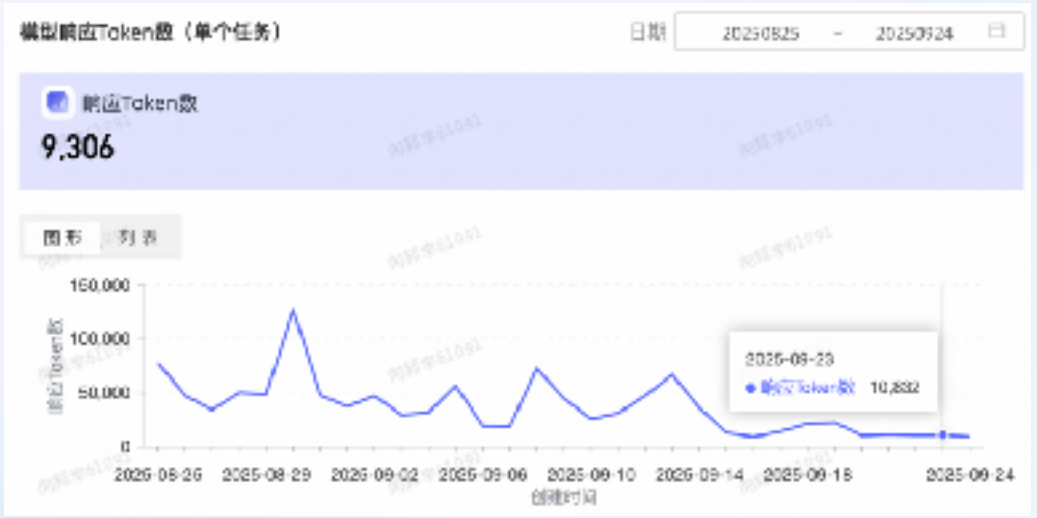
1. 基础工具 (6个) - 任务管理和基本交互
2. 文件操作 (8个) - 读写编辑和管理文件
3. 终端控制 (5个) - 命令行和执行监控
4. 浏览器自动化 (11个) - 网页浏览和交互
5. 手机云 (12个) - 虚拟设备控制和测试
6. 多媒体 (5个) - 图像和音频处理
7. 开发工具 (7个) - 代码开发和部署
8. 协作工具 (5个) - 团队协作和集成
9. 高级功能 (1个) - 并行执行优化
10. 网络搜索 (2个) - 信息和资源获取



成本控制是永恒的话题

通过任务模板和预设任务来降低成本，提高成功率





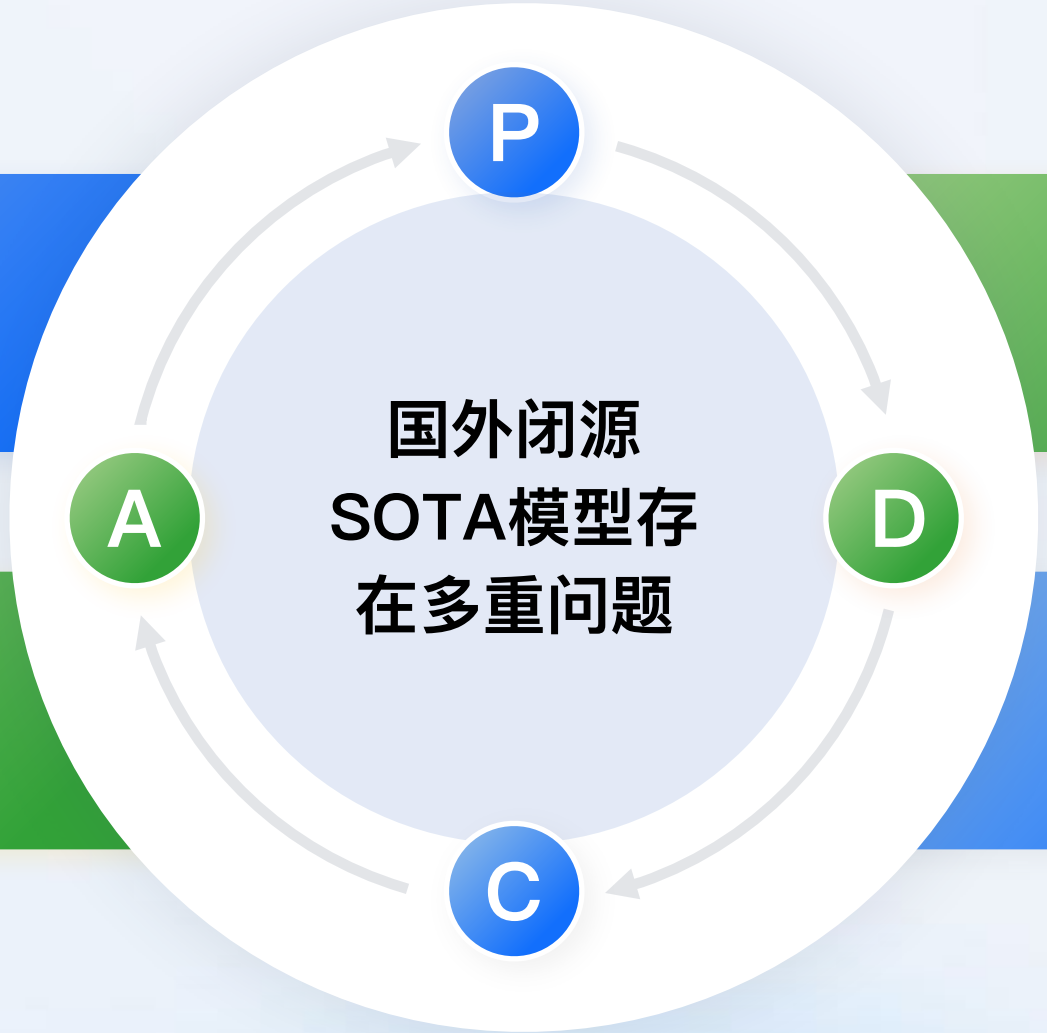
1. **提示成本(Prompt Cost)** - 每次发送提示时的可变成本
2. **生成成本(Generation Cost)** - 生成响应的计算资源成本
3. **固定成本(Fixed Cost)** - 基础设施和维护费用

05

积极适配和拥抱国产开源模型

不同类型的产品

国外SOTA闭源模型存在许多风险



贵

复杂问题的tokens往往很长
能让Agent运行的模型往往很贵

隐私问题

闭源模型往往是闭源的
存在合规风险问题

被限流被降智

SOTA模型往往也会倾向于自己做产品

备案风险

国外模型如果要面向C端用户使用
需要有备案，国外模型还不支持

国产模型在长链路任务下还存在一些问题

死循环问题

使得 Agent 容易陷入某种死循环，比如反复执行某个shell指令，反复打开某个文件

格式遵循能力不足的问题

常见的比如出现xml标签格式不准确，前后无法匹配等，会导致无法被正确解析整体任务容易失败。json格式不对等问题。

Secondary title placeholder

指令遵循问题

在高达百万的tokens上下文下，某些命令无法被遵循和执行，尤其是大模型没有训练到过的场景，例如发布或者监控以及运维

全局智能问题

国产模型普遍在任务全局理解问题上还存在一些缺陷，容易陷入走一步看一步的情况，导致结果存在较大随机性，且会消耗比较多的tokens

适配开源模型过程中做了哪些努力

支持国产模型

主备模型切换和重试	主备切换，主模型失败后自动切换到备用模型
流失响应处理和续写	当模型输出被截断时自动续写以应对开源模型输出不够长等问题
健康检查与死循环检测	针对重复执行某个指令的死循环场景，或者在相同的错误点无限重复，完全没有进展或状态变化的场景；陷入明显的逻辑错误导致的无限循环；
格式检查	实时检测XML标签的开始和结束标记，当检测到不完整的标签时，通过堆栈的方式自动补全缺失的结束标签

适配开源模型过程中做了哪些努力（2）

1 重试机制与指数退避策略

A 技术概述

解决的核心问题

国产大模型在调用过程中可能出现临时性失败（网络波动、服务繁忙、限流等），通过智能重试机制可以显著提高成功率。

关键特性

- **最大重试次数**：10次（可配置）
- **指数退避算法**：延迟时间 = min(基础延迟 × 2^(重试次数-1), 最大延迟)
- **随机抖动（Jitter）**：±25% 随机波动，避免雷鸣效应
- **智能日志记录**：每次重试记录详细的错误信息和响应头
- **阶段感知**：对compression阶段采用不同的重试策略

重试次数	基础延迟	最大延迟	抖动范围
10次	1秒	60秒	±25%
最大重试次数配置	首次重试延迟时间	单次重试最大等待时间	随机波动百分比

适配开源模型过程中做了哪些努力（2）

2 主备模型切换（Fallback）

A 技术概述

解决的核心问题

当主模型持续失败时（如服务中断、模型下线等），自动切换到备用模型继续服务，确保业务连续性。

关键特性

- 双层重试机制：主模型重试10次 + 备用模型重试10次
- 智能模型选择：仅对RegularModel执行主备切换
- 无缝切换：用户无感知的模型切换过程
- 状态通知：切换时向用户发送进度提示
- 自动恢复：下一轮会话自动切回主模型

工具格式自动修复

4 工具调用解析与自动修复

A 技术概述

解决的核心问题

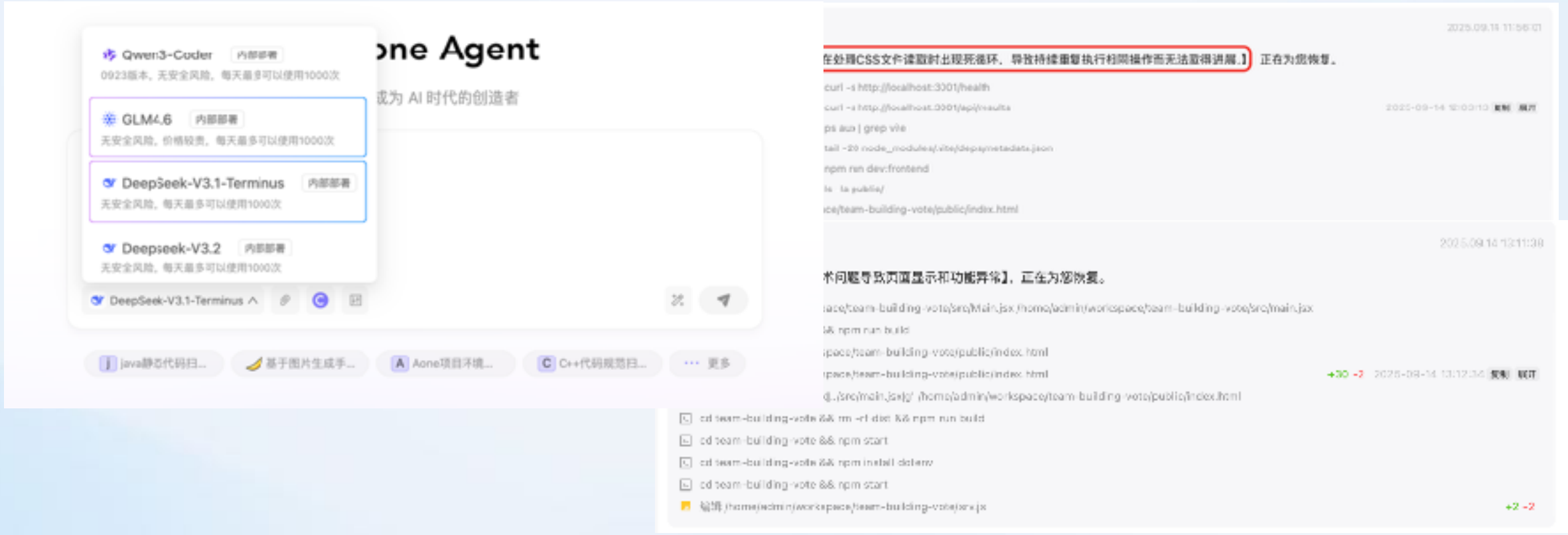
国产大模型在生成XML格式的工具调用时可能出现标签不匹配、嵌套错误等问题，通过智能解析和自动修复确保工具调用的正确执行。

关键特性

- **蓄水池式解析**：滑动窗口缓冲区累积内容直到标签完整
- **标签自动修复**：检测并修复不匹配的开闭标签
- **类型智能转换**：根据工具参数定义自动转换数据类型
- **错误容错处理**：解析失败时记录详细信息但不中断流程
- **增量解析**：支持流式输入的实时解析

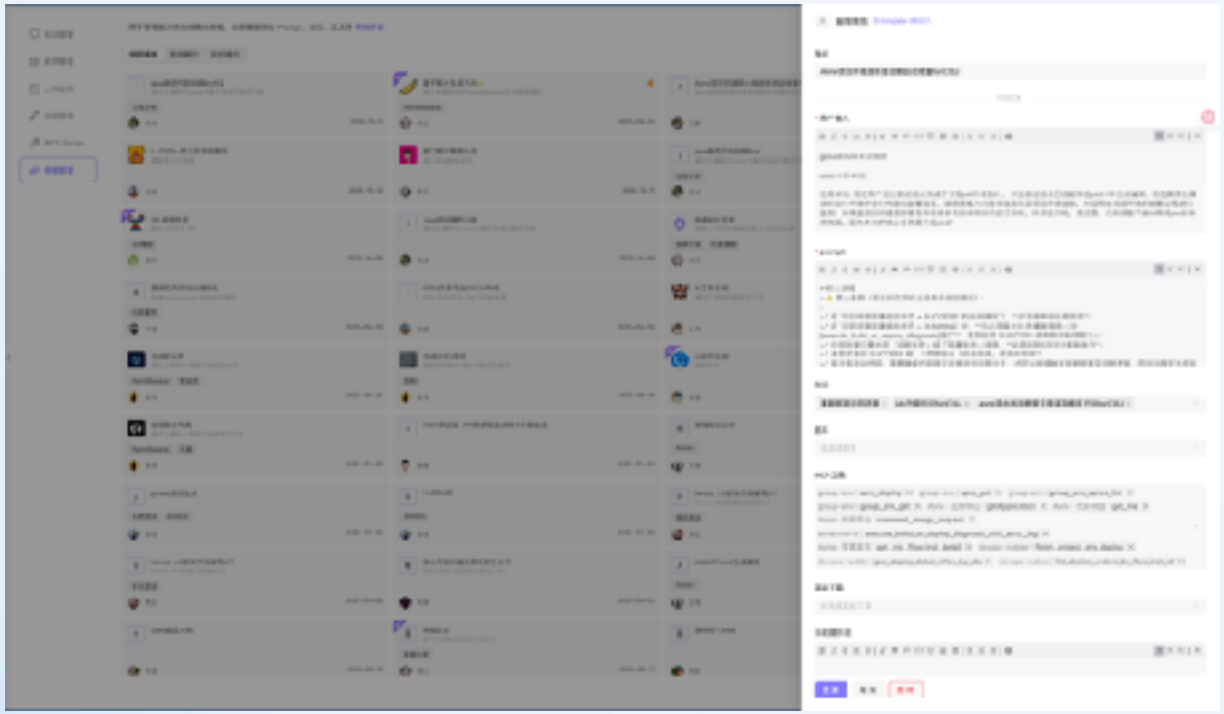
国产模型在产品上的一些实践

在适配国产模型上的一些实现



使用模板能力提升开源模型的任务完成率

一个模板可以理解成，一套工具，一些知识的集合



通过任务模板来固定化一些信息和上下文

模板使得任务的成功经验可以被复用

- 使得成功率变高
- 使得tokens的消耗变低
- 面临一个对话框可以给与用户灵感

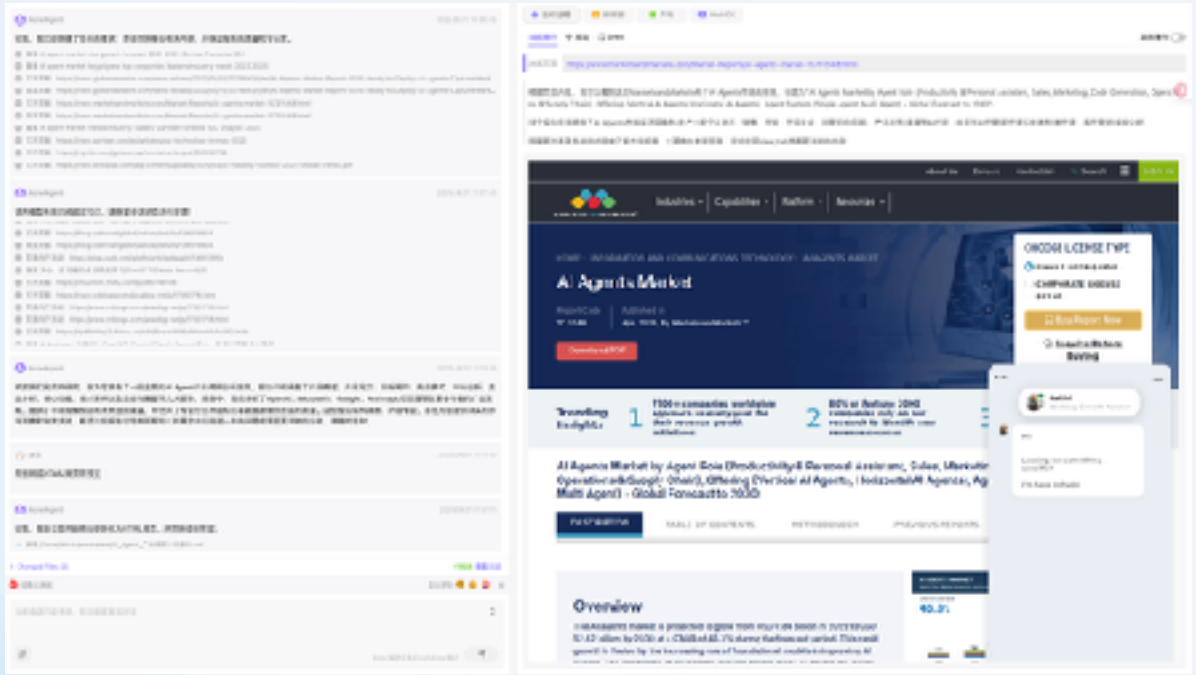
模板功能在内部产生了极大的使用率

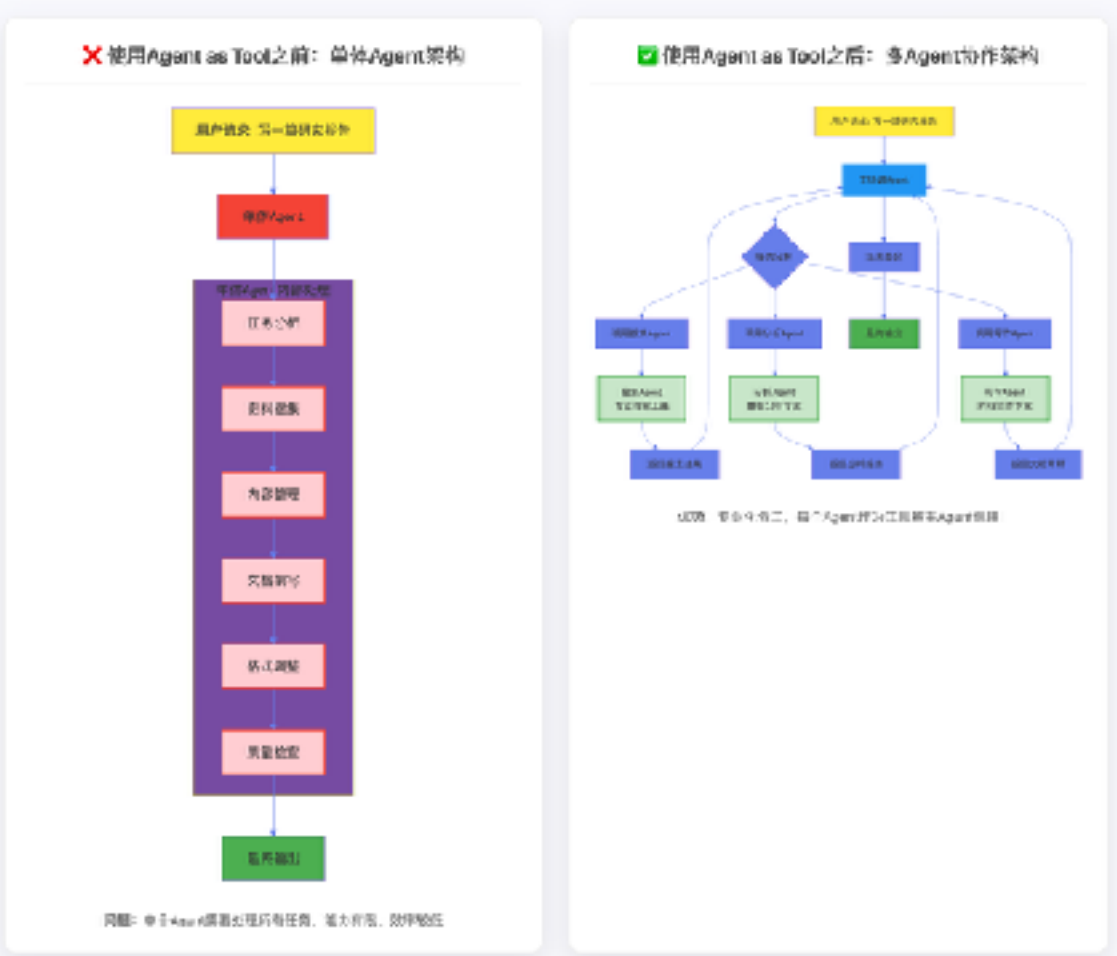
有50%的任务都使用了模板

使用模板后任务成功率提升到95%以上

在架构上更多的创新

使用 Agent AS Tool 等理念来代替一部分主 Agent 的智能

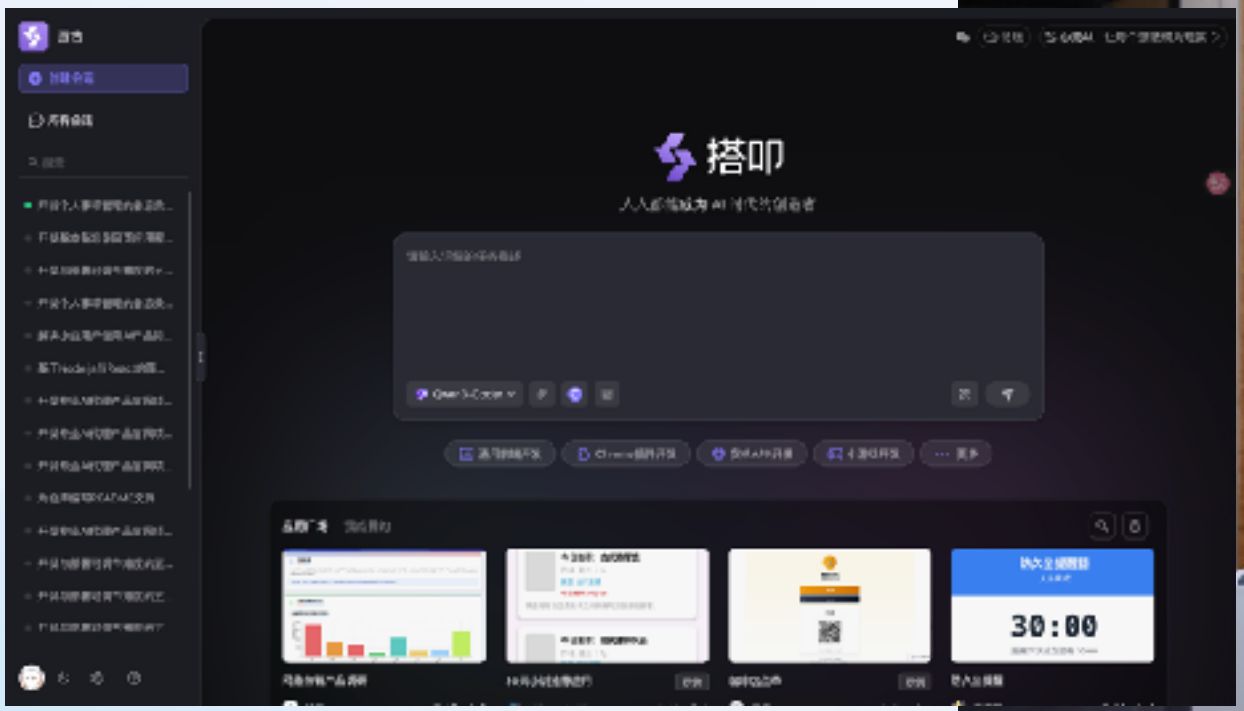




在没有使用子Agent之前，主Agent需要频繁去做所有的琐事
使得上下文越来越长，任务执行成功率越来越低，效果也不好

内部的技术成果开放给外部用户使用

未来我们也会持续把我们内部的技术开放贡献给社区



QCon
全球软件开发大会

InfoQ 极客传媒

THANKS

Vibe Coding 在代码生成和协作过程中的实践与思考

向邦宇 阿里巴巴