Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

Agent or not?

从程序自动修复评估智能体

硕士研究生 王怡男

2026年03月08日

问题回溯

- 问题回溯
  - 对内容熟悉程度低，准备不足
  - 难点部分把握不准，解释过多
  - 讲解缺乏具体实例，生硬枯燥
- 相关内容
  - 2025.06.16 王怡男《大模型支持的程序崩溃故障定位方法》
  - 2025.03.09 段学明《源代码补丁正确性测试》
  - 2024.08.11 张钊《自动化程序修复及其应用研究》
  - 2021.12.19 于浩淼《软件缺陷自动修复方法》

内容提要

- 预期收获
- 案例引入
- 题目内涵解析
- 研究背景与意义
- 研究历史与现状
- 知识基础
- 算法原理
  - SWE-agent
  - Agentless
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 1. 了解智能体的工作流程，以及为什么程序自动修复适合评估智能体
  - 2. 理解agent的价值不仅来自于模型本身
  - 3. 理解人工设计在LLM时代的不可替代性
  - 4. 形成评估智能体的基本视角

案例引入 OpenClaw的爆火

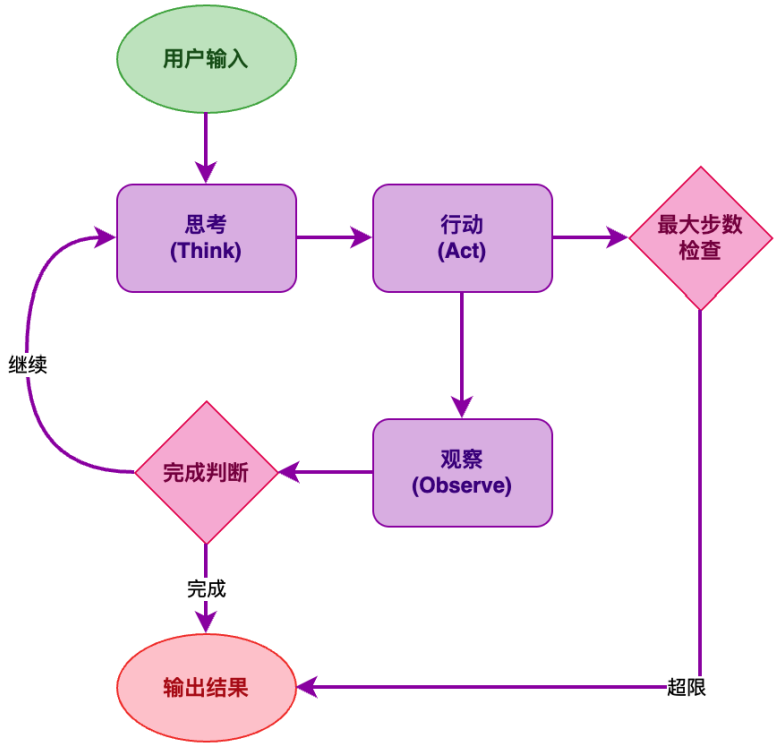
- OpenClaw
  - https://openclaw.ai/
  - 持续在线、可接入聊天渠道、可调用工具
- LLM解决问题的方式
  - AI不再只是在网页里回答问题
  - 从文本生成走向目标执行
- 更关键的问题
  - 当任务变得复杂时，我们需要
    - 自主的Agent?
    - 人工设计的良好流程系统?

腾讯宣布免费安装OpenClaw，近千名爱好者在鹅厂门口排长龙

最近，OpenClaw爆火，仅用百天便超越Linux登顶GitHub星标历史第一，引发全民“养虾”热潮。

题目内涵解析 智能体

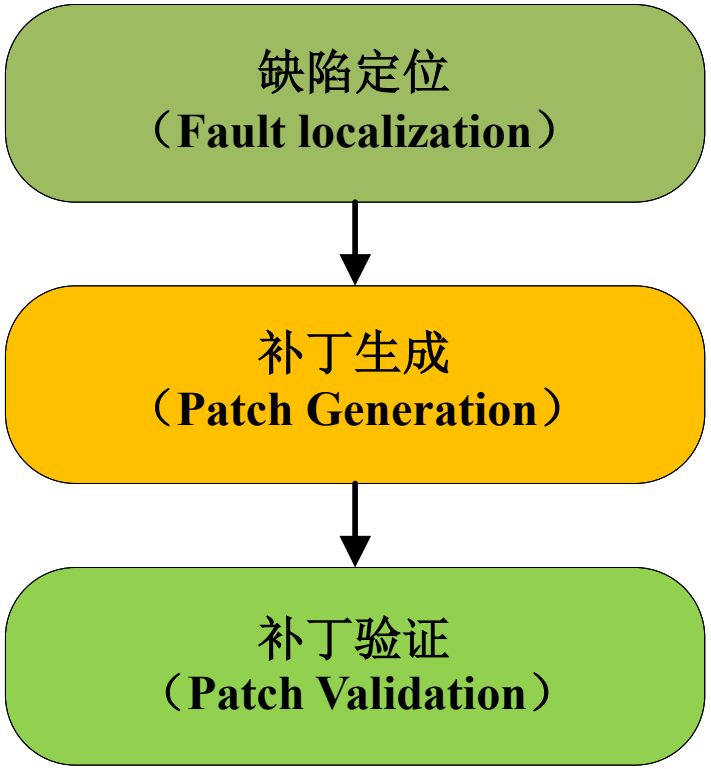
- 智能体（Agent）
  - 以目标为导向
  - 能够感知环境、调用工具、根据反馈迭代决策并持续执行任务的系统
- 是Agent？不是Agent？
  - 普通大语言模型（LLM）
    - 单次生成，无环境交互、无持续决策、无执行闭环
  - 工作流（Workflow）
    - 有业务逻辑，明确任务流转，明确状态如何改变
    - 可加入人工交互
  - 智能体（Agent）
    - 系统根据当前信息，自主决定下一步行动



LLM Agent解决问题的流程

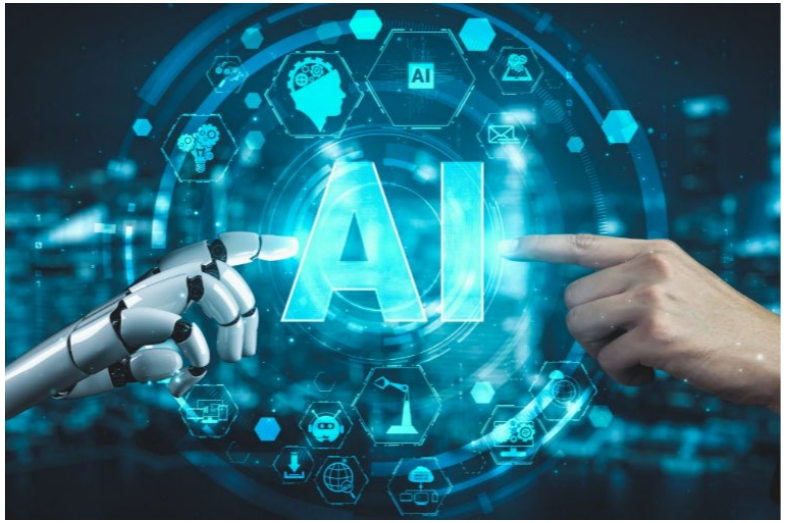
题目内涵解析 程序自动修复

- 程序自动修复（Automated Program Repair, APR）
  - 不是一次性代码生成任务
  - 一个包含缺陷定位、补丁生成、补丁验证的闭环任务
    - 缺陷定位（Fault localization）：改哪里?
      - 找到可疑文件、函数、代码行
    - 补丁生成（Patch Generation）：怎么改?
      - 生成候选补丁
    - 补丁验证（Patch Validation）：改得对?
      - 找到通过验证的最终补丁
- 适合讨论Agent
  - 软件工程关注的重要任务
  - 需要持续观察环境、采取动作、接收反馈、迭代修改



研究背景与意义

- 研究背景
  - 从代码生成到软件工程智能体
    - LLM在代码生成任务进步神速，但仓库级软件工程更复杂
  - 真实软件工程需要闭环
    - 程序缺陷修复带反馈、有验证、需迭代，是一个天然闭环的任务
  - 由于闭环复杂，学术界、工程界均走向Agent模式
    - 代码生成转向目标执行
  - Agent带来新的争议
    - 复杂的自主性Agent是否真的必要
- 研究意义
  - 将Agent能力放在可验证的场景内讨论
  - Agent能力 or 设计流程？谁更关键？

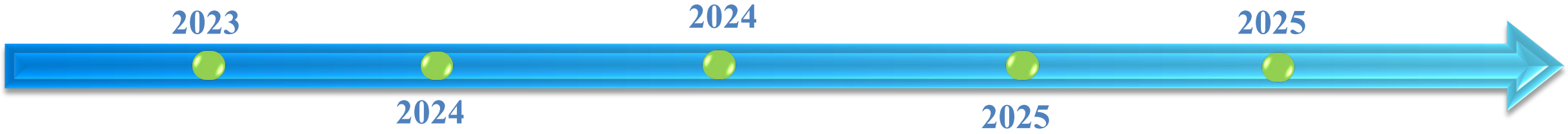


研究历史与现状 LLM Agent赋能的程序缺陷自动修复

SWE-bench：针对传统代码生成评测难以真实反映大模型软件工程能力的问题，提出面向真实仓库缺陷修复的评测框架SWE-bench。该基准首次将真实仓库、真实issue、跨文件修改与 执行环境中的补丁验证统一起来，使评测从小规模代码生成转向真实软件工程问题解决

AutoCodeRover：将LLM与结构化代码搜索相结合，基于AST利用类、方法等程序结构进行迭代式上下文检索，并在测试可用时结合SBFL进一步收缩搜索范围，从而生成最终补丁。该方法通过程序结构感知与精细化上下文检索提升issue理解与定位效果，体现了较强的软件工程导向

Agentless：针对现有agent方法在工具使用复杂、决策规划失控和错误累积放大等方面的不足，刻意不允许LLM自主进行工具调用或行动规划。该方法表明，真实软件工程任务中的性能提升未必依赖更复杂的自主智能体，也可能来自更简单、可控且可解释的流程设计；实验在开源方法中表现最优，并进一步推动了对“是否真的需要agent”的反思



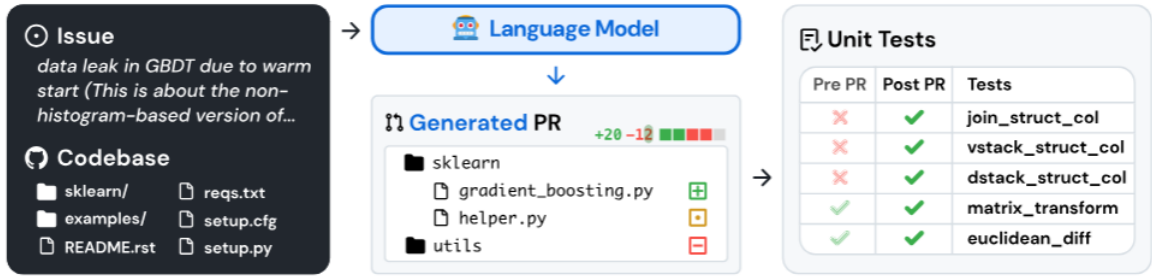
SWE-agent：首次系统引入ACI概念，为LLM设计专门的仓库搜索、文件查看、代码编辑与反馈机制。该方法表明，真实软件工程性能提升不仅来自更强模型，也来自更合适的接口设计；其实验在SWE-bench上达到12.47%、在Lite上达到18.00%

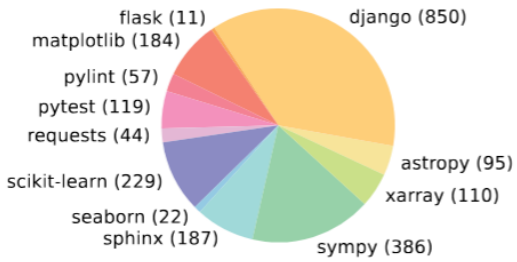
SpecRover：通过迭代式规格推断联合代码搜索，从项目结构与运行行为中推断预期语义，并利用reviewer agent审核补丁与给出置信信号。该方法将“意图理解”进一步前置到自动修复流程中，在SWE-bench上相较AutoCodeRover取得超过50%的效果提升，同时保持较低成本，说明规格推断在LLM时代仍是提升补丁质量的重要方向

知识基础 SWE-bench评测基准

- SWE-bench
  - https://www.swebench.com/
  - 面向真实软件工程问题的评测基准
  - 2294个任务，分布于12个Python仓库
    - 真实Github issue描述 + 对应Python仓库
    - 修改仓库以解决issue
  - 评测方法
    - 模型需要提交一个patch
    - 把patch应用到仓库后运行测试
    - *% Resolved*：修复成功的任务占总任务数的比例
    - *pass@k*：提交*k*个patch，能修复成功的比例
- SWE-bench Lite
  - SWE-bench的子集，包含300个质量更高、聚焦功能性的缺陷修复

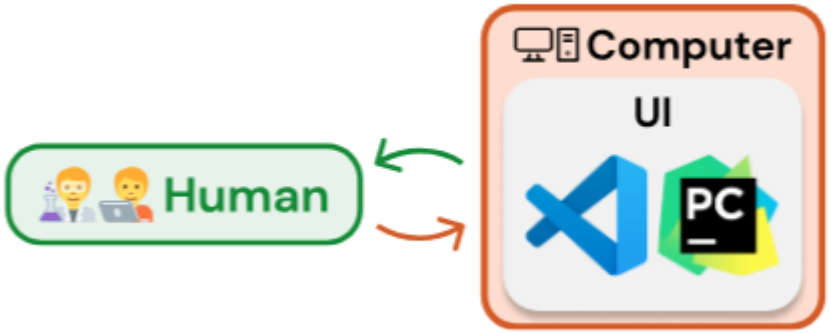
		Mean	Max
Issue Text	Length (Words)	195.1	4477
Codebase	# Files (non-test)	3,010	5,890
	# Lines (non-test)	438K	886K
Gold Patch	# Lines edited	32.8	5888
	# Files edited	1.7	31
	# Func. edited	3	36
Tests	# Fail to Pass	9.1	1633
	# Total	120.8	9459

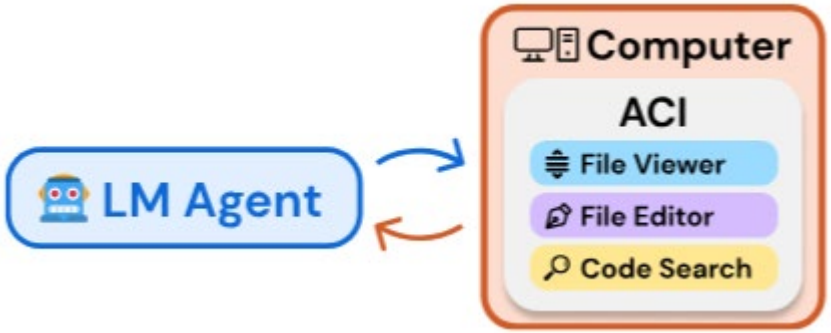




知识基础 ACI

- 智能体—计算机接口（Agent-Computer Interface, ACI）
  - Agent与计算机环境之间的抽象接口层
    - 指定Agent可用的命令
    - 规定环境状态如何反馈给Agent
    - 跟踪历史命令与历史观察
    - 总结内容，生成下一步输入
  - 为什么需要ACI?
    - 现有工具多为人类或API设计，不一定适合Agent
    - 直接操作shell往往动作琐碎、反馈不足，容易影响agent表现
  - 不影响模型参数





SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering

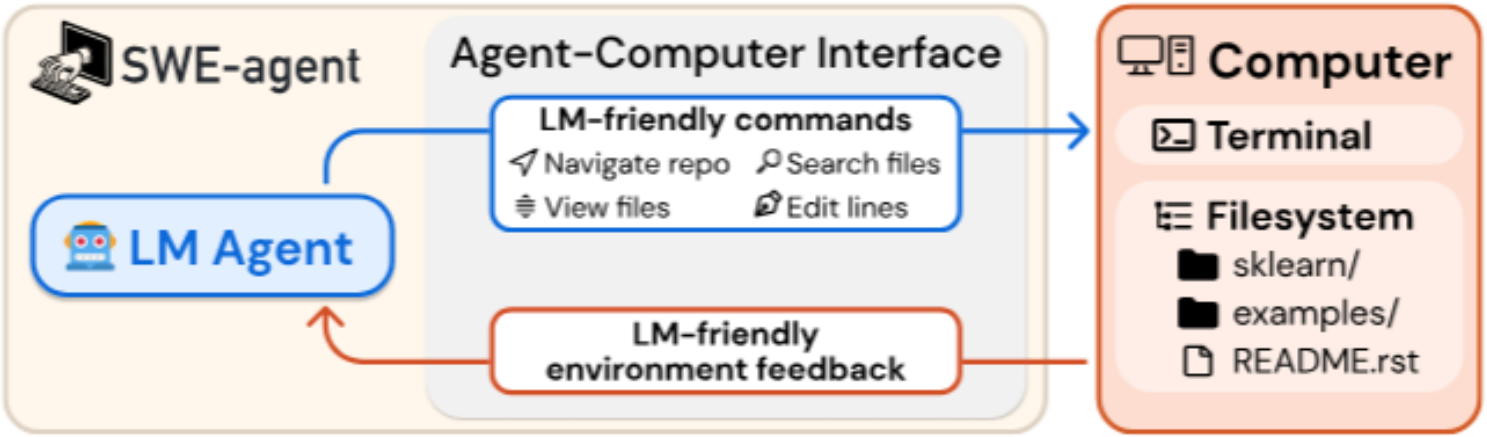
算法原理 TIPO

T	目标	解决真实代码仓库中的issue
I	输入	2294个issue描述，12个目标代码仓库
P	处理	1. 搜索与导航：使用命令在仓库中搜索文件名、字符串、目录级上下文，完成定位 2. 阅读并修改代码：使用命令查看文件并按行修改代码，编辑后系统反馈更新结果 3. 多轮交互：每一步先生成thought，再输出command，并接收环境反馈进入下一轮 4. 控制历史与错误恢复：管理历史上下文，通过简洁反馈与guardrails减少错误传播
O	输出	1个最终补丁、1条完整交互轨迹

P	问题	直接使用仅Shell或面向人类的UI并不适合LLM； 仓库级issue修复要求同时完成定位、编辑与验证
C	条件	需要Linux Shell执行环境；不修改LLM参数；需要预先设计的ACI
D	难点	ACI如何设计；仓库级上下文长，历史信息管理困难
L	水平	NeurIPS, 2024（CCF A）

算法原理 创新说明

- SWE-agent
  - 首次系统提出并验证ACI在agent中的关键作用
  - 围绕LLM特点设计面向软件工程的专用交互层，不直接依赖shell-only工具
    - 通过搜索摘要、行级编辑与guardrails显著提升交互稳定性与修复效果
  - 总结出一套可复用的ACI设计原则
    - 简单动作、紧凑操作、简洁反馈、guardrails防错



算法原理 ACI设计-搜索与导航

- 搜索与导航
  - 修复首先依赖高效定位
  - Shell原生命令对LLM并不友好
    - grep、find这类shell搜索命令容易返回过多结果，占满上下文窗口
    - cd + ls + cat逐层浏览会消耗多轮交互，效率很低
  - 从文件级、目录级到文件内搜索，形成由粗到细的定位
    - find_file：按文件名搜索仓库中的文件
    - search_dir：在目录级范围内搜索字符串
    - Search_file：在单文件范围内搜索字符串
  - 摘要化与结果裁剪
    - 搜索结果采用统一摘要格式
    - 单次查询结果限制50条以内，超过上限时建议模型改为更具体查询

Find Files

```
find_file atmosphere.py
```

```
Found 2 matches for "atmosphere.py" in /pvlib__pvlib-python/pvlib:
/pvlib__pvlib-python/pvlib/tests/atmosphere.py
/pvlib__pvlib-python/pvlib/atmosphere.py
```

Search Directory

```
search_dir APPARENT_ZENITH_MODELS
```

```
Found 4 matches for "APPARENT_ZENITH_MODELS" in /pvlib__pvlib-
python:
/pvlib__pvlib-python/docs/sphinx/source/whatsnew/v0.3.0.txt (1
matches)
/pvlib__pvlib-python/pvlib/atmosphere.py (2 matches)
/pvlib__pvlib-python/pvlib/location.py (1 matches)
End of matches for "APPARENT_ZENITH_MODELS" in /pvlib__pvlib-
python
```

Search File

```
search_file APPARENT_ZENITH_MODELS
```

```
Found 2 matches for "APPARENT_ZENITH_MODELS" in /pvlib__pvlib-
python/pvlib/atmosphere.py:
Line 12:APPARENT_ZENITH_MODELS = ('simple', 'kasten1966',
'kastenyoung1989',
Line 15:AIRMASS_MODELS = APPARENT_ZENITH_MODELS +
TRUE_ZENITH_MODELS
End of matches for "APPARENT_ZENITH_MODELS" in /pvlib__pvlib-
python/pvlib/atmosphere.py
```

算法原理 ACI设计-文件查看与编辑

- 文件查看
  - Shell原生命令不适合LLM做文件阅读与文件内导航
    - Cat、printf容易一次输出过多内容
    - head、tail不方便在文件内部灵活跳转
  - 提供open、scroll_down、scroll_up、 goto4个命令
    - 每次最多显示100行

Observation (File Viewer)

```
[File: solvers/diophantine.py (3093 lines total)]
(400 lines above)
401:    elif total_degree == 3:
402:        diop_type = "cubic_thue"
403:
404:    elif (total_degree > 3):
405:        if all(coeff[k] == 1):
406:            diop_type = 'general_sum_even'
407:
408:    if diop_type is not None:
409:        return var, coeff, diop_type
(2684 lines below)
```

- 文件编辑
  - Shell原生命令不适合LLM做文件编辑
    - 参数复杂，容易出错
    - 编辑结果“静默”，需额外确认是否修改成功
  - 提供edit、create2个编辑命令
  - 提供submit命令用于提交最终patch
  - Guardrails：编辑后自动检查语法错误，有错则本次编辑不执行

Response (Thought + Edit Action)

We need to add a check to ensure that all powers are the same. Let's edit the code to include this check.

```
edit 404:407
    elif (total_degree > 3 and
          len(set(k.exp[k.is_Pow])) == 1):
        if all(coeff[k] == 1):
            diop_type = 'general_sum_even'
end_of_edit
```

算法原理 ACI设计-上下文管理

- 上下文管理
  - 多轮闭环任务，历史越长，不代表信息越有用
    - 累计旧的目录输出、文件内容、命令反馈，上下文膨胀
    - 旧的观察结果已经过时或重复，保留会干扰后续决策
  - *Thought* + *Action*格式约束
    - 每一步都要求模型同时生成*thought*和*action*
    - 如果输出格式错误，系统返回错误消息，要求重新生成
  - 环境反馈
    - 当前命令输出、当前打开文件、当前工作目录
    - 命令成功执行但无输出时，系统明确告知
  - 旧观察结果处理
    - 保留历史顺序和交互结构
    - 旧观察结果的具体内容折叠成一行占位符

Next Step Template

```
{OBSERVATION}
(Open file: /path/to/open/file.py)
(Current directory: /path/to/cwd)
bash-$
```

Error Message

```
Your output was not formatted correctly. You must always include one
discussion and one command as part of your response. Make sure you do
not have multiple discussion/command tags.
Please make sure your output precisely matches the following format:
DISCUSSION
Discuss here with yourself about what your planning and what you're
going to do in this step.

```
command(s) that you're going to run
```
```

Environment Response (collapsed) Template

```
Old output omitted (101 lines)
```

实验设计 数据资源

- 数据资源
  - SWE-bench：完整版 2294个任务，Lite版本300个任务
  - HumanEvalFix：聚焦代码编辑与调试的数据集，164个任务
- 评估标准
  - *% Resolved / pass@1*
  - *$ Avg. Cost*：平均API推理成本
- 对比方法
  - SWE-bench作者给出的RAG基线
  - Shell-only（与shell进程交互解决问题）
  - 一些针对代码任务优化过的LLM
    - GPT-4、CodeLLaMa-instruct-13B、DeepseekCoder-CodeAlpaca-6.7B、WaveCoder-DS-6.7B

Dataset	Released	License	Splits	Count	Languages	GitHub Repo
SWE-bench	10/10/2023	MIT	Test Lite Dev	2294 300 225	Python	princeton-nlp/ SWE-bench
HumanEvalFix	07/23/2023	MIT	Test	164	Python, JS, Go Java, C++, Rust	bigcode-project/ octopack

实验设计 对比实验

- 对比实验结果
  - SWE-agent在SWE-bench全量和Lite版本的*% Resolved*分别达到12.47和18.00，均优于RAG方法和Shell-only方法
  - SWE-agent在基座模型为GPT-4和Claude 3时均有发挥，表明ACI不仅对单一模型有效
  - 有ACI帮助的SWE-agent效果优于针对代码任务优化过的LLM

Model	SWE-bench		SWE-bench Lite	
	% Resolved	$ Avg. Cost	% Resolved	$ Avg. Cost
RAG				
w/ GPT-4 Turbo	1.31	0.13	2.67	0.13
w/ Claude 3 Opus	3.79	0.25	4.33	0.25
Shell-only agent				
w/ GPT-4 Turbo	-	-	11.00	1.46
w/o Demonstration	-	-	7.33	0.79
SWE-agent				
w/ GPT-4 Turbo	**12.47**	1.59	**18.00**	1.67
w/ Claude 3 Opus	10.46	2.59	13.00	2.18

Model	Python	JS	Java
CodeLLaMa-instruct-13B	29.2	19.5	32.3
GPT-4	47.0	48.2	50.0
DeepseekCoder-CodeAlpaca-6.7B	49.4	51.8	45.1
WaveCoder-DS-6.7B	57.9	52.4	57.3
SWE-agent w/ GPT-4 Turbo	**87.7**	**89.7**	**87.9**

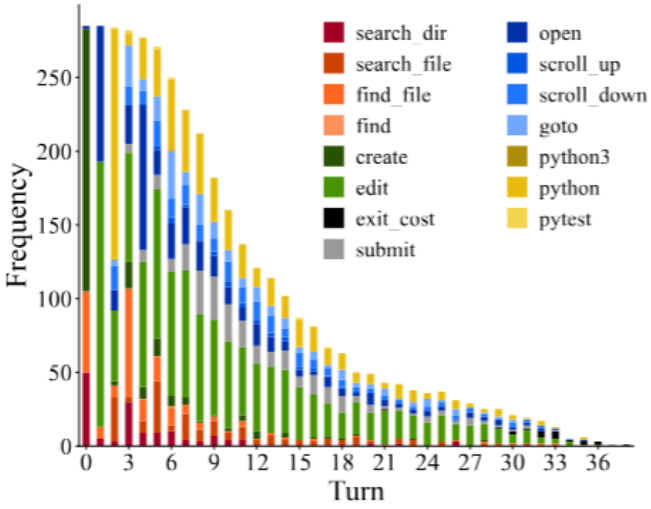
实验设计 消融实验

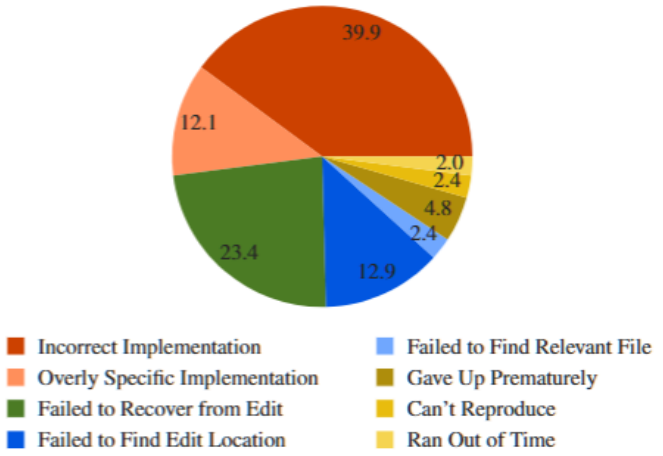
- 消融实验结果
  - 摘要式搜索优于逐条翻看的搜索，逐条搜索甚至不如不搜索
    - 成本和上下文的浪费
  - 编辑能力最为重要，guardrails能进一步减少错误传播
  - 文件查看的窗口存在适中最优值
  - 多轮agent的关键是保留最近、最相关的交互信息

Editor		Search		File Viewer		Context	
`edit` action	15.0 ↓ 3.0	Summarized	18.0	30 lines	14.3 ↓ 3.7	Last 5 Obs.	18.0
w/ linting	18.0	Iterative	12.0 ↓ 6.0	100 lines	18.0	Full history	15.0 ↓ 3.0
No `edit`	10.3 ↓ 7.7	No search	15.7 ↓ 2.3	Full file	12.7 ↓ 5.3	w/o demo.	16.3 ↓ 1.7

实验设计 Agent行为分析

- Agent行为分析结果
  - 成功轨迹
    - 通常从create复现问题，或find_file / search_dir 定位问题开始
    - Turn 5之后，最常见的两个动作是edit和python
      - 大多进入“编辑—执行—再编辑”的循环
  - 失败模式
    - 实现不对或实现过于具体
    - 即使应用了上下文管理，编辑恢复仍然困难
      - 有51.7%的轨迹出现过至少一次failed edit
      - 一次failed edit后，最终成功编辑的概率会从90.5%降到57.2%





算法总结

- 算法贡献
  - 提出ACI视角
    - 提出接口设计是智能体性能的核心问题，而不仅仅是模型能力
  - 构建面向软件工程的交互式闭环系统
    - 围绕搜索、查看、编辑、上下文管理设计LLM友好的软件工程接口
  - 总结出一套可复用的ACI设计原则
    - 简单动作、紧凑操作、简洁反馈、guardrails防错
- 算法局限
  - ACI设计过程高度依赖人工经验
    - 开发与分析过程主要是手工完成，自动化程度不足
    - 依赖对Agent行为分析后的反复修改、迭代

算法原理 Agentless

Demystifying LLM-Based Software Engineering Agents

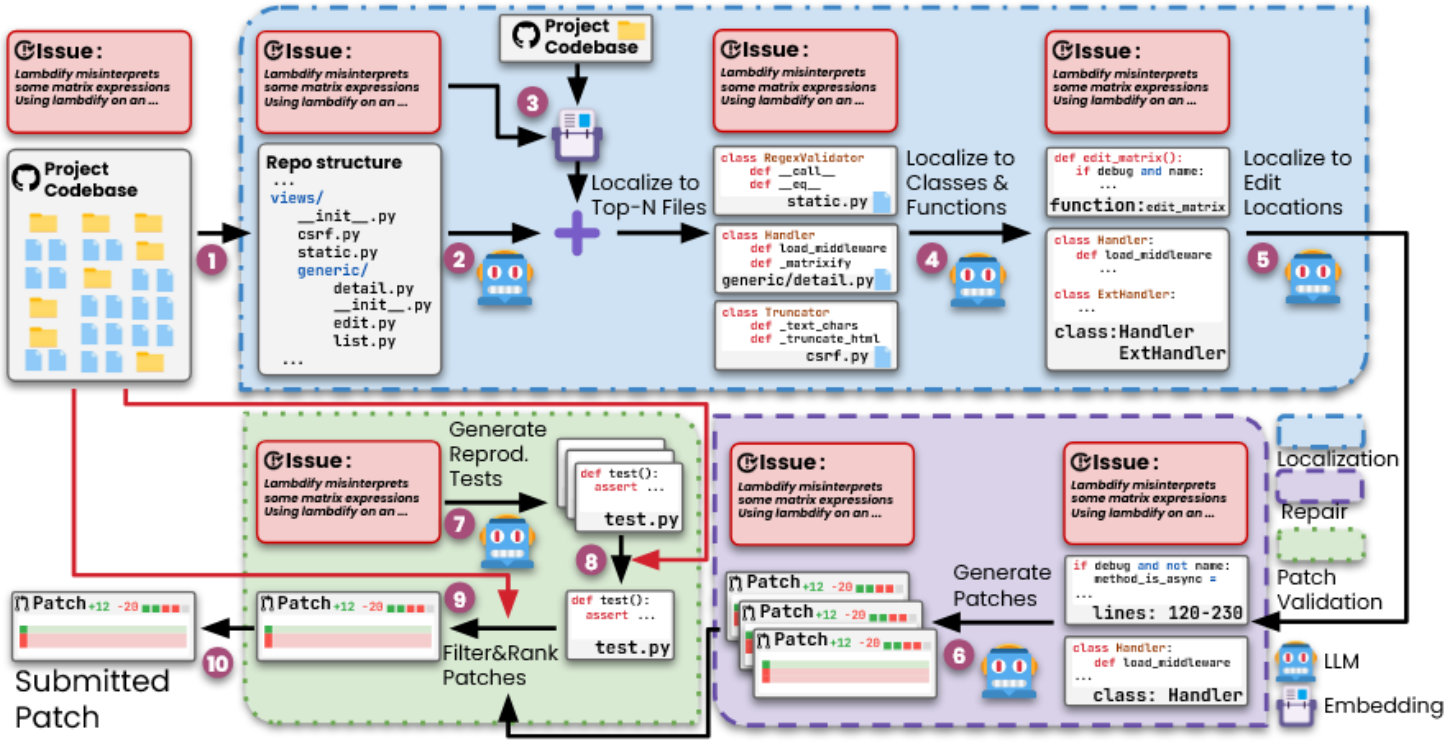
算法原理 TIPO

T	目标	解决真实代码仓库中的issue
I	输入	2294个issue描述，12个目标代码仓库，已有测试与运行结果
P	处理	1. 缺陷定位：从文件到元素、再到编辑位置，完成层次化定位 2. 补丁生成：基于编辑位置完成小范围替换，生成多个候选补丁 3. 补丁验证：结合复现测试与回归测试，选出最终补丁
O	输出	1个最终补丁

P	问题	Agent存在工具设计复杂、决策失控和错误放大等问题
C	条件	依赖测试环境做复现测试与回归测试
D	难点	原始仓库通常没有现成的复现测试
L	水平	FSE, 2025(CCF A)

算法原理 创新说明

- Agentless
  - 从“做更复杂的agent”转向“反思是否真的需要agent”
  - 将层次化定位与简单diff patch结合，降低上下文与编辑复杂度
  - 引入复现测试与回归测试的补丁验证机制，提升最终patch质量



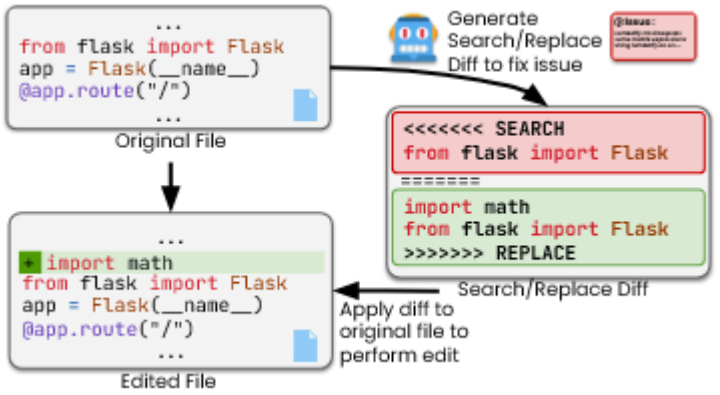
算法原理 缺陷定位

- 层次化定位
  - 定位可疑文件
    - 提示词检索
      - 将整个仓库转成存储库结构，类似Linux的tree命令
      - 将issue描述和存储库结构一起输入LLM，要求输出Top-N可疑文件
    - 基于嵌入的检索
      - LLM过滤无关文件夹、剩余文件分块嵌入、与issue描述做相似度匹配
  - 定位相关元素
    - 保留骨架格式：类、函数、变量声明，类级、模块级注释
  - 定位具体编辑位置
    - 根据LLM找到的类/函数，给出这些位置的完整代码
    - LLM输出更小的编辑位置
      - 类、函数、具体代码行

```
class UUIDField(CharField):
    default_error_messages = {
        'invalid': _('Enter a valid UUID.'),
    }
    def prepare_value(self, value):
...
    def to_python(self, value):
...
class JSONField(CharField):
    default_error_messages = {
        'invalid': _('Enter a valid JSON.'),
    }
    widget = Textarea
    def __init__(self, encoder=None, decoder=None):
...
    def to_python(self, value):
    # Process JSON path from the right-hand side
...
def slugify(value, allow_unicode=False):
```

算法原理 补丁生成

- 补丁生成
  - 局部上下文构造
    - 对定位好的编辑位置，截取局部上下文窗口 [40−x, 78+x]
    - 若存在多个编辑位置，将多个局部窗口拼接起来
  - Search/Replace diff
    - Search：原始代码片段
    - Replace：替换后的代码片段
    - 不生成完整代码，只聚焦小范围修改
  - 多候选补丁生成
    - 先贪心（temperature=0）
      - 稳定、保守
    - 再高温采样多个补丁（temperature=0.8）
      - 跳跃、发散，增加“撞”到正确补丁的概率



算法原理 补丁验证

- 补丁验证
  - 复现测试：补丁是否使原有的fail变为pass
  - 回归测试：补丁是否使原有的pass变成fail
  - 复现测试生成
    - 原始仓库通常只有回归测试，没有现成的复现测试
    - LLM自动生成完整复现测试，测试输出为3种信号
      - 问题重现了、问题已解决、其他问题
  - 回归测试过滤
    - 先运行仓库里所有已有测试，找出原始代码就能pass的测试
    - LLM筛选哪些测试不应作为回归测试，移除之
  - 补丁选择
    - 过滤回归测试失败的补丁、保留复现测试成功的补丁
    - 补丁规范化后，提交出现类型最多的补丁

```python
from sqlfluff import lint

def test__rules__std_L060_raised() -> None:
    try:
        sql = "SELECT   IFNULL(NULL, 100),
            NVL(NULL,100);"
        result = lint(sql, rules=["L060"])
        assert len(result) == 2
    except:
        print("Other issues")
        return

    try:
        assert result[0]["description"] ==
"Use 'COALESCE' instead of 'IFNULL'."
        assert result[1]["description"] ==
"Use 'COALESCE' instead of 'NVL'."
        print("Issue resolved")
    except AssertionError:
        print("Issue reproduced")
        return

    return

test__rules__std_L060_raised()
```

```python
# patch A
if x is None:
    return 0
```

```python
# patch B
if x is None:
    # handle empty case
    return 0
```

实验设计 数据资源

- 数据资源
  - SWE-bench Lite：300个任务
- 评估标准
  - *% Resolved / pass@1*
  - *$ Avg. Cost*：平均API推理成本
  - *$ Avg. Tokens*：平均token使用量
  - *% Correct Location*：补丁是否覆盖ground-truth的真实编辑位置
- 对比方法（26个）
  - 闭源/商业方法：CodeStory Aide、MarsCode等
  - 开源方法：AutoCodeRover(-v2)、SWE-agent等
  - SWE-bench作者给出的RAG基线

Tool	LLM
CodeStory Aide [Codestory 2024]	GPT-4o+Claude 3.5 S
Bytedance MarsCode [Liu et al. 2024c]	NA
Honeycomb [Honeycomb 2024]	NA
MentatBot [MentatBot 2024]	GPT-4o
Gru [Gru 2024]	NA
Isoform [Isoform 2024]	NA
SuperCoder2.0 [SuperCoder 2024]	NA
Alibaba Lingma Agent [Lingma 2024]	GPT-4o+Claude 3.5 S
Factory Code Droid [Factory 2024]	NA
Amazon Q Developer-v2 [Amazon 2024]	NA
SpecRover [Ruan et al. 2024]	GPT-4o+Claude 3.5 S
CodeR [Chen et al. 2024]	GPT-4
MASAI [Arora et al. 2024]	NA
SIMA [SIMA 2024]	GPT-4o
IBM Research Agent-101 [IBM 2024]	NA
OpenCSG StarShip [OpenCSG 2024]	GPT-4
Amazon Q Developer [Amazon 2024]	NA
RepoUnderstander [Ma et al. 2024]	GPT-4
AutoCodeRover-v2 [AutoCodeRover 2024]	GPT-4o
RepoGraph [RepoGraph 2024]	GPT-4o
Moatless [Örwall 2024]	Claude 3.5 S GPT-4o
OpenDevin+CodeAct v1.8 [OpenDevin 2024]	Claude 3.5 S
Aider [Gauthier 2024]	GPT-4o+Claude 3.5 S
SWE-agent [Yang et al. 2024a]	Claude 3.5 S GPT-4o GPT-4
AppMap Navie [AppMap 2024]	GPT-4o
AutoCodeRover [Zhang et al. 2024c]	GPT-4
RAG [Yang et al. 2024a]	Claude 3 Opus GPT-4 Claude-2 GPT-3.5
AGENTLESS	GPT-4o

实验设计 对比实验

- 对比实验结果
  - Agentless在SWE-bench Lite上达到32.00%（96/300）的*% Resolved*，平均成本$0.70，优于所有开源方法
  - Agentless优于部分闭源/商业方法
  - Agentless的*% Correct Location*值表现良好，定位质量也具有竞争力

Tool	LLM	% Resolved	Avg. $ Cost	Avg. # Tokens	% Correct Location Line	Function	File
CodeStory Aide [Codestory 2024]	GPT-4o+Claude 3.5 S	129 (43.00%)	-	-	28.7%	56.3%	72.0%
Bytedance MarsCode [Liu et al. 2024c]	NA	118 (39.33%)	-	-	29.3%	57.0%	79.7%
Honeycomb [Honeycomb 2024]	NA	115 (38.33%)	-	-	34.0%	56.0%	69.3%
MentatBot [MentatBot 2024]	GPT-4o	114 (38.00%)	-	-	24.0%	52.3%	68.7%
Gru [Gru 2024]	NA	107 (35.67%)	-	-	24.0%	51.0%	74.3%
Isoform [Isoform 2024]	NA	105 (35.00%)	-	41,963	24.3%	53.7%	72.0%
SuperCoder2.0 [SuperCoder 2024]	NA	102 (34.00%)	-	-	29.7%	63.3%	65.7%
Alibaba Lingma Agent [Lingma 2024]	GPT-4o+Claude 3.5 S	99 (33.00%)	-	-	28.3%	57.3%	74.7%
Factory Code Droid [Factory 2024]	NA	94 (31.33%)	-	-	23.0%	55.0%	72.7%
Amazon Q Developer-v2 [Amazon 2024]	NA	89 (29.67%)	-	-	29.3%	54.3%	73.7%
SpecRover [Ruan et al. 2024]	GPT-4o+Claude 3.5 S	93 (31.00%)	$0.65	-	-	-	-
CodeR [Chen et al. 2024]	GPT-4	85 (28.33%)	$3.34	323,802	23.0%	50.7%	66.3%
MASAI [Arora et al. 2024]	NA	84 (28.00%)	-	-	25.0%	55.7%	75.0%
SIMA [SIMA 2024]	GPT-4o	83 (27.67%)	$0.82	-	22.0%	51.7%	78.7%
IBM Research Agent-101 [IBM 2024]	NA	80 (26.67%)	-	-	27.7%	55.7%	72.7%
OpenCSG StarShip [OpenCSG 2024]	GPT-4	71 (23.67%)	-	-	22.3%	60.3%	88.3%
Amazon Q Developer [Amazon 2024]	NA	61 (20.33%)	-	-	22.3%	43.0%	71.3%
RepoUnderstander [Ma et al. 2024]	GPT-4	64 (21.33%)	-	-	-	-	-
AutoCodeRover-v2 [AutoCodeRover 2024]	GPT-4o	92 (30.67%)	-	-	23.0%	49.7%	69.0%
RepoGraph [RepoGraph 2024]	GPT-4o	89 (29.67%)	-	-	20.7%	49.0%	71.0%
Moatless [Örwall 2024]	Claude 3.5 S	80 (26.67%)	$0.17	-	25.0%	53.7%	78.3%
	GPT-4o	74 (24.67%)	$0.14	-	21.7%	50.0%	72.7%
OpenDevin+CodeAct v1.8 [OpenDevin 2024]	Claude 3.5 S	80 (26.67%)	$1.14	-	23.3%	48.7%	66.3%
Aider [Gauthier 2024]	GPT-4o+Claude 3.5 S	79 (26.33%)	-	-	21.3%	47.3%	69.3%
SWE-agent [Yang et al. 2024a]	Claude 3.5 S	69 (23.00%)	$1.62	521,208	28.3%	54.0%	71.7%
	GPT-4o	55 (18.33%)	$2.53	498,346	21.0%	42.3%	58.0%
	GPT-4	54 (18.00%)	$2.51	245,008	20.0%	43.7%	60.3%
AppMap Navie [AppMap 2024]	GPT-4o	65 (21.67%)	-	-	19.7%	43.7%	58.0%
AutoCodeRover [Zhang et al. 2024c]	GPT-4	57 (19.00%)	$0.45	38,663	18.0%	40.7%	62.0%
RAG [Yang et al. 2024a]	Claude 3 Opus	13 (4.33%)	$0.25	-	7.0%	24.0%	40.3%
	GPT-4	8 (2.67%)	$0.13	-	3.7%	16.3%	34.0%
	Claude-2	9 (3.00%)	-	-	7.0%	19.7%	34.7%
	GPT-3.5	1 (0.33%)	-	-	0.3%	3.7%	13.0%
AGENTLESS	GPT-4o	96 (32.00%)	$0.70	78,165	20.7%	50.3%	69.7%

实验设计 消融实验

- 消融实验结果
  - 缺陷定位的消融分析
    - 提示词检索和基于嵌入的检索互补，两者结合才能达到最佳效果
  - 补丁生成的消融分析
    - 高温采样补丁可以增加Agentless的性能
  - 补丁验证的消融分析
    - 复现测试和回归测试的结合是最终性能提升的关键来源

Method	Performance	Avg. $
Greedy location (40 samples)	88 (29.33%)	$0.22
Multi-samples merged (40 samples)	85 (28.33%)	$0.24
Multi-samples (4 x 10 samples)	96 (32.00%)	$0.29

Method	Contains GT	LoC	Avg. $
File level localization			
Prompting-based	78.67%	3221	$0.02
Embedding-based (w/o irrelevant filtering)	67.67%	3388	$0.05
Embedding-based (w/ irrelevant filtering)	70.33%	3622	$0.04
Combined	81.33%	3424	$0.06
Related element localization			
Complete file	56.67%	778	$0.15
Skeleton format	62.00%	698	$0.02
Edit location localization			
Greedy	53.67%	189	$0.06
Direct from file-level	52.00%	208	$0.18
Multi-samples merged	59.33%	342	$0.07
Multi-samples individual	53.33% 53.33% 52.67% 51.33%	165 180 168 213	$0.07

Method	Performance	Avg. $
Majority voting	77 (25.67%)	$0.00
+Regression test	81 (27.00%)	$0.01
+Reproduction test	96 (32.00%)	$0.25

算法总结

- 算法贡献
  - 反向提出“Agentless”视角，重新追问复杂自主agent的必要性
    - 简单的三阶段流程也可能更有效、更易复现
  - 将复现测试与回归测试引入补丁验证和选择
  - 对SWE-bench Lite数据的局限性做出了分析，构造了更严格的SWE-bench Lite-S
- 算法局限
  - 补丁验证与选择仍然依赖启发式规则
  - 结果主要建立在SWE-bench Lite上，泛化性需进一步验证
  - 存在数据泄漏风险
    - 使用GPT-4o，无法排除ground-truth出现在训练数据中的可能性

特点总结与未来展望

特点总结与未来展望

- 特点总结
  - SWE-agent
    - agent性能不仅取决于模型，也取决于系统设计
  - Agentless
    - 复杂自主agent并不一定是唯一答案
- 未来展望
  - 何时需要怎样的agent?
    - 哪些任务需要强自主性
    - 哪些任务更适合分阶段流程
  - ACI设计从人工经验走向自动化设计

参考文献

[1] Jimenez C E, Yang J, Wettig A, et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?[A]. arXiv, 2024.

[2] Yang J, Jimenez C E, Wettig A, et al. Swe-agent: Agent-computer interfaces enable automated software engineering[J]. Advances in Neural Information Processing Systems, 2024, 37: 50528-50652.

[3] Xia C S, Deng Y, Dunn S, et al. Demystifying LLM-Based Software Engineering Agents[J]. Proceedings of the ACM on Software Engineering, 2025, 2(FSE): 801-824.

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！