RepairAgent: An Autonomous, LLM-Based Agent for Program Repair

Info

- Tags: APR, Agents framework, LLM engineering
- Author: Bouzenia I, Devanbu P, Pradel M.
- CCF Rank: N/A
- Conference: Arxiv
- Institution: University of Stuttgart
- Link: https://arxiv.org/abs/2403.17134
- Repo: https://github.com/sola-st/RepairAgent
- Types: Paper
- Year: Mar, 2024

RepairAgent: Pioneering Autonomous Program Debugging & Repair with AI [Part 1]

RepairAgent：基于 LLM 的自主程序修复代理 | BriefGPT - AI 论文速递

- 场景：APR修复bug
- 方法：
    - 交互循环

    - 动态prompt
    - 有限状态机架构
    - 工具调用和命令执行
    - 可解释性
- 特点：
    - 错误定位
    - 信息收集和分析
    - 简单修复和复杂修复
    - 迭代改进

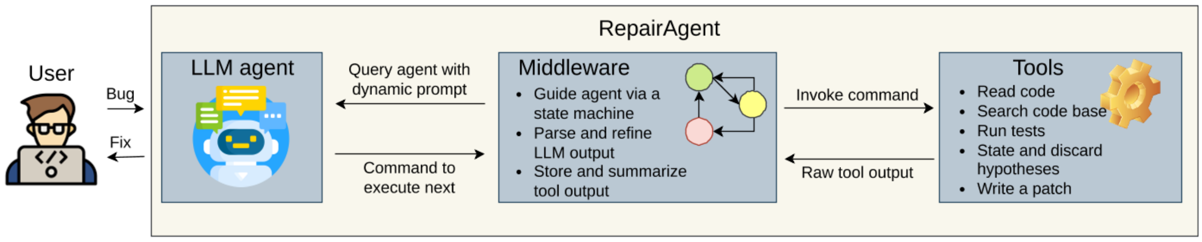
Intro

- Software bugs
- APR/semi-APR
- LLM-based APR
    - Gen1:和模型的一次性交互
    - Gen2:根据输出迭代重复查询LLM
    - Limitation：硬编码反馈循环不符合人类dev修复的方式，人类会收集信息以理解错误，搜索可能有帮助修复的代码和实验patch
- Contribution
    - Repair Agent：第一个用于APR的基于LLM的Agent（FixAgent？）
    - 动态的prompt格式更新
    - 一组用于bug修复的工具，LLM自主使用
    - 中间件

    - 新的SOTA
- Main components
    - 通用LLM：GPT3.5
    - 动态prompt
    - 14种工具集
    - 中间件：协调LLM和工具的通信
    - 有限状态机：引导合理的工具调用和状态转移
- 评估结果：
    - 和SOTA（keep the conversation going...）（ITER: Iterative Neural Repair for Multi-Location Patches）比较
    - 数据集：Defects4J v1.2/2.0 共835个bug
    - 成功修复了74/90个，其中49个涉及多行修复
    - 其中39个为SOTA没有修复
    - 成本：0.14$/bug
    - 结论：新的SOTA

Approach

Framework


RepairAgent
User
Bug
Fix
LLM agent
Query agent with dynamic prompt
Command to execute next
Middleware
Guide agent via a state machine
Parse and refine LLM output
Store and summarize tool output
Invoke command
Raw tool output
Tools
Read code
Search code base
Run tests
State and discard hypotheses
Write a patch
Fig. 1: Overview of RepairAgent.

- Component

    - Agent
    - Middleware
    - Tools
- Workflow
    1. Query the agent using prompt
    2. Process the response
    3. Execute the command
    4. Update prompt based on output
    5. Stop when bug has been fixed or reach the budget

Dynamic prompt

Prompt contains *static* and *dynamic* prompt section.

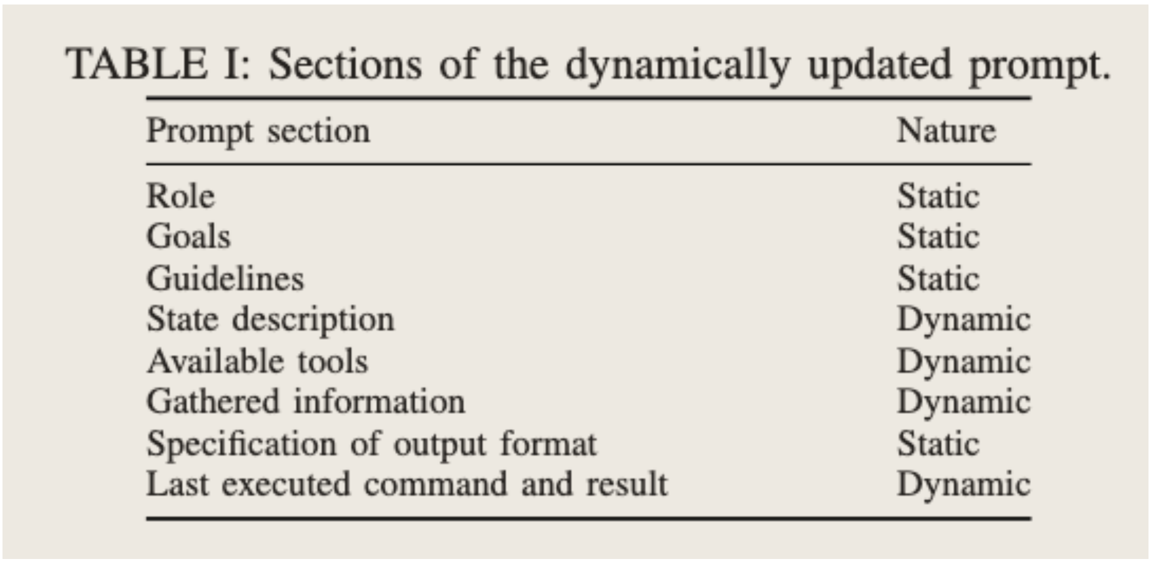


TABLE I: Sections of the dynamically updated prompt.

Prompt section	Nature
Role	Static
Goals	Static
Guidelines	Static
State description	Dynamic
Available tools	Dynamic
Gathered information	Dynamic
Specification of output format	Static
Last executed command and result	Dynamic

Here is how a prompt built:

1. Role : define the area of agent’s expertise. Rely on auto analysis
2. Goals: locate the bug/gather info about the bug/suggest simple fixes/suggest complex fixes/**Iterate** over the previous goals
3. Guidelines:

    1. Diverse kinds of bugs；Single-line or multi-lines
    2. Provide a simple example to illustrate how to fix
    3. Ask LLM to insert comments when trying to fix to enhance the reasoning abilities
    4. Ask LLM to define what is the next step
    5. Ask LLM to select tools efficiently

4. State description:

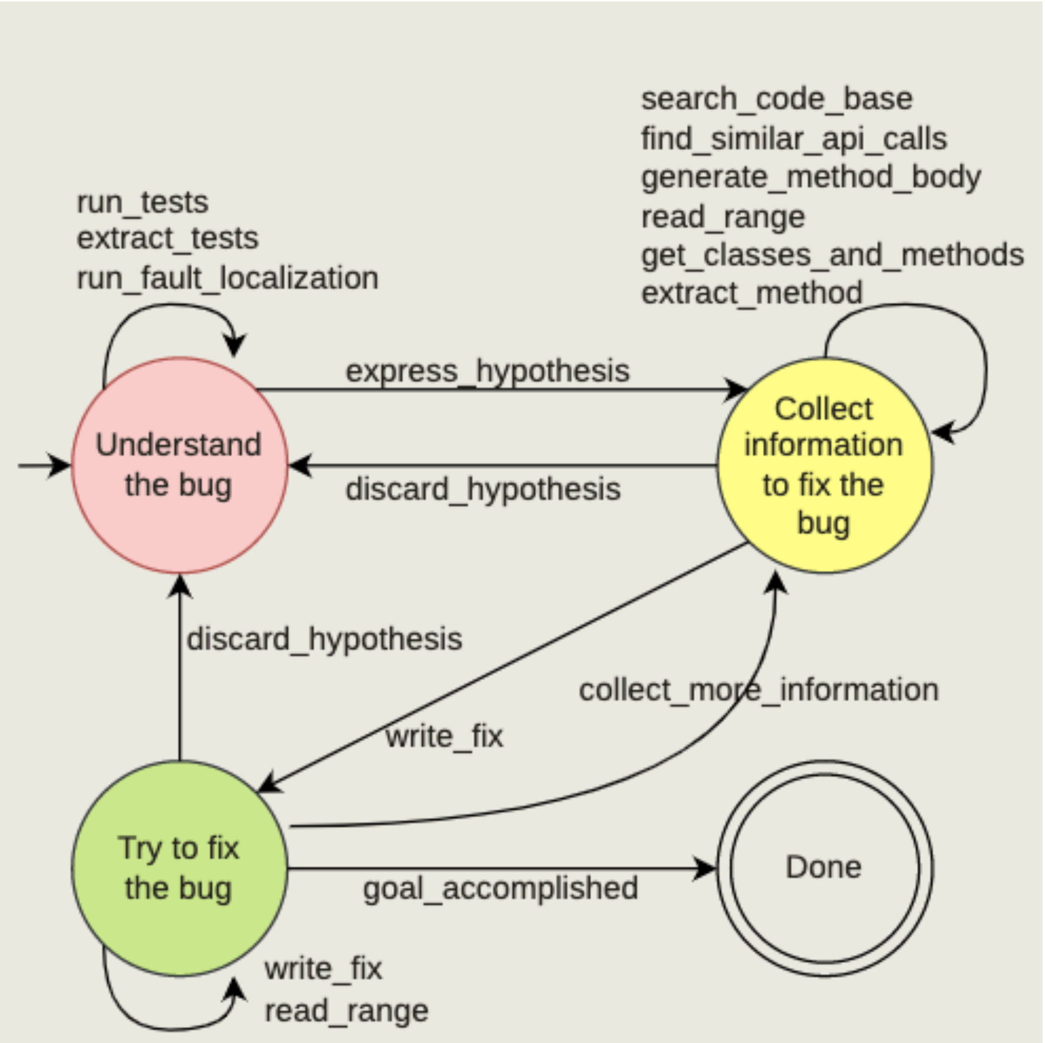
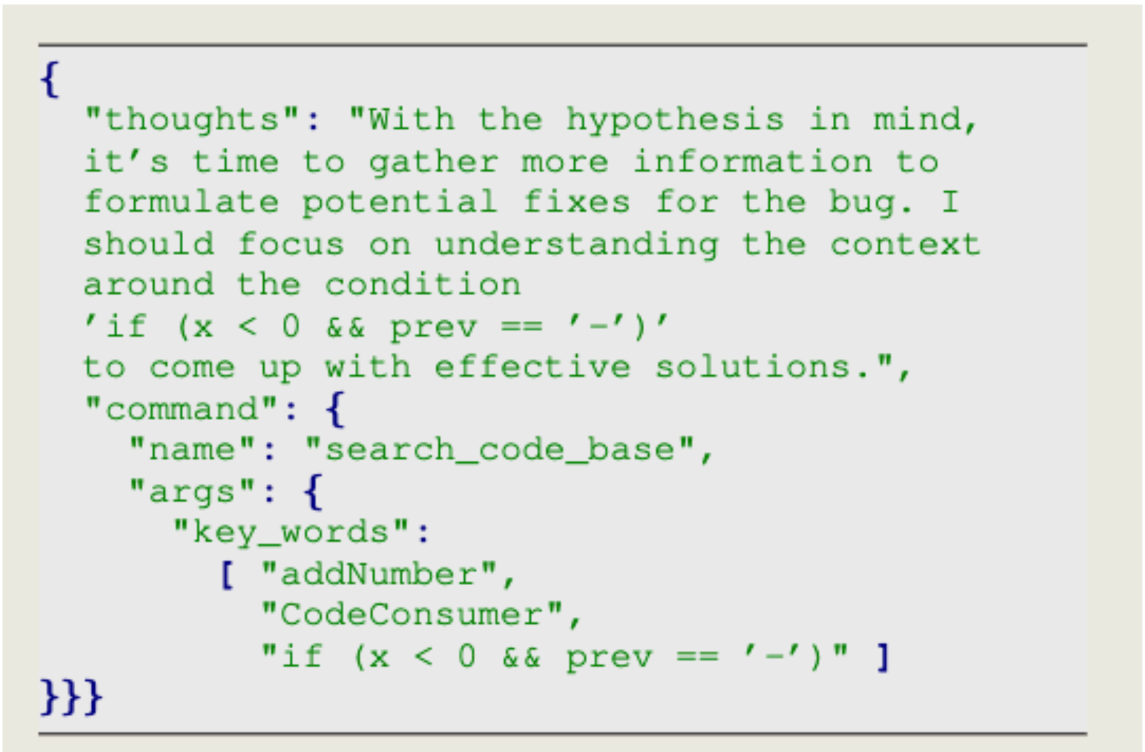


Fig. 2: State machine to guide selection of tools.

    1. Setup a finite state machine
    2. Limit available tools for each state

    3. Three options:
        1. Understand the bug: give a assumption about the bug and the reason
        2. Collect information: search similar fix code/read relevant code
        3. Fix: attempt to fix the bug based on collected information or back to the previous states
5. Toolkits
6. Gathered information: infos about the bug and the codebase. Maintain a prompt section to list the information gathered by various tools, allowing LLM to recall the memory
7. Specs of output format
8. Last execute commands/result: to notice how many cycles left



```
{
  "thoughts": "With the hypothesis in mind,
  it’s time to gather more information to
  formulate potential fixes for the bug. I
  should focus on understanding the context
  around the condition
  ’if (x < 0 && prev == ’-’)’
  to come up with effective solutions.",
  "command": {
    "name": "search_code_base",
    "args": {
      "key_words":
        [ "addNumber",
          "CodeConsumer",
          "if (x < 0 && prev == ’-’)" ]
}}}
```

Fig. 4: Example of a response of the repair agent.

Tools

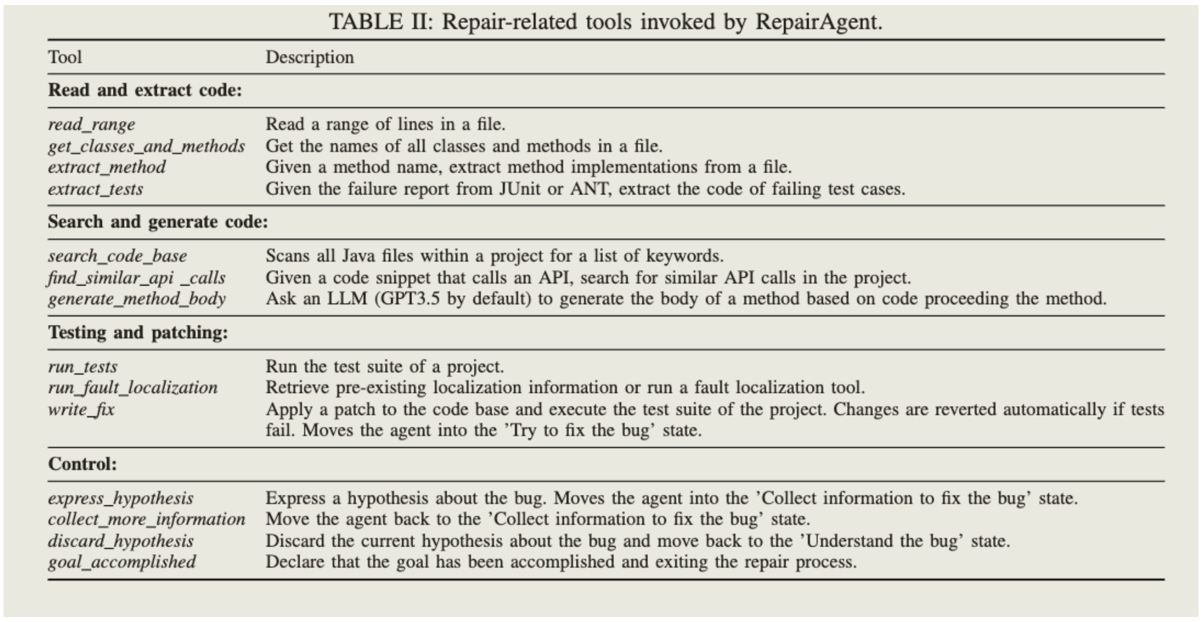


TABLE II: Repair-related tools invoked by RepairAgent.

Tool	Description
Read and extract code:	
read_range	Read a range of lines in a file.
get_classes_and_methods	Get the names of all classes and methods in a file.
extract_method	Given a method name, extract method implementations from a file.
extract_tests	Given the failure report from JUnit or ANT, extract the code of failing test cases.
Search and generate code:	
search_code_base	Scans all Java files within a project for a list of keywords.
find_similar_api _calls	Given a code snippet that calls an API, search for similar API calls in the project.
generate_method_body	Ask an LLM (GPT3.5 by default) to generate the body of a method based on code proceeding the method.
Testing and patching:	
run_tests	Run the test suite of a project.
run_fault_localization	Retrieve pre-existing localization information or run a fault localization tool.
write_fix	Apply a patch to the code base and execute the test suite of the project. Changes are reverted automatically if tests fail. Moves the agent into the 'Try to fix the bug' state.
Control:	
express_hypothesis	Express a hypothesis about the bug. Moves the agent into the 'Collect information to fix the bug' state.
collect_more_information	Move the agent back to the 'Collect information to fix the bug' state.
discard_hypothesis	Discard the current hypothesis about the bug and move back to the 'Understand the bug' state.
goal_accomplished	Declare that the goal has been accomplished and exiting the repair process.

Middleware

1. Understand and refine LLM output：
    1. 修正工具参数的格式偏移（文件路径/路径）（使用模糊字符串匹配）
    2. LLM输出中提供的工具参数映射到工具的参数
    3. 处理替换无效参数
    4. 监视并解决重复调用相同参数的问题
2. Calling the tool:
    1. 调用相关工具
    2. 工具在隔离环境中执行
3. Update prompt
    1. 收集并解析工具的输出
    2. 更新prompt的动态部分

Implementation

- Py3.10
- Docker
- GPT 3.5-0125
- ANTLR： Java交互

Evaluation

Setup

- Dataset：Defects4J: 17个Java项目中835个bug
- Baselines：Chatrrepair（Keep the conversation going...），ITER，SELF-APR 这些方法不使用agnet
- Metrics：
    - plausible：pass all test cases
    - correct: 和开发者给出的修复 语义一致或语法一致

Result

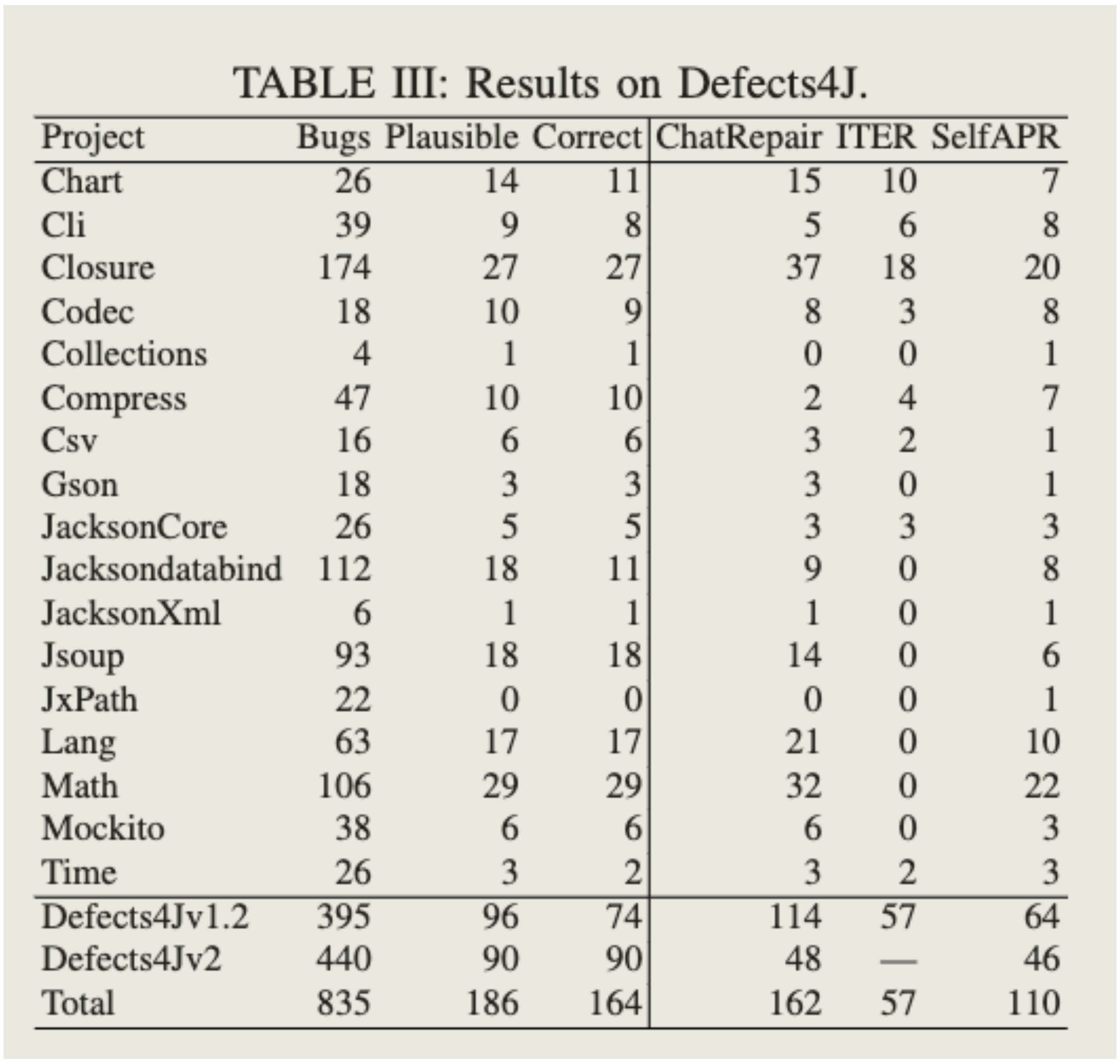


TABLE III: Results on Defects4J.

Project	Bugs	Plausible	Correct	ChatRepair	ITER	SelfAPR
Chart	26	14	11	15	10	7
Cli	39	9	8	5	6	8
Closure	174	27	27	37	18	20
Codec	18	10	9	8	3	8
Collections	4	1	1	0	0	1
Compress	47	10	10	2	4	7
Csv	16	6	6	3	2	1
Gson	18	3	3	3	0	1
JacksonCore	26	5	5	3	3	3
Jacksondatabind	112	18	11	9	0	8
JacksonXml	6	1	1	1	0	1
Jsoup	93	18	18	14	0	6
JxPath	22	0	0	0	0	1
Lang	63	17	17	21	0	10
Math	106	29	29	32	0	22
Mockito	38	6	6	6	0	3
Time	26	3	2	3	2	3
Defects4Jv1.2	395	96	74	114	57	64
Defects4Jv2	440	90	90	48	—	46
Total	835	186	164	162	57	110

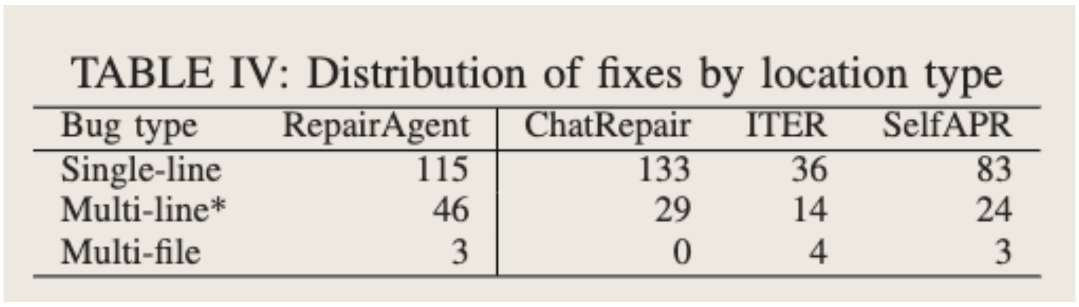
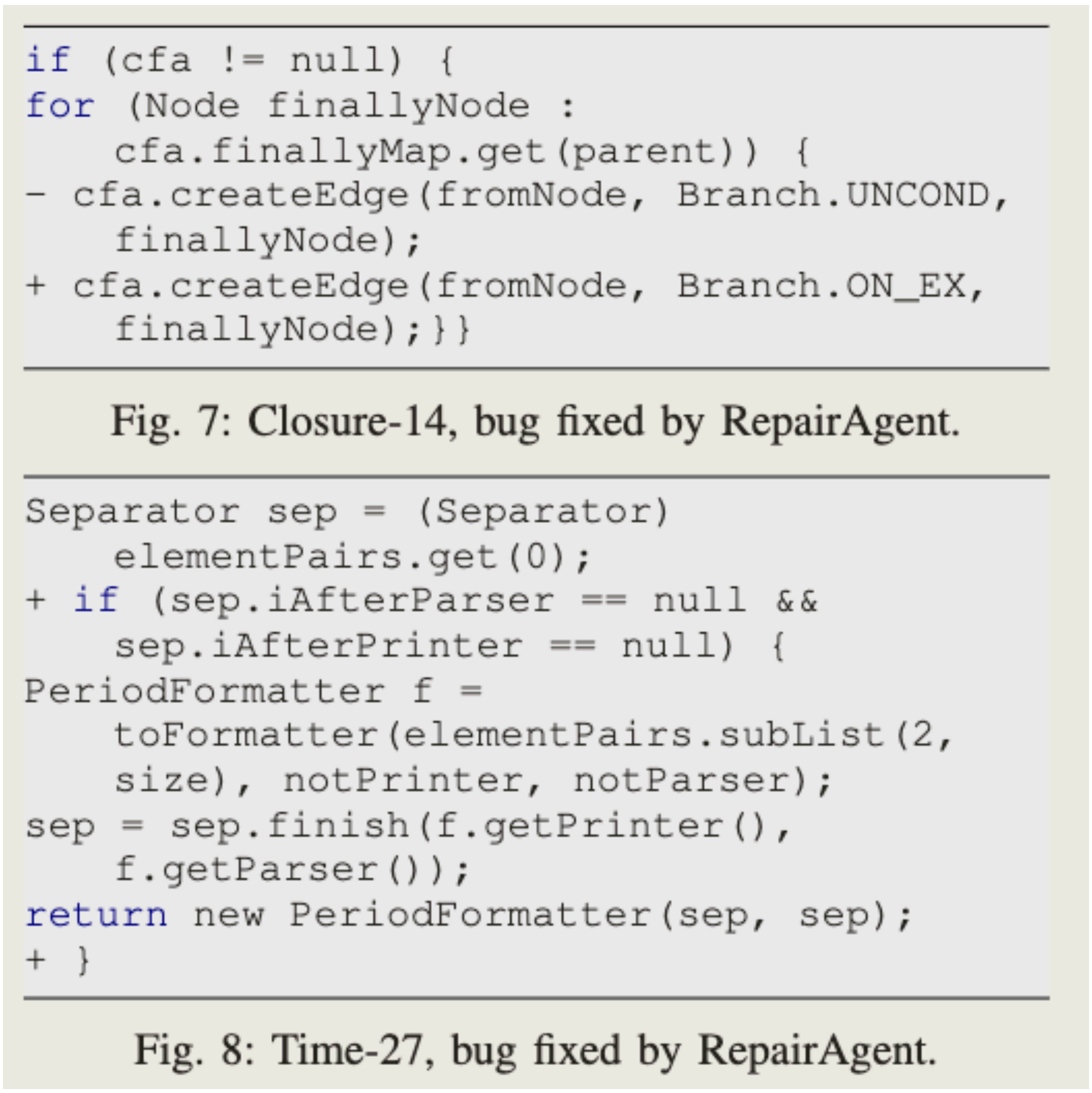


TABLE IV: Distribution of fixes by location type

Bug type	RepairAgent	ChatRepair	ITER	SelfAPR
Single-line	115	133	36	83
Multi-line*	46	29	14	24
Multi-file	3	0	4	3

- 针对多行复杂bug修复完成度较好
- 归因于RepairAgent**自主检索**合适修复成分的能力，以及Agent可以对**任意数量**的行和文件执行编辑
- 不同工作的方法可以是“互补”的
- 修复了数个其他工作没有修复的bug



```
if (cfa != null) {
for (Node finallyNode :
    cfa.finallyMap.get(parent)) {
- cfa.createEdge(fromNode, Branch.UNCOND,
    finallyNode);
+ cfa.createEdge(fromNode, Branch.ON_EX,
    finallyNode);}}
```

Fig. 7: Closure-14, bug fixed by RepairAgent.

```
Separator sep = (Separator)
    elementPairs.get(0);
+ if (sep.iAfterParser == null &&
    sep.iAfterPrinter == null) {
PeriodFormatter f =
    toFormatter(elementPairs.subList(2,
    size), notPrinter, notParser);
sep = sep.finish(f.getPrinter(),
    f.getParser());
return new PeriodFormatter(sep, sep);
+ }
```

Fig. 8: Time-27, bug fixed by RepairAgent.

- Example1:在其他文件中找到类似的调用并修复
- Example2:工具生成的缺失的if语句

Costs

- 中位数0.14USD每个漏洞
- 修复成功的花费明显少于未成功

Usage of tools

输出格式修正

- 2.3%的输出完全无法解析为JSON
- 9.9%的输出JSON中域名不正确

工具命令修正

- 1.4%不存在的命令
- 0.7%命令错误但被middleware映射修正
- 1.9%参数错误
- 8%可被修复的参数错误

平均每个bug修复需要35次工具调用

Discussion

- Data leakage：Chatrepair也有相同的风险
- Test case：数据集中至少包含一个测试用例，现实中可能没有，工作可能强依赖测试用例
- Fault location：问题定位不准确可能导致修复失败
- LLM的不稳定性导致了复现的难度：提供了交互日志供查询
- 工具扩展可能：更多更好的工具可能有助于增强性能

Notes

Pros

1. 比较完善的prompt示例
2. 引入了外部工具协助分析代码
3. 代码生成使用了单独的action而不是包含在决策Agent中，增强了专注性
4. middleware设计可以有效保证工具使用的稳定性和输出格式的稳定性
5. 更强的检索能力可以更好获取上下文
6. 需要格外关注输出格式和工具命令的格式偏移的问题 人工参与可能降低APR的效率，包括文中列出的问题类别
7. 可以参考实验设计，related work可以继续深挖

Cons

1. 一种伪multi agent：有限状态机
2. memory和context是通过Gathered Information传递的,没有经过准确设计
3. 预先确定了有bug，FP的问题没有注意到
4. 只针对Java的单个数据集做了测试
5. 没有测试多种不同底座模型下的效能，也没有测试原生LLM下的修复性能作为对比
6. 缺少消融实验评估每个组件的功能
7. 评估效果不突出（100+/800+）

计划

- 看一下具体实现，prompt和middleware有没有可以参考的地方
- **DeepCode AI Fix: Fixing Security Vulnerabilities with Large Language Models 今年2月**