Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

大模型支持的
程序崩溃故障定位方法

硕士研究生 王怡男

2025年06月15日

问题回溯

- 问题回溯
  - 缺少对方法局限性的讲解
  - 两个方法的关联性讲解不足
  - 图表的坐标解释不足
- 相关内容
  - 2025.01.12 王怡男《程序崩溃的故障定位方法》
  - 2024.06.30 赵智洋《程序崩溃的根本原因分析技术》

内容提要

- 预期收获
- 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
  - AutoFL
  - FlexFL
- 特点总结与工作展望
- 参考文献

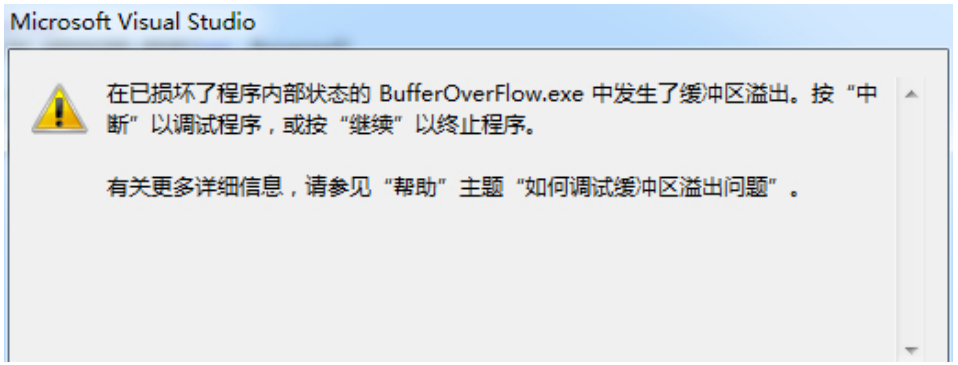
预期收获

- 预期收获
  - 1. 了解LLM-based故障定位方法相较传统方法的优势
  - 2. 了解LLM-based故障定位方法缓解LLM输入长度受限问题的解决方案
  - 3. 理解函数调用在大型代码库情境下的重要作用
  - 4. 了解现有LLM-based故障定位方法的局限及未来发展方向

题目内涵解析 基本概念

- 程序崩溃（Program Crash）
  - 程序在运行时由于未预料到的异常情况而终止
  - 崩溃发生后，通常有两种描述崩溃的方式
    - 崩溃报告
      - 开发者或用户对崩溃的自然语言描述
    - 崩溃触发测试用例
      - 一个能稳定触发崩溃的测试用例，通常配有断言和失败栈信息

```
Traceback (most recent call last):
  File "D:\test.py", line 4, in <module>
    result = numerator / denominator
ZeroDivisionError: division by zero
```



- 故障定位（Fault Localization）
  - 在程序崩溃后，分析崩溃原因、确定导致崩溃的具体代码或程序状态

Bug Report	*Title:* #90 DateTimeZone.getOffsetFromLocal error during DST transition *Description:* This may be a failure of my understanding, but the comments in DateTimeZone.getOffsetFromLocal lead me to believe that if an ambiguous local time is given, the offset corresponding to the later of the two possible UTC instants will be returned - i.e. the greater offset. . . . . (More details in [49])

Trigger Test	public void test_DateTime_constructor_Moscow_Autumn() {
	DateTime dt = new DateTime(2007, 10, 28, 2, 30, ZONE_MOSCOW);
	assertEquals("2007-10-28T02:30:00.000+04:00", dt.toString());
	The last line shown above failed with the following stack trace.
	junit.framework.ComparisonFailure: expected:<...10-28T02:30:00.000+0[4]:00> but was:<...10-28T02:30:00.000+0[3]:00>
	~~at junit.framework.Assert.assertEquals(Assert.java:100)~~
	at org.joda.time.TestDateTimeZoneCutover.test_DateTime_constructor_Moscow_Autumn(TestDateTimeZoneCutover.java:922)

题目内涵解析 Why LLM?

- 从传统测试方法到LLM-based方法
  - 传统方法存在较大物理开销
    - 传统方法使用模糊测试等技术扩展输入，对硬件要求较高
    - 使用LLM的API调用或本地部署，可以大幅缓解硬件层面造成的限制

All of our experiments are conducted within a cloud VM with 32 cores (based on Intel Xeon Silver 4114, 2.20 GHz) and 224 GiB RAM. We use the Ubuntu 18.04 operating system.

某传统方法的巨大硬件限制

  - 传统方法对输入信息的依赖大
    - SBFL、MBFL等方法依赖完整测试套件、覆盖信息或崩溃报告，自动化程度低
    - 使用LLM有希望仅从少数崩溃触发测试中启动自动调试流程
  - 传统方法结果不可解释
    - 大多数传统方法仅输出一组可疑代码位置及数值打分，无法告诉开发者为何出错
    - LLM具备推理链能力，能从崩溃信息出发分析逻辑因果链，生成自然语言解释

研究背景与意义

- 研究背景
  - LLM在多个软件工程的相关领域得到大量应用，展现出处理代码问题的强大能力
  - 开发者越来越重视结果的解释性，而传统FL工具难以解释可疑位置为何可疑
  - 传统FL方法可扩展性差，难以大规模应用
- 研究意义
  - LLM支持下的故障定位技术，为传统FL工具的实用性瓶颈提供有力突破
    - 能生成自然语言解释
    - 缓解输入信息依赖
    - 具备可扩展性
  - 从“打分排序”走向“智能对话式定位”的关键转型

研究历史与现状 大模型支持的程序崩溃故障定位

Wu等人结合提示词工程与上下文裁剪策略，探索了大模型对于故障定位的支持作用，利用大模型对缺陷代码的上下文感知能力，在语句级别有效提升故障定位的准确率，但在方法、类级别场景下性能明显下降

AgentFL：借助多智能体协同机制模拟开发者调试行为，构建理解-导航-确认三阶段流程，结合最佳行为追踪、文档引导搜索、多轮对话策略，有效突破大模型在长上下文处理能力不足的瓶颈，在Defects4J-V1.2.0上实现Top-1准确定位157个缺陷，体现出与其他方法的强互补性与实用性

AutoFL：面向大模型缺乏解释能力与上下文限制问题，提出结合自然语言解释生成与导航函数调用的故障定位方法，实现跨大规模代码库的高效定位。其在798个真实缺陷上相较基线精度提升达233.3%，并在用户访谈中获得积极反馈，具有实际可用性

FlexFL：面向现有LLM-based方法依赖触发测试与私有模型的局限，先基于多源缺陷线索缩小搜索空间，再借助开源LLM精细定位，成功融合传统方法与LLM优势。实验表明，其使用轻量级开源LLM即可超过AutoFL与AgentFL，展示出卓越的通用性与灵活性

知识基础 传统的故障定位方法原理

- 基于频谱的故障定位方法（SBFL）
  - 通过对比通过用例与崩溃触发用例中各语句/方法的执行频率，计算怀疑度
  - 核心思想：如果某个语句/方法主要出现在崩溃触发用例中，而较少或不出现在通过用例中，那么它可能就是崩溃根因的位置
- 基于变异的故障定位方法（MBFL）
  - 通过对程序进行细微修改得到多个变异体
  - 比较变异前后的测试行为，统计每个变异点的修复能力
  - 核心思想：如果一个变异能让崩溃的测试变成通过（或反过来），它修改的位置可能与崩溃根因有关
- 基于信息检索的故障定位方法（IBFL）
  - 将崩溃报告视为“查询”，源代码文件视为“文档”，基于文本相似度计算每个方法与崩溃报告之间的相关性

A Quantitative and Qualitative Evaluation of LLM-Based Explainable Fault Localization

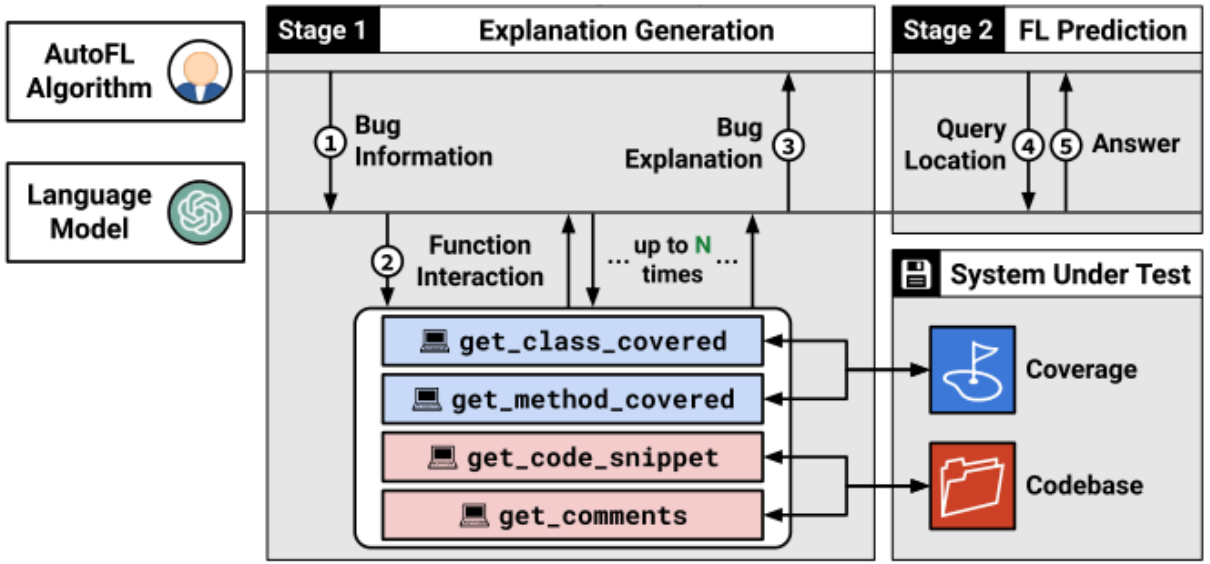
算法原理 TIPO

T	目标	实现程序崩溃的故障定位，并给出解释说明
I	输入	1个崩溃测试（包括代码片段和错误栈信息） 程序源代码及覆盖信息 多个预设的调试函数
P	处理	1. LLM多轮交互调用调试函数，生成自然语言的根因解释 2. 基于解释，LLM推理输出可疑方法名 3. 多次重复1和2，结果合并、排序，输出最终定位结果
O	输出	1组按置信度排序的方法级根因候选列表、每个候选方法对应的自然语言解释

P	问题	传统故障定位方法依赖大量覆盖信息、变异数据或多个测试
C	条件	需要程序源代码；依赖GPT，开源模型无法使用该方法
D	难点	LLM输入长度有限制；不完整源代码无法提供足够信息
L	水平	Proceedings of the ACM on Software Engineering, 2024

算法原理 创新说明

- AutoFL
  - 首次系统化地将LLM用于方法级别的故障定位
  - 仅需1个崩溃测试，相比传统方法降低了前处理开销
  - 提出利用调试函数引导模型与源代码进行交互、推理、逐步解释，平衡LLM输入长度与获取信息量两个限制
  - 可自动生成自然语言的根因解释，有助于开发者理解bug背后的因果关系



算法原理 根因解释生成

- 初始化提示词构建
  - 定义任务，进入状态：你是一个调试助手，你会看到一个失败的测试……
  - 自动生成一个提示词作为LLM的初始输入
  - 提示词内容
    - 失败测试名称
    - 测试代码片段
    - 错误消息与栈回溯
    - 一个初始建议函数调用
  - 提示词设计细节
    - 只保留崩溃前语句
    - 标记崩溃点
    - 删除已通过断言

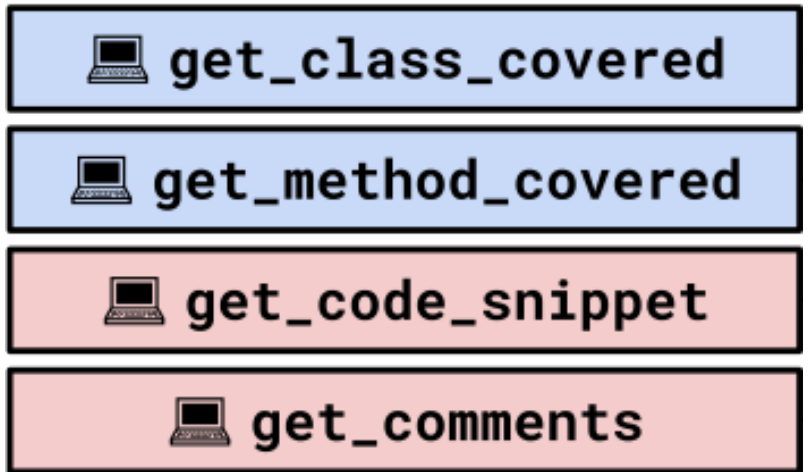
````
The test `...EqualsBuilderTest::testBigDecimal()` failed. The test looks like:

```java
381 : public void testBigDecimal() {
382 : BigDecimal o1 = new BigDecimal("2.0");
383 : BigDecimal o2 = new BigDecimal("2.00");
385 : assertTrue(new EqualsBuilder().append(o1, o2).isEquals()); // error occurred here
386 : }
```

It failed with the following error message and call stack:
```
junit.framework.AssertionFailedError
       at ...EqualsBuilderTest::testBigDecimal(EqualsBuilderTest.java:385)
```
Start by calling the `get_failing_tests_covered_classes` function.
````

算法原理 根因解释生成

- 函数交互
  - LLM可以调用四个预先定义的函数，用于获取调试信息
  - 函数调用由LLM自主发起，AutoFL执行函数并将结果反馈给LLM
  - LLM可获取信息
    - 失败测试所覆盖的类或方法
    - 某方法的源代码或注释
  - 方法名不完整时，向LLM给出提示
  - 最多调用函数N次



- 自然语言解释生成
  - LLM调用N次函数或自行结束推理后，基于当前对测试和代码的理解，生成一段自然语言文本

算法原理 故障定位预测

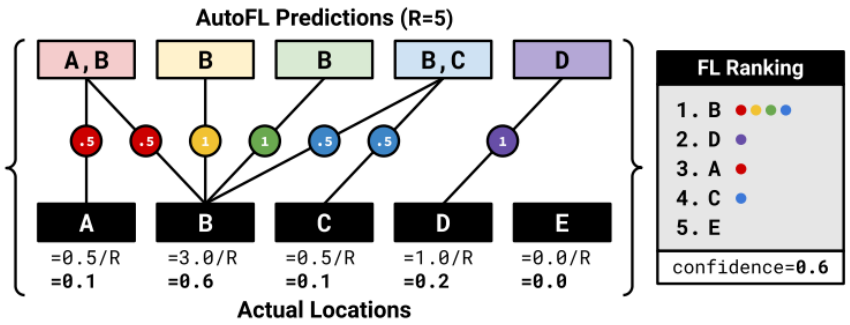
- 可疑方法列表生成
  - 在获得故障根因的自然语言解释后，引导LLM输出根因所在方法
  - 提示词构建
    - 明确告诉LLM只输出方法名
    - 不允许多余解释
    - 一行一个，形成列表

```
Based on the available information, provide the signatures of the most likely culprit methods for
↪ the bug. Your answer will be processed automatically, so make sure to only answer with the
↪  accurate signatures of the most likely culprit (in `ClassName.MethodName(ArgType1,
↪ ArgType2, ...)` format), without commentary (one per line).
```

- 故障定位结果整合
  - 对LLM在R轮运行中输出的可疑方法进行融合、赋分、排序、置信度评估
  - 整合步骤
    - 为每一轮预测赋分
    - 置信度计算

$$\text{score}(m) = \frac{1}{R}\sum_{k=1}^{R}\left(\frac{1}{|r_k|}\cdot[m \in r_k]\right)$$

$$\text{confidence} = \max_{m \in M} score(m)$$



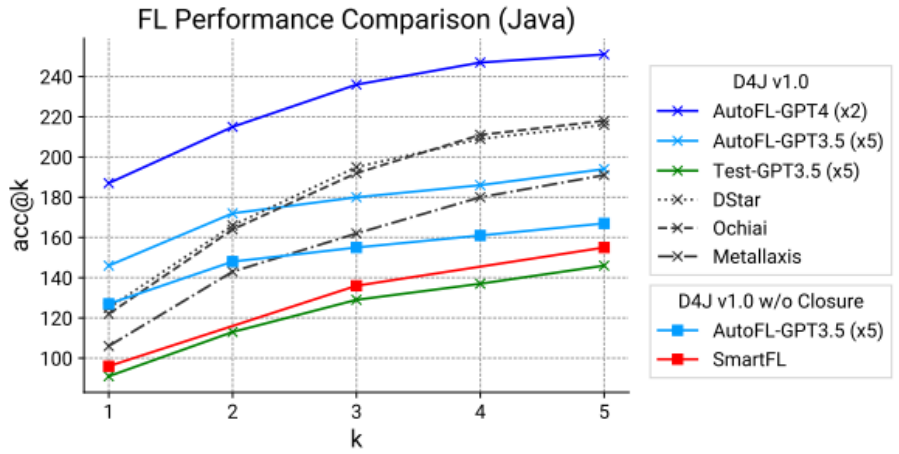
实验设计 数据资源

- 数据资源
  - Defects4J v1.0，共353个bug（Java）；改进版 BugsInPy，共445个bug（Python）
  - 共计21个开源项目的798个可复现bug
- 评估标准
  - 实验设置重复运行次数R=5，最大函数交互次数N=10，模型选用GPT-3.5和GPT-4
  - *acc@k*：正确定位根因所在方法的bug数
    - 对于每个bug，若故障定位结果中前k个候选方法中包含真实缺陷方法，则该值加1，反之不增加
- 对比方法
  - 基于变异的方法：Metallaxis
  - 基于语义的方法：SmartFL
  - 基于频谱的方法：Ochiai、Dstar
  - 消融实验：Test-LLM（AutoFL无函数交互的版本）

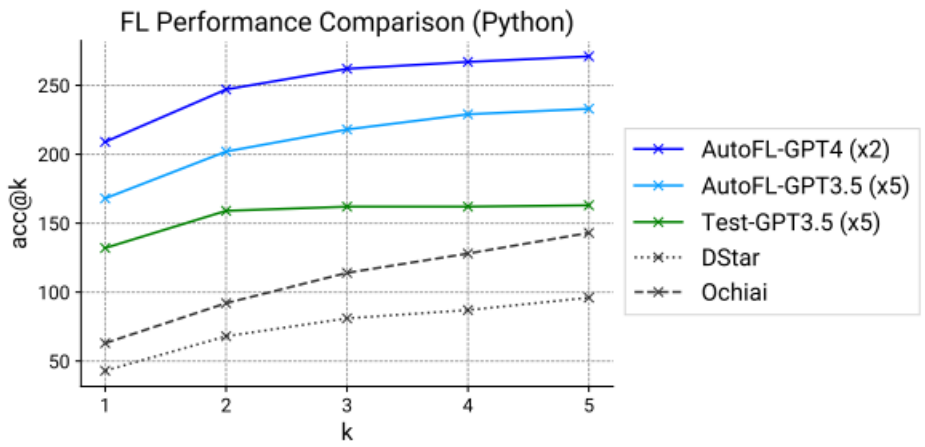
Project	#Bugs	#FTs	#PMs	Project	#Bugs	#FTs	#PMs	Project	#Bugs	#FTs	#PMs
PySnooper	2	1.00	2.00	luigi	28	1.29	1.65	tornado	14	1.14	1.15
ansible	11	1.27	1.45	matplotlib	27	1.04	1.58	youtube-dl	42	1.00	1.39
black	22	1.05	2.43	pandas	165	1.25	1.53	Chart†	26	3.54	1.52
cookiecutter	4	1.25	2.00	sanic	3	1.00	1.00	Closure†	131	2.63	1.25
fastapi	16	1.94	2.42	scrapy	38	1.45	1.17	Lang†	64	1.92	1.38
httpie	5	1.00	1.25	spacy	2	1.00	1.00	Math†	106	1.66	1.32
keras	36	1.17	2.37	thefuck	30	1.27	1.24	Time†	26	2.85	1.56

实验设计 实验结果

- 性能对比结果
  - AutoFL-GPT3.5在Java数据集的*acc@1*值优于经典方法，在Python数据集的*acc@1-5*均优于经典方法
  - AutoFL-GPT3.5在Java数据集的*acc@3-5*的性能弱于基于频谱的方法，而AutoFL-GPT4的性能优于所有对比方法，说明语言模型能力越强，效果越好
  - 函数交互对于性能有极大提升



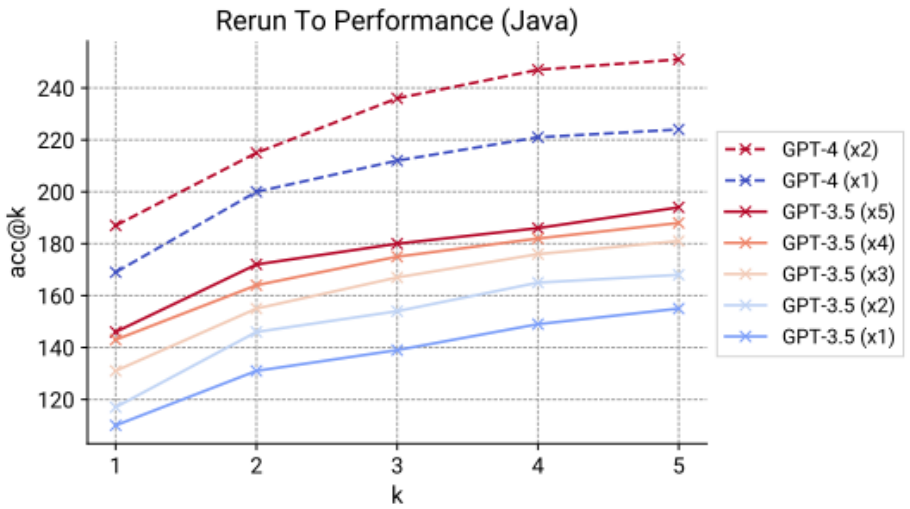
(a) FL evaluation on Defects4J



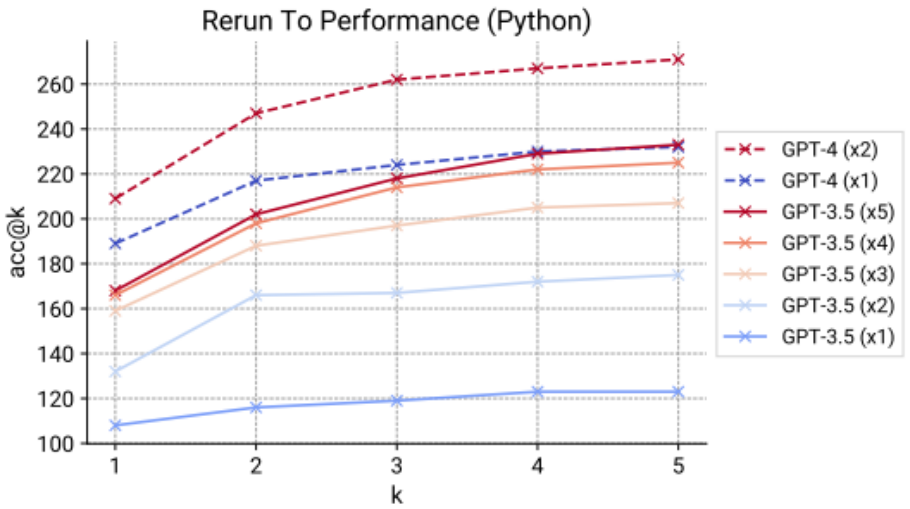
(b) FL evaluation on BugsInPy

实验设计 实验结果

- 性能对比结果
  - 随着重复运行次数R增大，性能提升显著
  - 2轮AutoFL-GPT4甚至优于5轮AutoFL-GPT3.5，进一步说明语言模型能力越强，效果越稳越好



(a) Defects4J



(b) BugsInPy

实验设计 实验结果

- 效率对比结果
  - AutoFL-GPT3.5
    - 单轮耗时为16.4s，5轮合计耗时为87.24s，低于基于频谱的方法最快记录112s
  - AutoFL-GPT4
    - 单轮耗时为152.48s，2轮合计耗时为310.35s
  - AutoFL-GPT3.5运行效率高，可作为轻量级工具

Benchmark	LLM	Prep. (①)	$R = 1$	$R = 2$ (②)	$R = 5$ (②)	Merge (③)	Total (①+②+③)
Defects4J	GPT-3.5	4.87	16.40	-	82.00	0.37	87.24
	GPT-4		152.48	304.96	-	0.52	310.35
BugsInPy	GPT-3.5	23.50	34.71	-	173.55	0.28	197.33
	GPT-4		65.80	131.6	-	0.20	155.38

算法总结

- 算法贡献
  - 提出首个支持自动解释的LLM-based方法级故障定位技术
  - 设计了函数交互式LLM架构，有效缓解了LLM的输入长度限制问题
- 算法局限
  - 极度依赖GPT，不支持开源模型，存在部署与隐私障碍
  - 对非触发式测试支持能力弱
    - 当前只能处理测试用例中显式调用故障方法或其栈信息可见的情况
  - 解释生成质量不稳定

函数调用是如何工作的?

你可以向模型提供一组函数定义（JSON Schema 形式），然后模型会在回答中生成建议调用某个函数（带参数）的结构化响应，例如：

json 复制 编辑

```
{
  "name": "get_weather",
  "arguments": {
    "location": "Beijing",
    "unit": "celsius"
  }
}
```

然后你用外部程序去执行这个函数并返回结果给模型，模型继续对话。

FlexFL: Flexible and Effective Fault Localization with Open-Source Large Language Models

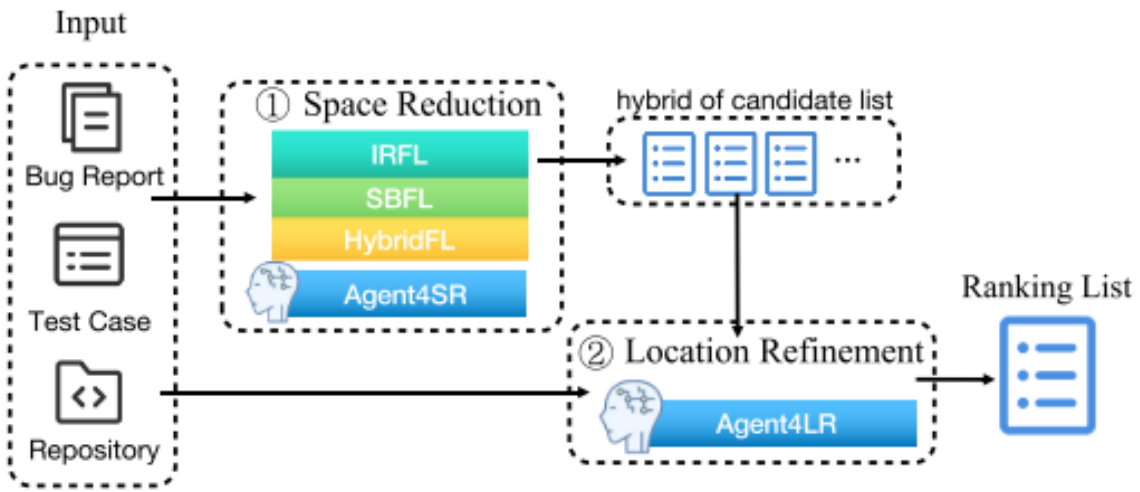
算法原理 TIPO

T	目标	实现程序崩溃的故障定位
I	输入	1份崩溃报告或1个崩溃测试、程序源代码、一些通用函数
P	处理	1. 定义2个智能体，一个用于空间缩减，一个用于位置精炼 2. 空间缩减：结合多种故障定位技术，用智能体1选出m个候选方法 3. 位置精炼：利用智能体2对m个候选方法进行代码推理和验证，重新排序
O	输出	1组按可疑度从高到低的方法级根因候选列表

P	问题	LLM难以处理大规模代码库；开源LLM无函数调用能力
C	条件	需要程序源代码
D	难点	构建函数调用，让开源LLM有效读懂源码
L	水平	IEEE Transactions on Software Engineering, 2025(CCF A)

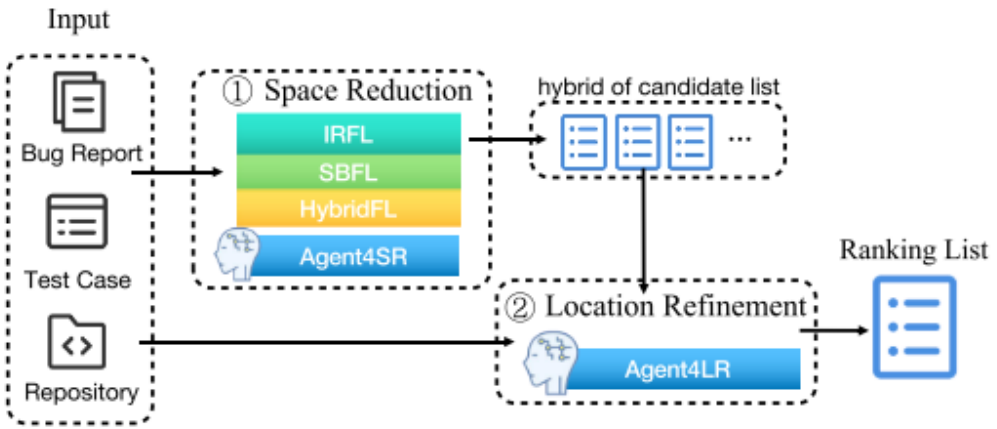
算法原理 创新说明

- **FlexFL**
  - 提出可同时使用崩溃报告和崩溃测试作为输入，且具备应对输入缺失的鲁棒性
  - 融合传统故障定位方法与LLM-based方法，使LLM在输入长度受限情况下提升性能和效率
  - 为开源LLM设计了可解析的函数调用格式，解决了开源模型无法原生调用函数的问题



算法原理 创新说明

- **FlexFL**
  - 设计了Agent4SR和Agent4LR两个智能体，进行二次分析
  - Agent4SR：Agent for Space Reduction
    - 在整个项目代码库中，全局搜索与崩溃相关的函数方法，生成候选列表供下一步精细分析使用
  - Agent4LR：Agent for Location Refinement
    - 在已筛选出的候选方法列表中，进一步精细分析每个方法的代码语义，推理哪个才是真正的根因方法，最终输出排名列表



算法原理 输入设计

- 提示词工程
  - 定义任务，进入状态：你是一个调试助手，你会看到……
  - 自动生成一个提示词作为LLM的初始输入
  - 提示词内容
    - 崩溃相关描述
      - 崩溃报告、崩溃测试二者至少一个
      - 建议入手的方法名或
        前一步发现的可疑方法名

The bug report is as follows:```
<bug report>```
The trigger test is as follows:```
<trigger test>```
The suggested methods are as follows:```
1.org.joda.time.DateTime.DateTime(long, DateTimeZone)……```

- 源代码预处理
  - 提取所有类和方法的路径及全名，供函数调用时使用

算法原理 空间缩减

- 初始推理
  - 提示智能体先输出对崩溃根因的初步思考
- 函数调用
  - 开源LLM无法像GPT一样自带函数调用
  - 设计函数调用协议，通过提示词引导LLM发出统一格式的调用语言
    - 仅包含函数名和参数：FunctionName(Argument)
  - 设置最大交互次数MAX次

Name	Arguments	Description
get_paths	None	Get the paths of the Java software system
get_classes_of_path	path_name	Get the classes in the path of the Java software system
get_methods_of_class	class_name	Get the methods belonging to the class of the Java software system
get_code_snippet_of_method	method_name	Get the code snippet of the Java method
find_class	class_name	Find the class through fuzzy search
find_method	method_name	Find the method through fuzzy search
exit	None	Exit function calling

算法原理 空间缩减

- 方法名修复
  - 缓解因LLM幻觉可能产生的方法名输出模糊的问题
    - 将LLM输出的方法名视为查询，将源码中提取的全名进行‘.’, ‘/’,‘(’分隔符分隔
      - 片段完全包含——直接匹配
      - 计算查询与片段的编辑距离——模糊匹配
- 方法级故障定位技术融合
  - 将LLM-based方法与其他技术融合
    - 若输入中含有崩溃报告，则融合基于信息检索的方法
    - 若输入中含有崩溃测试，则融合基于频谱的方法
  - 将Agent4SR给出的Top-k方法名与其他技术给出的Top-k方法名融合成新的可疑方法名候选集

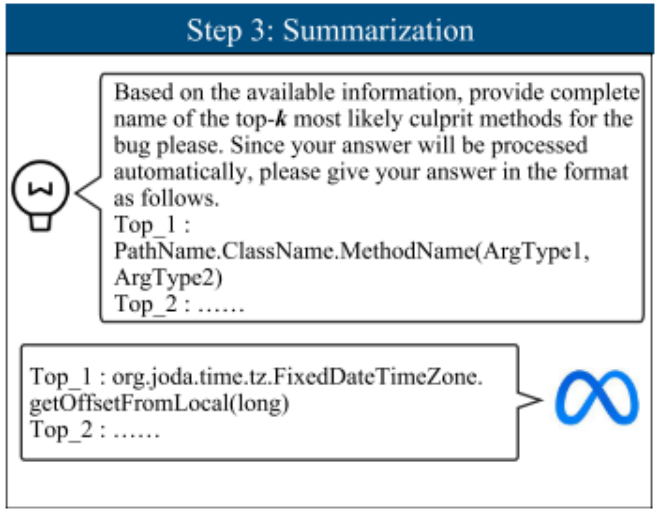
Algorithm 1 Postprocessing Process

Input: The name of query *query*; The names of entities to search *entities*.
Output: The matching names *names*.

```
1:  names = []
2:  query_split = split(query)
3:  for entity in entities do
4:      entity_splits = split(entity)
5:      if query_split all in entity_splits then
6:          names.add(entity)
7:  if length(names) > 0 then return names
8:  for entity in entities do
9:      edit_distance = Levenshtein.distance(query, entity)
10:     edit_distances.add(edit_distance)
11:     if edit_distance < 5 then
12:         names.add(entity)
13: if length(names) > 0 then return names
14: return names[:5]
```

算法原理 定位细化

- 语义理解与因果推理
  - 输入为上一步给出的可疑方法名候选集
  - Agent4LR不再做代码探索，只专注于可疑方法的代码语义理解
    - 仅能调用get_code_snippet_of_method或exit
  - 结合崩溃报告、崩溃测试、可疑方法语义，给出最终的Top-k可疑方法



Name	Arguments	Description
get_paths	None	Get the paths of the Java software system
get_classes_of_path	path_name	Get the classes in the path of the Java software system
get_methods_of_class	class_name	Get the methods belonging to the class of the Java software system
get_code_snippet_of_method	method_name	Get the code snippet of the Java method
find_class	class_name	Find the class through fuzzy search
find_method	method_name	Find the method through fuzzy search
exit	None	Exit function calling

实验设计 数据资源

- 数据资源
  - Defects4J v2.0，共835个bug，其中835个都带有崩溃触发测试用例，814个有开发者编写的崩溃报告
- 评估标准
  - *Top-N*：Top-N中是否包含至少一个故障方法
  - *MAP*：多个故障方法在推荐列表中的平均排序质量
  - *MRR*：第一个定位正确方法的平均排名倒数
- 对比方法
  - 基于变异的方法：Metallaxis，MUSE
  - 基于信息检索的方法：BoostNsift
  - 基于频谱的方法：Ochiai，Dstar
  - 基于大模型的方法：AutoFL，AgentFL

项目名	缺陷数量
Chart	26
Closure	176
Lang	65
Math	106
Time	27
Mockito	38
Collections	4
Codec	18
Compress	47
Gson	18
JacksonCore	26
JacksonDatabind	112
Jsoup	93
JxPath	22
Wicket	43

实验设计 实验结果

- 对比实验结果
  - FlexFL在各项评价指标上显著超过所有传统方法
  - 使用重复策略（Repetition）可以进一步提升性能
  - 无特殊标注时，FlexFL使用开源模型Meta-Llama-3-8B-Instruct，在几乎所有*Top-N*指标上优于AutoFL
  - FlexFL框架具有吸收其他方法的能力

FlexFL vs other FL techniques on Defects4J (v2.0.0)

Family	Technique	Top-1	Top-3	Top-5	MAP	MRR
IRFL	BoostN	149	241	280	0.206	0.235
SBFL	Ochiai	167	316	389	0.259	0.297
HybridFL	SBIR	222	377	433	0.309	0.362
LLM-based	FlexFL	**350**	**478**	**529**	**0.439**	**0.501**
	FlexFL +Repetition	**363**	**511**	**558**	**0.463**	**0.526**

FlexFL vs vs other FL techniques on Defects4J (v1.0)

Family	Technique	Top-1	Top-3	Top-5
MBFL	MUSE	73	139	161
	Metallaxis	106	162	191
SBFL	Ochiai	122	192	218
	DStar	125	195	216
LLM-based	AgentFL	144	169	173
	AutoFL-Llama3-8B	105	157	172
	AutoFL-GPT-3.5#	146	180	194
	AutoFL-GPT-3.5-1106 ($22.26)*	151	209	221
	FlexFL-GPT-3.5-1106 ($9.71)*	154	220	240
	FlexFL	164	214	236
	FlexFL + Repetition	**164**	**220**	**245**
	AutoFL-GPT-4	**187**	236	251
	FlexFL + AutoFL-GPT-4	176	**239**	**261**

实验设计 实验结果

- 消融实验结果
  - 输入来源的消融分析
    - 缺少任意一个输入来源都会显著影响性能，最严重的是缺少程序源代码
  - 两智能体设计的消融分析
    - Agent4LR可以增强任意一种传统方法的效果
    - Agent4SR单独使用时也优于传统方法
  - 函数调用交互机制的消融分析
    - 每项设计单独去除，都会造成性能下降
    - 函数交互机制设计如何，决定LLM的上限

Variant	Top-1	Top-3	Top-5	MAP	MRR
w/o trigger test	257	345	387	0.323	0.365
w/o bug report	266	395	442	0.339	0.399
w/o buggy program	45	64	75	0.062	0.066
FlexFL	**350**	**478**	**529**	**0.439**	**0.501**

Variant	Top-1	Top-3	Top-5	MAP	MRR
BoostN	149	241	280	0.206	0.235
BoostN + Agent4LR	255	336	364	0.317	0.356
Ochiai	167	316	389	0.259	0.297
Ochiai + Agent4LR	303	421	475	0.387	0.442
SBIR	222	377	433	0.309	0.362
SBIR + Agent4LR	319	429	473	0.400	0.453
Agent4SR	232	318	348	0.293	0.333
FlexFL w/o Agent4SR	338	467	510	0.422	0.485
FlexFL	**350**	**478**	**529**	**0.439**	**0.501**

Variant	Top-1	Top-3	Top-5	MAP	MRR
w/o reasoning	330	466	512	0.418	0.480
w/o postprocessing	330	462	512	0.417	0.478
w/o focus	332	440	479	0.401	0.464
FlexFL	**350**	**478**	**529**	**0.439**	**0.501**

- 其他实验结果
  - FlexFL在不同开源大模型下的泛化能力
    - 虽然不同LLM有轻微性能差异，但各项指标均保持在高位
    - FlexFL的框架并不依赖某个特定模型，具有良好的LLM迁移适应性
  - FlexFL在真实场景中的泛化能力
    - GHRB数据集的子集：28个bug
      - 崩溃数据形成晚，模型没有“看过”
      - 代码规模更大
    - FlexFL能够有效泛化到实际项目中的大型代码库，适应不同项目结构与缺陷分布，表现出良好的可扩展性与实用性

Base Model	Top-1	Top-3	Top-5	MAP	MRR
Mistral-Nemo-12B	322	464	507	0.411	0.473
Qwen2-7B	329	456	500	0.408	0.474
Llama3-8B	**350**	**478**	**529**	**0.439**	**0.501**

Technique	Top-1	Top-3	Top-5	MAP	MRR
BoostN	3	6	7	0.102	0.158
Ochiai	4	12	15	0.273	0.297
SBIR	12	15	18	0.434	0.505
AutoFL-Llama3-8B	12	15	15	/	/
AutoFL-GPT-3.5	13	16	16	/	/
Agent4SR-Llama3-8B	15	18	20	0.557	0.601
FlexFL-GPT-3.5	16	20	21	0.570	0.644
FlexFL-Llama3-8B	**19**	**22**	**22**	**0.648**	**0.732**

算法总结

- 算法贡献
  - 提出融合LLM与传统故障定位技术的双阶段定位框架
  - 支持灵活输入的提示词编排机制
  - 设计函数调用的工具调度机制
  - 验证了方法在不同开源LLM和大型项目上的有效性
- 算法局限
  - Agent4LR只能在Agent4SR产生的候选方法中定位，如果第一阶段未包含真实故障方法，第二阶段无法弥补
  - 推理能力仍然受LLM基础模型限制
  - 项目规模更大时，可能导致上下文长度超限

特点总结与未来展望

特点总结与未来展望

- 特点总结
  - AutoFL && FlexFL
    - LLM-based
      - 大模型支持下的故障定位方法
    - 函数调用
      - 使用该机制逐步获取源码信息，平衡输入长度限制与获取信息多少
    - 推理解释
      - 对崩溃根因、故障定位具备一定的推理与解释能力
- 未来展望
  - 当前仍缺乏有效机制判断LLM输出的解释是否可靠
  - 函数调用还处于浅层阶段，可进一步扩展深层交互
  - 持续与其他工具集成

参考文献

[1] Kang S, An G, Yoo S. A quantitative and qualitative evaluation of LLM-based explainable fault localization[J]. Proceedings of the ACM on Software Engineering, 2024, 1(FSE): 1424-1446.

[2] Xu C, Liu Z, Ren X, et al. FlexFL: Flexible and Effective Fault Localization with Open-Source Large Language Models[J]. IEEE Transactions on Software Engineering, 2025.

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！