AiDD AI+研发数字峰会
AI+ Development Digital summit
第5届

# 基于大模型的软件缺陷定位与修复

罗 丹 | 杭州逻界科技有限公司

# 科技生态圈峰会 + 深度研习

## ——1000 + 技术团队的共同选择

KEYLINK ing

### K+峰会

**K+峰会 · 敦煌站**
K+ 思考周®研习社
时间：2025.08.29-30

**K+峰会 · 上海站**
K+ 金融专场
时间：2025.10.17-18

**K+峰会 · 香港站**
K+ 思考周®研习社
时间：2025.11.25-26

K+峰会详情

### AiDD峰会

**AiDD峰会 · 上海站**
AI+研发数字峰会
时间：2025.05.17-18

**AiDD峰会 · 北京站**
AI+研发数字峰会
时间：2025.08.08-09

**AiDD峰会 · 深圳站**
AI+研发数字峰会
时间：2025.11.28-29

AiDD峰会详情

# 演讲嘉宾



## 罗丹

杭州逻界科技有限公司 CEO

杭州逻界科技有限公司创始人，长期从事程序分析和编译优化领域的研究，研制多款特定领域的编译优化与缺陷修复工具。

# 目录
# CONTENTS

# PART 01

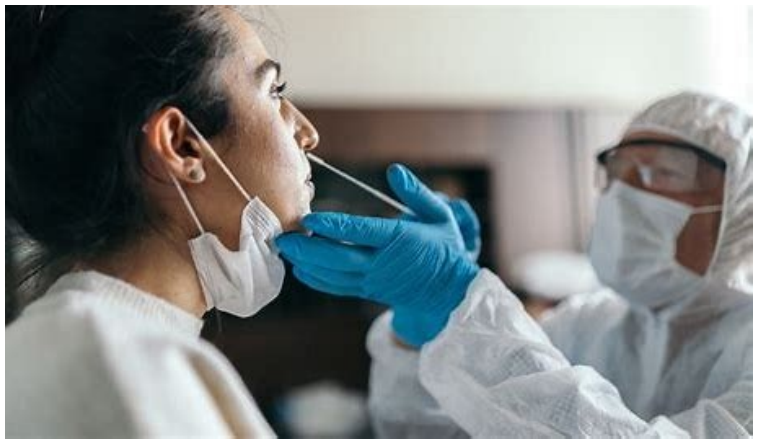
# 背景

# 背景

## 软件缺陷

- 随着软件使用范围的不断扩大和复杂度的不断提高，软件缺陷问题频发，造成巨大损失

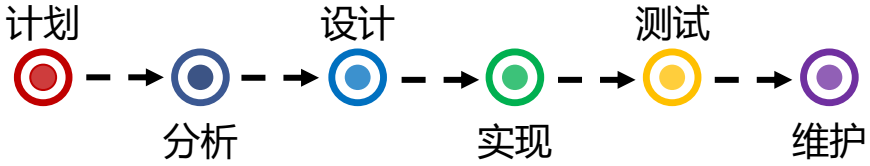

美国大选投票机系统缺陷导致6000张选票登记错误



英国卫生服务系统缺陷导致密接人群未得到提醒

程序缺陷导致Starliner飞船首飞失败

# 背景



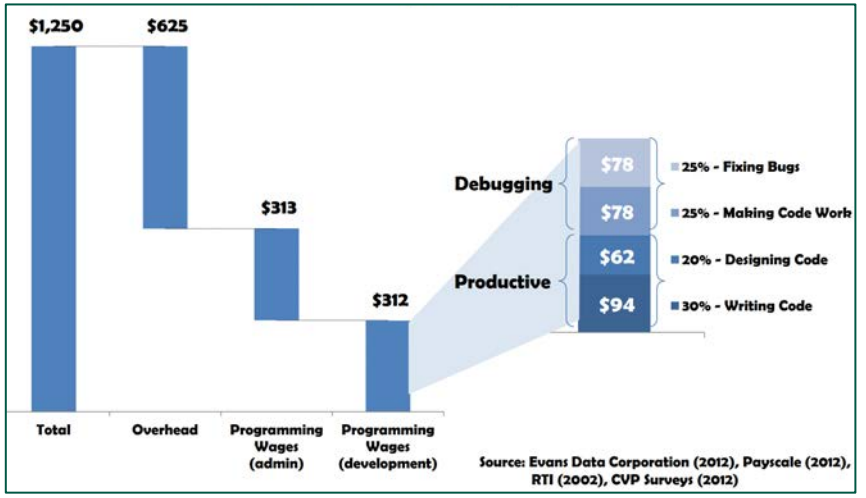

## 软件调试

- 软件调试是解决软件缺陷的主要手段，在软件实现过程中占据约**50%**的成本

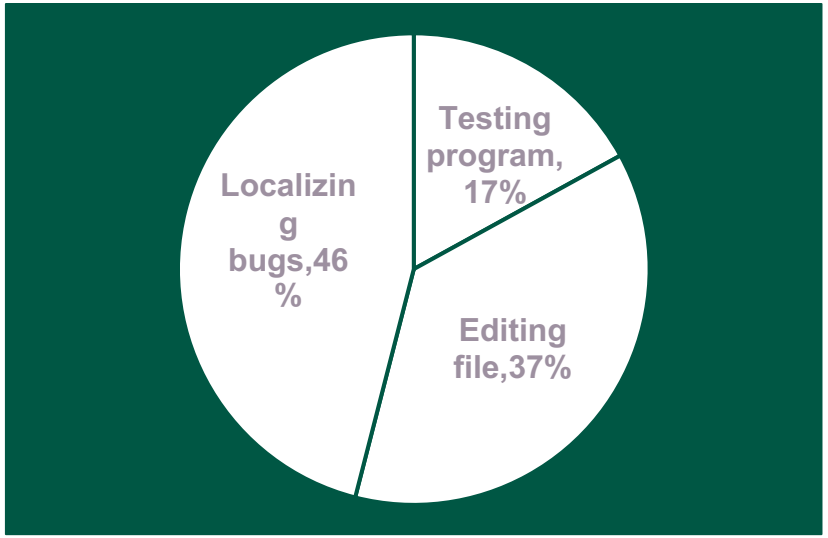
## 软件缺陷定位与修复

- 缺陷定位是软件调试的关键步骤，在软件调试中可占据**46%**的时间

**软件缺陷定位与修复** 对企业软件研发具有重要意义




软件实现过程中软件调试开销占50%[1]




缺陷定位在软件调试时间中占46%[2]

# 背景

## 传统软件缺陷自动定位方法面临瓶颈

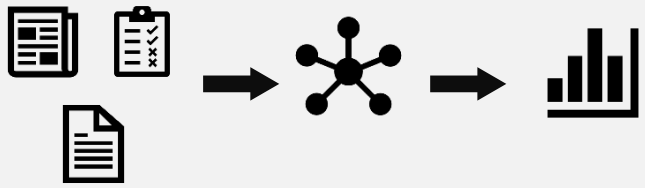
- 对程序语义信息运用不充分

基于程序频谱的缺陷定位方法

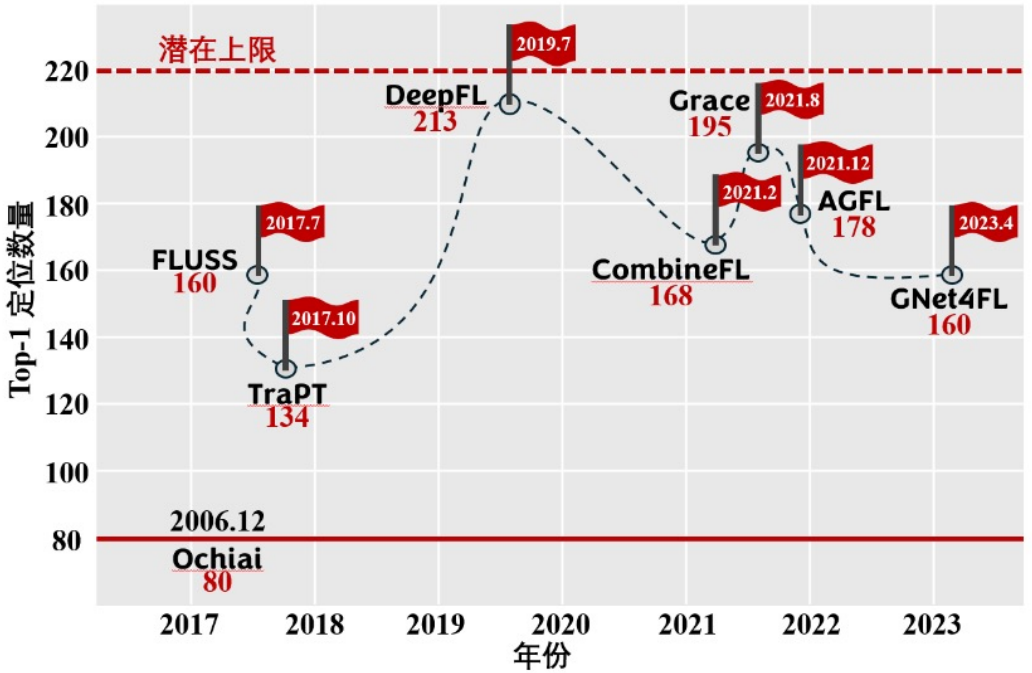
- 缺乏与缺陷相关的专家知识

基于程序依赖关系的缺陷定位方法

- 场景局限于软件项目内部



基于机器学习的缺陷定位方法




传统缺陷定位方法发展情况

如何突破当前缺陷定位方法面临的性能瓶颈?

# 背景

## ◈ 大语言模型提供了新的机遇

“Where Is the Bug and How Is It Fixed?” [1]

缺陷定位需要具备的能力

### 代码理解

```
private boolean canRemoveArgFromCallSites(
        Node function, int argIndex) {
  Definition definition =
getFunctionDefinition(function);
  for (UseSite site :
defFinder.getUseSites(definition)) {
    if (isModifiableCallSite(site)) {
      Node arg =
getArgumentForCallOrNewOrDotCall(
              site, argIndex);
      if (arg != null &&
NodeUtil.mayHaveSideEffects(
              arg, compiler)) {
        return false;
      }
    } else {
      return false;
    }
}
```

这段Java代码的**功能是判断在JavaScript代码分析或修改的上下文中，是否可以从一个函数的所有调用点（call sites）中安全地移除一个指定的参数**。具体的功能解释如下：

1. **获取函数定义**：首先，通过调用`getFunctionDefinition(function)`获取传入的`function`节点所代表的JavaScript函数的定义。
2. **遍历使用点**：接着，使用`defFinder.getUseSites(definition)`获取这个函数定义的所有使用点（即函数被调用的地方）。
3. **检查每个使用点**：对于函数的每个使用点，代码检查这个点是否是可修改的调用点，通过调用`isModifiableCallSite(site)`。如果某个调用点不可修改，则函数直接返回`false`，表示无法移除参数。
4. **分析参数**：如果调用点可修改，代码会使用`getArgumentForCallOrNewOrDotCall(site, argIndex)`获取该点的指定索引（`argIndex`）上的参数。
5. **副作用检查**：最关键的逻辑是检查参数是否可能有副作用，通过调用`NodeUtil.mayHaveSideEffects(arg, compiler)`。副作用是指在执行过程中对状态的改变或与外部系统的可观察交互。如果参数有副作用，移除它是不安全的，方法将返回`false`。
6. **最终决定**：如果在所有调用点中的参数都没有副作用，方法将得出结论，认为安全地移除参数是可行的，并将返回`true`。

大语言模型能够将**代码功能**转化为清晰、详细的**自然语言解释**

# 背景

## ◈ 大语言模型提供了新的机遇

**缺陷定位需要具备的能力**

**代码逻辑推理**

```
public class Calculator {

    public static void main(String[] args) {
        double result = cal(10, 5);
        System.out.println("Final Result: " + result);
    }

    private static double cal(double a, double b) {
        double sum = a + b;
        return mul(sum, 2);
    }

    private static double mul(double a, double b) {
        double product = a * b;
        return div(product, 2);
    }

    private static double div(double a, double b) {
        double quotient = a / b;
        return quotient;
    }
}
```

文心一言

分析这个程序的函数调用关系和计算结果:

1. 主方法main()调用了cal()方法,传入参数10和5
2. cal()方法对传入的参数做加法运算,计算和sum为15
3. cal()方法调用mul()方法,将sum和2作为参数传入
4. mul()方法对参数做乘法运算,得到product为30
5. mul()方法调用div()方法,将product和2作为参数传入
6. div()方法对参数做除法运算,得到quotient为15
7. div()方法将quotient返回给mul()方法
8. mul()方法将返回的quotient返回给cal()方法
9. cal()方法将返回的quotient(也就是15)返回给main()方法
10. main()方法将返回的结果15赋值给result

**所以最终结果result为15。**

**整个调用链是:**
**main() -> cal() -> mul() -> div() -> mul() -> cal() -> main()**

**代码理解**

大语言模型能够分析函数间的**调用关系**，并**“推理”**得出结果

# 背景

## ◈ 大语言模型提供了新的机遇

**缺陷定位需要具备的能力**

**经验知识**

```
public class ArrayOutOfBoundsExample {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        for (int i = 0; i <= arr.length; i++) {
            System.out.println(arr[i]);
        }
        int index = -1;
        System.out.println(arr[index]);
    }
}
```

AI

这段代码存在多个问题：

**数组越界**：在for循环中，你试图访问数组的长度，这会导致数组越界异常（ArrayIndexOutOfBoundsException）。循环应该只迭代到arr.length - 1。
修改后的for循环如下：
for (int i = 0; i < arr.length; i++) { System.out.println(arr[i]); }

**数组下标为负数**：你试图访问数组的一个负数索引，这也会导致数组越界异常。数组的索引不能是负数。
修改后的代码：

int index = -1;
System.out.println(arr[index]); // 这行代码是无效的，并且会导致ArrayIndexOutOfBoundsException。

如果你想在数组中存储负数索引，你需要重新考虑你的数据结构或算法。在Java中，数组的索引是从0开始的，并且不能是负数。

**代码逻辑推理**

**代码理解**

大语言模型具备与缺陷相关的**经验知识**，可以识别程序中的**缺陷模式**

# 背景

## 大语言模型提供了新的机遇

**缺陷定位需要具备的能力**

**经验知识**

大语言模型具备与缺陷相关的**经验知识**

可以识别程序中的**缺陷模式**

**前/后向推理**

大语言模型能够分析函数间的**调用关系**

并通过**推理**得出结果

**代码理解**

大语言模型能够将**代码功能**转化为清晰

详细的**自然语言解释**

数据驱动 → “智能”驱动?

# 背景


> *“软件工程要利用业界经过广泛**实践检验**的工程化方法指导软件系统的开发和运维。”*
>
> *——毛新军 董威*

# PART 02

# 缺陷定位技术

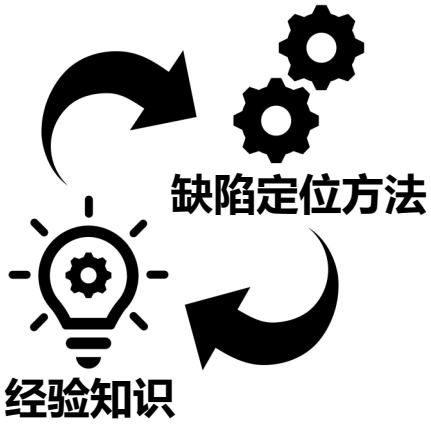
# 缺陷定位技术

## 使用大语言模型进行缺陷定位仍面临严峻挑战

- 上下文管理困难
- 难以表示程序执行信息
- 缺陷定位的复杂思维逻辑
- **经验知识重用**

如何解决这些挑战，实现基于大语言模型的软件缺陷自动定位系统?

如何重用历史定位经验知识?




在程序缺陷定位过程中产生的经验知识有助于指导开发人员定位新的缺陷[7,8]。

**如何对从历史缺陷定位案例中总结出的经验知识进行持久化保存，并在面对新到来的缺陷时合理地重用历史经验**，是一个值得探索的重要问题

# 缺陷定位技术

## 面临挑战

- 上下文管理困难
- 难以表示程序执行信息
- 缺陷定位的复杂思维逻辑
- 经验知识重用

## 提出问题

- 如何令大语言模型关注于与软件缺陷相关的上下文?
- 如何将动态程序执行信息转化为大语言模型易于理解的静态表示形式?
- 如何让大语言模型重用历史缺陷定位案例中的经验知识?

## 核心技术

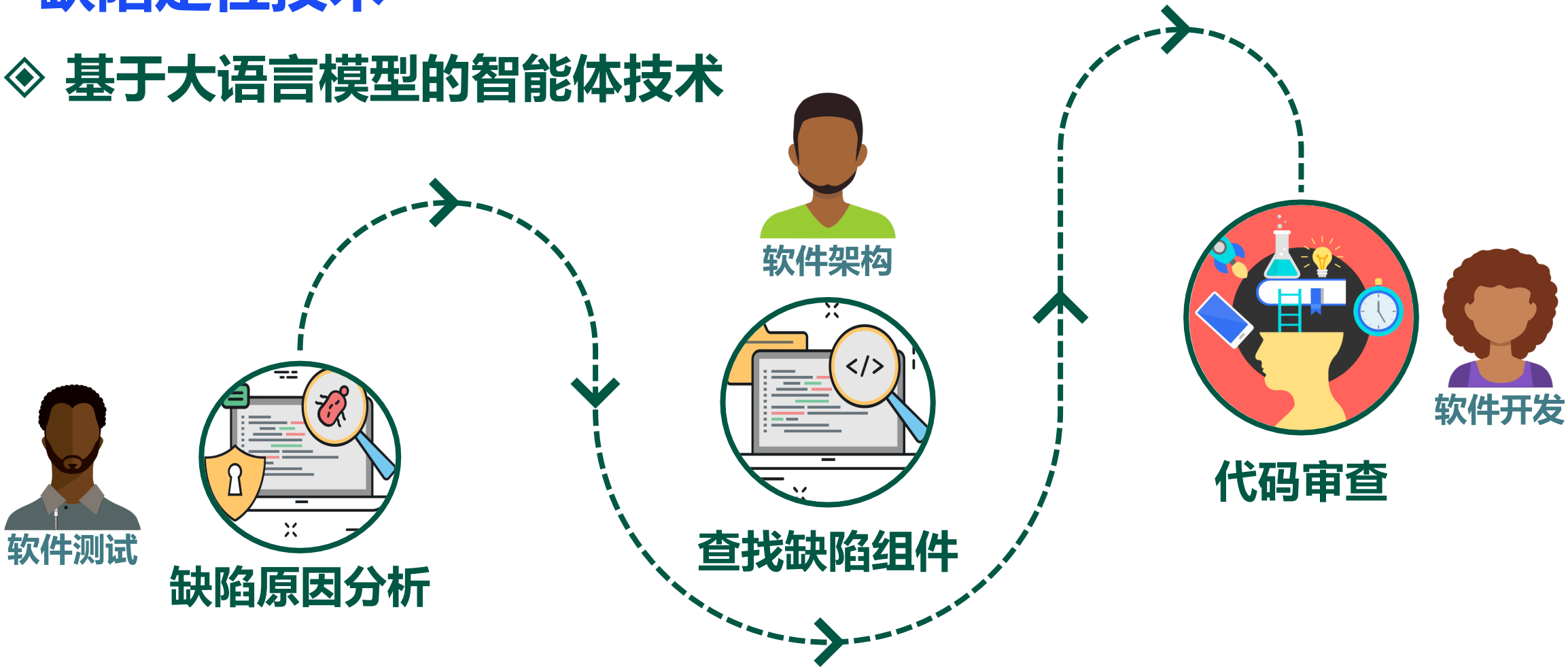
- 结合程序分析和大语言模型的程序缺陷定位技术
- 基于统一建模语言（UML）的程序执行信息表示技术
- 基于检索增强生成的缺陷定位知识重用技术

## 解决方案

- 基于大语言模型的软件缺陷自动定位系统

# 缺陷定位技术

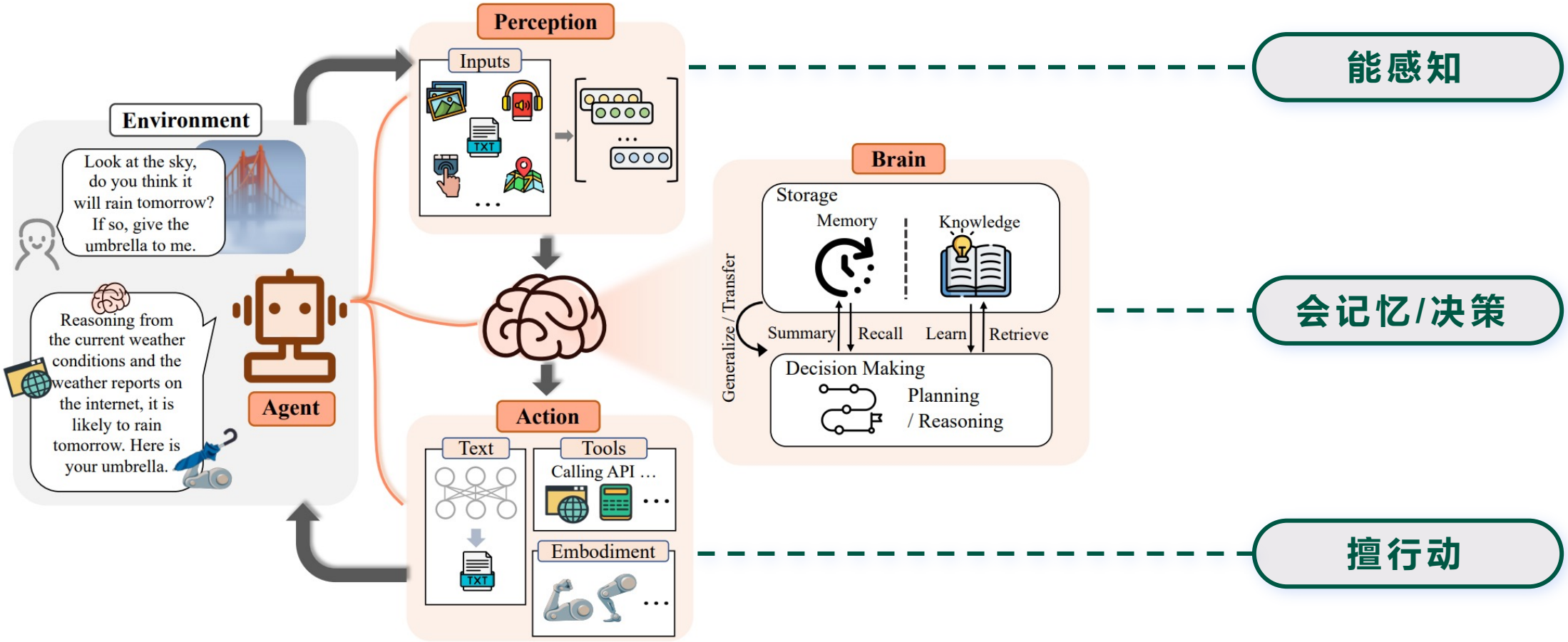
## 基于大语言模型的智能体技术




缺陷定位是一个复杂任务，可能需要多种专家知识的共同参与

# 缺陷定位技术

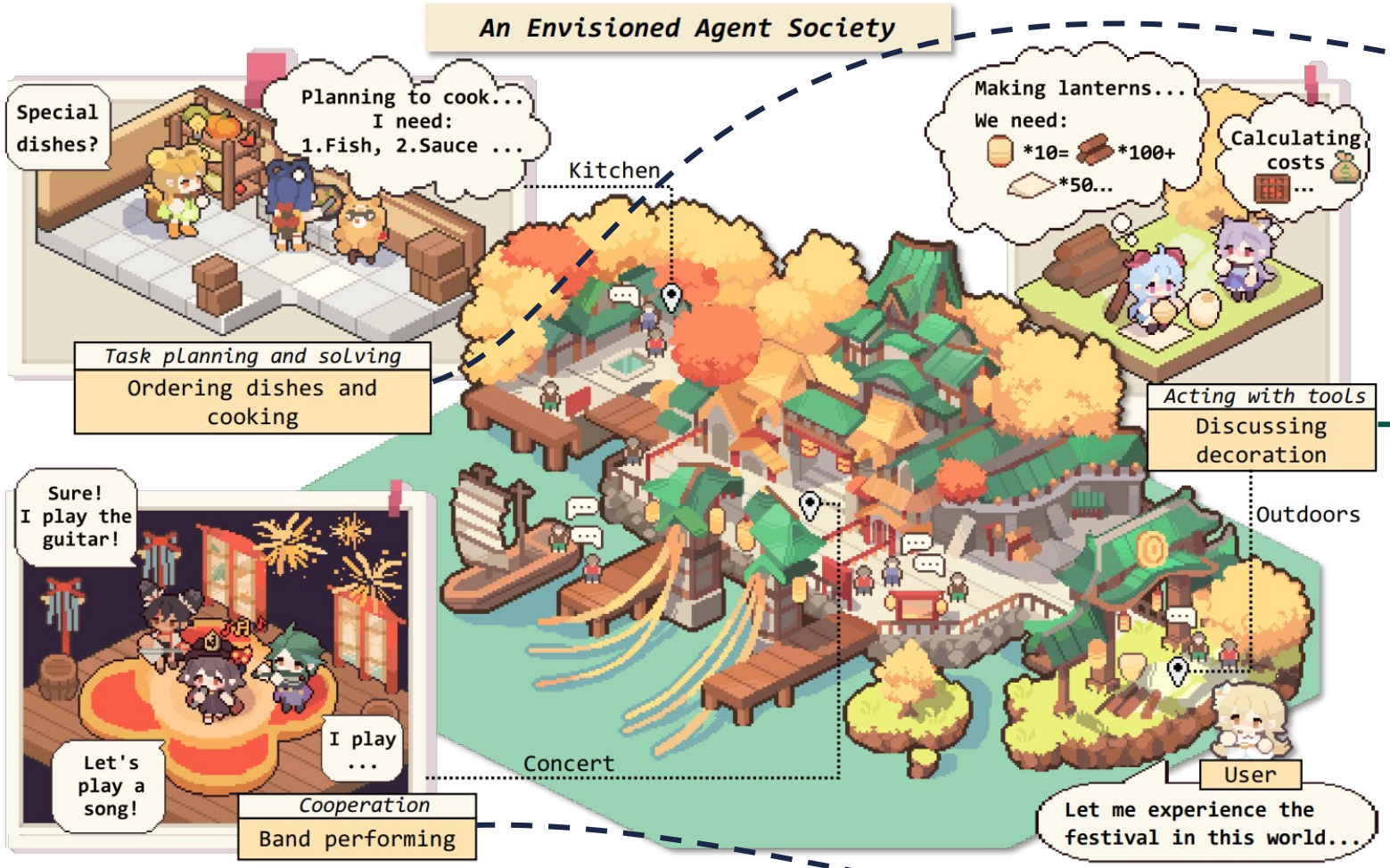
## 基于大语言模型的智能体技术




基于大语言模型的智能体的概念框架[11]

# 缺陷定位技术

## 基于大语言模型的智能体技术




设想的多智能体社会[11]

任务规划

工具使用

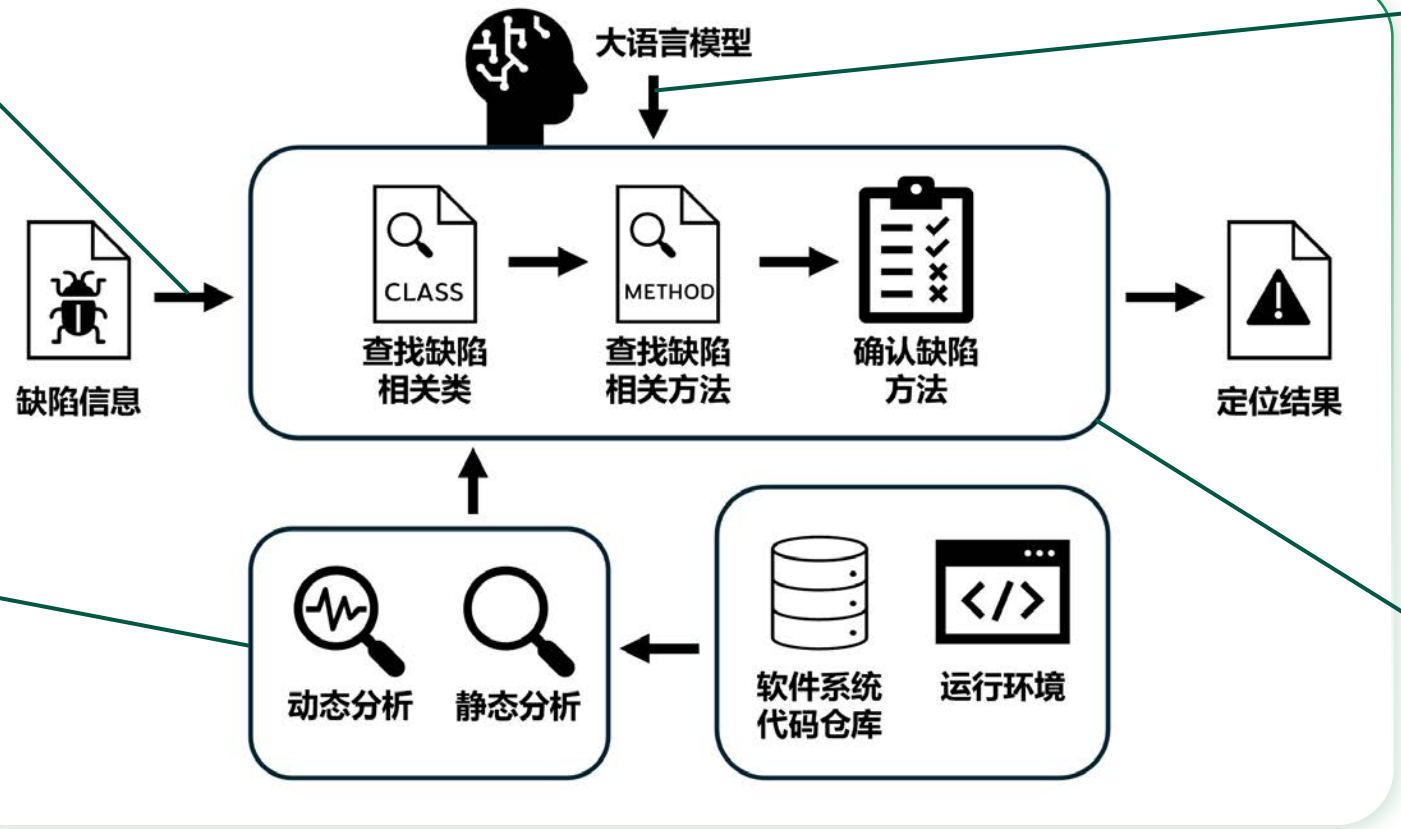
任务协作

# 缺陷定位技术

## 结合程序分析和大语言模型的程序缺陷定位技术

- 缺陷信息的处理；
- 不同缺陷信息的作用；
- ……

- 探究最优的提示格式；
- 智能体设计；
- ……




- 动态分析的成本控制；
- 静态分析获取的数据；
- 处理分析得到的数据；
- ……

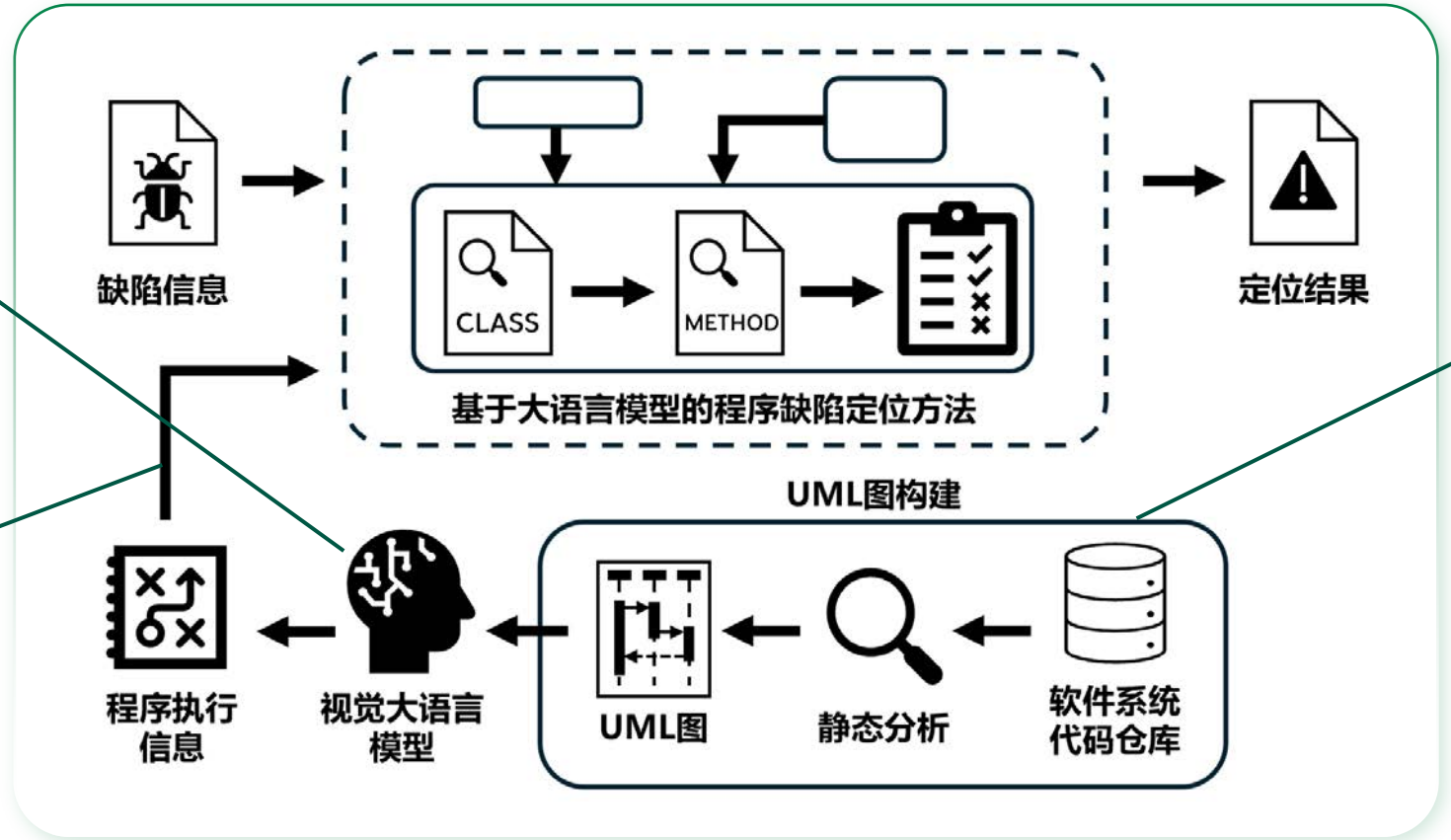
- 设计合理的思维逻辑；
- 不同步骤之间数据传递格式；
- ……

# 缺陷定位技术

## 基于统一建模语言（UML）的程序执行信息表示技术

- 提示设计；
- 生成数据处理；
- ……

- 输入格式设计；
- 数据质量评估；
- 删减冗余信息；
- ……



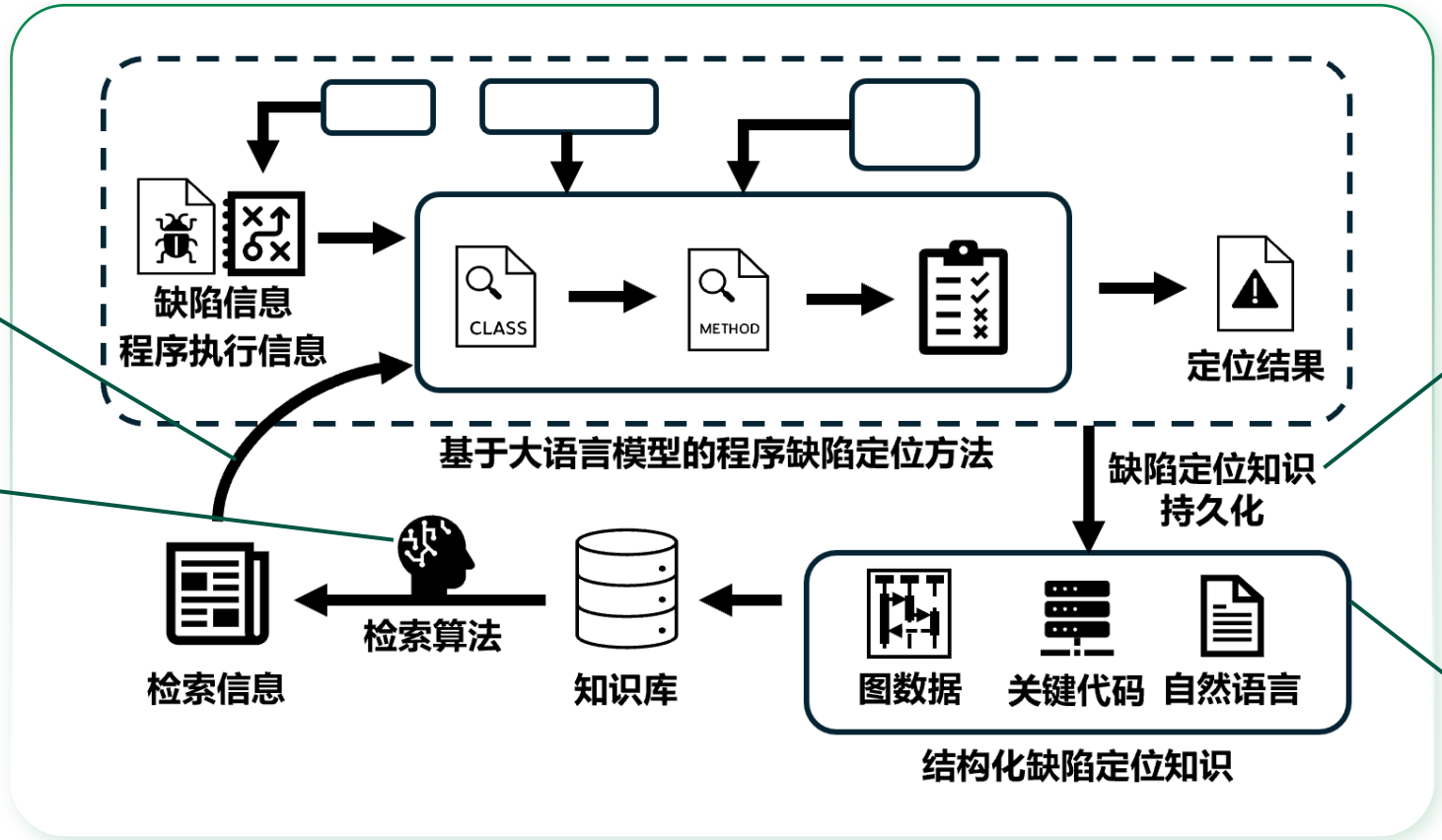

- 静态分析数据处理；
- UML图自动构建；
- ……

# 缺陷定位技术

## 基于检索增强生成的缺陷定位知识重用技术

- 输入格式设计；
- 数据质量评估；
- 删减冗余信息；
- ……

- 检索算法设计；
- 检索质量评估；
- 删减冗余信息；
- ……




- 缺陷知识识别；
- 缺陷知识抽取；
- 删除无用信息；
- ……

- 知识数据格式设计；
- 知识质量评估；
- 知识转化与生成；
- ……

# 缺陷定位技术

- 开发人员进行缺陷定位的过程是分阶段进行的

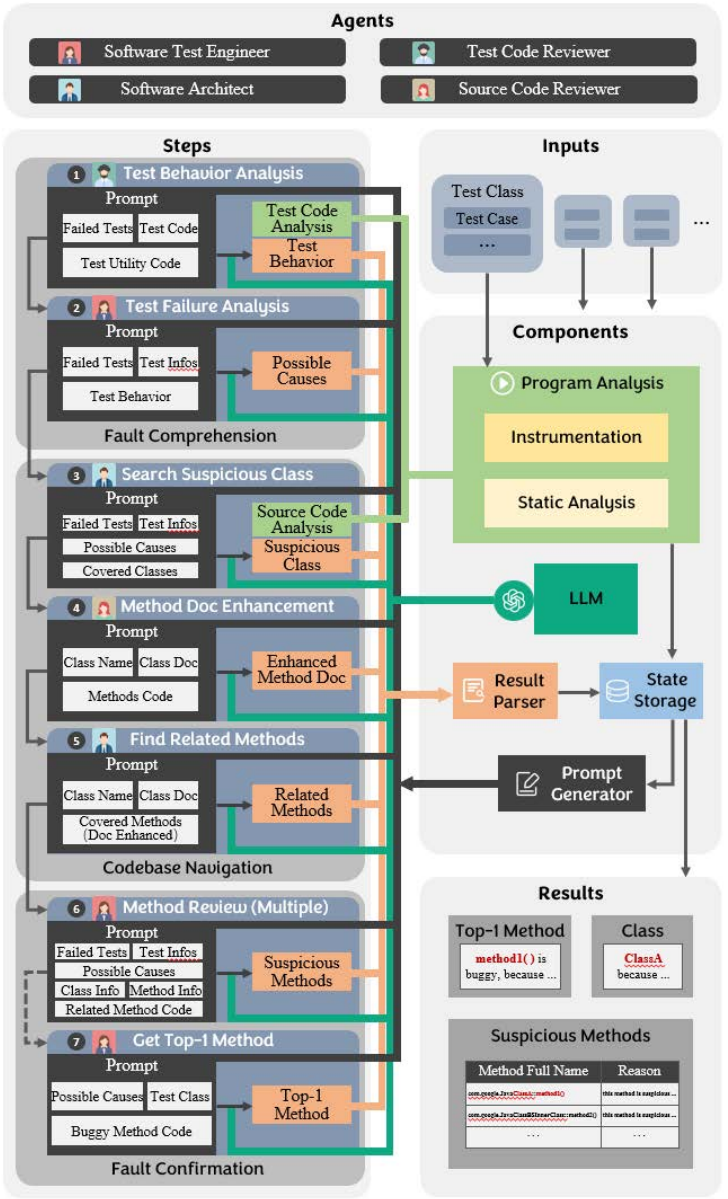

开发者缺陷定位行为模式

# 缺陷定位技术

## AgentFL——基于多智能体的缺陷定位

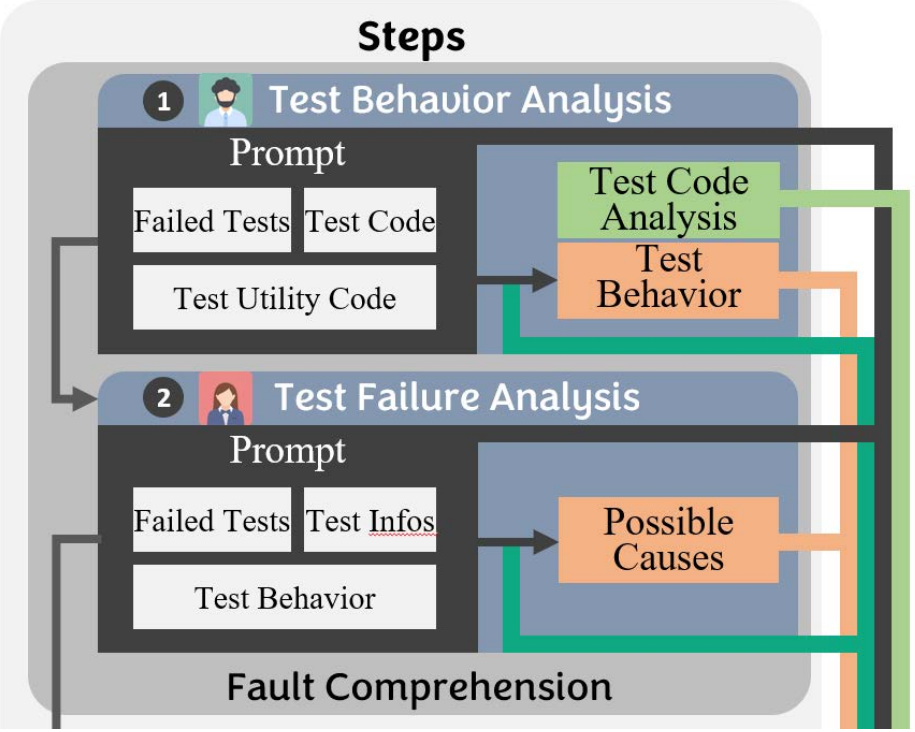
- 分阶段实现项目级缺陷定位任务


构建相互协作的多智能体，每一个智能体
专门负责某一具体的子任务



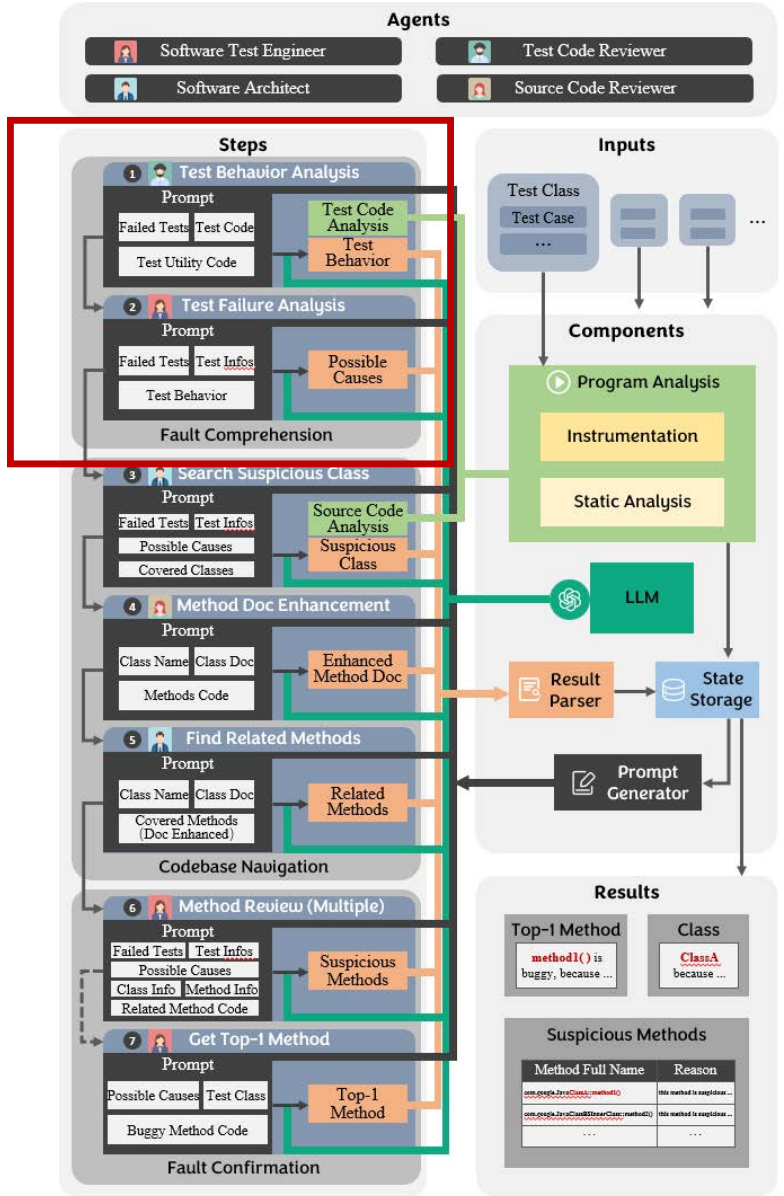
# 缺陷定位技术

## AgentFL—Phase-I




**分析测试行为** → **推测失败原因**

阶段1：分析测试失败原因

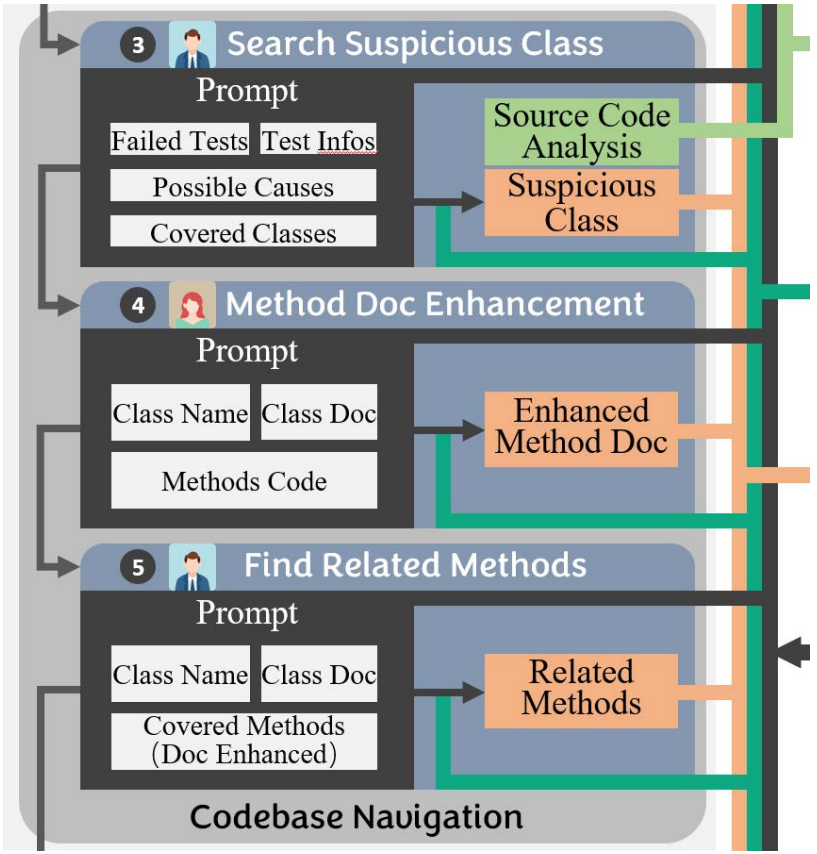


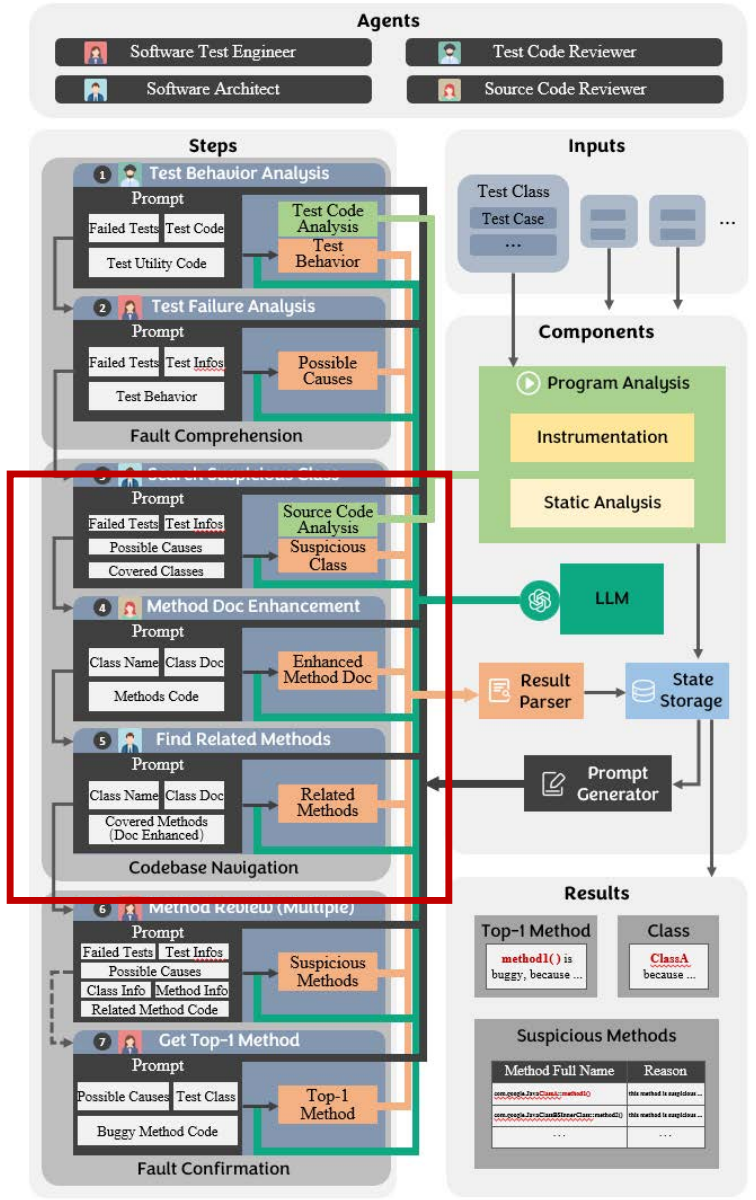
# 缺陷定位技术

## AgentFL—Phase-II

**搜索可疑类**

**寻找缺陷相关函数**

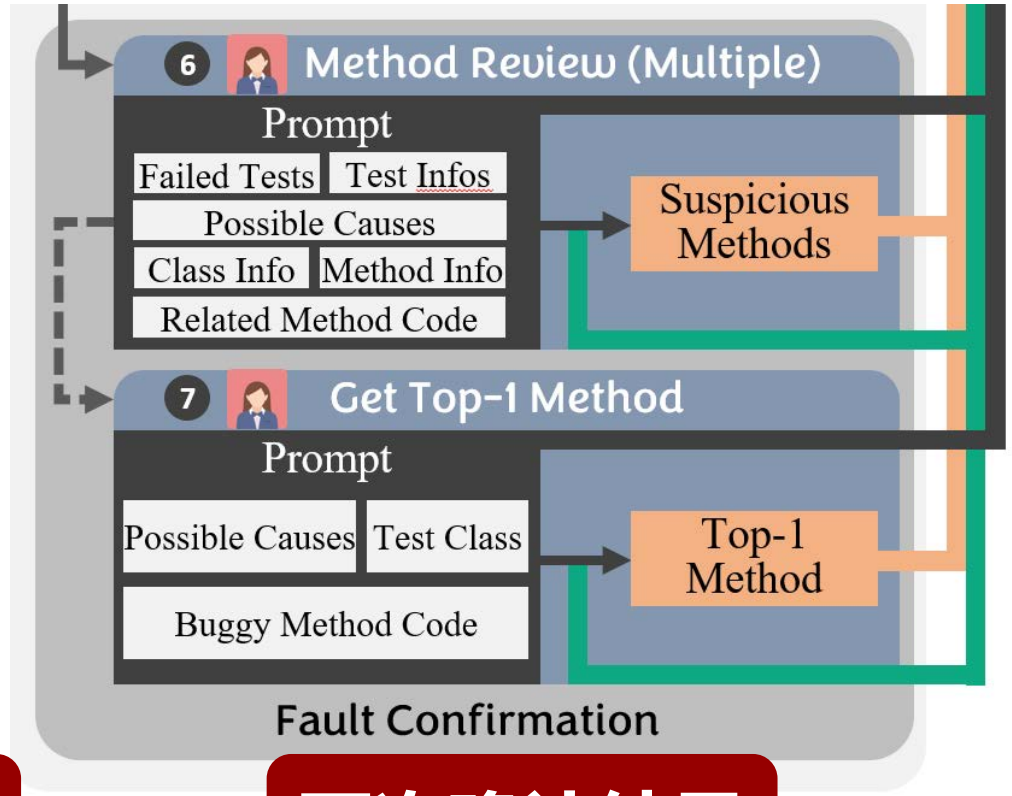


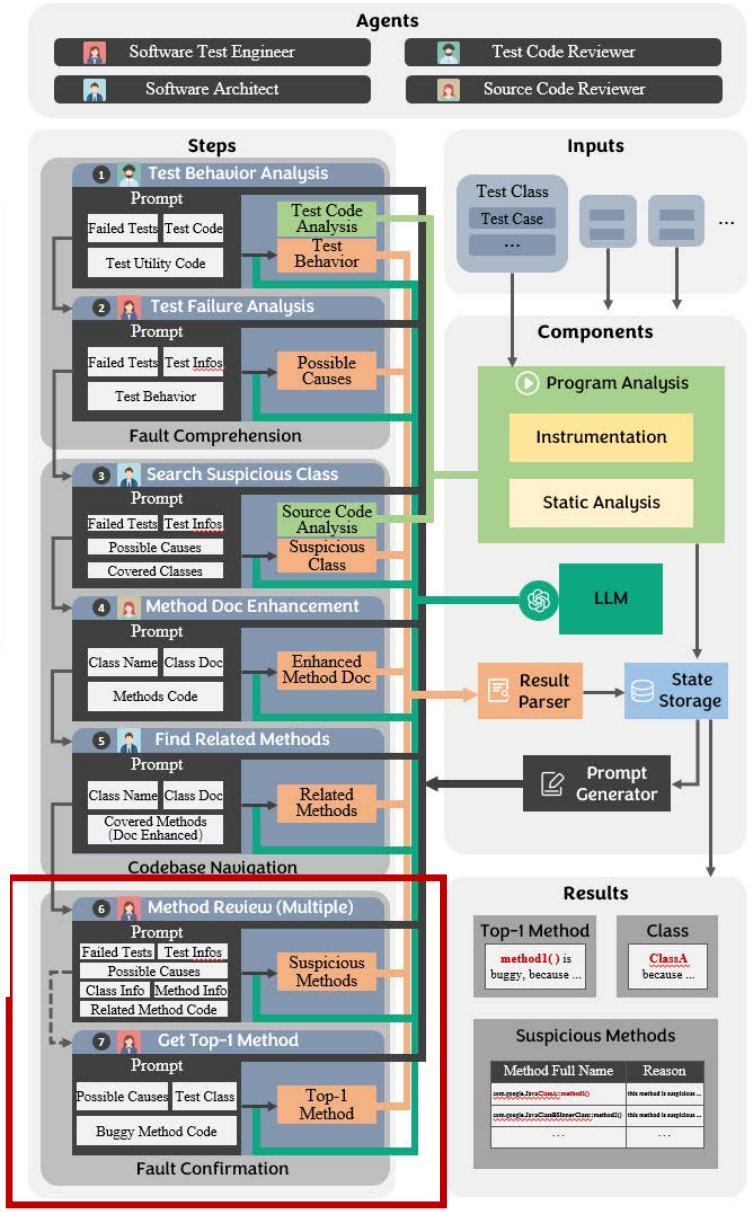

阶段2：借助软件文档，自顶向下地在整个软件项目中搜寻与缺陷相关的代码

# 缺陷定位技术

## AgentFL—Phase-III




**确定可疑函数** → **再次确认结果**




阶段3：检查源码并确认可疑函数

# PART 03

# 自动化生成修复建议

# 自动化生成修复建议——即时缺陷检测

## 通过挖掘历史数据训练分类器模型


高质量数据集
仅包含2332个缺陷实例

依赖模型自动提取隐式特征信息[2]

F1仅为0.26!

# 自动化生成修复建议——即时缺陷检测

通过挖掘历史数据训练分类器模型




**人类实践过程中是否存在可以辅助判断的特征?**

新代码变更

软件演化时间线



引入缺陷的代码变更

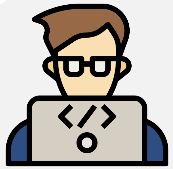
修复缺陷的代码变更

高质量数据集
仅包含2332个缺陷实例

依赖模型自动提取
隐式特征信息[3]

F1仅为0.26!

# 自动化生成修复建议——即时缺陷检测



## 人类实践：当代码变更存在以下特征时更易引入缺陷[3]

| 特征 | 原因 |
| --- | --- |
| 变更内容多 | 增加代码复杂性和产生问题的可能性 |
| 变更分布广 | 变更分布于文件系统中多个位置，测试难度大 |
| 开发人员经验不足 | 开发人员对项目和子系统不熟悉，会增加出错的可能性 |
| 代码变更之间间隔时间短 | 频繁更改，缺少时间进行彻底审查和测试 |

**linux/commit/9a0bf52**

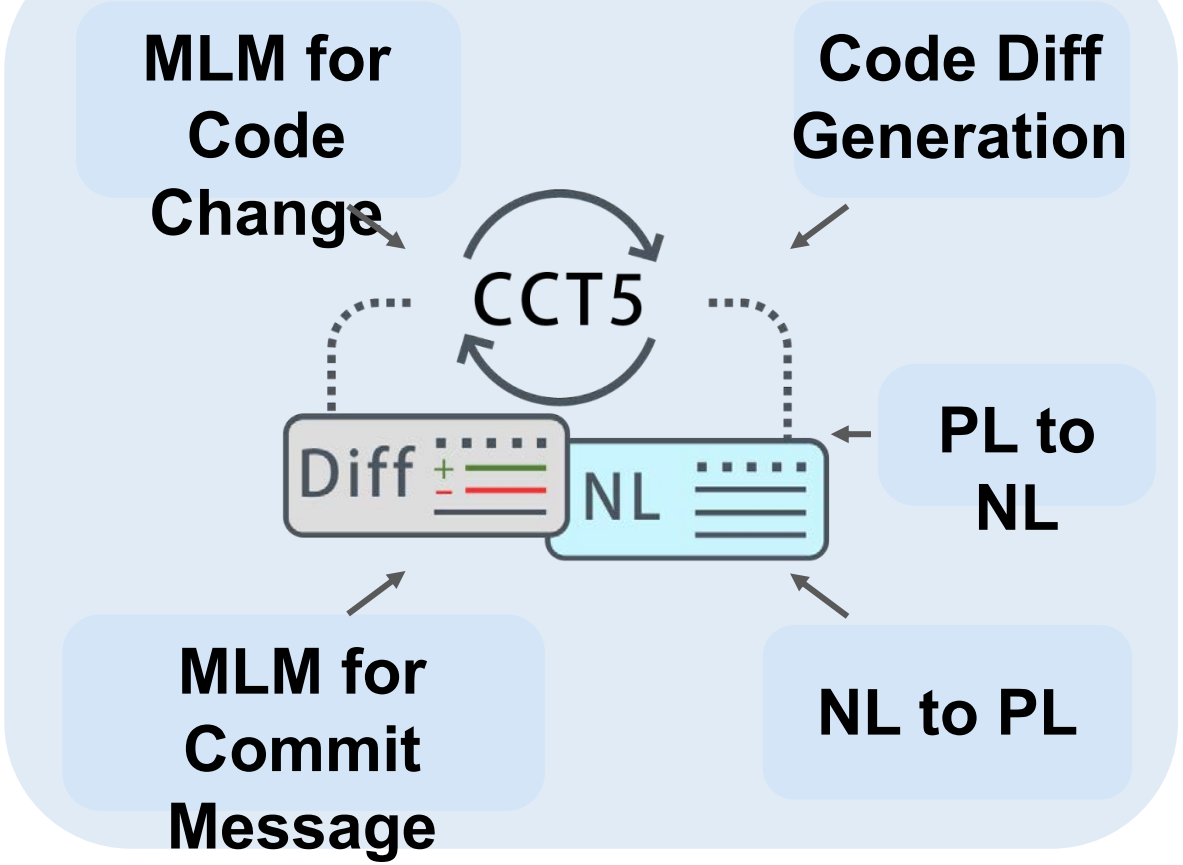
**272 changed files** with
**37 additions** and **38 deletions**

**An out-of-bounds read (3年后修复, 1fa23)**

**A buffer overflow (3年后修复, e6431)**

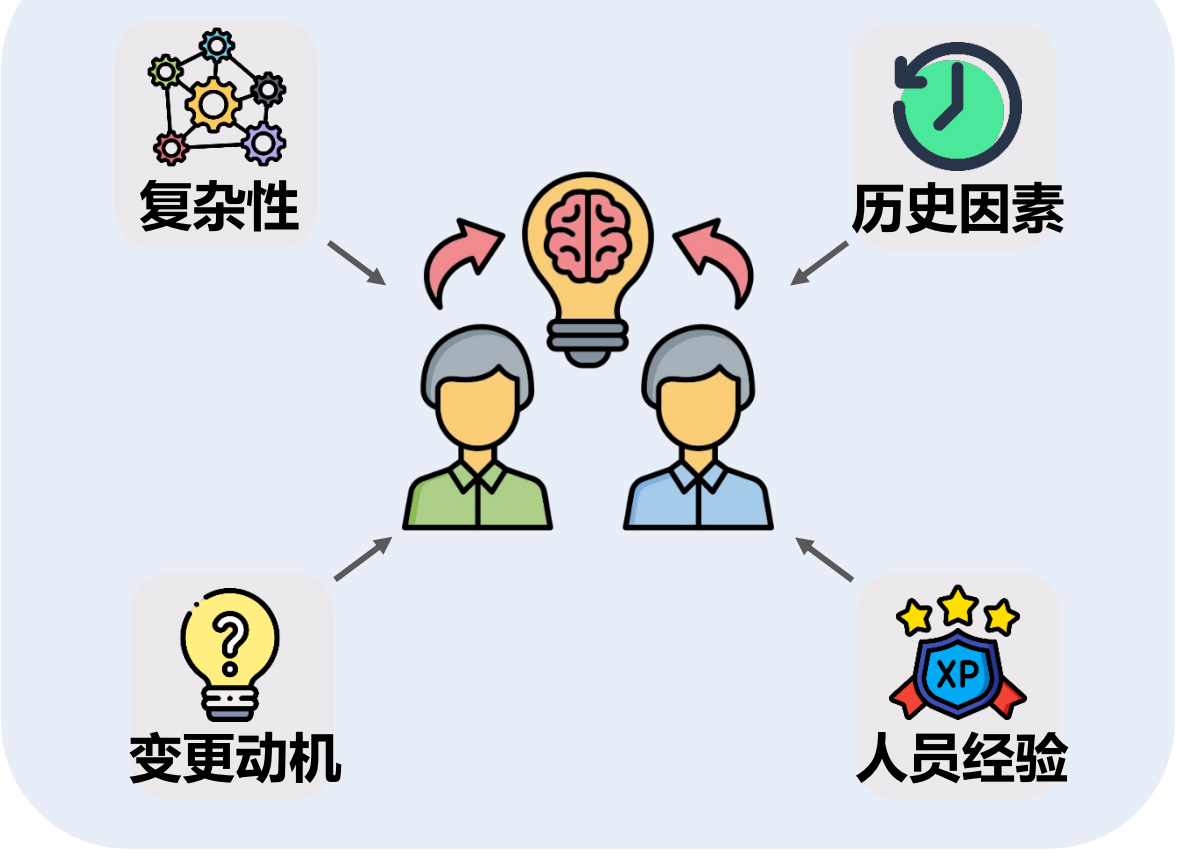
# ▶▶自动化生成修复建议——即时缺陷检测

## AI




构建面向代码变更的预训练模型

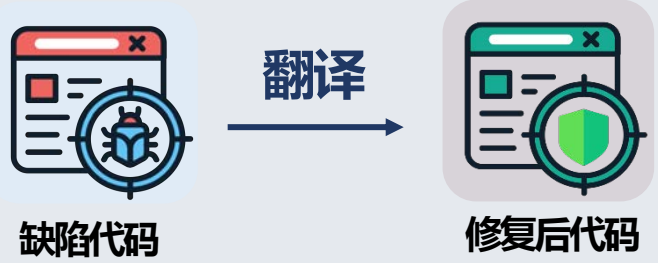
## Human




引入缺陷相关的专家知识

# 自动化生成修复建议——多维缺陷修复

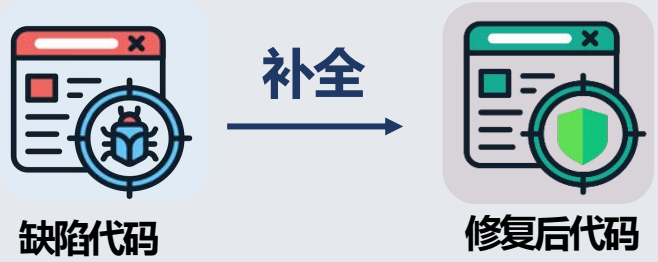
## 基于NMT的缺陷自动修复方法




规约缺陷修复为**翻译**任务[4]

依赖包含**大量**
缺陷代码 → 修复后代码
案例的**高质量**数据集

## 基于MLM的缺陷自动修复方法

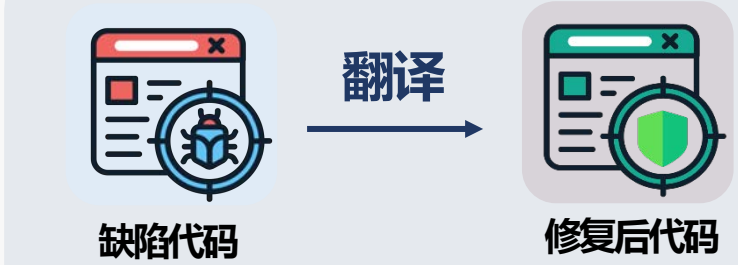



规约缺陷修复为**补全**任务[5]

依赖模型在预训练过程中
掌握的**掩码语言建模**能力

# 自动化生成修复建议——多维缺陷修复

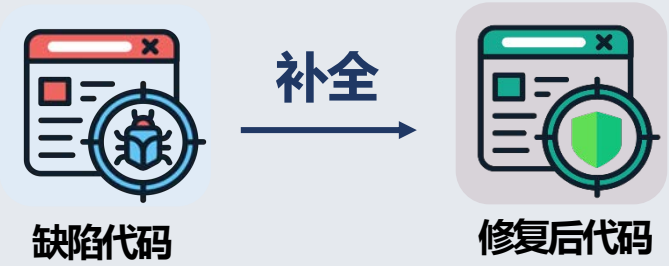
## 基于NMT的缺陷自动修复方法




规约缺陷修复为**翻译**任务

关注于**语句**粒度的修复

## 基于MLM的缺陷自动修复方法




规约缺陷修复为**补全**任务

关注于**令牌**粒度的修复

**现有方法通常针对单一的修复粒度进行补丁生成**

# 自动化生成修复建议——多维缺陷修复

基于NMT的缺陷自动修复方法


关注于语句粒度的修复

基于MLM的缺陷自动修复方法


关注于令牌粒度的修复

开发人员只在特定粒度上修bug吗?

现有方法通常针对单一的修复粒度进行补丁生成

# 自动化生成修复建议——多维缺陷修复

语句级

表达式级

令牌级

**实证分析得到的修复粒度分布**

| | 类别 | 占比(%) |
|---|---|---|
| 语句类型 | Expression Statement | 17.3 |
| | If Statement | 7.8 |
| | Return Statement | 3.3 |
| | 其他 | 2.4 |
| | 总计 | **30.8** |
| 表达式类型 | Method Invocation | 20.1 |
| | Variable Declaration | 5.7 |
| | Assignment | 3.1 |
| | 其他 | 4.7 |
| | 总计 | **35.8** |
| 令牌类型 | Identifier | 19.4 |
| | Type Literal | 5.7 |
| | Modifiers | 5.7 |
| | 其他 | 4.8 |
| | 总计 | **33.4** |

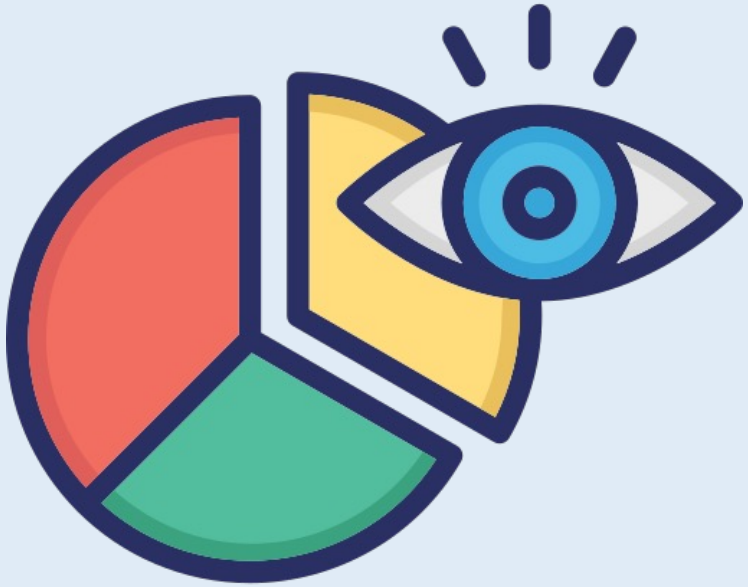
# 自动化生成修复建议——多维缺陷修复

语句级

表达式级

令牌级

调研得到的修复粒度分布

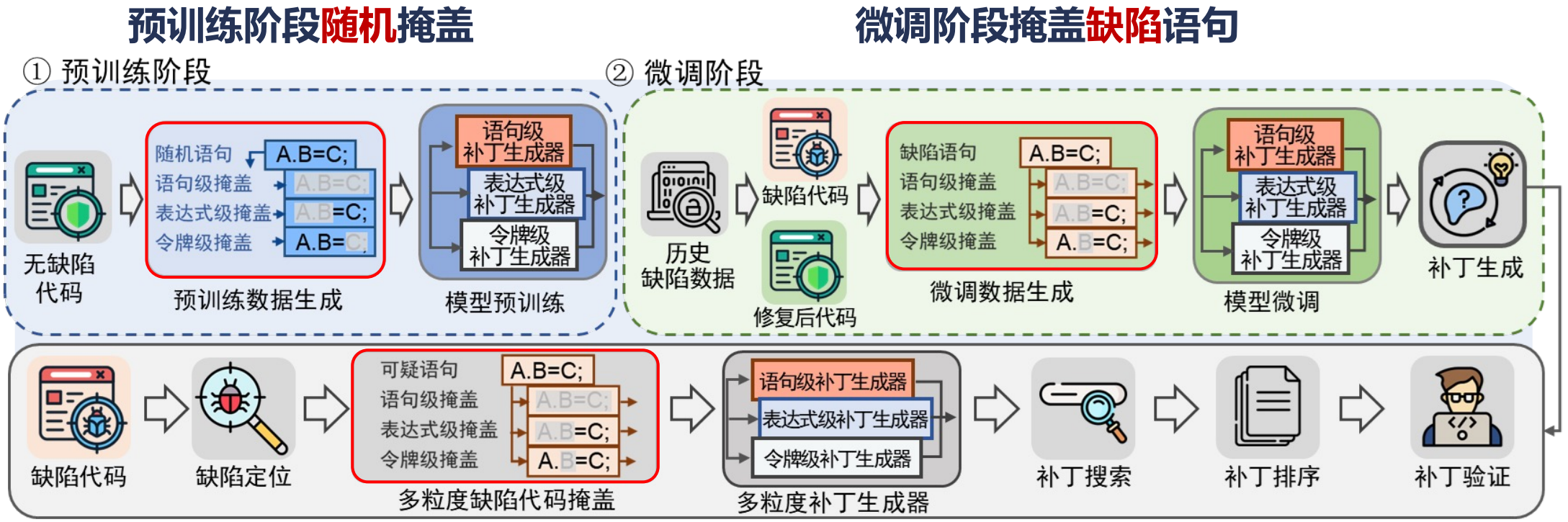


仅关注单个粒度不足以应对所有的缺陷

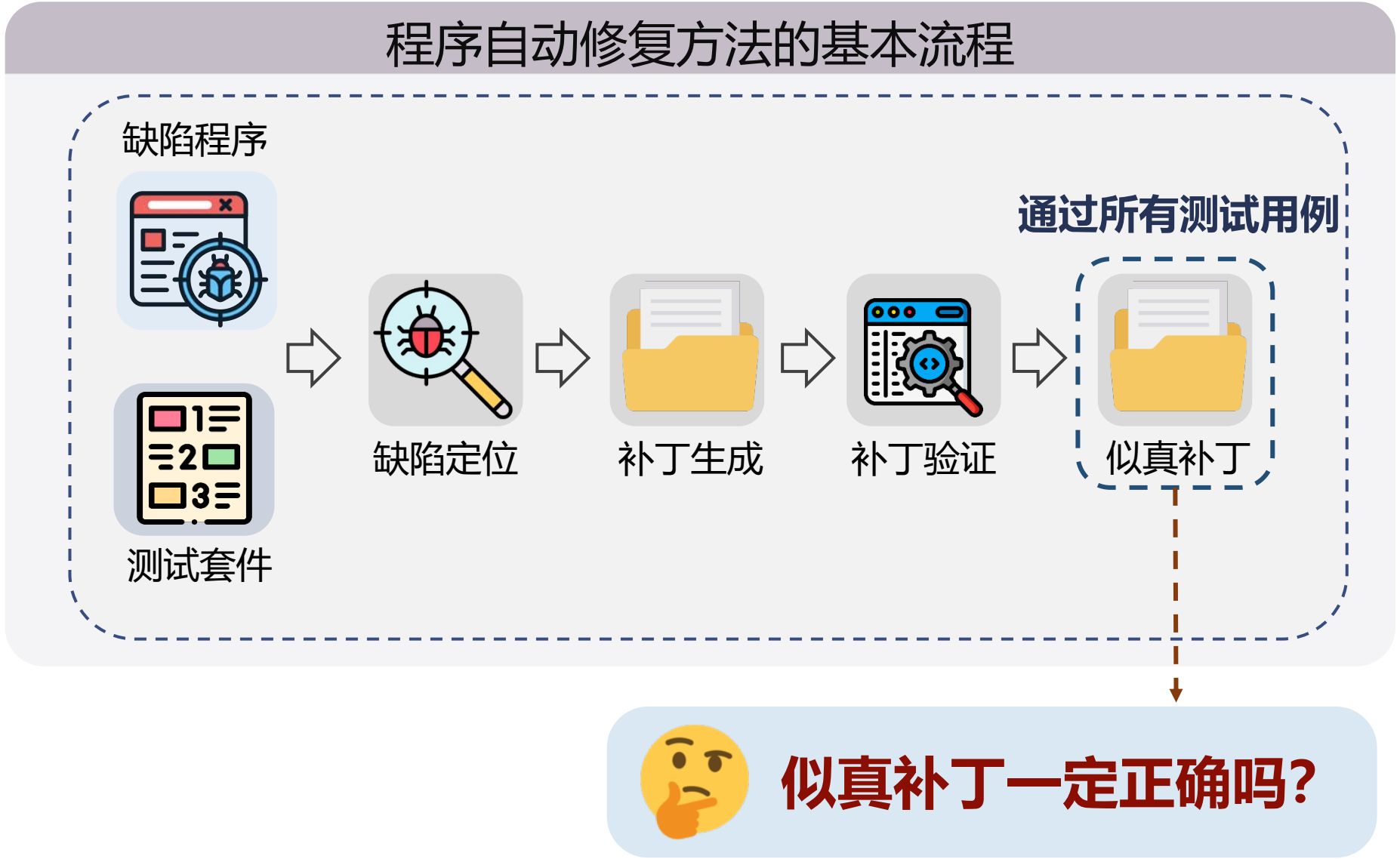
| 类型 | | |
|---|---|---|
| | 其他 | 4.8 |
| | 总计 | 33.4 |

# 自动化生成修复建议——多维缺陷修复

- 预训练多个擅于在不同粒度上生成补丁的模型
- 基于启发式对多粒度上生成的补丁进行排序



# 自动化生成修复建议——补丁正确性评估



# 自动化生成修复建议——补丁正确性评估

❑通过输入**缺陷代码、修复后代码**等信息，对生成的补丁进行正确性评估[6]

Lang-61 patch

```
    if (strLen > size) {
        return -1;
    }
    char[] thisBuf = buffer;
-   int len = thisBuf.length - strLen;
+    int len = size - strLen + 1;
    outer:
    for(int i = startIndex; i < len; i++){
        for (int j = 0; j < strLen; j++) {
```

**当前先进方法将生成的补丁当作token序列进行处理**

**Changed Lines:**

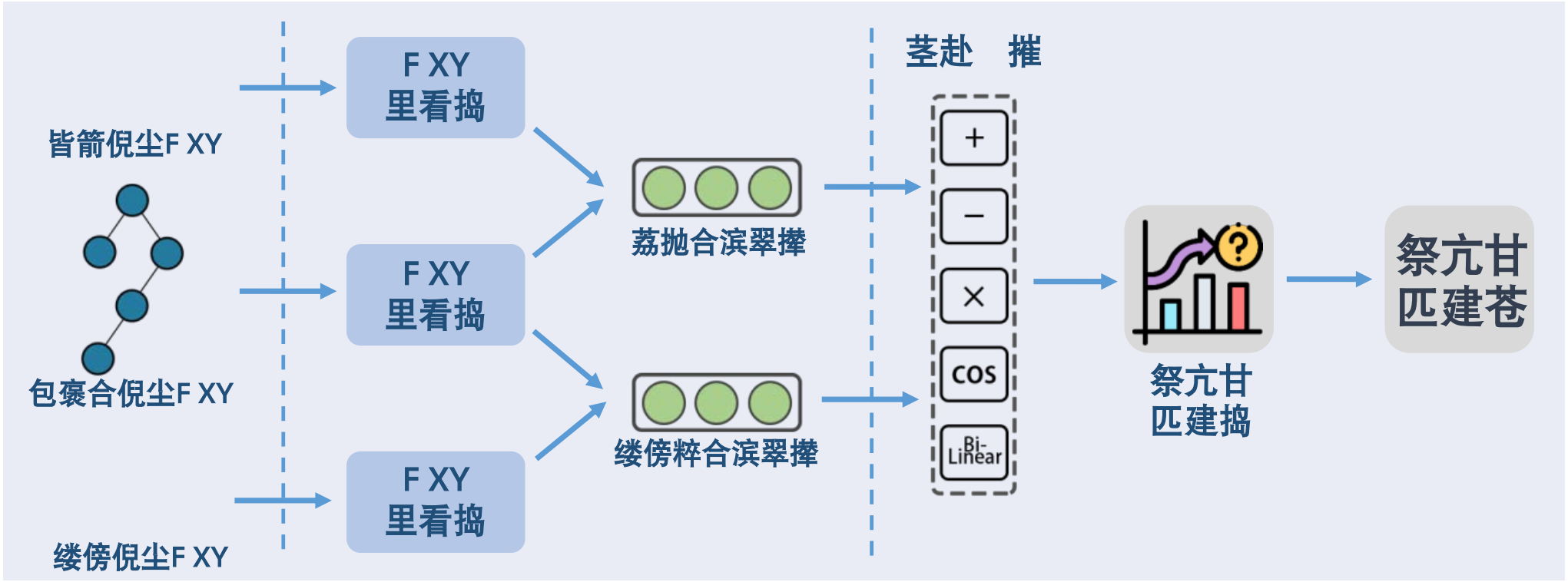
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| int | len | = | thisBuf | length | - | strLen | |
| int | len | = | size | - | strLen | + | 1 |

# 自动化生成修复建议——补丁正确性评估

- 开发人员不仅基于代码变更文本评估补丁正确性

# 自动化生成修复建议——补丁正确性评估

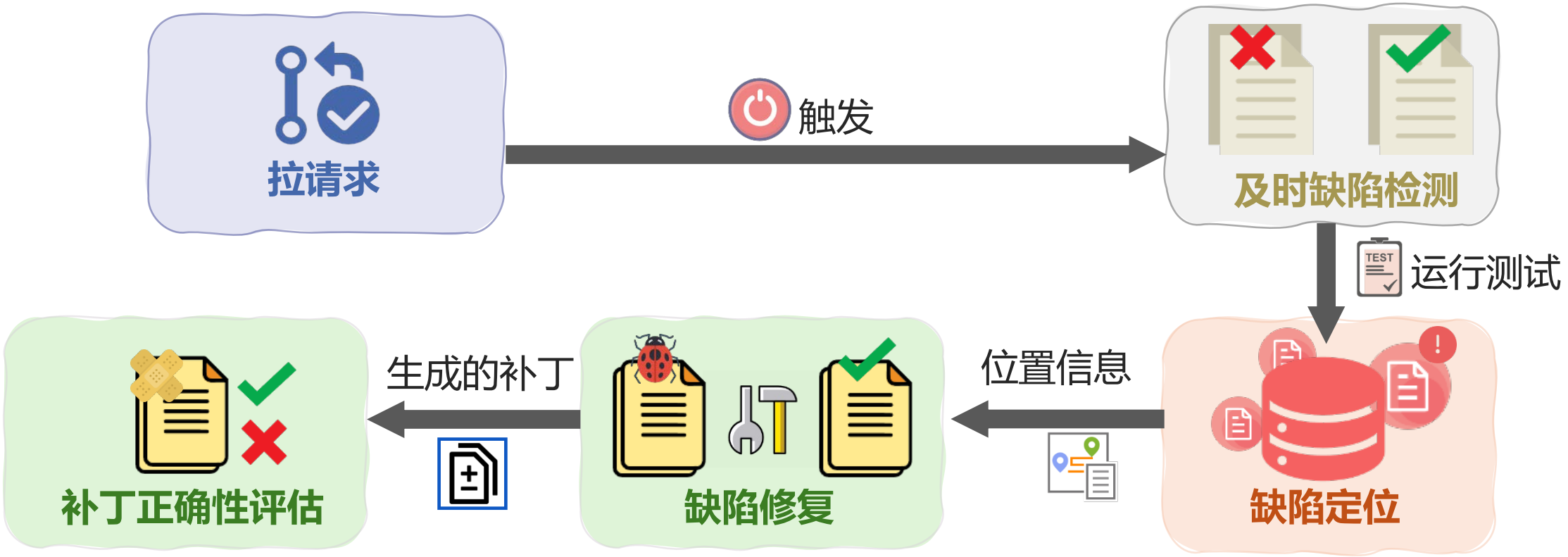
## CACHE——上下文感知的补丁正确性评估技术



# PART 04

# 总结与展望

# 总结与展望

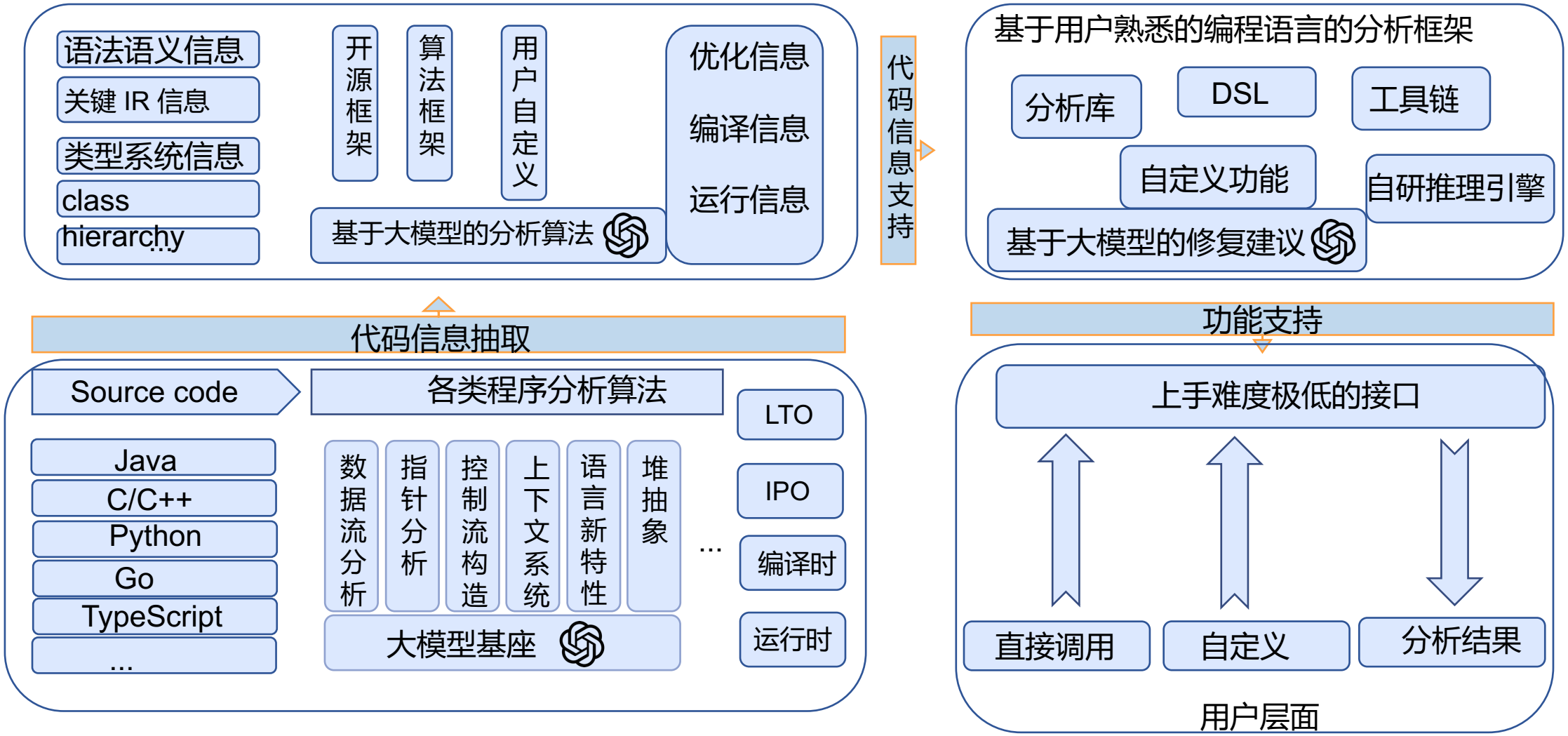


# 总结与展望

软件可靠性保障—代码审查


程序理解—注释更新

# 总结与展望


语法语义信息
关键 IR 信息
类型系统信息
class hierarchy
...
开源框架
算法框架
用户自定义
基于大模型的分析算法
优化信息
编译信息
运行信息
代码信息支持
基于用户熟悉的编程语言的分析框架
分析库
DSL
工具链
自定义功能
自研推理引擎
基于大模型的修复建议
代码信息抽取
功能支持
Source code
各类程序分析算法
Java
C/C++
Python
Go
TypeScript
...
数据流分析
指针分析
控制流构造
上下文系统
语言新特性
堆抽象
...
大模型基座
LTO
IPO
编译时
运行时
上手难度极低的接口
直接调用
自定义
分析结果
用户层面

# 科技生态圈峰会 + 深度研习

## ——1000 + 技术团队的共同选择

KEYLINK ing

### K+峰会

**K+峰会 · 敦煌站**
K+ 思考周®研习社
时间：2025.08.29-30

**K+峰会 · 上海站**
K+ 金融专场
时间：2025.10.17-18

**K+峰会 · 香港站**
K+ 思考周®研习社
时间：2025.11.25-26

K+峰会详情

### AiDD峰会

**AiDD峰会 · 上海站**
AI+研发数字峰会
时间：2025.05.17-18

**AiDD峰会 · 北京站**
AI+研发数字峰会
时间：2025.08.08-09

**AiDD峰会 · 深圳站**
AI+研发数字峰会
时间：2025.11.28-29

AiDD峰会详情

AI+研发数字峰会 第5届
AI+ Development Digital summit

利用AI技术深化计算机对现实世界的理解

# 推动研发进入智能化时代