

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

SUNGMIN KANG*, KAIST, South Korea

GABIN AN*, KAIST, South Korea

SHIN YOO, KAIST, South Korea

Proceedings of the ACM on Software Engineering, Volume 1, Issue FSE
Article No.: 64, Pages 1424 - 1446

発表者：松尾 和輝

Fault Localization (FL)

- コードのどの部分に不具合があるかを特定する作業
 - デバッグで最も**時間がかかる作業**であると知られている^[8]
- 既存の自動FL技術^[45]
 - スペクトルベースFL (SBFL)
 - テストケースの成否と実行経路を元に特定する手法
 - 情報検索ベースFL (IRFL)
 - バグ報告を用いて特定する手法
 - 突然変異ベースFL (MBFL)
 - SBFLにミューテーション解析を導入した手法
- 実務者が求めるものとの乖離があるため、ほとんど利用されていない

[8] Marcel Böhme, Ezekiel O. Soremekun, Sudipta Chattopadhyay, Emamurho Ugherughe, and Andreas Zeller. 2017. Where is the Bug and How is It Fixed? An Experiment with Practitioners. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (Paderborn, Germany) (ESEC/FSE 2017). Association for Computing Machinery, New York, NY, USA, 117–128. <https://doi.org/10.1145/3106237.3106255>

[45] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A Survey on Software Fault Localization. IEEE Transactions on Software Engineering 42, 8 (2016), 707–740. <https://doi.org/10.1109/TSE.2016.2521368>

既存手法の課題

- 説明の欠如^[20]
 - 約90%の開発者が、自動FL技術に対して合理的な説明を提供する能力を重視
 - 既存の自動FL技術は説明を提供する能力を持っていない
- LLMを用いることで不具合の位置特定と説明を自動生成可能
 - ソフトウェア工学と自然言語処理タスクに長けているため
 - FLはリポジトリ全体を対象とするが、LLMにはコンテキスト長の制約がある
 - 32,000トークンを処理できるLLMがあるが、処理しきれない

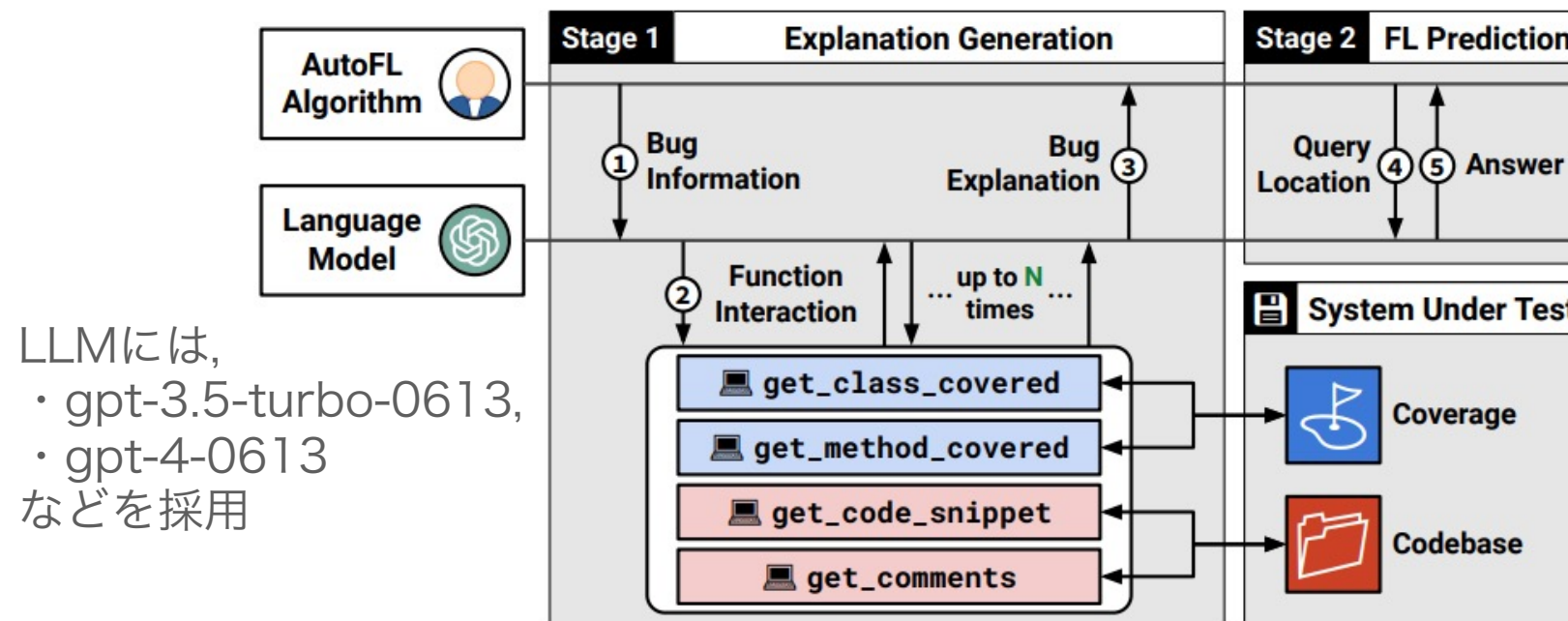
[20] Pavneet Singh Kochhar, Xin Xia, David Lo, and Shanping Li. 2016. Practitioners' Expectations on Automated Fault Localization. In Proceedings of the 25th International Symposium on Software Testing and Analysis (Saarbrücken, Germany) (ISSTA 2016). Association for Computing Machinery, New York, NY, USA, 165–176. <https://doi.org/10.1145/2931037.2931051>

目的・アプローチ

- 目的
 - LLMを用いた新しいFL技術であるAutoFLの提案
- アプローチ
 - プロンプトを用いて，説明を自動生成し不具合の位置特定

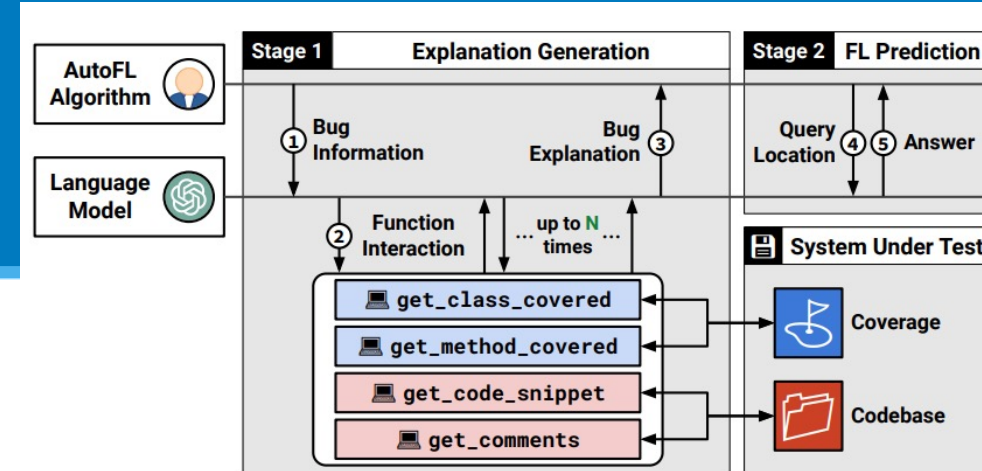
AutoFL | 概要図

- Stage1 | 説明の生成
- Stage2 | 不具合位置の特定
- 不具合位置の最終決定



- 入力 | テスト名, テストコード, エラーメッセージとスタックトレース
- 出力 | 不具合を含むと予測したメソッド, 説明

Stage 1 | 説明の生成



- システムプロンプトの入力
 - LLMに対して役割とタスクを明示するためのメッセージ
 - 最初にLLMに何をして欲しいのかを説明する

Listing 1. System Prompt for LLM

You are a debugging assistant. You will be presented with a failing test, and tools (functions) to

- ↪ access the source code of the system under test (SUT). Your task is to provide a step-by-
- ↪ step explanation of how the bug occurred, based on the failing test and the information
- ↪ you retrieved using tests about the SUT. You will be given N chances to interact with
- ↪ functions to gather relevant information. An example answer would look like follows.

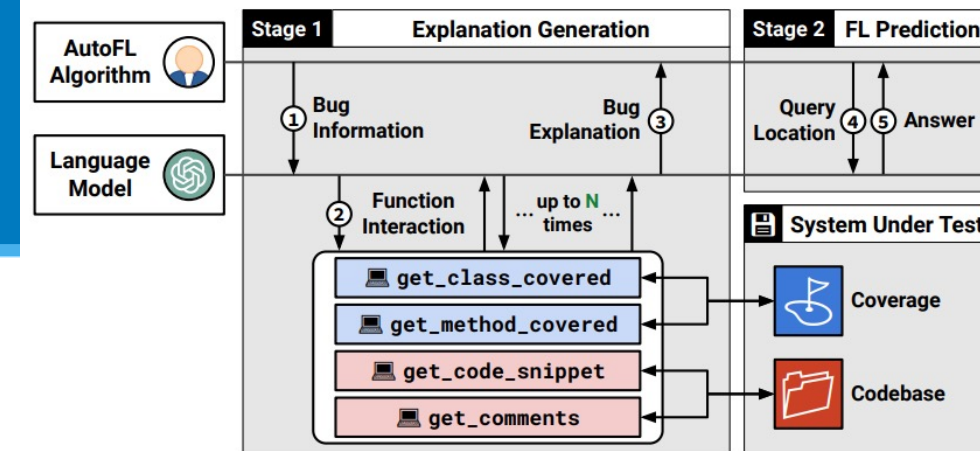
<HANDCRAFTED ROOT CAUSE ANALYSIS EXAMPLE>

あなたはデバッグアシスタントです。あなたには、失敗したテストと、テスト対象システム (SUT) のソースコードにアクセスするためのツール (関数) が与えられます。あなたの仕事は、失敗したテストと、SUT に関するテストを使用して取得した情報に基づいて、バグがどのように発生したかを段階的に説明することです。関連する情報を収集するために、関数と対話する機会が N 回与えられます。回答例は次のようになります。

<HANDCRAFTED ROOT CAUSE ANALYSIS EXAMPLE>

HANDCRAFTED ROOT CAUSE ANALYSIS EXAMPLE：手作りの例，バグの原因を2文で説明する簡潔なもので，AutoFLが簡潔な説明を生成するように誘導

Stage 1 | 説明の生成

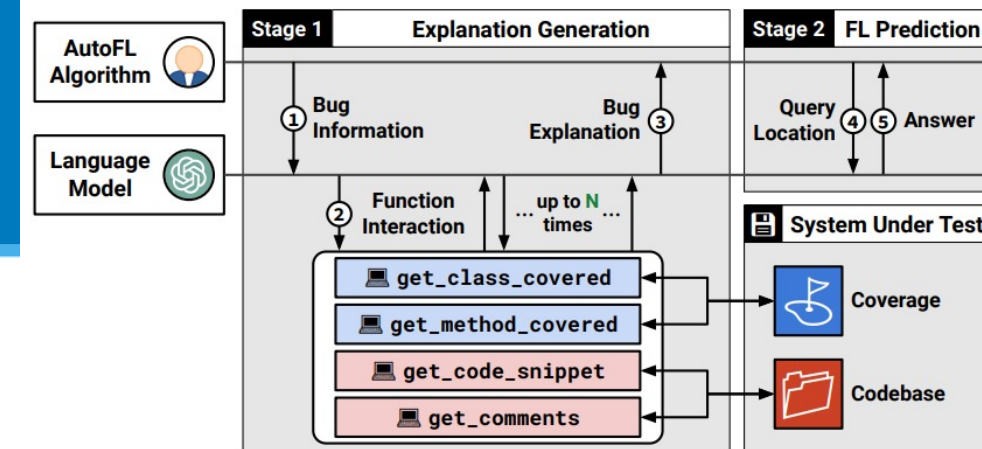


- ①初期プロンプトの生成
 - 特定のバグに関する詳細な情報をLLMに提供するためのメッセージ

Listing 2. Example prompt from Defects4J Lang-48

テスト名	1 The test <code>...EqualsBuilderTest::testBigDecimal()</code> failed. The test looks like:
	2
	3 <code>```java</code>
	4 <code>381 : public void testBigDecimal() {</code>
テストコード	5 <code>382 : BigDecimal o1 = new BigDecimal("2.0");</code>
	6 <code>383 : BigDecimal o2 = new BigDecimal("2.00");</code>
	7 <code>385 : assertTrue(new EqualsBuilder().append(o1, o2).isEquals()); // error occurred here</code>
	8 <code>386 : }</code>
	9 <code>```</code>
	10
	11 It failed with the following error message and call stack:
	12 <code>```</code>
エラーメッセージと スタックトレース	13 <code>junit.framework.AssertionFailedError</code>
	14 <code>at ...EqualsBuilderTest::testBigDecimal(EqualsBuilderTest.java:385)</code>
	15 <code>```</code>
	16 Start by calling the <code>get_failing_tests_covered_classes</code> function.

Stage1 | 説明の生成



- ②関数呼び出しの実行

- 情報収集のために，初期プロンプトに基づいて，予め用意された関数を呼び出す

- クラスレベルのカバレッジ情報取得 (`get_class_covered`)
 - 失敗したテストに関連するクラスの情報を取得
- メソッドレベルのカバレッジ情報取得 (`get_method_covered`)
 - 失敗したテストに関連するメソッドの情報を取得
- コードスニペットの取得 (`get_code_snippet`)
 - 関連のあるメソッドのコードスニペットを取得
- ドキュメントの取得 (`get_comment`)
 - 関連のあるメソッドのドキュメントを取得

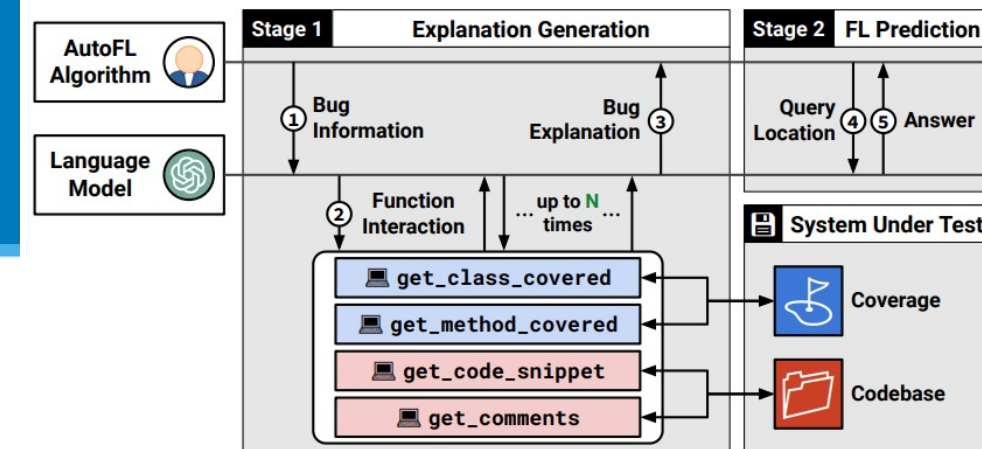
Stage 1 | 説明の生成

- ②関数呼び出しの実行

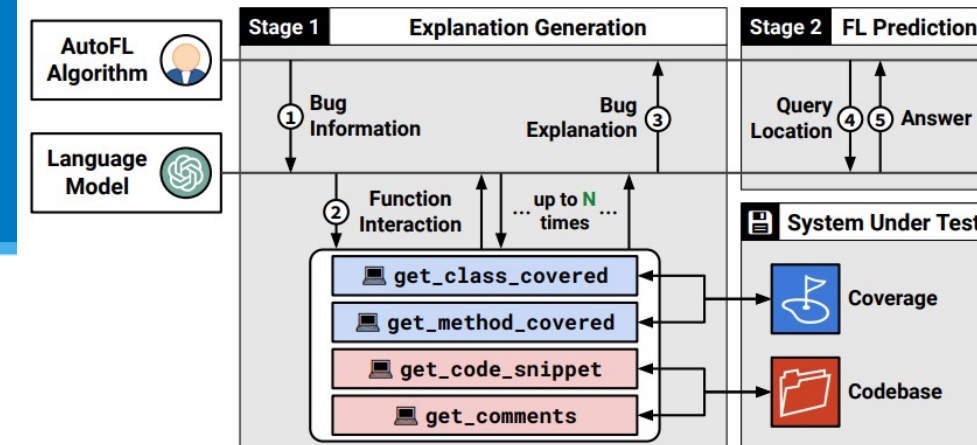
Listing 2. Example prompt from Defects4J Lang-48

```
1 The test `...EqualsBuilderTest::testBigDecimal()` failed. The test looks like:
2
3 ```java
4 381 : public void testBigDecimal() {
5 382 : BigDecimal o1 = new BigDecimal("2.0");
6 383 : BigDecimal o2 = new BigDecimal("2.00");
7 385 : assertTrue(new EqualsBuilder().append(o1, o2).isEqual()); // error occurred here
8 386 : }
9 ```
10
11 It failed with the following error message and call stack:
12 ```
13 junit.framework.AssertionFailedError
14   at ...EqualsBuilderTest::testBigDecimal(EqualsBuilderTest.java:385)
15 ```
16 Start by calling the get_failing_tests_covered_classes function.
```

- get_class_coveredを呼び出すよう指示
- 関連するクラスの情報（ここではBigDecimal, EqualsBuilder）を取得
- 取得した情報をもとに、さらに関数呼び出しを実行（例えば、 EqualsBuilderに対してget_method_coveredを実行）
- 必要な情報を得られるまで、最大N回関数呼び出しを実行



Stage 1 | 説明の生成



③不具合の説明生成

- 収集した情報をもとに不具合の説明を自動生成

当たり障りのない説明

(a) Defects4J Math-105 Run#2

The failing test `SimpleRegressionTest.testSSENonNegative()` belongs to the `org.apache.commons.math.stat.regression` package and is testing the `SimpleRegression` class.

Bland

失敗したテスト
`SimpleRegressionTest.testSSENonNegative()`
 は `org.apache.commons.math.stat.regression`
 パッケージに属し、`SimpleRegression` クラスをテ
 ストしています。

正確で簡潔な、有用な説明

(b) Defects4J Lang-47 Run#1

(...)
 The `NullPointerException` occurred at line 1186, where `str` is attempted to be accessed. This indicates that the `obj` argument passed to the method is `null`, causing the `getNullText` method to be called, which returns `null`. Consequently, calling the `length` method on a `null` object (`str`) results in a `NullPointerException`.

To fix this issue, the `appendFixedWidthPadLeft` method should handle the case when the `obj` argument is `null` and provide a proper value for `str` to avoid the `NullPointerException`.

Accurate, Concise, Useful

(...)
`str` にアクセスしようとした 1186 行目で `NullPointerException` が発生しました。これは、メソッドに渡された `obj` 引数が `NULL` であるため、`getNullText` メソッドが呼び出され、`NULL` が返されたことを示しています。その結果、`null` オブジェクト (`str`) に対して `length` メソッドを呼び出すと、`NullPointerException` が発生します。

この問題を解決するには、`appendFixedWidthPadLeft` メソッドは、`obj` 引数が `NULL` の場合を処理し、`NullPointerException` を回避するために `str` に適切な値を提供する必要があります。

一部不正確な説明

(c) Defects4J Math-30 Run#2

(...) The test `testBigDataSet` in the class `MannWhitneyUTest` fails when asserting that the calculated Mann-Whitney U statistic (`result`) is larger than 0.1. The failure occurs in the line 113 of the test case. Upon investigation, the failure seems to be caused by incorrect calculations in the methods `mannWhitneyU` and `calculateAsymptoticPValue`.

To fix the issue, we suggest the following: (...)

- Confirm the correctness of the calculations for EU (expected value) and VarU (variance) of the Mann-Whitney U statistic.
- Ensure that the computation of the standard score `z` is correct based on the observed `Umin`, `EU`, and `VarU`. (...)

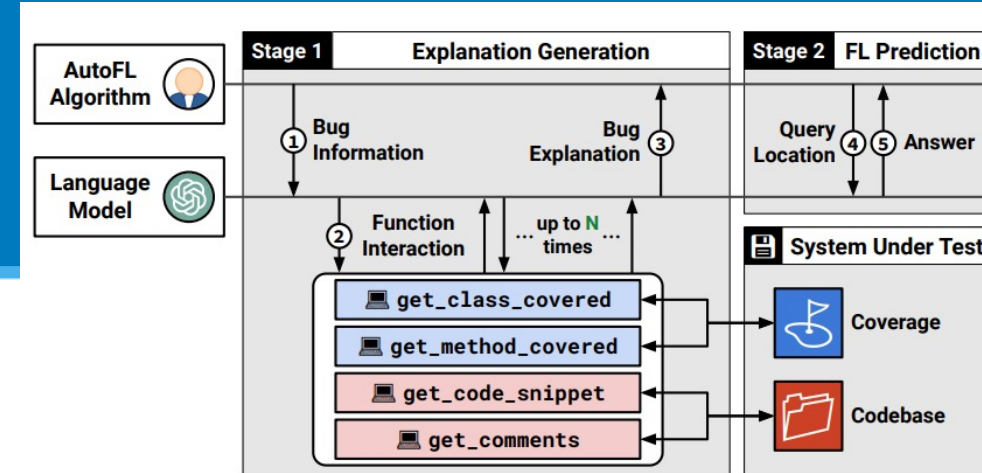
Accurate, but also Imprecise

(...)
`MannWhitneyUTest` クラスの `testBigDataSet` は、計算された Mann-Whitney U 統計量 (`result`) が 0.1 より大きいことを表明する際に失敗します。失敗はテスト・ケースの 113 行目で発生します。調べてみると、この失敗は、`mannWhitneyU` と `calculateAsymptoticPValue` メソッドでの間違っ
 た計算が原因のようです。この問題を修正するため
 に、以下のことを提案します: (...)

- Mann-Whitney U 統計量の EU (期待値) と VarU (分散) の計算が正しいことを確認する。
- 標準得点 `z` の計算が、観測された `Umin`、`EU`、`VarU` に基づいて正しいことを確認する。(...)

緑 : 不具合の発生原因
 青 : 解決方法
 赤 : 不正確な解決方法

Stage2 | 位置の特定



- ④予測と⑤結果

- Stage1で収集した情報に基づいて、以下のプロンプトを用いて不具合の位置を一つ以上予測

Listing 3. Prompt to Request the Fault Location

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).

利用可能な情報に基づいて、バグの原因である可能性が最も高いメソッドのシグネチャを提供してください。あなたの回答は自動的に処理されますので、必ず、最も可能性の高い原因メソッドの正確なシグネチャ（`ClassName.MethodName(ArgType1, ArgType2, ...)` 形式）だけを、コメントなしで（1行に1つ）記入してください。

不具合位置の最終決定

- AutoFLでは複数回の実行結果を統合して、最終的な結果を出力
 - AutoFLを R 回繰り返し実行した結果を組み合わせて、各メソッドにスコアを割り当てる
 - 予測に合計 n 個のメソッドが含まれる場合、これらのメソッドにそれぞれ $1/n$ のスコアを与える
 - メソッド m のスコアは以下の式で表される

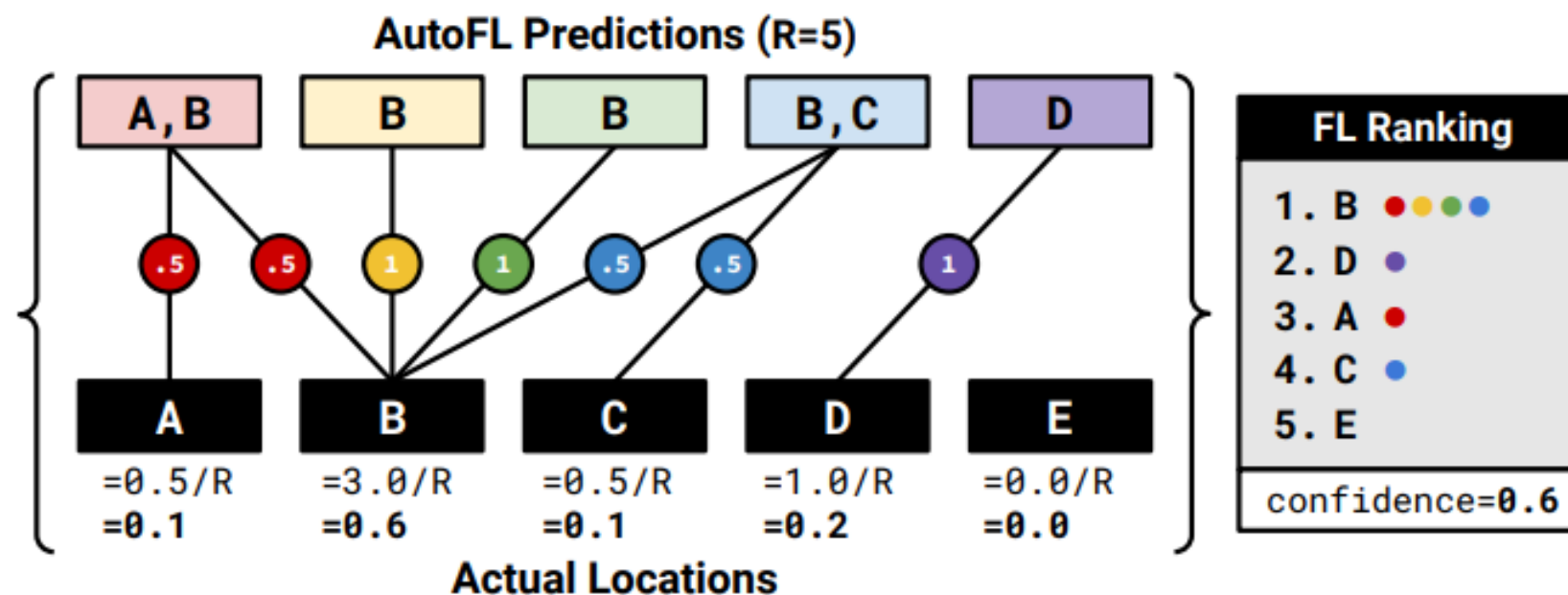
$$\text{score}(m) = \frac{1}{R} \sum_{k=1}^R \left(\frac{1}{|r_k|} \cdot [m \in r_k] \right)$$

r_i : i 番目の実行で予測されたメソッドの集合
[$\cdot$] : 中の述語が真ならば1, そうでなければ0

- すべてのメソッドでスコアを計算後、降順にランキング
- スコアが0でテストに含まれるメソッドはランク付けされたりストの最後に追加

不具合位置の最終決定

- 例：Rは5, メソッドBのスコアを求める
 - 1回目はA, Bを予測, それぞれ0.5のスコアを割り当て
 - 2回目はBを予測, 1のスコアを割り当て
 - メソッドBのスコアは：
 $(0.5 + 1 + 1 + 0.5 + 0) / 5 = 0.6$

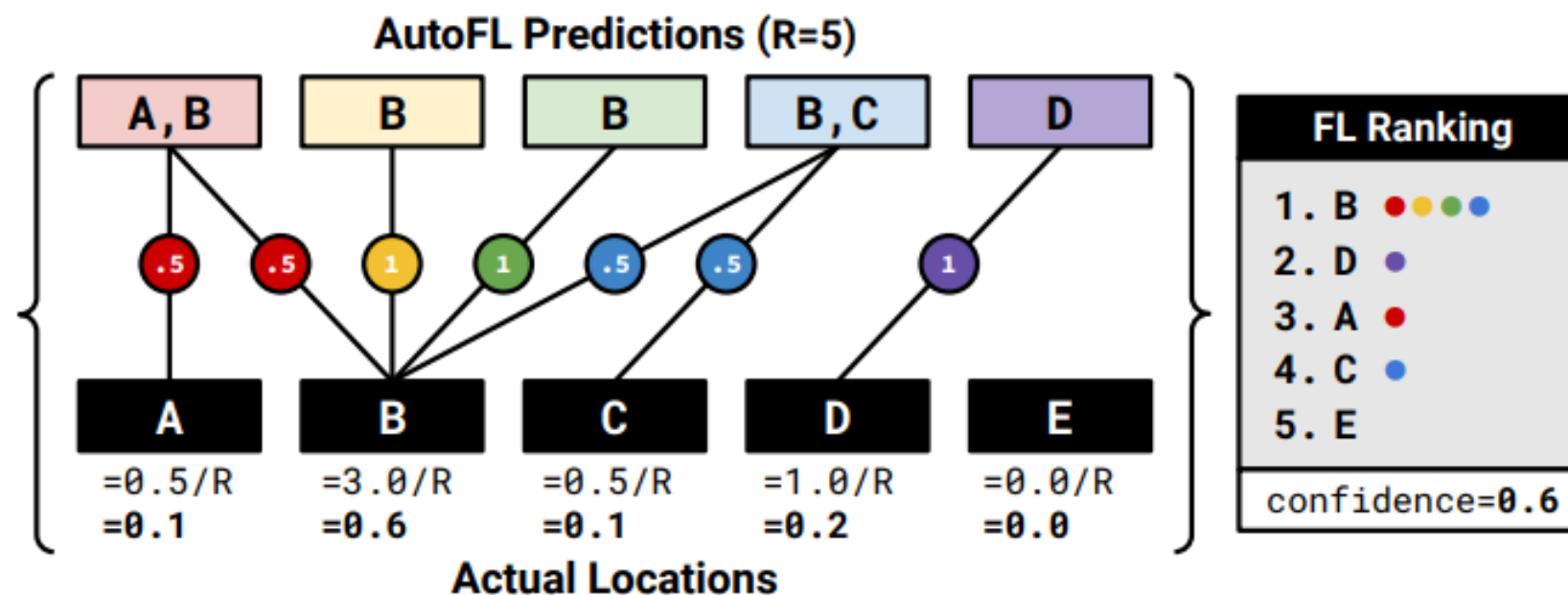


不具合位置の最終決定

- 信頼度

- ランク付けリストを作成後，予測の信頼度を推定する
- 最も高いスコアを，信頼度とする
 - AutoFLが一貫して同じ予測をすると信頼度が高くなる

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



リサーチクエスチョン

- RQ1：AutoFLの精度
 - SBFL, MBFL手法よりも精度が同等かそれ以上
- RQ2：AutoFLの信頼度とFL性能の関係
 - 信頼度とFL性能には正の相関がある
- RQ3：AutoFLの説明の質
 - 不具合の原因を正確に説明しているものは約20%
- RQ4：開発者からのフィードバック
 - AutoFLの説明はエラーメッセージや各関数の意図を説明することで役に立っていると示唆
 - 一部の開発者は説明が冗長であると感じた

まとめと所感

- まとめ

- LLMを用いた新しいFL技術であるAutoFLの提案
- プロンプトを活用し説明の生成と不具合の位置特定を行う
- LLMを用いることで、既存手法よりも高い精度で不具合の位置を特定することが可能
- AutoFLが生成する説明は、概ね開発者に好意的に受け止められた

- 所感

- ハルシネーションがあるので、生成された説明を本当に信用していいのかが難しいところ
- どこまでを自動で行なっているのかがよくわからん