Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

源代码补丁正确性测试

博士研究生 段学明

2025年03月09日

问题回溯

- 总结反思
  - 部分词汇表达不严谨
  - 理论层面对方法存在的问题分析欠缺

- 相关内容
  - 2024.08.11 张钊：《自动化程序缺陷修复及其应用研究》
  - 2021.12.19 于浩淼：《软件缺陷自动修复方法》

内容提要

- 预期收获
- 内容引入
- 内涵解析与研究目标
- 研究背景与研究意义
- 研究历史与现状
- 知识基础
- 算法原理
  - APOSTLE
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 1. 了解源代码补丁正确性测试的基本概念和研究方向
  - 2. 理解补丁正确性测试在自动程序修复流程中的必要性
  - 3. 了解源代码补丁正确性测试的前沿方法和未来发展

内容引入

- 自动程序修复（Automated Program Repair, APR）旨在自动修复软件错误，在软件开发和维护中发挥着重要作用
- 随着深度学习技术的不断进步，越来越多的APR技术被提出，利用神经网络从海量开源代码库中学习错误修复模式
- APR生成的补丁仍然存在过度拟合的问题，这对实际应用构成了极大的威胁
  - 有效性≠正确性
  - 需要能够预测补丁正确性的方法

有效性和正确性测试均是APR流程中的环节

```
  public void draw(...) {
+   if (true) return ;
    ...
```

(a) An incorrect patch produced by jKali [34]

```
public void testDrawWithNullDataset() { ...
  JFreeChart chart = ChartFactory.
    createPieChart3D("Test", null,...);
  try {...
    chart.draw(...);
    success = true; }
  catch (Exception e) {
    success = false; }
  assertTrue(success);
}
```

(b) A failing test checking for a null dataset

```
public void testNullValueInDataset() { ...
  dataset.setValue(..., null);
  JFreeChart chart = createPieChart3D(dataset);
  try {...
    chart.draw(...);
    success = true; }
  catch (Exception e) {
    success = false; }
  assertTrue(success);
}
```

(c) A passing test checking for a null value in a dataset

内涵解析与研究目标

- 研究目标
  - 在APR流程中，验证所生成的补丁是否真正解决了目标缺陷且未引入新的问题，即解决APR生成补丁存在的过拟合问题
- 内涵解析
  - APR流程：缺陷定位、生成补丁、补丁排序过滤及验证
  - 过拟合问题
    - 补丁可能符合有限测试套件的要求，但在测试套件之外的表现却与开发人员的初衷不同
    - 过度拟合补丁可能会解决测试用例中的特定错误，但会在应用程序的不同部分引入新的错误，从而导致不可靠的软件行为

背景意义

- 研究背景
  - APR技术的广泛研究和应用
  - APR生成的补丁经有效性验证后仍存在不可用隐患，对实际应用构成极大的威胁
  - 补丁正确性测试概念被提出，以研究和应对以上隐患
- 研究意义
  - 提高软件质量
    - 通过准确预测由APR生成的补丁的正确性，可以在不引入新缺陷的情况下有效解决软件缺陷问题，从而提高软件质量
  - 降低维护成本
    - 提前准确预测补丁的正确性，尽量减少错误补丁的使用，可以有效地大幅降低整体维护成本
  - 提高开发效率
    - 短时间内评估和验证多个代码补丁

研究历史

基于静态

Tan等人通过考虑代码删除和程序转换等静态代码特征，优先选择正确的补丁，而不是不合适的补丁。与以往“直接给定修复模板”不同，提出了一组“反模式（anti-patterns）”来约束修复搜索过程，从而提高补丁质量。

基于动态

Xiong等人认为，原始程序和修补程序的通过测试可能表现相似，而原始程序和补丁程序的失败测试可能表现不同。基于这一观点，提出PATCH-SIM和TEST-SIM，使用执行轨迹来近似估计补丁程序的正确性，即使没有标准的正确性信息。

基于学习

Lin等人探索了考虑程序结构的上下文感知代码更改嵌入的想法，用于补丁正确性评估。使用抽象语法树（AST）路径来表示补丁，并构建了一个深度学习分类器来预测补丁的正确性。

2016 2017 2018 2020 2022 2022

基于动态

Xin等人提出DiffTGen，通过首先生成新的测试输入来识别过度拟合的修补程序，这些输入揭示了原始故障程序和修补程序之间的语义差异，然后根据语义差异测试修补程序，最后生成测试用例。

基于学习

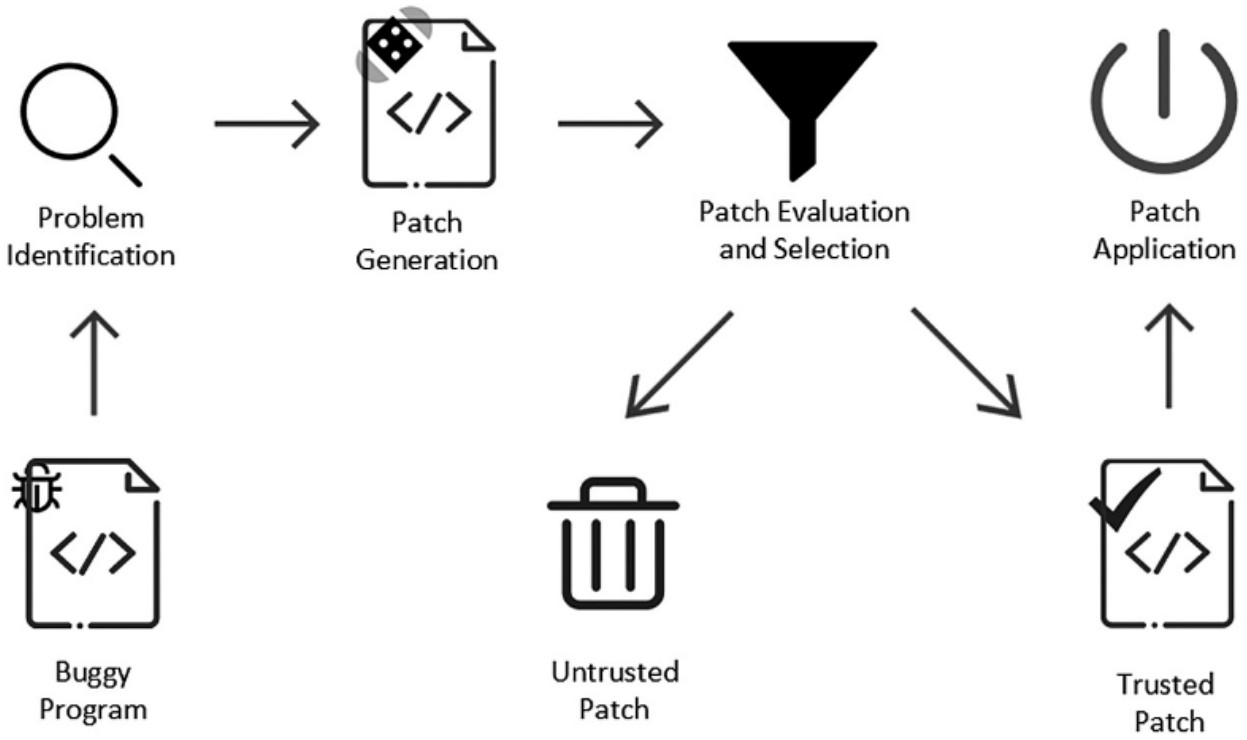
Tian等人使用基于深度学习的补丁嵌入进行静态检查。研究了通过学习代码表示来学习编码补丁正确性属性的深度特征的好处以及代码变化的不同表示学习方法，以获得适合相似性计算的嵌入，证明与依赖动态信息的最先进PATCH-SIM相比，学习表示可以带来合理的性能。

基于学习

Tian等人提出一种基于无监督学习的方法BATS，通过根据失败的测试规范检查补丁行为来预测补丁的正确性。利用代码和补丁的深度表示学习模型：对于给定的失败测试用例，生成的嵌入用于在搜索历史相似测试用例时计算相似性度量，以识别相关的应用补丁，然后将其用作评估APR生成补丁正确性的代理。

研究现状

- 补丁正确性研究的重要性显现
  - APR一般步骤
    - 缺陷定位
    - 补丁生成
    - 补丁排序过滤及验证：
      - 有效性（可信补丁）
      - 正确性
      - 可读性
      - 自然性



- 基于静态和动态的方法劣势明显，随着基于学习模型的代码表征技术发展，基于学习的补丁正确性测试方法研究广泛

知识基础 有效性与正确性

- 补丁验证（有效性）
  - 目的：输出可信补丁以供部署
  - 标准：
    - 能使错误程序通过的测试套件在打了补丁的程序上仍能通过
    - 在错误程序上失败的故障触发测试套件在打了补丁的程序上也能通过
  - 常用方法：基于测试的验证策略，针对每个候选补丁执行可用的测试套件，可过滤掉无法编译或无法通过可用测试套件的候选补丁
  - 缺点：耗时；测试套件是一种不完整的规范，因为它只描述了程序行为空间的一部分，现有测试套件的可信补丁可能无法泛化到其他潜在测试套件
- 补丁正确性测试

  条件：输入为可信补丁
  - 目的：在补丁验证后进一步过滤过拟合补丁，提高返回补丁的质量
  - 核心思想：相似性（基于静态的方法、基于学习的方法）

知识基础 补丁和测试用例

- 补丁（patch）
  - 定义：指对源代码进行修改以修复软件缺陷的代码片段，包括代码变更、修复描述
  - 代码变更：对源代码具体修改，如增加新代码行、删除错误代码行、修改现有代码
- 测试用例（test case）
  - 可以触发并验证程序缺陷的代码片段

删除

增加

修复描述

```
@@ -421,7 +421,7 @@ public class TestFormAuthenticator
extends TomcatBaseTest {
-   private class FormAuthClientBase extends
SimpleHttpClient {
+   private abstract class FormAuthClientBase extends
SimpleHttpClient {
        protected static final String LOGIN_PARAM_TAG =
        "action=";
        protected static final String LOGIN_RESOURCE =
        "j_security_check";
```

Commit Message:
Make base class **abstract**

```
  public void draw(...) {
+   if (true) return ;
    ...
```

(a) An incorrect patch produced by jKali [34]

```
public void testDrawWithNullDataset() { ...
  JFreeChart chart = ChartFactory.
    createPieChart3D("Test", null,...);
  try {...
    chart.draw(...);
    success = true; }
  catch (Exception e) {
    success = false; }
  assertTrue(success);
}
```

(b) A failing test checking for a null dataset

算法原理 APOSTLE

Automated patch correctness predicting to fix software defect

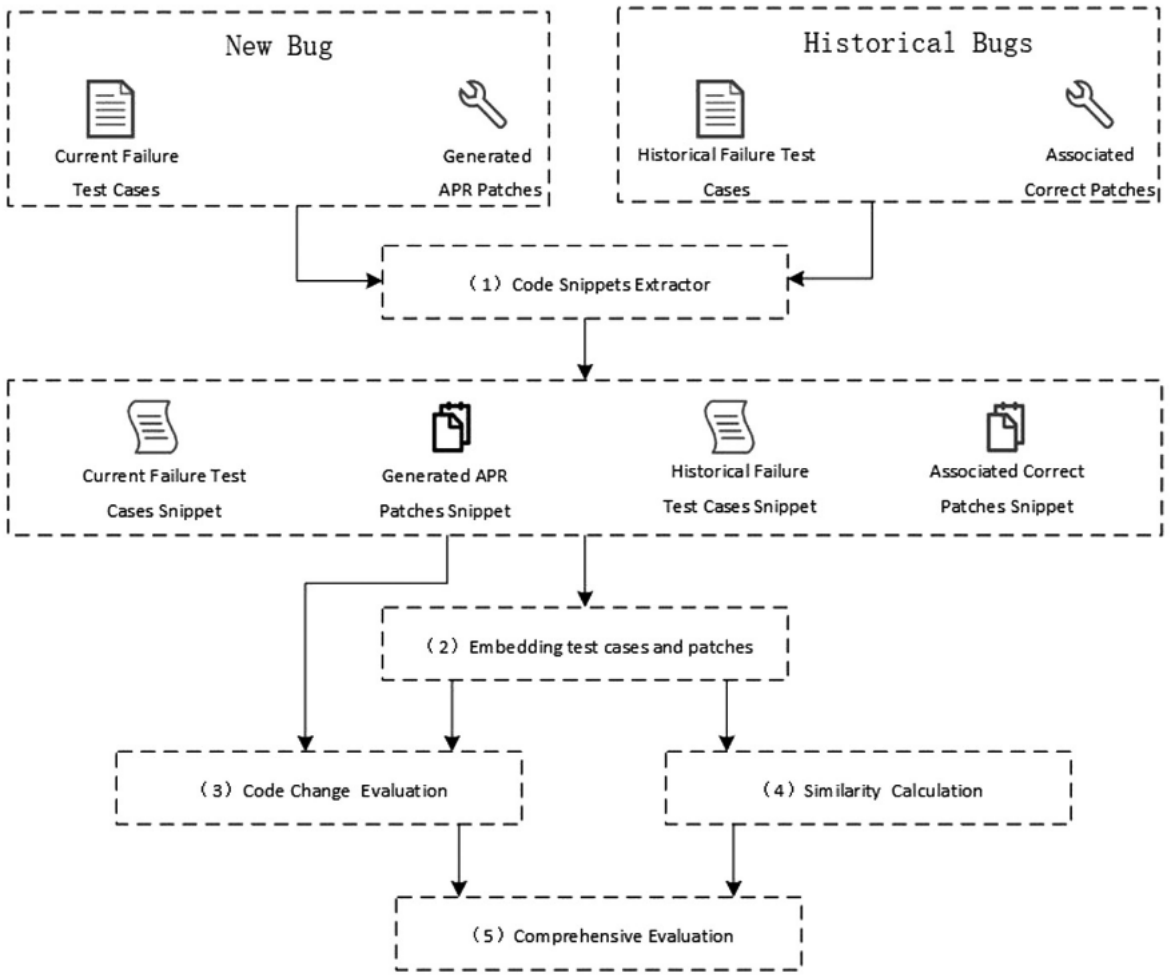
算法原理 TIPO

T	目标	预测APR生成补丁的正确性
I	输入	历史失败测试用例、相关的历史正确补丁、当前失败测试用例、APR生成的当前补丁
P	处理	1. 代码向量化 2. 相似度和代码变化程度计算 3. 综合评估
O	输出	APR生成的当前补丁正确性（是/否）、补丁依正确性概率的排序

P	问题	基于静态的方法预测性能低，基于动态的方法预测成本高，基于学习的方法在代码向量化时难以提取高级语义，且现有方法预测标准相对单一，缺乏多角度预测
C	条件	需存在历史失败测试用例及相应的历史正确补丁；当前补丁是有效的（可信补丁）
D	难点	如何综合多角度建立补丁正确性测试体系
L	水平	ESWA2024（SCI一区）

算法原理

- 算法原理图
  - 代码向量化
    - UniXcoder用于历史和当前失败测试用例
    - CodeBERT用于历史和当前补丁
  - 相似度和代码变化度计算
    - 计算当前补丁与相关历史补丁之间的语义相似度和变化程度，从先前正确补丁的经验中为预测提供指导
  - 综合评估
    - 从相似度、代码变化度两个角度进行综合评估



算法原理 代码向量化

- 模块概述
  - 输入：测试用例、补丁
  - 操作：
    - 对于测试用例，将单个用例的源代码视为token序列
    - 对于补丁，将补丁代码中添加和删除的代码行视为token序列
  - 输出：测试用例和补丁的向量表示

```
1  --- a/source/org/jfree/chart/renderer/category/AbstractCategoryItemRenderer.java
2  +++ b/source/org/jfree/chart/renderer/category/AbstractCategoryItemRenderer.java
3  @@ -1794,7 +1794,7 @@ public abstract class AbstractCategoryItemRenderer extends AbstractRenderer
4           }
5           int index = this.plot.getIndexOf(this);
6           CategoryDataset dataset = this.plot.getDataset(index);
7  -        if (dataset != null) {
8  +        if (dataset == null) {
9               return result;
10           }
11          int seriesCount = dataset.getRowCount();
12
13 #this is the buggy part
14      if ( dataset ! = null ) {
15 #this is the patched part
16      if ( dataset = = null ) {
```

- 向量化模型
  - UniXcoder、CodeBERT

整个补丁的向量如何表示?

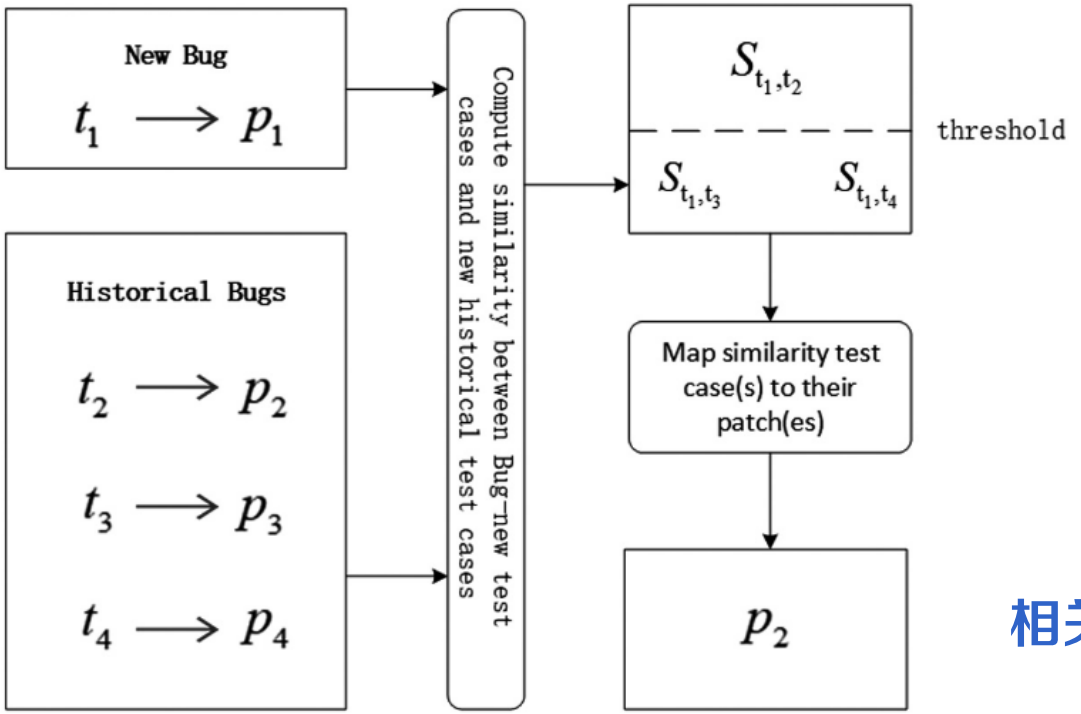
- 原理：预训练模型将每个token映射到一个数字向量，同时保持相似标记之间的语义相似性，这构成了后续相似性计算的基础

算法原理 代码向量化

- 向量化模型选择原因
  - 测试用例： UniXcoder
    - 一个统一的跨模态预训练编程语言模型
    - 该模型利用带有前缀适配器的掩码注意矩阵来控制模型的行为，并利用 AST 和代码注释等跨模态内容来增强代码表示
    - 猜想：对代码片段有很强的特征提取能力
  - 补丁： CodeBERT
    - 是一种用于编程语言（PL）和自然语言（NL）的双模预训练模型
    - 可学习通用表征，支持下游的 NL-PL 应用，如自然语言代码搜索、代码文档生成等
    - 猜想：不连续的代码行代码特征更少，相对与自然语言更为接近

算法原理　相似度和代码变化程度

- 动机
  - 如果触发的测试用例相似，修复代码的补丁也应相似
  - 代码变化量大的补丁很可能是不正确的
  - 因此，决定使用相似度和代码变化程度来评估补丁的正确性
- 相似度：指当前APR生成的补丁与相关历史补丁间的相似度



相似的测试用例

相关历史补丁

算法原理　相似度和代码变化程度

- 计算相似度
  - 计算相似的测试用例

$$S_{t_1,t_2} = d_{t_1,t_2}/(1 + d_{t_1,t_2})$$

  - 问题：一个测试用例可能对应多个相似的历史测试用例，导致一个补丁对应多个相关历史补丁
  - 解决：平衡相关历史补丁

$$p = \frac{1}{n}\sum_{i=1}^{n}(p_i \times s_{t_i})$$

其中，p_i表示第i个测试用例对应的补丁的向量形式，s_{t_i}表示第i个测试用例与当前测试用例之间的相似度，n表示补丁的数量

  - 计算相关历史补丁与当前补丁之间的相似度

$$S_{p_1,p_2} = d_{p_1,p_2}/(1 + d_{p_1,p_2})$$

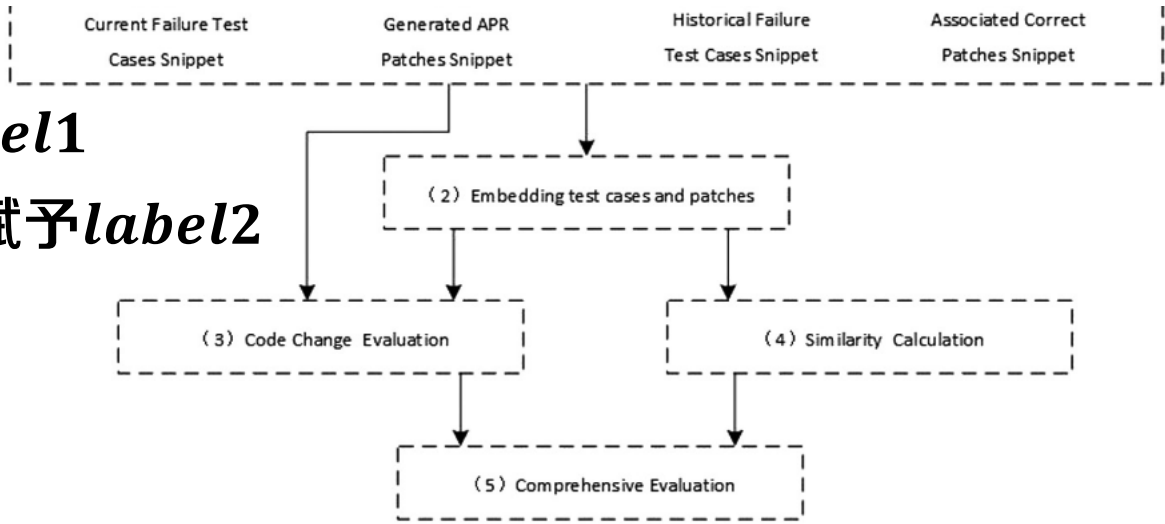
算法原理　相似度和代码变化程度

- 计算代码变化程度
  - 2个指标：代码变化量、代码语义变化程度
  - 计算当前代码变化量（非语义），与阈值进行比较，若高于阈值，则进行标记
  - 代码语义变化程度的计算与之相似

$$a = d_{patched,buggy}/(1 + d_{patched,buggy})$$

其中，$d_{patched,buggy}$表示打补丁部分和出错部分向量之间的欧氏距离

- 流程梳理
  - 检测补丁的代码变化量，决定是否赋予$label1$
  - 检测补丁的代码语义变化程度，决定是否赋予$label2$
  - 计算该补丁与相关历史补丁之间的相似度



算法原理　综合评估

- 实现方案
  - 预期降低系数：
    - 为代码的过度更改和代码语义的过度更改设定
    - 该系数表示代码变化过度时补丁正确性的预期下降
    - 仅当代码改动过大时，才会参与综合评估
  - 综合评估阈值
    - 综合评估得分高于这个阈值，则预测APR生成的补丁是正确的，否则错误
- 综合评估得分与其他指标的关系

```
result = s
if label1:
    result *= coefficient1
if label2:
    result *= coefficient2
if result > 0.5:
    return true
else:
    return false
```

实验设计

- 数据资源
  - 数据集：领域权威Defects4J数据集、个别领域研究提供的数据集
  - 语言：领域常用目标语言Java
- 基线
  - 基于无监督学习的方法：BATS (2022)
  - 基于动态的方法：PATCH-SIM (2018)
  - 基于深度学习的方法：Tian et al. (2020)
- 指标
  - Recall、F1-score、AUC
  - $+Recall = \frac{TP}{TP+FN}$，$-Recall = \frac{TN}{TN+FP}$
  - 平均精确度$MAP = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{j=1}^{m}(P_{ij}\times Rel_{ij})}{\#\ correct\ patches}$，$MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{rank_i}$

Rel_{ij}是指示函数，若列表i中第j个补丁正确则为1，P_{ij}是列表i中阈值j的精确度

实验设计　对比实验

- 实验结果

Effectiveness of APOSTLE compared with BATS.

Model	Test case embedding	Patch embedding	*AUC*	*F1-score*	*MAP*	*MRR*
BATS	code2vec	CC2Vec	0.718	0.722	0.798	0.806
	code2vec	BERT	0.676	0.626	0.770	0.854
APOSTLE	**UniXcoder**	**CodeBERT**	**0.801**	**0.730**	**0.858**	**0.944**

Qualitative analysis results.

Prediction	Reference	
	Positive	Negative
Positive	TP(37)	FN(27)
Negative	FP(7)	TN(46)

- 分析
  - F1分数没有明显差异：数据集不平衡导致
  - 方法对阳性样本的识别能力相对较弱

实验设计　对比实验

- 实验结果

Comparison with a state-of-the-art dynamic-based patch assessment.

Method	*AUC*	*F1-score*	*+Recall*	*-Recall*
APOSTLE	**0.801**	**0.730**	**0.868**	**0.578**
PATCH-SIM	0.634	0.597	0.952	0.316

Comparison with a state-of-the-art supervised classifier.

Method	*AUC*	*F1-score*	*+Recall*	*-Recall*
APOSTLE	**0.801**	**0.730**	**0.868**	**0.578**
Tian et al.(LR[a])	0.668	0.592	0.698	0.453
Tian et al.(RF[b])	0.758	0.635	0.755	0.484

[a] LR refers to Logistic Regression.

[b] RF refers to Random Forest.

- 分析
  - PATCH-SIM通过生成额外测试用例，并比较执行轨迹的相似度来验证补丁正确性，额外测试用例数量有限，很多错误补丁并没有暴露问题，导致更多 “正确”判断
  - 更高的+Recall必然意味着对补丁看似正确的倾向性，在真正的错误补丁也易误认为正确

实验设计 消融实验

- 消融实验

The result of APOSTLE based on different pre-trained models.

Test case embedding	Patch embedding	*AUC*	*F1-score*	*+Recall*	*-Recall*	*MAP*	*MRR*
code2vec	CodeBERT	0.757	0.718	0.825	0.526	0.921	1.000
	UniXcoder	0.782	0.667	0.649	0.702	0.817	0.861
	GraphCode-BERT	0.783	0.750	0.789	0.684	0.852	0.903
CodeBERT	CodeBERT	0.646	0.596	0.635	0.578	0.692	0.700
	UniXcoder	0.596	0.457	0.387	0.737	0.644	0.623
	GraphCode-BERT	0.604	0.521	0.486	0.676	0.653	0.645
UniXcoder	CodeBERT	**0.801**	**0.730**	**0.868**	**0.578**	**0.858**	**0.944**
	UniXcoder	**0.824**	**0.721**	**0.755**	**0.719**	**0.817**	**0.870**
	GraphCode-BERT	**0.805**	**0.763**	**0.849**	**0.688**	**0.818**	**0.852**
GraphCode BERT	CodeBERT	0.568	0.434	0.368	0.720	0.637	0.637
	UniXcoder	0.373	0.084	0.051	0.859	0.508	0.467
	GraphCode-BERT	0.413	0.266	0.202	0.729	0.543	0.523

- 参数实验、计算补丁相似方法实验

特点总结与未来展望

算法总结

- 算法创新
  - 选取合适的预训练模型分别向量化测试用例和补丁
  - 设计相似度和代码变化程度计算方法
  - 考虑多种相似性结果进行综合评估（多阈值、多系数）
- 算法优劣
  - 优势：无监督、速度快、成本低
  - 劣势：无法预测所有补丁（没有使用与当前测试用例相似度低于阈值的历史测试用例）、各类阈值系数需按照数据集做最优标定、可解释性不强（语义变化程度）
- 未来工作
  - 利用其他代码表征方式（如抽象语法树）更好地捕捉补丁程序的语义信息
  - 探索可能影响补丁正确性预测的其他因素，如补丁程序的复杂性、可读性和可维护性

参考文献

[1] Zheng Z, Wang R, Tao Z, et al. Automated patch correctness predicting to fix software defect[J]. Expert Systems with Applications, 2024, 256: 124877.

[2] Zhang Q, Fang C, Ma Y, et al. A survey of learning-based automated program repair[J]. ACM Transactions on Software Engineering and Methodology, 2023, 33(2): 1-69.

道德经

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！