计算机科学
COMPUTER SCIENCE

基于CodeBERT和Stacking集成学习的补丁正确性验证方法

韩威, 姜淑娟, 周伟

引用本文

韩威, 姜淑娟, 周伟. 基于CodeBERT和Stacking集成学习的补丁正确性验证方法[J]. 计算机科学, 2025, 52(1): 250-258.

HAN Wei, JIANG Shujuan, ZHOU Wei. Patch Correctness Verification Method Based on CodeBERT and Stacking Ensemble Learning [J]. Computer Science, 2025, 52(1): 250-258.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240100019

基于 CodeBERT 和 Stacking 集成学习的补丁正确性验证方法

韩 威 姜淑娟 周 伟
中国矿业大学计算机科学与技术学院 江苏 徐州 221116
中国矿业大学矿山数字化教育部工程研究中心 江苏 徐州 221116
(ts21170066p31@cumt.edu.cn)

摘 要 近年来,自动程序修复已成为软件工程领域的重要研究课题。然而,现有的自动修复技术大多是基于补丁生成和测试的,在补丁验证环节时间成本很高。此外,由于测试套件的不完备,许多候选补丁虽然能通过测试,但实际上并不正确,从而导致补丁过拟合。为提高补丁验证的效率并缓解补丁过拟合的问题,提出了一种静态的补丁验证方法。该方法首先使用大型预训练模型 CodeBERT 自动提取缺陷代码片段和补丁代码片段的语义特征,然后使用历史缺陷修复补丁数据训练 Stacking 集成学习模型,训练之后的模型可以对新的缺陷修复补丁进行有效验证。在 Defects4J 缺陷数据集相关的 1 000 个补丁数据上对所提方法的验证能力进行评估。实验结果表明,该方法可以有效地验证补丁的正确性,从而提高补丁验证的效率。

关键词:自动程序修复;补丁验证;预训练模型;集成学习;Defects4J 缺陷数据集
中图分类号 TP311

Patch Correctness Verification Method Based on CodeBERT and Stacking Ensemble Learning

HAN Wei,JIANG Shujuan and ZHOU Wei
School of Computer Science and Technology,China University of Mining and Technology,Xuzhou,Jiangsu 221116,China
Engineering Research Center of Mine Digitalization of Ministry of Education,China University of Mining and Technology,Xuzhou,Jiangsu 221116,China

Abstract In recent years,automatic program repair has become an important research topics in the field of software engineering. However,most of the existing automatic repair technologies are based on patch generation and testing,which consumes a significant amount of time and cost in the patch verification process. In addition,because the test suite is not completeness,many candidate patches can pass the test,but the test results are not consistent with the facts,which leads to the patch overfitting problem. To improve the efficiency of patch verification and alleviate patch overfitting issues,a static patch verification method is proposed. The method first uses the large pre-training model CodeBERT to automatically extract the semantic features of defect code fragments and patch code fragments,and then uses the historical defect repair patch data to train a Stacking ensemble learning model. The trained model can effectively verify the new defect repair patch. The verification ability of the proposed method is evaluated on the 1 000 patch data related to the Defects4J defect dataset. Experimental results show that the static patch verification method can effectively verify the correctness of the patch,thereby improving the efficiency of patch verification.

Keywords Automatic program repair,Patch verification,Pre-training model,Ensemble learning,Defects4J defect dataset

1 引言

近年来,自动程序修复(Automatic Program Repair,APR)已经发展成为软件工程领域的一个重要研究课题,大量的自动修复工具[1-3]应运而生,旨在减少软件维护成本,提高软件安全性。如今的自动修复技术大多基于生成和验证,即首先利用错误定位工具对错误所在位置进行定位,然后使用修复技术对错误进行修复,得到候选补丁,最后使用测试套件对所有候选补丁进行正确性验证。如果候选补丁通过所有的测试,则该补丁被视为正确补丁[4]。然而,由于现有测试套件的不完备,许多候选补丁虽然可以通过测试套件,但实际上并不正确,甚至会引入更多的错误,这种现象被称为补丁过拟合。过拟合问题已经成为自动程序修复的关键性挑战之一。

为解决补丁过拟合问题,提高补丁的可靠性,许多研究人员正在探索使用有效的技术和方法来自动识别补丁的正确性。目前,已经有各种补丁正确性评估技术被提出,用于判定生成的补丁是否正确。根据补丁验证过程中是否需要执行测试用例,补丁验证方法分为静态的补丁验证方法和动态的

到稿日期:2024-01-02 返修日期:2024-06-04
基金项目:国家自然科学基金(61673384)
This work was supported by the National Natural Science Foundation of China(61673384).
通信作者:姜淑娟(shjjiang@cumt.edu.cn)

补丁验证方法。静态的补丁验证方法，如 Csuvik 等[5]通过假设正确的补丁比不正确的补丁修改更小，依次对候选补丁进行排名。而动态的补丁验证技术通常通过测试生成工具（如 Evosuite[6]和 Randoop[7]等）生成额外的用例对现有测试套件进行补充之后执行合理补丁。例如，Xiong 等[8]利用工具生成新的测试用例，并根据补丁在测试用例上执行的行为相似性确定补丁的正确性。相比之下，静态的补丁验证技术通常精度不够；而动态的补丁验证技术虽然精度很高，但每次都需要运行测试套件，非常耗时。另外，一些研究人员将重点放在基于学习的静态补丁验证技术上。例如，Lin 等[9]同时考虑更改的代码片段和相关的未更改代码片段，然后使用抽象语法树（Abstract Syntax Tree，AST）路径表示技术对其进行表示，从而捕获代码的结构信息，并对补丁进行评估。Ye 等[10]在补丁的 AST 级别提取 202 个代码特征，然后使用监督学习自动学习一个补丁验证概率模型。然而，无论是提取 AST 路径表示还是手动设计代码特征，工作量都会很大，成本较高。

为解决上述问题，本文提出一种静态的补丁验证方法。首先，使用大型预训练模型 CodeBERT 作为编码器，对补丁代码片段和缺陷代码片段进行表征，从而得到二者的特征向量；然后，建立一个 Stacking 模型集成多个类型的分类算法，通过将特征向量和对应的标签输入 Stacking 模型中，可以使模型更好地理解哪些特征对补丁的正确性更有利，从而学习补丁过拟合和正确性的概率；最后，训练完成的集成学习模型可以更好地对新的补丁数据进行验证。

本文的主要贡献是提出了一种结合 CodeBERT 和 Stacking 集成策略来验证补丁正确性的方法。与 BERT 等在自然语言语料上训练的模型相比，CodeBERT 在大量的代码语料上训练，可以更好地表示代码数据。与使用单个分类模型相比，本文在 Stacking 中集成了可以减小方差的基于 Bagging 集成的分类器、可以减小偏差的基于 Boosting 集成的分类器和拟合能力更好的基于深度学习的分类器。在自动程序修复工具产生的真实修复补丁数据集上进行五折交叉验证，结果证明了所提方法在缺陷修复补丁验证方面的准确性和效率总体上都得到了提高。

2　基础知识

所提方法主要基于预训练模型 CodeBERT 和 Stacking 集成学习算法，下文就相关概念和基本知识予以介绍。

2.1　CodeBERT

CodeBERT 是 Feng 等[11]在 2020 年提出的一种面向编程语言（Programming Language，PL）和自然语言（Natural Language，NL）的双峰预训练模型，其基于多层双向的 Transformer 架构[12]。CodeBERT 是在大规模数据集 CodeSearchNet[13]上进行预训练的，该数据集包括 2.1M 双峰 NL-PL 对数据和 6.4 M 单峰代码数据。其中代码数据基于 6 种编程语言（Go，Java，Javascript，PHP，Python 和 Ruby），模型设置类似于 MultiBERT[14]，不同之处是数据输入没有标记显示以指示输入数据使用的语言。

为了利用大规模双峰和单峰数据，CodeBERT 提出了一种结合掩模语言模型（Masked Language Modeling，MLM）和替换令牌检测（Replaced Token Detection，RTD）的混合目标损失函数。MLM 预训练任务使用双峰数据，即给定一个 NL-PL 对的数据点（$x=\{w,c\}$）作为输入，其中 w 是一个 NL 单词序列，c 是一个 PL 标记序列，随机为 NL 和 PL 选择一组位置，将该位置的序列替换为特殊的掩码令牌[mask]。MLM 的目标是预测被屏蔽的原始标记。RTD 预训练任务使用双峰和单峰数据，首先分别使用单峰 NL 和 PL 训练数据生成器来恢复随机被屏蔽的令牌，然后用 CodeBERT 作为判别器来判断该令牌是否为原始被屏蔽的令牌，这是一个二元分类问题。预训练后的 CodeBERT 模型可以捕获 NL 和 PL 之间的语义联系，在 NL-PL 理解任务和生成任务中都取得了不错的效果。

2.2　Stacking 集成学习

Stacking 是机器学习中的一种模型集成技术，由 Wolpert[15]于 1992 年提出。其基本思想是结合多个基础模型的优点，从而获得更好的综合性能。具体来说，Stacking 是将使用不同学习算法 $L_1,\cdots,L_N$ 生成的多个分类器组合在一个数据集 S 上的方法。该数据集由示例 $s_i=(x_i,y_i)$ 组成，即由特征向量（$\boldsymbol{x}_i$）和它们的类别（$\boldsymbol{y}_i$）组成的一对。在第一阶段，生成一组基本分类器 $C_1,C_2,\cdots,C_N$，其中 $C_i=L_i(S)$。在第二阶段，Stacking 学习一个元分类器，它将基本分类器的输出组合起来，进行最终预测。由于集成后模型具有平滑特性，因此集成后的模型在性能上通常优于任何一个用于集成的基模型[16]。Stacking 的架构如图 1 所示。

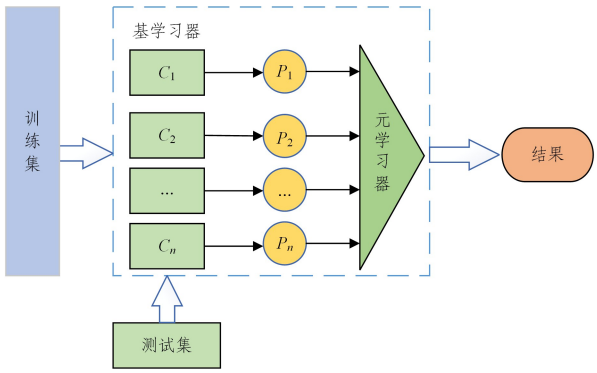

图 1　Stacking 架构

Fig. 1　Stacking architecture

为了生成用于学习元分类器的训练集，一般使用留一法（Leave-One-Out）或 k 折交叉验证过程。对于留一法，将每个基本学习算法应用于几乎整个数据集，只留出一个示例进行测试，即 $\forall i=1,\cdots,n;\forall k=1,\cdots,N:C_k^i=L_k(S-s_i)$。然后使用学习的分类器为 si 生成预测：$\hat{y}_i^k=C_k(x_i)$。元级数据集由形式为 $((\hat{y}_i^1,\cdots,\hat{y}_i^n),y_i)$ 的示例组成，其中特征是基本分类器的预测，类是该示例的正确类别。对于 k 折交叉验证过程，不是每次留出一个示例，而是留出原始数据集大小的 $1/k$ 子集，并在这些子集上获得学习到的基本分类器的预测。

3　本文方法

所提软件修复补丁正确性验证方法的整体框架如图 2 所示。该方法共分为 3 个阶段。1）数据预处理阶段。通过对收集的补丁数据集进行拆分和转换，使其符合模型可以

接收的数据形式。2)特征提取阶段。使用在代码语料上预训练的表示学习模型 CodeBERT 对补丁数据进行表征，并使用 4 种比较函数将表示进行组合。3)模型学习与分类。将组合之后的补丁训练集输入 Stacking 集成学习模型中进行训练，训练完成的模型可以有效地识别正确补丁和过拟合补丁。

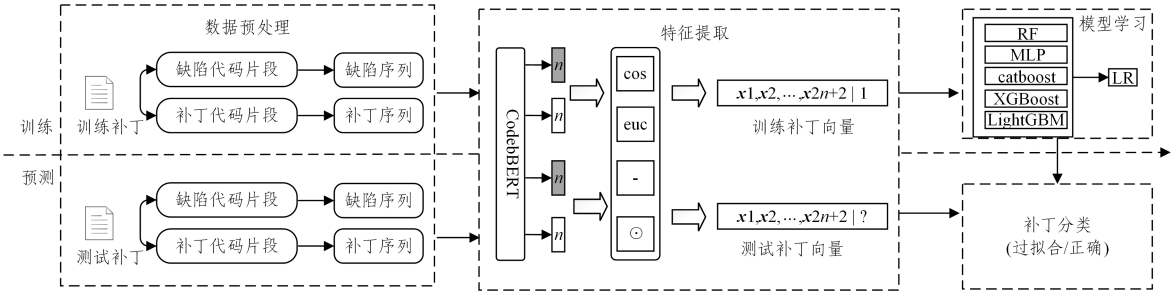

图 2 所提方法的总体框架

Fig. 2 Overall framework of the proposed method

3.1 数据预处理

由于本文方法在训练和验证时需要输入缺陷代码和其相应的正确补丁或错误补丁，从而判断补丁是否正确，因此将补丁样本拆分为打补丁之前的缺陷代码片段和打补丁之后的补丁代码片段，即缺陷-补丁代码对。同时，考虑到补丁常常只对很小部分的代码片段进行修改，对整个源代码文件进行嵌入时差异非常小，因此只考虑修改的代码片段。另外，为了适应嵌入模型，将多行代码扁平化为一行并输入 CodeBERT 模型中生成特征向量。本文数据集中的某个补丁样本如图 3 所示，这是 APR 工具 TBar[1] 为 Defects4j 缺陷数据集中 Chart-1 错误生成的修复补丁，其中“+”代表增加的代码行，“-”代表删除的代码行。缺陷代码片段由删除的行和其上下文组成，如图 4 所示。补丁代码片段由增加的行和同样的上下文组成，如图 5 所示。

```
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
4. extends AbstractRenderer
5.             }
6.             int index = this.plot.getIndexOf(this);
7.             CategoryDataset dataset = this.plot.getDataset(index);
8. -           if (dataset != null) {
9. +           if (dataset == null) {
10.                return result;
11.            }
12.            int seriesCount = dataset.getRowCount();
```

图 3 TBar 为 Chart-1 生成的修复补丁

Fig. 3 Repair patches generated by TBar for Chart-1

```
1.  public abstract class AbstractCategoryItemRenderer extends AbstractRenderer
2.              }
3.              int index = this.plot.getIndexOf(this);
4.              CategoryDataset dataset = this.plot.getDataset(index);
5.  -           if (dataset != null) {
6.                  return result;
7.              }
8.              int seriesCount = dataset.getRowCount();
```

图 4 缺陷代码片段

Fig. 4 Defect code snippet

```
1.  public abstract class AbstractCategoryItemRenderer extends AbstractRenderer
2.              }
3.              int index = this.plot.getIndexOf(this);
4.              CategoryDataset dataset = this.plot.getDataset(index);
5.  +           if (dataset == null) {
6.                  return result;
7.              }
8.              int seriesCount = dataset.getRowCount();
```

图 5 补丁代码片段

Fig. 5 Patch code snippet

3.2 特征提取

在特征提取阶段，使用预训练的 12 层 CodeBERT-base 模型分别对缺陷代码片段和补丁代码片段进行表征。CodeBERT 模型是在大量的代码语料库上预训练过的，能够充分表示代码的特征，从而应用于下游相关任务。为了对代码片段进行表征，首先将缺陷代码片段和补丁代码片段视为令牌序列，然后使用分词器将其分解为令牌列表。对于缺陷列表 $b=[b_1,b_2,b_3,\cdots,b_n]$ 和补丁列表 $p=[p_1,p_2,p_3,\cdots,p_n]$，使用 CodeBERT-base 模型对其进行编码。取模型最后一层[CLS]令牌对应的向量用作整个代码语句的语义表示，从而得到带有语义信息的嵌入向量 $\boldsymbol{e}_b$ 和 $\boldsymbol{e}_p$，其维度为 768。为捕获补丁代码片段与缺陷代码片段的差异，从而更好地进行分类[17-18]，使用减法、乘法、余弦相似度和欧几里得相似度 4 种组合函数对 $\boldsymbol{e}_b$ 和 $\boldsymbol{e}_p$ 进行集成，从而得到交叉特征 $\boldsymbol{E}_{\text{mix}}$。4 种组合函数如下所示：

$$\boldsymbol{E}_{\text{sub}}=\boldsymbol{e}_p-\boldsymbol{e}_b \tag{1}$$

$$\boldsymbol{E}_{\text{mul}}=\boldsymbol{e}_p\odot\boldsymbol{e}_b \tag{2}$$

$$E_{\text{euc}}=\|\boldsymbol{e}_p-\boldsymbol{e}_b\|_2 \tag{3}$$

$$\boldsymbol{E}_{\cos}=\frac{\boldsymbol{e}_b\boldsymbol{e}_p}{\|\boldsymbol{e}_b\|\|\boldsymbol{e}_p\|} \tag{4}$$

$$\boldsymbol{E}_{\text{mix}}=\boldsymbol{E}_{\text{sub}}\oplus\boldsymbol{E}_{\text{mul}}\oplus\boldsymbol{E}_{\text{euc}}\oplus\boldsymbol{E}_{\cos} \tag{5}$$

构建交叉特征的目的在于捕获缺陷代码片段和补丁代码片段之间的差异，不同的比较函数从不同的角度考虑其差异性。$\boldsymbol{E}_{\text{sub}}$ 只需将补丁代码片段和缺陷代码片段相减；$\boldsymbol{E}_{\text{mul}}$ 将补丁代码片段和缺陷代码片段逐元素相乘；而 $\boldsymbol{E}_{\text{euc}}$ 和 $\boldsymbol{E}\cos$ 分别代表欧几里得相似度和余弦相似度，从相似度的角度出发考虑二者的差异。通过将 4 个组合特征进行连接，得到最终的交叉特征 $\boldsymbol{E}_{\text{mix}}$，其中⊕表示矩阵拼接。最终的特征向量具有 $2\times k+2$ 维，其中 k 表示代码嵌入的维数($k=768$)。

3.3 学习与分类

在所有的补丁特征提取融合完成后，将交叉特征输入由多种类型分类器集成的 Stacking 模型中进行学习和分类。Stacking 模型具有取长补短的特性，在选择第一层的基模型时应该遵循“强而不同”的原则，即第一层的基模型应该尽量选择准确性较高、结构差异较大的强学习器。此外，在有监督学习中，模型的误差主要来源于方差和偏差两个方面。机器学习中的 Bagging 策略通过对样本重采样，训练多个不同的

模型，最后对多个模型结果进行平均，达到减小方差的效果。Boosting 策略通过每次聚焦上一个弱分类器的估计结果，不断调整权重和改进模型，来有效减小偏差。因此，所提方法在 Stacking 中综合了基于 Bagging、基于 Boosting 和拟合性较好的基于深度学习的强学习器作为分类模型的第一层，分别对补丁进行预测。

在基于 Bagging 策略的强分类器中，选取典型的随机森林(Random Forests，RF)模型[19]。该模型中每个决策树在建立时随机选择一部分特征进行分裂，从而降低了单个决策树的过拟合风险，提高了整体模型的泛化能力和准确性。在基于 Boosting 策略的强分类器中，选取 XGBoost[20]，LightGBM[21]和 CatBoost[22] 3 种经典的 SOTA(State-Of-The-Art) Boosting 算法。3 种模型采用不同的策略构造决策树，XGBoost 基于 level-wise 策略，LightGBM 基于 leaf-wise 策略，而 CatBoost 使用了对称决策树结构。在基于深度学习的分类器中，选取多层感知机(Multilayer Perceptron，MLP)。MLP 由多个感知层组成，每个感知层由多个神经元组成。每个神经元接收来自其他神经元的输入，并通过一个非线性激活函数将这些输入映射为输出。MLP 的训练通常采用反向传播算法，该算法使用梯度下降来最小化损失函数。在训练期间，模型通过反复调整权重和偏置来优化预测结果。

由于 Stacking 的第一层都是强学习器，特征在学习的过程中经历了复杂的非线性变换。为了防止模型过拟合，第二层选取简单的逻辑回归模型对第一层输出结果进行最终分类。

Stacking 学习和分类的具体执行步骤如下。

步骤 1　将输入划分为训练集和测试集，再使用五折交叉验证将训练集划分为 5 份。

步骤 2　针对每一个基模型，轮流选取训练集中 5 份数据的 1 份作为测试集，其余 4 份作为训练集，重复 5 次。5 个基模型分别得到测试结果 A_1，A_2，A_3，A_4 和 A_5。将所有结果进行纵向拼接作为第二层元模型的训练集，记为 A。

步骤 3　基模型在进行五折交叉验证时，每次都针对步骤 1 中的测试集进行测试，然后对得到的 5 个不同的结果取平均值。5 个基模型分别得到 B_1，B_2，B_3，B_4 和 B_5。将其进行纵向拼接作为第二层元模型的测试集，记为 B。

步骤 4　A 作为训练数据，B 作为测试数据。由元学习器进行最终的预测分类。

使用 Stacking 集成策略集成多个模型对样本进行学习，可以更充分地利用样本中的特征。本文提出结合 CodeBERT 预训练模型和 Stacking 集成策略对补丁进行验证是同时从模型的两端进行考虑，即如何有效地提取更丰富的补丁特征以及如何更充分地利用提取到的特征，从而提升整个模型的性能。

4　实验分析

为了评估所提方法的有效性，本章进行了大量的实验探究及结果分析。实验的运行环境为 Linux Ubuntu 20.04.1 系统，4 块 2.20 GHz 的 Intel(R) Xeon(R) Gold 5120 CPU，24 GB 物理内存，1 块 GeForce RTX 2080 Ti GPU，编译环境为 Python3.7。

4.1　实验对象

为验证所提方法的有效性，需要使用已标记的历史修复补丁数据集。使用 Liu 等[23]和 Xiong 等[8]收集的补丁集合对所提方法进行验证，这些补丁是来自各类 APR 工具对 Defects4J 缺陷数据集中的缺陷进行修复产生的真实补丁。同时，为了避免数据不平衡的问题，数据集中添加了从 Defects4J[24]中获取的开发人员为其编写的正确修复补丁。最后，对收集到的所有补丁样本进行去重。数据集中的补丁分布如表 1 所列。

表 1　补丁数据集

Table 1　Patch datasets

来源	正确补丁	错误补丁	补丁总数
Liu 等[23]	137	502	639
Xiong 等[8]	30	109	139
Defects4J 中的开发者补丁[24]	356	0	356
合计(去重)	468	532	1 000

4.2　评价标准

为了对所提方法的效果进行评估，使用准确率(Accuracy)、精准率(Precision)、召回率(Recall)、F1 分数(F1 Score)和 AUC(Area Under the Curve)等评价指标。

准确率指在所有分类样本中被正确分类的样本所占的比例，如式(6)所示：

$$Accuracy=\frac{TP+TN}{TP+TN+FP+FN} \tag{6}$$

其中，TP(True Positive)表示被预测为正例样本且实际为正例样本的个数；FP(False Positive)表示被预测为正例样本但实际为负例样本的个数；TN(True Negative)表示被预测为负例样本且实际为负例样本的个数；FN(False Negative)表示被预测为负例样本但实际为正例样本的个数。

精准率指在所有被分类为正例的样本中，真正的正例样本所占的比例，如式(7)所示。精准率越高，表示分类器预测为正例的样本中真正的正例样本越多，误判率越低。

$$Precision=\frac{TP}{TP+FP} \tag{7}$$

召回率指在所有实际为正例的样本中被正确预测为正例样本所占的比例，如式(8)所示。召回率越高，表示分类器能够正确识别出的真正的正例样本越多，遗漏率越低。

$$Recall=\frac{TP}{TP+FN} \tag{8}$$

F1 分数是综合考虑精准率和召回率的指标，即精准率与召回率的调和平均值，用于衡量分类器的综合表现，如式(9)所示：

$$F1=\frac{2*Precision*Recall}{Precision+Recall} \tag{9}$$

AUC 指分类算法绘制的 ROC 曲线下的面积。AUC 值越大，表示分类器的性能越好，其越能够正确地判断正例和负例，如式(10)所示：

$$AUC=\frac{\sum I(P_{正},P_{负})}{M\times N}$$

$$I(P_{正},P_{负})=\begin{cases}1, & P_{正}>P_{负}\\ 0.5, & P_{正}=P_{负}\\ 0, & P_{正}<P_{负}\end{cases} \tag{10}$$

其中，M 和 N 分别表示正例样本和负例样本的数量，$P_{正}$ 和 $P_{负}$ 分别表示预测正例样本的精准率和预测负例样本的精准率。

4.3 实验结果和分析

为了评估所提软件修复补丁正确性验证方法的有效性，希望通过实验回答以下 3 个问题。

研究问题 1：使用 Stacking 策略集成多个分类模型是否比使用单个模型进行补丁验证的效果更好？

研究问题 2：使用不同的表示学习模型对数据进行表征，对补丁验证的整体效果有多大的影响？

研究问题 3：与其他的补丁验证技术相比，所提模型的性能如何？

4.3.1 研究问题 1

如 3.3 节所述，CatBoost，XGBoost，LightGBM，MLP，RF 和 LR(逻辑回归)进行 Stacking 模型集成。在研究问题 1 中，旨在探索所提方法与这些只使用单个模型的分类技术相比，其性能如何。为保证实验的准确性，实验在 4.1 节所述的去重数据集上进行五折交叉验证，即随机将数据集划分成 5 个部分，其中 4/5 用于训练，1/5 用于测试，实验进行 5 次求平均值。

表 2 列出了除分类器以外其余部分不变的情况下，Stacking 集成模型与所有单个基模型的比较结果。第 1 列是用于表征的表示学习模型，第 2 列展示了 6 种基模型分类算法和集成分类算法，第 3—7 列分别列出了准确性、精准率、召回率、F1 分数和 AUC 指标的详细值。从表中可以看出，在使用相同表征技术的前提下，单个分类算法中，准确率、召回率、F1 分数和 AUC 表现最佳的是 CatBoost 模型，分别为 0.780，0.754，0.761，0.855；精准率表现最佳的是 MLP，为 0.784。相比之下，Stacking 集成分类算法在准确率、精准率、召回率、F1 分数和 AUC 方面都具有最佳性能，分别为 0.800，0.799，0.762，0.780 和 0.865，在各项指标上比最佳基模型分别提高了 2%，1.5%，0.8%，1.9%，1%。由此可见，在补丁正确性验证方面，Stacking 集成模型可以取得比单个基模型更好的分类效果。

表 2 不同分类器的性能比较

Table 2 Performance comparison of different classifiers

表征模型	分类器	准确率	精准率	召回率	F1	AUC
CodeBERT	LR	0.764	0.763	0.720	0.740	0.826
	RF	0.755	0.763	0.694	0.726	0.836
	LightGBM	0.760	0.752	0.732	0.741	0.844
	CatBoost	**0.780**	0.771	**0.754**	**0.761**	**0.855**
	MLP	0.769	**0.784**	0.719	0.743	0.841
	XGBoost	0.776	0.776	0.736	0.754	0.851
	Stacking	**0.800**	**0.799**	**0.762**	**0.780**	**0.865**

4.3.2 研究问题 2

为研究不同表示学习模型对补丁验证总体效果的影响，本小节选用了 Doc2Vec[25]，BERT[26]，CC2Vec[17] 和 CodeBERT 4 种表示学习模型对补丁数据的特征进行提取组合，然后将其输入到相同的 Stacking 模型中。Doc2Vec 和 CC2Vec 遵循前人的设定[18]，使用 36 364 个补丁的代码数据来训练 Doc2Vec 模型，使用 CC2Vec 的 CNN-3D 层作为提取错误代码片段和补丁代码片段特征的最佳位置。BERT 直接使用在大型语料库上预训练好的模型 bert-large-uncase。实验在 4.1 节的去重数据集上进行。为保证实验结果的准确性，进行五折交叉验证，运行 5 次，取平均值。

表 3 列出了使用不同表示学习模型的对比结果。第 1 列为分类模型，第 2 列是不同的表示学习模型，第 3－7 列是 5 个评估指标。根据表中的数据，使用 CodeBERT 进行特征提取，使得整个方法在准确率、精准率、召回率、F1 分数和 AUC 方面都取得了最佳效果，其略优于 CC2Vec 模型的效果，Doc2Vec 效果最差。这是因为 CodeBERT 在大型的代码语料库上使用 MLM 和 RTD 任务预训练过，能够充分理解代码的语义，从而更好地捕捉代码的语义特征。CC2Vec 则是专门为代码更改表示所设计的深度学习框架，可以更好地捕获代码修改前后的差异，但因为用于预训练的高质量代码更改较少，该模型难以把握代码的整体语义信息，所以效果略次于 CodeBERT。Doc2Vec 虽然也使用了许多代码数据进行训练，但是该模型是为文档或段落数据开发的表示学习技术，未能充分地学习到代码的语义信息，效果较差。BERT 虽然是为自然语言开发的表示学习模型，但是其网络结构很深且训练语料库很大，可以捕获语句的关系，因此效果优于 Doc2Vec。但由于该模型没有在代码语料上训练过，因此次于 CC2Vec 和 CodeBERT。综上所述，CodeBERT 在补丁验证领域取得了良好的效果。

表 3 不同表示学习模型的对比

Table 3 Comparison of different representation learning models

分类器	表征模型	准确率	精准率	召回率	F1	AUC
Stacking	Doc2Vec	0.734	0.729	0.690	0.707	0.797
	BERT	0.756	0.741	0.737	0.739	0.828
	CC2Vec	0.796	0.790	0.771	0.779	0.847
	CodeBERT	**0.800**	**0.799**	**0.762**	**0.780**	**0.865**

4.3.3 研究问题 3

为更好地回答研究问题 3，进一步将其分解为两个子问题分别进行探究。

研究问题 3-1：与静态的补丁验证技术相比，所提模型的性能如何？

研究问题 3-2：与动态的补丁验证技术相比，所提模型的性能如何？

为了回答研究问题 3-1，将所提模型分别与 Tian 等[18]提出的补丁验证方法、Ye 等[10]提出的 ODS 进行比较。

Tian 等[18]的方法性能最好的一组实验是使用 BERT 预训练模型对数据进行表征，然后将其输入到逻辑回归分类器中进行分类。因此，使用 BERT＋LR 的组合与本文方法进行对比。除此之外，还分别添加了研究问题 1 中的 CodeBERT＋LR 组合和研究问题 2 中的 BERT＋Stacking 组合作为基线进行对比，从而更好地验证本文方法的总体有效性。实验在 4.1 节的去重数据集上进行。为保证实验结果的准确性，进行五折交叉验证，运行 5 次，取平均值。

实验结果如表 4 所列。BERT＋LR 组合的准确率、

精确率、召回率、F1 分数和 AUC 分别为 0.740，0.727，0.715，0.720，0.802；BERT＋Stacking 的组合在其基础上各个指标分别提升了 1.6%，1.4%，2.2%，1.9%，2.6%；CodeBERT＋LR 的组合在其基础上各个指标分别提升了 2.4%，3.6%，0.5%，2%，2.4%。相比之下，所提补丁验证模型效果最佳，对比 BERT＋LR 的组合，其各项指标分别提升了 6%，7.2%，4.7%，6%，6.3%，优于其他两个基线提升之和。综上所述，所提方法在各个指标上均显著优于基线方法，也再次证明了 CodeBERT 和 Stacking 集成学习结合的必要性。

表 4　所提方法与 Tian 等方法的对比

Table 4　Comparison between the proposed method and Tian et al.'s methods

模型	准确率	精准率	召回率	F1	AUC
BERT＋LR	0.740	0.727	0.715	0.720	0.802
BERT＋Stacking	0.756	0.741	0.737	0.739	0.828
CodeBERT＋LR	0.764	0.763	0.720	0.740	0.826
CodeBERT＋Stacking	**0.800**	**0.799**	**0.762**	**0.780**	**0.865**

此外，将所提方法与 Ye 等[10]提出的静态补丁验证方法 ODS 进行比较。ODS 手动设计并提取补丁特征构建预测模型，从而对补丁进行过拟合检测。将所提方法与 ODS 在其报告中测试的 139 个补丁[8]上进行简单的比较，其余补丁作为训练集。

实验结果如表 5 所列。除精准率外，所提方法在其他 3 个指标上均优于 ODS。ODS 精准率更高，可能是因为手动设计的某些特征与过拟合补丁具有更高的相关性。所提方法对比 ODS，除了拥有更好的综合性能外，还具有可以自动提取代码特征的优势，降低了成本。

在后续的工作中我们将扩充数据集并分析手动设计的特征中与过拟合高度相关的部分，将其与自动提取的特征相结合，进一步提高模型性能。

表 5　所提方法与 ODS 的对比

Table 5　Comparison between the proposed method and ODS

模型	准确率	精准率	召回率	F1
ODS	0.676	**0.933**	0.636	0.756
CodeBERT＋Stacking	**0.683**	0.865	**0.700**	**0.774**

为回答研究问题 3-2，将所提方法与最先进的动态补丁验证技术 PATCH-SIM[8]进行比较。该技术通过比较补丁程序的执行轨迹来识别补丁的正确性。主要从以下两个方面将所提方法与其进行比较。1）识别过拟合补丁的数量：验证所提方法是否可以对动态补丁验证技术进行补充。2）补丁验证所消耗的时间：验证所提方法相比动态补丁技术在时间上是否有优势。由于动态补丁验证技术非常耗时，两种方法都在 Xiong 等[8]使用的 139 个标记补丁上进行评估。在对所提方法进行评估时，为避免训练数据和测试数据重复，从 4.1 节中的数据集上去除测试集所用的补丁，其余补丁用于模型训练。此外，为保证公平，在比较二者的运行性能时，在同一环境下执行所提方法和 PATCH-SIM。

表 6　所提方法与 PATCH-SIM 在识别过拟合补丁数量方面的对比

Table 6　Comparison between the proposed method and PATCH-SIM in identifying the number of overfitting patches

测试集			PATCH-SIM				CodeBERT＋Stacking			
项目	不正确	正确	TP	FP	TN	FN	TP	FP	TN	FN
Chart	23	3	14	0	3	9	17	1	2	6
Lang	10	5	6	0	4	5	5	2	3	5
Math	63	20	33	0	20	30	46	8	12	17
Time	13	2	9	0	2	4	9	1	1	4

为了对所提方法和动态补丁验证技术 PATCH-SIM 进行分析，表 6 列出了两种方法在测试数据集中 4 个项目上的详细分类结果。前 3 列给出了测试集的项目名称、不正确补丁数量和正确补丁数量，第 4—7 列展示了 PATCH-SIM 的预测结果，第 8—11 列给出了所提方法的预测结果。例如，第 3 行最后 4 列表示所提方法在 Chart 项目上可以正确检测到 23 个过拟合补丁中的 17 个、3 个正确补丁中的 2 个，以及 1 个假阳性和 6 个假阴性。

为了验证所提方法和 PATCH-SIM 是否具有互补性，图 6 展示了两种方法在 4 个项目上可以正确检测到的过拟合补丁的交集。左边圆形区域、右边圆形区域和中间交叉区域分别代表本文方法所能检测到的过拟合补丁个数、PATCH-SIM 所能检测到的过拟合补丁个数和两种方法检测到的相同过拟合补丁个数。例如，对于 Chart 项目（左上图），所提方法和 PATCH-SIM 检测到 10 个相同的过拟合补丁，并且它们还可以分别检测到 7 个和 4 个不同的过拟合补丁。综合考虑 4 个项目，实验结果表明所提方法和 PATCH-SIM 总共可以识别出 98 个过拟合补丁，其中 41 个可以被两种方法共同识别。由上述实验结果可以看出，在验证过拟合补丁数量方面，所提方法和 PATCH-SIM 可以相互补充。这是因为两种方法从不同的角度评估补丁的正确性。其中，动态补丁验证技术 PATCH-SIM 通过分析补丁的执行轨迹识别补丁的正确性；而所提方法通过自动提取补丁特征构建概率模型，从静态的角度评估补丁的正确性。

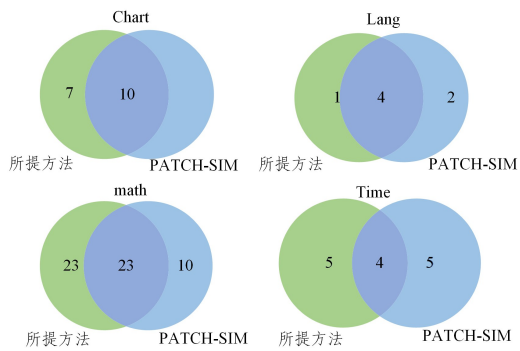

图 6　所提方法和 PATCH-SIM 检测到的过拟合补丁的交集

Fig. 6　Intersection of overfitting patches detected by our method and PATCH-SIM

表 7 列出了所提方法和 PATCH-SIM 运行时性能的对比。第 1 列介绍了两种补丁评估方法，第 2 列给出了进行特征提取和模型训练所需要的时间，因为动态补丁验证技术 PATCH-SIM 不需要特征提取和训练，所以用“－”表示；第 3 列是两种方法对测试数据集的 139 个补丁的总预测时间；第 4 列是平均每个补丁验证所消耗的时间。

表 7 所提方法与 PATCH-SIM 运行时的性能对比

Table 7 Runtime performance comparison between the proposed method and PATCH-SIM

方法	特征提取+训练	预测总时间	预测时间/补丁
PATCH-SIM	—	约 70 h	30 min
CodeBERT+Stacking	427.0 s	38.5 s	0.3 s

所提模型在特征提取和训练阶段消耗了约 7 min，其中大部分时间消耗在特征提取阶段。但是只要确定训练数据集，则模型只需要训练一次便可重复使用。在测试阶段，PATCH-SIM 由于需要运行测试套件，消耗大约 70 h 才能对所有补丁完成验证；而所提方法只需要 38.5 s。平均到每个补丁，PATCH-SIM 验证需要花费 30 min；本文方法只需要 0.3 s，比 PATCH-SIM 快了 6 000 倍。所提方法在模型训练完成的基础上，几乎在补丁生成的那一刻便可判断出缺陷修复补丁是否过拟合。

上述实验分析表明，所提方法和动态补丁验证技术 PATCH-SIM 在进行补丁验证时可以检测到不同的过拟合补丁，二者互补。此外，由于所提方法是静态方法，不用执行测试用例，因此可以节省大量的编译和运行时间，这将极大地提高缺陷修复人员的工作效率。

4.3.4 结果与分析

实验结果表明，在自然语言和代码语言双语料上预训练的 CodeBERT 在验证补丁正确性时表现出最佳性能，在补丁代码上训练过的 CC2Vec 效果次之。这意味着面向代码的表示学习模型可以为补丁验证产生更好的特征表示。在未来，研究人员应该重点关注面向代码中先进的预训练表示学习模型。此外，使用预训练模型在补丁正确性验证任务上进行微调，可能是提高性能的有效手段。

在分类阶段，Stacking 集成模型实现了比单个分类算法更优的性能，因为不同分类算法的侧重点不同。实验结果表明，对不同分类器的预测概率进行组合是有价值的。未来，研究人员应通过组合异构强学习器的预测结果，对补丁进行更有效的验证。

5 有效性分析

本文主要从内部有效性和外部有效性两个方面对所提方法的有效性进行讨论。

1）影响内部有效性的因素。首先，表示学习模型 CodeBERT 没有在我们的数据集上进行微调，这可能导致其在补丁正确性验证方面的性能被低估。未来将收集更多的补丁数据对 CodeBERT 进行微调，从而减少这种威胁。其次，大量文献提出了各种各样的分类算法，实验可能会因为本文在 Stacking 模型中只考虑了 6 种而造成选择偏差。但通过考虑由不同策略形成的代表性强学习器（基于 Bagging、基于 Boosting 和基于深度学习），减轻了这种威胁。

2）影响外部有效性的因素。本文只使用了 Defects4J 缺陷数据集上的相关补丁数据，这是因为目前大多数的缺陷修复工作都是在 Defects4J 上进行验证的，方便数据获取和实验对比。所使用的补丁数据集中包含多种 APR 工具为 Defects4J 中缺陷产生的修复补丁，其真实且多样化，减轻了数据集方面的威胁。

6 相关工作

APR 技术的目标是自动识别和修复程序错误。如今的自动修复技术大多基于补丁生成和测试，通常包含 3 个步骤。1）利用现成的错误定位技术对可疑的代码语句或方法进行识别，从而对错误所在位置进行定位；2）使用一系列代码转换策略对错误进行修复，得到程序变体，也就是候选补丁；3）使用测试套件对所有的候选补丁进行正确性验证。如果候选补丁通过所有的测试，则该补丁被视为合理补丁。如果一个合理补丁的语义符合开发者的意图，则该补丁为正确补丁，否则为过拟合补丁。

在如今的 APR 领域，解决补丁过拟合问题刻不容缓。Qi 等[27]证明了绝大多数由 GenProg[28]，RSRepair[29] 和 AE[30] 生成的合理的补丁都只是过度拟合测试套件，实际上并不正确。因此，许多研究学者开始提出各种补丁验证技术自动识别补丁的正确性。

为更好地预测补丁的正确性，最早的思想是增加额外的测试用例，使测试套件尽可能完备。例如，Xin 等[31] 提出的 DiffTGen 使用测试生成工具 Evosuite 生成测试用例。如果 APR 生成的候选补丁程序的输出和人工编写的正确程序的输出不同，则认为该候选补丁是过拟合补丁。Yang 等[32] 提出使用模糊测试来生成新的测试用例，并设置测试断言（崩溃和内存安全）对补丁进行正确性验证。Xiong 等[8] 使用测试生成工具生成新的测试用例，并根据测试用例执行的相似性来评估补丁的正确性。其基于正确的补丁不会改变最初通过测试用例的行为，但会修改最初失败测试用例的行为这一合理观察，实现了动态补丁验证技术 PATCH-SIM。上述这类通过不断地增加额外测试用例用于补丁验证的技术非常依赖测试用例的质量。在应用中，高覆盖率的测试可能并不可用[10]。Tian 等[33] 从自然语言的角度出发，考虑错误报告和修复补丁的描述之间的联系，从而确定补丁的正确性。所提方法不依赖任何的测试用例和错误报告，而是通过预训练的表示学习模型对补丁代码和缺陷代码进行静态特征提取。

为了识别 APR 工具产生的过拟合补丁，Ye 等[10] 在缺陷代码和补丁代码的抽象语法树级别手动设计并提取了 202 个代码特征。这些代码特征和标签被输入 3 种机器学习模型中，构建出一个概率模型。ODS 与所提方法都使用了代码的静态特征和机器学习算法。然而，ODS 依赖手动设计的代码特征，成本较高，可能无法推广到其他数据集。在 ODS 的基础上，Tian 等[34] 对手动设计特征的有效成分进行分析，并结合自动提取的特征对补丁进行验证。此外，Csuvik[5] 等尝试使用 BERT 生成嵌入向量来确定缺陷代码和补丁代码之间的相似阈值，从而对补丁进行过滤。在这之后，Tian 等[18] 将 BERT 模型和机器学习相结合来进行补丁正确性验证。相比在自然语言上训练的 BERT 模型，使用在代码语料上训练的 CodeBERT 模型可以更好地对补丁代码和缺陷代码进行表示。此外，相比上述补丁验证方法所使用的单个机器学习模型，使用 Stacking 算法集成多个异构机器学习模型可以更有效地对提取到的特征进行分类，从而提高补丁验证的性能。

结束语 本文针对软件缺陷修复补丁的正确性验证这一

关键问题，提出了一种基于 CodeBERT 预训练模型和 Stacking 集成学习的方法，并在 Defects4J 相关的补丁数据集上进行了验证。实验结果表明，所提方法在 F1 和 AUC 两个指标上均取得了不错的效果，其分别为 0.780 和 0.865，这表明该方法可以有效地对补丁进行正确性验证。本文的研究为自动修复领域提供了一种新的补丁验证工具，可以提高补丁的质量和可靠性，加快自动修复的速度。未来，计划进一步结合补丁的静态特征和动态执行信息，旨在在降低时间成本的同时提升补丁验证的准确度，为自动修复领域作出更大的贡献。

参 考 文 献

[1] LIU K，KOYUNCU A，KIM D，et al. TBar：revisiting template-based automated program repair[C]// Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. NY：ACM，2019：31-42.

[2] WONG C，SANTIESTEBAN P，KÄSTNER C，et al. VarFix：balancing edit expressiveness and search effectiveness in automated program repair[C]// Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. NY：ACM，2021：354-366.

[3] JIANG N，LUTELLIER T，TAN L. Cure：code-aware neural machine translation for automatic program repair[C]// 2021 IEEE/ACM 43rd International Conference on Software Engineering. NJ：IEEE，2021：1161-1173.

[4] JIANG J J，CHEN J J，XIONG Y F. Survey of automatic program repair techniques[J]. Journal of Software，2021，32(9)：2665-2690.

[5] CSUVIK V，HORVÁTH D，HORVÁTH F，et al. Utilizing source code embeddings to identify correct patches[C]// 2020 IEEE 2nd International Workshop on Intelligent Bug Fixing. NJ：IEEE，2020：18-25.

[6] FRASER G，ARCURI A. Evosuite：automatic test suite generation for object-oriented software[C]// Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. NY：ACM，2011：416-419.

[7] PACHECO C，ERNST M D. Randoop：feedback-directed random testing for Java[C]// Companion to the 22nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion. NY：ACM，2007：815-816.

[8] XIONG Y F，LIU X Y，ZENG M H，et al. Identifying patch correctness in test-based program repair[C]// Proceedings of the 40th International Conference on Software Engineering. NY：ACM，2018：789-799.

[9] LIN B，WANG S W，WEN M，et al. Context-aware code change embedding for better patch correctness assessment[J]. ACM Transactions on Software Engineering and Methodology，2022，31(3)：1-29.

[10] YE H，GU J，MARTINEZ M，et al. Automated classification of overfitting patches with statically extracted code features[J]. IEEE Transactions on Software Engineering，2021，48(8)：2920-2938.

[11] FENG Z Y，GUO D Y，TANG D Y，et al. Codebert：a pre-trained model for programming and natural languages[J]. arXiv：2002.08155，2020.

[12] VASWANI A，SHAZEER N，PARMAR N，et al. Attention is all you need[J]. arXiv：1706.03762，2017.

[13] HUSAIN H，WU H H，GAZIT T，et al. Codesearchnet challenge：Evaluating the state of semantic code search[J]. arXiv：1909.09436，2019.

[14] KARTHIKEYAN K，WANG Z H，MAYHEW S，et al. Cross-lingual ability of multilingual bert：An empirical study[C]// International Conference on Learning Representations. OpenReview. net，2020.

[15] WOLPERT D H. Stacked generalization[J]. Neural Networks，1992，5(2)：241-259.

[16] DITTERRICH T G. Machine learning research：four current direction[J]. Artificial Intelligence Magzine，1997，4：97-136.

[17] HOANG T，Kang H J，LO D，et al. CC2Vec：distributed representations of code changes[C]// Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. NY：ACM，2020：518-529.

[18] TIAN H Y，LIU K，KABORÉ A K，et al. Evaluating representation learning of code changes for predicting patch correctness in program repair[C]// Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. NJ：IEEE，2020：981-992.

[19] BREIMAN L. Random forests[J]. Machine Learning，2001，45：5-32.

[20] CHEN T Q，GUESTRIN C. Xgboost：a scalable tree boosting system[C]// Proceedings of the 22nd acm sigkdd International Conference on Knowledge Discovery and Data Mining. NY：ACM，2016：785-794.

[21] KE G L，MENG Q，FINLEY T，et al. Lightgbm：A highly efficient gradient boosting decision tree[J]. Neural Information Processing Systems，2017，30：3149-3157.

[22] PROKHORENKOVA L，GUSEV G，VOROBEV A，et al. CatBoost：unbiased boosting with categorical features[J]. Neural Information Processing Systems，2018，31：6639-6649.

[23] LIU K，WANG S W，KOYUNCU A，et al. On the efficiency of test suite based program repair：a systematic assessment of 16 automated repair systems for java programs[C]// Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. NY：ACM，2020：615-627.

[24] JUST R，JALALI D，ERNST M D. Defects4J：a database of existing faults to enable controlled testing studies for Java programs[C]// Proceedings of the 2014 International Symposium on Software Testing and Analysis. NY：ACM，2014：437-440.

[25] LE Q，MIKOLOV T. Distributed representations of sentences and documents[C]// International Conference on Machine Learning. NY：ACM，2014：1188-1196.

[26] DEVLIN J，CHANG M W，LEE K，et al. Bert：pre-training of deep bidirectional transformers for language understanding[J]. arXiv：1810.04805，2018.

[27] QI Z C，LONG F，ACHOUR S，et al. An analysis of patch plausibility and correctness for generate-and-validate patch genera-

tion systems[C]// Proceedings of the 2015 International Symposium on Software Testing and Analysis. NY: ACM, 2015: 24-36.
[28] LE G C, NGUYEN T V, FORREST S, et al. Genprog: a generic method for automatic software repair[J]. IEEE Transactions on Software Engineering, 2011, 38(1): 54-72.
[29] QI Y H, MAO X G, LEI Y, et al. The strength of random search on automated program repair[C]// Proceedings of the 36th International Conference on Software Engineering. NY: ACM, 2014: 254-265.
[30] WEIMER W, FRY Z P, FORREST S. Leveraging program equivalence for adaptive program repair: Models and first results [C]// 2013 28th IEEE/ACM International Conference on Automated Software Engineering. NJ: IEEE, 2013: 356-366.
[31] XIN Q, REISS S P. Identifying test-suite-overfitted patches through test case generation[C]// Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. NY: ACM, 2017: 226-236.
[32] YANG J, ZHIKHARTSEV A, LIU Y, et al. Better test cases for better automated program repair[C]// Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. NY: ACM, 2017: 831-841.
[33] TIAN H, TANG X, HABIB A, et al. Is this change the answer to that problem? Correlating descriptions of bug and code changes for evaluating patch correctness[C]// Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. Rochester, IEEE/ACM, 2022: 1-13.
[34] TIAN H, LIU K, LI Y, et al. The Best of Both Worlds: Combining Learned Embeddings with Engineered Features for Accurate Prediction of Correct Patches[J]. ACM Transactions on Software Engineering and Methodology, 2023, 32(4): 1-34.

HAN Wei, born in 1998, master. His main research interest is automatic program repair.

JIANG Shujuan, born in 1966, Ph. D, professor, Ph.D supervisor, is a member of CCF (No. 15801M). Her main research interests include software analysis and test and compilation techniques.

(责任编辑: 刘钰枭)

关于征集 CCF 产学合作基金优秀项目案例的通知

一、征集范围

通过验收 1 年以上的 CCF 产学合作基金项目，由同时符合以下条件的 CCF 产学合作基金合作伙伴推荐：1、合作伙伴年度合作基金资助项目不少于 8 个，且年度基金资助规模不少于 200 万元。2、合作伙伴已具有规范的优秀项目申请、评审、推荐机制。

二、评选流程

1)由基金合作伙伴与立项项目负责人联合填写产学合作基金优秀项目案例申请表，提交给 CCF 产学合作基金办公室(以下简称“基金办公室”)。2)基金办公室对提交的材料进行规范性审核后提交给 CCF 产学合作基金技术委员会(以下简称“基金技术委员会”)。3)基金技术委员会召开会议进行评选。每支基金评选出的候选优秀项目案例数量不超过年度资助项目数的 10%。4)由 CCF 组织评审委员会进行优秀项目案例终评，终审结论经秘书长批准后公布。

三、评选标准

按照成果的创新性及难度(30%)、应用及推广情况(50%)、其他可以证明项目优秀的因素(20%)进行评选。

四、征集及评选时间安排

1)2024 年 12 月 12 日：征集截止
2)2024 年 12 月 30 日：完成评选
3)2025 年 01 月 02 日：公示评选结果
4)2025 年 01 月 18 日：颁发证书

五、申报途径

扫码下面二维码，填写相关信息并发送申报表至基金办公室(fund@ccf.org.cn)。

据 CCF 微信公众号