计算机科学
COMPUTER SCIENCE

# 基于细粒度代码表示和特征融合的即时软件缺陷预测方法

朱晓燕，王文格，王嘉寅，张选平

引用本文

朱晓燕，王文格，王嘉寅，张选平. 基于细粒度代码表示和特征融合的即时软件缺陷预测方法[J]. 计算机科学，2025，52(1)：242-249.

ZHU Xiaoyan, WANG Wenge, WANG Jiayin, ZHANG Xuanping. Just-In-Time Software Defect Prediction Approach Based on Fine-grained Code Representationand Feature Fusion [J]. Computer Science, 2025, 52(1): 242-249.

---

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240200046

# 基于细粒度代码表示和特征融合的即时软件缺陷预测方法

朱晓燕　王文格　王嘉寅　张选平
西安交通大学计算机科学与技术学院　西安 710049

**摘　要**　即时软件缺陷预测指在软件更改初次提交之际预测该更改引入缺陷的倾向。此类预测针对单一程序变更,而非在粗粒度上进行。由于其即时性和可追溯性,该技术已在持续测试等领域得到广泛应用。目前的研究中,提取变更代码表示的方法粒度较粗,仅标出了变更行,而没有进行细粒度的标记。此外,现有的使用提交内容进行缺陷预测的方法,仅仅是把提交消息与变更代码的特征进行简单拼接,缺失了在特征空间上的深度对齐,这使得在提交消息质量参差不齐的情况下,会出现预测结果易受噪声干扰的情形,并且现有方法也未将领域专家设计的人工特征以及变更内容中的语义语法信息综合起来进行预测。为了解决上述问题,提出了一种基于细粒度代码表征和特征融合的即时软件缺陷预测方法。通过引入新的变更嵌入计算方法来在细粒度上表示变更代码。同时,引入特征对齐模块,降低提交消息中噪声对方法性能的影响。此外,使用神经网络从人工设计的特征中学习专业知识,充分利用现有特征进行预测。实验结果表明,相较于现有方法,该方法在3个性能指标上均有显著提升。

**关键词**:即时软件缺陷预测;特征融合;软件工程;深度学习;代码表示
**中图分类号**　TP311

# Just-In-Time Software Defect Prediction Approach Based on Fine-grained Code Representation and Feature Fusion

ZHU Xiaoyan,WANG Wenge,WANG Jiayin and ZHANG Xuanping
School of Computer Scienceand Technology,Xi'an Jiaotong University,Xi'an 710049,China

**Abstract**　Just-in-time software defect prediction(JIT-SDP) aims to predict the defect tendency of software changes at the time when they are first committed. Such predictions are made on a single program change rather than on a coarse granularity. It has been widely used in fields such as continuous testing due to its immediacy and traceability. Existing JIT-SDP studies extract features from code changes at a coarse granularity,merely marking the changed lines without fine-grained tagging. Moreover,studies based on commit content are limited to simple concatenation of features extracted from commit messages and code changes,lacking deep alignment in feature space. This makes the prediction results tend to be disturbed by noise when the quality of committed message cannot be guaranteed. Existing methods also fail to fully utilize artificial features designed by domain experts and semantic syntax structure information in commit content at the same time,thus not fully leveraging existing features. To address these problems,a JIT-SDP approach based on fine-grained code changes and feature fusion is proposed. The method introduces new change embeddings to represent code changes at a fine granularity. By designing a feature alignment module,the impact of noise in low-quality commit message on performance is reduced. Meanwhile,neural networks are used to learn domain-specific knowledge from artificial features and fully utilize existing features. Experimental results show that compared to existing methods,this approach improves significantly on three performance metrics.

**Keywords**　Just-in-time software defect prediction,Feature fusion,Software engineering,Deep learning,Code representation

## 1　引言

软件已经成为日常生活中不可或缺的工具。然而,几乎所有的软件系统都不可避免地存在一些缺陷。为了在软件开发过程中尽早地预测到缺陷并进行修复,研究人员针对不同粒度的软件部分提出了多种软件缺陷预测技术,如文件粒度[1-2]、模块粒度[3-5]、包粒度[6]和变更粒度[7-9]。其中,针对变更粒度的软件缺陷预测又被称为即时软件缺陷预测(Just-In-Time Software Defect Prediction,JIT-SDP)。不同于传统的软件缺陷预测技术,即时软件缺陷预测专注于对代码变更的部分进行缺陷预测,从而避免将包含缺陷的更改合并到最终发布的软件版本中。此外,其他软件缺陷预测方法只能在

到稿日期:2024-02-19　返修日期:2024-07-02
基金项目:国家自然科学基金(72274152)
This work was supported by the National Natural Science Foundation of China(72274152).
通信作者:朱晓燕(zhu.xy@xjtu.edu.cn)

文件级别或者模块级别进行粗粒度的预测[10-11]，而在变更粒度上的即时软件缺陷预测技术提供了更加精细化的预测，同时减少了测试流程所需的资源[12]。

Kamei 等[13]认为即时软件缺陷预测有如下优势：首先，软件变更代码的占比通常较小，而且每次提交都有一定的提交消息作为描述性文本附于提交内，当发现这个提交为缺陷提交时，开发者无须花费太大功夫进行缺陷的定位和更正；其次，每一个提交通常只有一个开发者，在需要进行缺陷定位和更正时也更容易找到熟悉这一部分的开发者来开展维护工作；此外，如果开发者在生成一个提交后收到了即时软件缺陷预测系统的提示，就可以快速检查自己编写的代码。通常在这种情况下，开发者对于其编写的代码有着较为清晰的记忆，完成缺陷定位和更正的难度也会更小。

自即时软件缺陷预测被提出以来，研究人员提出了不同的模型。其中较为经典的是从项目版本管理系统中提取出人工特征，并对这些特征使用机器学习分类器进行分类的模型[13-15]。这些特征是由人工设计的属性归纳出来的，例如添加和删除的代码行数、修改的文件数量、相关开发者的经验值等。这些特征忽略了变更代码和提交消息所包含的语义语法信息，可能会导致预测产生错误。例如，假设两个提交都只在同一个文件内进行添加和删除操作，且添加的代码行数和删除代码行数一致，则这两个提交在人工特征中的表现是完全相同的，但执行时却可能会产生完全不同的结果，产生缺陷的可能性也不相同。因此，只依赖人工特征并不能精确预测提交的缺陷倾向。

最近，研究人员提出了一些利用自然语言处理技术从变更代码和提交消息中自动提取特征来进行即时软件缺陷预测的方法，如文献[7-9]。这些方法分析了变更内容的语义语法信息，相较于先前仅使用人工特征的方法获得了一定的性能提升。但此类方法在计算变更代码嵌入时，仅将代码变更行作为带有添加或删除标记的文本串进行处理，没有将其中的变化进行细粒度的表示；而且现有的方法只是将提取出的变更代码特征和提交消息特征进行了简单的拼接，没有对两者进行特征对齐操作。先前的研究表明[16]，这种做法可能会导致模型的性能受到来自低质量提交消息中所包含噪声的影响。此外，现有的方法也未能将包含专业知识的人工特征利用起来。例如，改动过同一个文件的开发者越多，那么这个文件包含缺陷的可能性就越大[17]，只对单次提交的语义语法信息进行分析并不能了解到这些知识。

为了解决以上问题，本文提出了基于细粒度代码表示和特征融合的即时软件缺陷预测方法(JIT-SDP model based on Fine-grained Code Representation and Feature Fusion，FCRFF)。本文的主要贡献如下：

1)设计了一种细粒度的变更代码嵌入计算方式来精确表示代码变更，在嵌入计算中考虑变更行中代码单词的变更方式。

2)设计了一种用于在提交消息和变更代码之间进行特征对齐的模块，以减轻低质量的提交消息中包含的噪声对 JIT-SDP 任务的影响。

3)在 JIT-SDP 的研究中同时考虑了代码变更的具体内容和人工特征。

4)通过实验验证了本文方法的整体有效性，以及每个改进的有效性。

## 2 相关工作

Kim 等在 2008 年首次对代码的变更进行缺陷预测，并在 2013 年将这种软件缺陷预测方法称作即时软件缺陷预测[13]。随后出现的模型以依赖于人工特征的模型为主，这些模型的通常做法是将人工特征输入逻辑回归和随机森林等机器学习分类器中来预测提交消息中包含缺陷的可能性。Kamei 等[13]提出了 LR-JIT，其使用逻辑回归分类器对人工特征进行预测；Jiang 等[18]提出了一种针对每个开发者的不同特点进行缺陷预测的方法；Zhao 等[19]将传统森林模型用于 Android 应用程序的缺陷检测；Chen 等[20]构建了一个工作量感知的即时软件缺陷模型。但是这些研究都忽略了变更行的信息。为此，Kondo 等[21]提出了基于行数的上下文度量指标。然而，基于机器学习的方法仅依赖于人工特征，而仅从这些特征中无法得到变更代码内容中所包含的语义语法信息。

随着深度学习技术的发展，Yang 等[22]提出的 DBN-JIT 使用深度信念网络(Deep Belief Network，DBN)从人工特征中提取高级特征并使用逻辑回归分类器进行缺陷预测；Qiao 等[23]使用人工特征训练了一个三层的神经网络来进行工作量感知的即时软件缺陷预测；Zhu 等[24]将卷积神经网络(CNN)和核极限学习机进行整合，用于即时软件缺陷预测任务。但是这些方法仍没有考虑到变更代码和提交消息中所包含的语义语法信息。

基于提交消息和变更代码内容，Hoang 等[7]提出了 DeepJIT 模型。DeepJIT 是一个端到端的深度学习模型，其使用卷积网络来自动提取特征。它将提交消息和变更代码作为输入，分别传入两个 CNN 模型中以进行特征提取，最后将两个 CNN 模型输出的向量进行拼接，输入全连接层并获得提交消息中包含缺陷的倾向值。随后，Hoang 等[8]提出了 CC2Vec 模型，用于从提交消息中学习变更代码的分布式特征表示，其使用分层注意力网络(Hierarchical Attention Network，HAN)来从变更代码和提交消息文本之间提取层次关系，并将其输出用于改进 DeepJIT 模型的性能[8]。然而，CC2Vec 在训练模型时，需要使用训练集和未标记的测试集进行训练，这违反了“即时”的定义和划分训练集和测试集的用意。Zhou 等[9]在即时软件缺陷预测任务上探究了 CodeBERT[25]的泛化能力，使用 CodeBERT 对变更代码和提交消息进行特征提取，并据此提出名为 CodeBERT4JIT 的 JIT-SDP 模型。

## 3 本文方法

### 3.1 模型概述

图 1 为本文提出的模型的框架图。该模型以提交消息、变更代码和即时软件缺陷预测的 14 项人工特征作为输入，输出该提交缺陷倾向的预测值。模型主要由 4 部分

组成：使用细粒度嵌入计算方法的变更代码特征提取部分、提交消息特征提取部分、特征对齐，以及对人工设计的特征进行融合的部分。对于一个输入样本，首先对其变更代码和提交消息分别进行变更代码特征提取和提交消息特征提取，将输出作为单个部分的特征表示；再将两部分的特征向量作为输入，通过一个特征对齐模块，将提交消息与变更代码的特征进行对齐来减轻提交消息中可能包含的噪声的影响，并提取两者更健壮的联合特征表示；最后与来自于人工特征的信息表示进行融合并输出样本提交的缺陷倾向的预测结果。

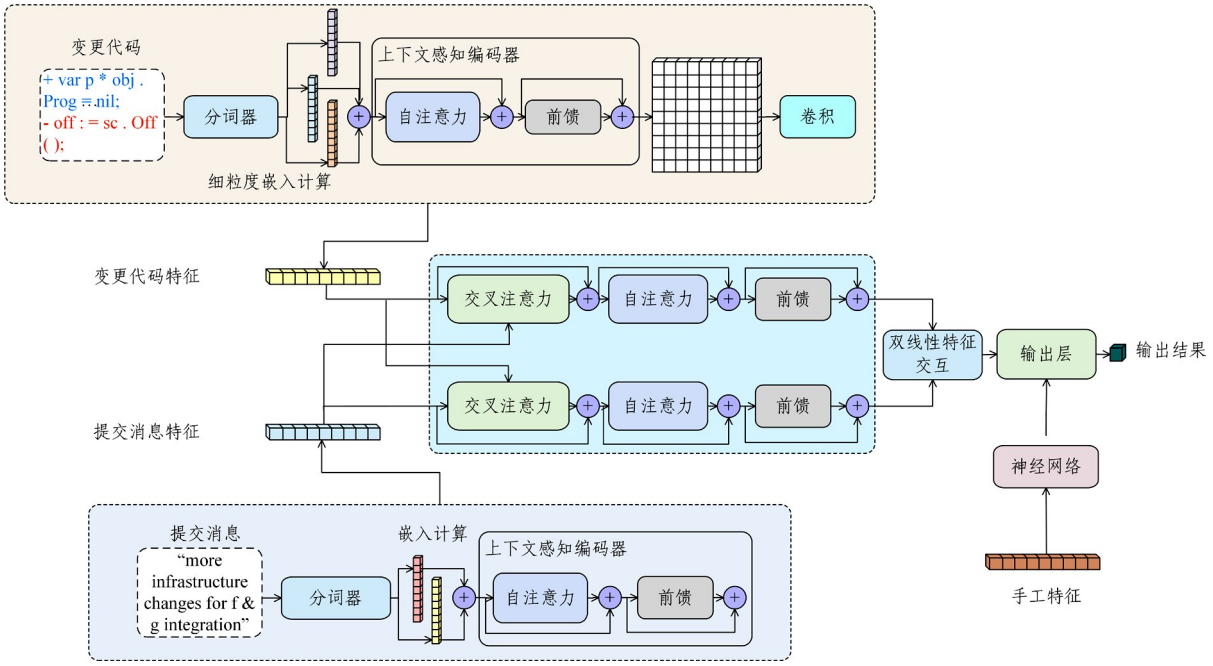


图 1 所提模型的架构图

Fig. 1 Framework of the proposed model

### 3.2 细粒度代码变更表示

图 2 给出了在本文方法中变更代码嵌入的计算过程。

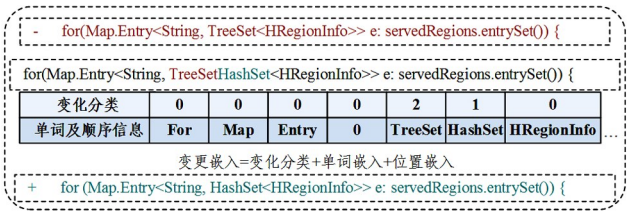


图 2 细粒度的代码更改表示

Fig. 2 Fine-grained code change representation

根据图中所展示的变更代码，首先进行文本串对比，找出分别在什么位置删除或添加了哪些单词。对于该例子，即删除了“TreeSet”，然后增加了“HashSet”。随后根据分析的结果，对这部分的代码进行重新排列，将所有未发生改变的单词和发生变动的单词按照两个变更行中相对位置合并为一行，并为所有代码单词设置变化分类标记。其中，将没有发生改变的单词的标记设置为 0，新增单词的标记设置为 1，删除的单词的标记设置为 2。根据合并后的变化分类串、每个代码单词的词嵌入以及位置嵌入，计算出该单词的嵌入表示。这 3 种嵌入均以独热向量(One-hot Vector)的形式表示。代码单词标记向量用根据字典计算出的独热向量表示；变化标签类型向量根据上述方法计算得到；位置表示向量则由该单词在整合后的变更表示行中的位置计算得到。此处设 $\boldsymbol{r}\in\boldsymbol{R}^{v_1}$ 为代码单词词嵌入向量，$\boldsymbol{t}\in\boldsymbol{R}^{v_2}$ 为变化类型标签向量，$\boldsymbol{p}\in\boldsymbol{R}^{L}$ 为单词位置嵌入向量，其中 $v_1$ 代表词汇表的大小，$v_2$ 代表变化类型标签的数量，$L$ 代表处理文本串的长度，则最终的细粒度的代码变更表示 $E_m$ 可由式(1)－式(4)计算得出。

$$\boldsymbol{E}_r=\boldsymbol{r}\boldsymbol{W}_r \tag{1}$$

$$\boldsymbol{E}_t=\boldsymbol{t}\boldsymbol{W}_t \tag{2}$$

$$\boldsymbol{E}_p=\boldsymbol{p}\boldsymbol{W}_l \tag{3}$$

$$\boldsymbol{E}_m=\boldsymbol{E}_r+\boldsymbol{E}_t+\boldsymbol{E}_p \tag{4}$$

其中，$\boldsymbol{W}_r\in\boldsymbol{R}^{v_1\times d}$，$\boldsymbol{W}_t\in R^{v_2\times d}$ 和 $\boldsymbol{W}_l\in R^{L\times d}$ 分别代表 3 个可训练的矩阵，$d$ 代表嵌入的维度大小。

### 3.3 特征提取模块

在特征提取模块中，本文方法使用编码器对输入的变更代码和提交消息进行特征提取。为了使编码器更加了解项目中代码的上下文信息，与先前直接使用预训练模型进行特征提取的研究[9]不同，在训练提出的 FCRFF 模型前，本文方法先对编码器进行自监督训练来得到一个上下文感知的编码器。

如图 3 所示，在训练编码器的阶段，我们为编码器搭配了一个解码器模块，将二者组合形成一个神经网络并通过反向传播进行优化。在训练时，将添加了随机噪声的细粒度变更代码嵌入表示作为输入，让网络学习如何重建未添加噪声的变更代码表示。添加噪声的主要步骤如下。

1)确定噪声的长度。本文从 $\lambda=2$ 的泊松分布中随机抽取该长度。

2)替换单词及顺序信息串中的内容。随机选择位置，将等同于噪声长度的内容删除并替换为单个 MASK 标志，且被替换掉的每个单词的变化分类标志保持不变。

3)重新按照式(1)－式(4)计算代码更改表示。

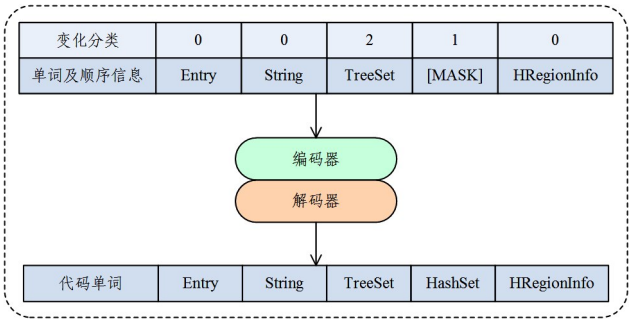


图 3 自监督的方式训练上下文感知的编码器

Fig. 3 Training context-aware encoder by self-supervised way

设 $\boldsymbol{X}'$ 为添加噪声后的变更代码嵌入表示，$\boldsymbol{X}$ 为添加原有的变更代码嵌入表示，$N$ 代表训练集中的数据样本数，$i$ 代表第 $i$ 个数据样本，$j$ 代表单个变更表示中的第 $j$ 位内容，则在训练上下文感知的编码器的过程中，目标函数可表示为式(5)：

$$\mathcal{L}_1=-\frac{1}{N}\sum_{i=1}^{N}\sum_{j}\log\mathcal{P}(\boldsymbol{X}_{ij}\mid\boldsymbol{X}_i') \tag{5}$$

通过自监督的方式训练得到上下文感知的编码器后，将该编码器作为提取变更代码和提交消息特征的模块使用。

### 3.4 特征对齐模块

虽然上述细粒度代码变更表示和上下文感知的编码器模块可以有效地提取出提交消息和变更代码的单项特征，但由于提交消息可能包含噪声[16]，因此二者的特征需要进行对齐操作来减轻噪声对模型性能的影响。

如图 4 所示，特征对齐模块的输入是先前分别提取出的提交消息特征向量和变更代码的特征向量。特征对齐模块由多个包含双向交叉注意力的子层组成，每个双向交叉注意力子层包含两个单向的交叉注意力子层、两个自注意力子层和一个前馈子层，其中一个双向交叉注意力子层负责从提交消息中关联特征到变更代码，另一个双向交叉注意力子层负责从变更代码中关联特征到提交消息。第 $k$ 层的双向交叉注意力子层的输入是第$(k-1)$层的输出。在单向的交叉注意力子层中，向特征 $A$ 的特征向量 $\hat{\boldsymbol{a}}^{k-1}$ 关联另一特征 $B$ 的特征向量 $\hat{\boldsymbol{b}}^{k-1}$ 的方法如式(6)－式(11)所示：

$$\hat{\boldsymbol{Q}}_A=\hat{\boldsymbol{a}}^{k-1}\boldsymbol{W}_h^Q \tag{6}$$

$$\hat{\boldsymbol{K}}_B=\hat{\boldsymbol{b}}^{k-1}\boldsymbol{W}_h^K \tag{7}$$

$$\hat{\boldsymbol{V}}_B=\hat{\boldsymbol{b}}^{k-1}\boldsymbol{W}_h^V \tag{8}$$

$$chead_h=Attention(\hat{\boldsymbol{Q}}_A,\hat{\boldsymbol{K}}_B,\hat{\boldsymbol{V}}_B) \tag{9}$$

$$chead=concat(chead_1,\cdots,chead_H) \tag{10}$$

$$CrossAtt_{B\to A}(\hat{\boldsymbol{a}}^{k-1},\hat{\boldsymbol{b}}^{k-1})=chead\,\boldsymbol{W}^O \tag{11}$$

其中，$\hat{\boldsymbol{Q}}_A$ 表示特征 $A$ 的查询向量，$\hat{\boldsymbol{K}}_B$ 表示特征 $B$ 的键向量，$\hat{\boldsymbol{V}}_B$ 表示特征 $B$ 的值向量，$\boldsymbol{W}_h^Q$，$\boldsymbol{W}_h^K$，$\boldsymbol{W}_h^V$，$\boldsymbol{W}^O$均表示权重矩阵，Attention 表示进行缩放点积注意力计算[26]，Concat 表示对向量进行拼接。对变更代码特征 $\hat{\boldsymbol{c}}_i^{k-1}$ 和提交消息特征 $\hat{\boldsymbol{m}}_j^{k-1}$ 进行双向交叉注意力特征关联的方法如式(12)和式(13)所示：

$$\hat{\boldsymbol{c}}^k=CrossAtt_{M\to C}(\hat{\boldsymbol{c}}^{k-1},\hat{\boldsymbol{m}}^{k-1}) \tag{12}$$

$$\hat{\boldsymbol{m}}^k=CrossAtt_{C\to M}(\hat{\boldsymbol{m}}^{k-1},\hat{\boldsymbol{c}}^{k-1}) \tag{13}$$

其中，$\hat{\boldsymbol{c}}^k$ 和 $\hat{\boldsymbol{m}}^k$ 是交叉注意力子层的输出。

接着，将经过交叉注意力子层处理的特征输入自注意力子层，以便在对齐操作后，进一步构建深度双向特征表示。具体处理方式如式(14)和式(15)所示：

$$\tilde{\boldsymbol{m}}^k=SelfAtt(\hat{\boldsymbol{m}}^k) \tag{14}$$

$$\tilde{\boldsymbol{c}}^k=SelfAtt(\hat{\boldsymbol{c}}^k) \tag{15}$$

其中，$\tilde{\boldsymbol{c}}^k$ 和 $\tilde{\boldsymbol{m}}^k$ 是自注意力子层的输出，$SelfAtt$ 表示进行自注意力计算。在自注意力子层后，$\tilde{\boldsymbol{c}}^k$ 和 $\tilde{\boldsymbol{m}}^k$ 还会被输入前馈子层中以获得第 $k$ 个特征对齐层的最终输出。对于最后一层双向交叉注意力子层的输出，本文使用池化层和一个非线性激活函数来进行处理，分别得到两个进行特征对齐后的提交消息向量 $\boldsymbol{C}$ 和变更代码特征向量 $\boldsymbol{M}$。

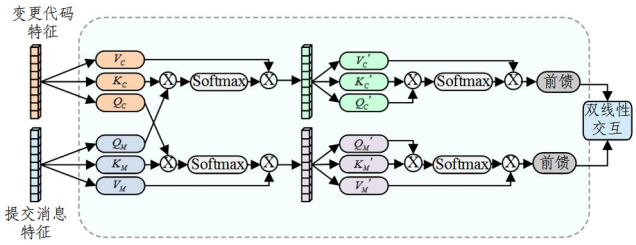


图 4 特征对齐模块

Fig. 4 Feature alignment module

受到 Huang 等[27]的启发，本文使用双线性特征交互层来对两个对齐后的特征进行融合。双线性特征交互层的思想如图 5 所示。我们设置了一个可训练的 $K$ 阶矩阵 $\boldsymbol{W}$ 用于融合两个模态的信息，首先将提交代码特征向量 $\boldsymbol{C}$ 与 $\boldsymbol{W}$ 进行内积乘法运算，再使用 Hadamard 乘法将结果与提交消息特征 $\boldsymbol{M}$ 相乘。设 $\boldsymbol{F}$ 为最终的特征向量，$i,j=[1,K]$，则运算的方式如式(16)所示：

$$\boldsymbol{F}_{ij}=\boldsymbol{C}_i\cdot\boldsymbol{W}_{ij}\odot\boldsymbol{M}_j \tag{16}$$

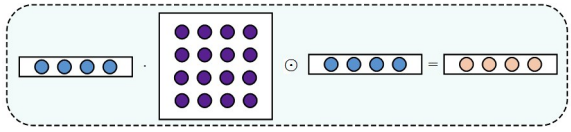

图 5 双线性特征交互层

Fig. 5 Bilinear feature interaction layer

### 3.5 人工 JIT-SDP 特征

为了充分利用先前 JIT-SDP 研究[13]中所设计的人工特征，本文提出的方法也考虑了代码提交的人工特征。具体而言，对于每一个样本提交，有 14 个人工机器学习特征进行输入。这 14 个特征如表 1 所列。对于这 14 项特征，本文使用三层神经网络进行特征提取，得到一组高维特征，用于和特征对齐模块的输出进行拼接。最后将拼接的结果输入全连接层，通过式(17)所示的 Sigmoid 激活函数得到最终的预测结果。

$$S(x)=\frac{1}{1+e^{-x}} \tag{17}$$

表 1 即时软件缺陷预测的 14 项人工特征

Table 1 14 artificial features of JIT-SDP

| 名称 | 说明 |
|---|---|
| NS | 修改的子系统数量 |
| ND | 修改的目录数量 |
| NF | 修改的文件数量 |
| Entropy | 描述修改的代码的分布情况 |
| LA | 增加的代码行数 |
| LD | 删除的代码行数 |
| LT | 在变更前,文件中的行数 |
| FIX | 标记此次提交是否修复了一个缺陷 |
| NDEV | 参与此次提交的开发者数量 |
| AGE | 两次变更之间的平均时间 |
| NUC | 变更文件中不同变更的数量 |
| EXP | 开发者与项目的相关经验 |
| REXP | 开发者最近参与该项目开发的程度 |
| SEXP | 开发者参与变更的子系统开发的程度 |

## 4 实验与性能分析

### 4.1 数据集收集

由于现有的数据集要么只包含变更代码和提交消息,要么只包含人工机器学习特征,为了验证所提出的模型,我们不得不重新收集新的数据集。参考先前研究中所测试的项目,我们在一些比较有代表性的开源项目上收集了一些数据集用于评估本文提出的模型。首先在 DeepJIT 和 CC2Vec 中出现的 Qt 和 OpenStack 项目上收集了数据集。Qt 是一个使用 C++ 编写的应用程序开发框架,被广泛用于创建图形用户界面和跨平台应用程序;OpenStack 是一个开源的云计算基础设施软件项目,是世界上最活跃的开源项目之一。为了在更多的数据集上进行测试,我们同样收集了来自于 Gerrit, Go, Platform 和 JDT 的数据。Gerrit 是一个被许多商业和开源项目用于代码审查的工具,其使用 Java 进行编写;Go 是由 Google 开发的一门编程语言,得到了大量的实际应用,其主要使用 Go 语言本身进行编写;Platform 和 JDT 均由 Eclipse 开发,在先前的基于人工特征的缺陷预测研究[13,28-29]中被广泛采用,主要使用 Java 编写。此外,我们还对最新的 JIT-SDP 数据集 ApacheJIT[30]进行了增强,收集到了既包含变更代码和提交消息,也包含 14 项人工特征的 ApacheJIT 数据集。收集到的数据集信息如表 2 所列。

表 2 所收集的项目数据的统计情况

Table 2 Statistics of the collected dataset

| 项目 | 提交个数 | 缺陷占比/% | 语言 |
|---|---|---|---|
| Qt | 52000 | 17.4 | C++ |
| OpenStack | 52000 | 36.3 | Java |
| JDT | 13348 | 41.2 | Java |
| Gerrit | 25081 | 10.6 | Java |
| Go | 43311 | 40.5 | Go |
| Platform | 31364 | 41.1 | Java |
| ApacheJIT | 106674 | 26.5 | Java |

在数据收集的过程中,我们参照了 Keshavarz 等[30]的做法,使用了 PyDriller[31]框架中所采用的 SZZ 算法来标记缺陷提交并进行过滤操作。对于 ApacheJIT,本文收集的数据集的时间跨度与 Keshavarz 等[30]的方法的时间跨度相同。对于其他 6 个数据集,我们收集的数据集的时间跨度为 2012—2020 年。最终得到了 7 个共计包含 359778 个提交的数据集。

### 4.2 实验设置

本文提出的模型的训练过程分为两个阶段。第一阶段是将编码器搭配一个解码器组成一个神经网络来进行自监督训练,使编码器学习上下文信息。在这一阶段,我们使用 CodeT5-base[32]网络的权重初始化模型,在训练数据集上迭代 10 次。使用 AdamW[33]优化算法作为优化器进行训练,学习率为 $1\times10^{-5}$,批次大小设置为 16。第二阶段训练整个网络时,交叉注意力模块中隐藏层的维度为 768,多头注意力统一使用 8 头注意力。每次训练时在数据集上迭代 15 次。

所有实验均在同一台服务器上完成,其配备了 Intel(R) Xeon(R) Platinum 8255C CPU @ 2.50GHz,40GB 内存和一个 32GB 显示内存的 NVIDIA Tesla V100。

### 4.3 评价指标

考虑到数据集中数据的不平衡性,我们倾向于使用独立于阈值的评价指标来评价模型的性能。首先使用的是在先前的即时软件缺陷预测研究[7-9]中被广泛使用的受试者工作特征曲线下面积(AUC-ROC)。受试者工作特征曲线(ROC)是一条以假阳性率(False Positive Rate,FPR)为横坐标、以真阳性率(True Positive Rate,TPR)为纵坐标的曲线,通过设置不同的阈值来展示正确分类的阳性样本与被错误分类的阴性样本率之间的关系,进而评估二分类模型的性能。AUC-ROC 指标的取值范围为[0,1],AUC-ROC 值越靠近 1,说明模型的性能越好。

除了 AUC-ROC 以外,我们还采用了查准率-查全率曲线下面积(AUC-Precision Recall Curve)指标进行模型的性能评估。在数据不平衡的情况下,AUC-ROC 曲线可能会对模型的性能产生过于乐观的评估[34]。先前 Saito 等[35]的研究表明,在不平衡的数据集上训练二分类模型时,查准率-查全率曲线比受试者工作特征曲线能够更准确地表达模型性能。

本文收集的数据集中,每个项目中的正负样本都是不平衡的,例如在 Qt 项目中,标记为包含缺陷的占比只有17.4%。因此,本文实验中使用 AUC-PRC 指标。与 AUC-ROC 指标相同,AUC-PRC 指标的计算过程中也不需要手动设置阈值。

为了更充分地进行模型性能的对比,本文实验中还使用了 F1-Score 作为评价指标。F1-Score 是评估分类任务算法的常用评价指标之一,也是先前软件工程领域相关研究中较为常见的评价指标[18,36]。F1-Score 平衡了模型的精确率和召回率,通过该指标可衡量模型性能的两个方面。

### 4.4 性能对比

本文选择了即时软件缺陷预测领域的一些模型进行对比,包括 DeepJIT[7], CC2Vec[8], DBN-JIT[22], LR-JIT[13] 和 CodeBERT4JIT[9]。下面简要介绍这些模型。

1)DeepJIT[7]通过卷积神经网络从提交消息和变更代码中提取特征并进行 JIT-SDP 任务,被广泛用于 JIT-SDP 的研究中作为对比基准。

2)CC2Vec[8]提取了变更代码的分布式特征,结合 DeepJIT 的网络进行 JIT-SDP 任务。

3)DBN-JIT[22]通过带有逻辑回归分类器的深度信念

网络从人工特征中提取高维特征用于 JIT-SDP 任务。

4）LR-JIT[13] 使用逻辑回归分类器对人工特征进行分类，输出样本的缺陷倾向。

5）CodeBERT4JIT[9] 是 DeepJIT 的一个变体，其使用 CodeBERT 预训练模型替换了 DeepJIT 中的卷积神经网络。

表 3－表 5 分别展示了在 3 个评价指标上的实验对比结果。可以观察到，DeepJIT，CC2Vec 和 CodeBERT4JIT 在 7 个测试的项目上的结果均优于仅以人工特征作为输入的 LR-JIT 和 DBN-JIT。这是因为 LR-JIT 和 DBN-JIT 无法根据人工特征获得变更内容的语义和语法信息，从而影响了模型性能。

表 3　AUC-ROC 指标上的实验结果

Table 3　Experimental results on AUC-ROC metric

| 模型 | Qt | OpenStack | JDT | Platform | Gerrit | Go | ApacheJIT |
|---|---|---|---|---|---|---|---|
| FCRFF | **0.7628** | **0.7562** | **0.6617** | **0.8067** | **0.7221** | **0.7678** | **0.8669** |
| DeepJIT | 0.7011 | 0.7148 | 0.6239 | 0.7574 | 0.6977 | 0.6631 | 0.7828 |
| CC2Vec | 0.7046 | 0.7101 | 0.6387 | 0.7754 | 0.6948 | 0.6679 | 0.7954 |
| CodeBERT4JIT | 0.7142 | 0.7260 | 0.6128 | 0.7401 | 0.6936 | 0.6751 | 0.8162 |
| LR-JIT | 0.6802 | 0.6814 | 0.5761 | 0.6373 | 0.6696 | 0.6227 | 0.7726 |
| DBN-JIT | 0.6825 | 0.6858 | 0.5859 | 0.6703 | 0.6823 | 0.6561 | 0.7763 |

表 4　AUC-PRC 指标上的实验结果

Table 4　Experimental results on AUC-PRC metric

| 模型 | Qt | OpenStack | JDT | Platform | Gerrit | Go | ApacheJIT |
|---|---|---|---|---|---|---|---|
| FCRFF | **0.3075** | **0.3527** | **0.6379** | **0.6016** | **0.1505** | **0.6890** | **0.5752** |
| DeepJIT | 0.2482 | 0.3353 | 0.5516 | 0.5665 | 0.1465 | 0.5480 | 0.4703 |
| CC2Vec | 0.2604 | 0.3348 | 0.5648 | 0.5664 | 0.1304 | 0.5546 | 0.5272 |
| CodeBERT4JIT | 0.2675 | 0.3176 | 0.5716 | 0.5057 | 0.1297 | 0.5986 | 0.4896 |
| LR-JIT | 0.2396 | 0.3098 | 0.5449 | 0.4088 | 0.1282 | 0.5394 | 0.4403 |
| DBN-JIT | 0.2328 | 0.3003 | 0.5426 | 0.4857 | 0.1274 | 0.4966 | 0.4667 |

表 5　F1-Score 指标上的实验结果

Table 5　Experimental results on F1-Score metric

| 模型 | Qt | OpenStack | JDT | Platform | Gerrit | Go | ApacheJIT |
|---|---|---|---|---|---|---|---|
| FCRFF | **0.3552** | **0.3624** | **0.6663** | **0.5886** | **0.2018** | **0.6541** | **0.4609** |
| DeepJIT | 0.3265 | 0.3289 | 0.6345 | 0.5419 | 0.1764 | 0.6146 | 0.4362 |
| CC2Vec | 0.3214 | 0.3256 | 0.6348 | 0.5564 | 0.1804 | 0.6193 | 0.4385 |
| CodeBERT4JIT | 0.3141 | 0.3229 | 0.6101 | 0.5601 | 0.1824 | 0.6127 | 0.4287 |
| LR-JIT | 0.2855 | 0.3020 | 0.5846 | 0.4761 | 0.1548 | 0.5132 | 0.4085 |
| DBN-JIT | 0.2675 | 0.2911 | 0.5501 | 0.4628 | 0.1618 | 0.5109 | 0.4139 |

此外，本文模型 FCRFF 在 3 个指标和 7 个数据集上均优于对比的基线。本文方法与基于人工特征的方法相比，在 AUC-ROC 指标上提升了 5.8%～20%，在 AUC-PRC 指标上提升了 13.8%～28.3%，在 F1-Score 指标上提升了 11.3%～24.4%；与基于深度学习提取语义语法的方法相比，在 AUC-ROC 指标上提升了 3.4%～13.7%，在 AUC-PRC 指标上提升了 2.7%～15.1%，在 F1-Score 指标上提升了 4.9%～10.7%。这些结果证明了本文针对先前研究存在的问题所提出的改进措施是有效的。一方面，FCRFF 进行了细粒度的变更代码嵌入计算，将变更代码的表示粒度由行级别降低为词汇级别，这有助于模型细粒度地学习发生变更的词汇、对应的变更类型和位置信息，从而提升模型学习变更代码的能力；另一方面，为了减轻提交消息中的噪声的影响，FCRFF 还对提交消息特征和变更代码特征进行了特征对齐的操作，并提取了更健壮的联合特征。此外，FCRFF 将语义语法信息和人工特征中的专业知识相结合，与先前的方法相比，能综合地提取到提交的信息，从而使模型的泛化性能更强。

### 4.5　消融实验

为了更好地探究本文所提出的不同改进的有效性，我们还开展了消融实验。本文提出的 FCRFF 模型的改进之处主要在于：1）设计了一种细粒度的变更代码嵌入计算方法来更好地表示变更操作；2）设计了一个特征对齐模块，以减轻提交消息中可能存在的噪声的影响；3）将人工特征与提交的语义语法信息同时运用到 JIT-SDP 任务中。本文的消融实验分别针对这 3 个方面开展。

本文设计了 4 种变体模型，相比 FCRFF，分别是：1）去掉细粒度变更代码表示，将提交中变更代码按照与 CodeBERT4JIT[9] 相同的方式进行处理的模型，表示为 NoFCR；2）去掉特征对齐模块，不考虑提交消息中可能包含噪声的影响的模型，表示为 NoFA；3）去掉人工特征，不考虑人工特征中所包含的专业知识的模型，表示为 NoHF；4）没有任何改进的模型，表示为 None，此时模型等同于 CodeBERT4JIT。

表 6－表 8 分别列出了在 3 个评价指标上的消融实验结果。通过对比 FCRFF 和 NoFCR 的实验结果可知，在 3 个指标上，FCRFF 均有明显的提升，这证明了所提出的细粒度变更代码嵌入计算方法对于 JIT-SDP 任务是有效的。现有的 JIT-SDP 模型在处理变更代码行时，无法让模型有效地捕捉变更的细节，而本文提出的变更代码表示可以通过细化词组变更类型来细粒度地告诉模型变更发生的具体位置。

表 6 AUC-ROC 指标上的消融实验结果

Table 6 Ablation experiment results on AUC-ROC metric

| 模型 | Qt | OpenStack | JDT | Platform | Gerrit | Go | ApacheJIT |
|---|---|---|---|---|---|---|---|
| FCRFF | **0.7628** | **0.7562** | **0.6617** | **0.8067** | **0.7221** | **0.7678** | **0.8669** |
| NoFCR | 0.7391 | 0.7328 | 0.6283 | 0.7756 | 0.7017 | 0.7382 | 0.8312 |
| NoFA | 0.7339 | 0.7304 | 0.6387 | 0.7745 | 0.7138 | 0.7198 | 0.8351 |
| NoHF | 0.7505 | 0.7359 | 0.6548 | 0.7862 | 0.7185 | 0.7541 | 0.8410 |
| None | 0.7142 | 0.7260 | 0.6128 | 0.7401 | 0.6936 | 0.6751 | 0.8162 |

表 7 AUC-PRC 指标上的消融实验结果

Table 7 Ablation experiment results on AUC-PRC metric

| 模型 | Qt | OpenStack | JDT | Platform | Gerrit | Go | ApacheJIT |
|---|---|---|---|---|---|---|---|
| FCRFF | **0.3075** | **0.3527** | **0.6379** | **0.6016** | **0.1505** | **0.6890** | **0.5752** |
| NoFCR | 0.2870 | 0.3406 | 0.6073 | 0.5204 | 0.1368 | 0.6656 | 0.5378 |
| NoFA | 0.2788 | 0.3193 | 0.6201 | 0.5637 | 0.1422 | 0.6517 | 0.5507 |
| NoHF | 0.2956 | 0.3447 | 0.6293 | 0.5865 | 0.1436 | 0.6716 | 0.5669 |
| None | 0.2675 | 0.3176 | 0.5716 | 0.5057 | 0.1297 | 0.5986 | 0.4896 |

表 8 F1-Score 指标上的消融实验结果

Table 8 Ablation experiment results on F1-Score metric

| 模型 | Qt | OpenStack | JDT | Platform | Gerrit | Go | ApacheJIT |
|---|---|---|---|---|---|---|---|
| FCRFF | **0.3552** | **0.3624** | **0.6663** | **0.5886** | **0.2018** | **0.6541** | **0.4609** |
| NoFCR | 0.3246 | 0.3386 | 0.6388 | 0.5798 | 0.1886 | 0.6479 | 0.4418 |
| NoFA | 0.3286 | 0.3315 | 0.6201 | 0.5781 | 0.1862 | 0.6380 | 0.4484 |
| NoHF | 0.3497 | 0.3447 | 0.6493 | 0.5852 | 0.1950 | 0.6518 | 0.4587 |
| None | 0.3141 | 0.3229 | 0.6101 | 0.5601 | 0.1824 | 0.6127 | 0.4287 |

通过对比 FCRFF 和 NoFA 的实验结果可知，FCRFF 在 3 个指标上均有提升，这证明了本文提出的特征对齐模块的有效性。先前的研究表明，提取提交消息中的内容有助于 JIT-SDP 任务的进行[16]，但尚未有研究对变更代码与提交消息进行特征对齐来减少提交消息中的噪声信息。本文设计的特征对齐模块解决了上述问题，也验证了先前的研究中所提出的理论。

通过对比 FCRFF 和 NoHF 的实验结果可知，FCRFF 在 3 个指标上均有提升，但相较于 NoFCR 和 NoFA 提升幅度较小。一方面，这证明了将人工特征与提交内语义语法信息一同考虑有助于进一步提升 JIT-SDP 模型的性能；另一方面也说明，相比人工特征，对语义语法信息的进一步挖掘在 JIT-SDP 任务中更加重要。

此外，通过对比 FCRFF 和 None 的实验结果可知，FCRFF 也有较为显著的性能提升，这表明了本文提出的 3 种方法均可为 JIT-SDP 模型贡献性能提升。

**结束语** 本文提出了一种基于细粒度代码表示和特征融合的即时软件缺陷预测方法，相较于先前的研究，本文提出的方法具有以下优势：1）使用了一种细粒度的变更代码嵌入计算方式，来具体地表示添加行相较于删除行中发生变化的位置；2）设计了一个特征对齐模块，用于减轻提交消息中包含的噪声对 JIT-SDP 任务的影响；3）同时考虑了人工特征以及变更代码和提交消息中的语义语法信息，充分利用二者所包含的不同种类信息进行 JIT-SDP 任务。实验证明，本文提出的方法优于现有方法，且每个改进都是有效的。后续研究将探索如何将所提出的方法运用于跨项目背景下的即时软件缺陷预测任务。

## 参 考 文 献


[1] WANG S, LIU T, NAM J, et al. Deep Semantic Feature Learning for Software Defect Prediction [J]. IEEE Transactions on Software Engineering, 2020, 46(12): 1267-1293.

[2] NUCCI D D, PALOMBA F, ROSA G D, et al. A Developer Centered Bug Prediction Model [J]. IEEE Transactions on Software Engineering, 2018, 44(1): 5-24.

[3] SHAO Y, LIU B, WANG S, et al. A novel software defect prediction based on atomic class-association rule mining [J]. Expert Systems with Applications, 2018, 114: 237-254.

[4] ASANO T, TSUNODA M, TODA K, et al. Using Bandit Algorithms for Project Selection in Cross-Project Defect Prediction [C]// Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2021: 649-653.

[5] ZHAO Y, WANG Y, ZHANG D, et al. Eliminating the high false-positive rate in defect prediction through BayesNet with adjustable weight [J]. Expert Systems, 2022, 39(6): e12977.

[6] HATA H, MIZUNO O, KIKUNO T. Bug prediction based on fine-grained module histories [C] // Proceedings of the 2012 34th International Conference on Software Engineering (ICSE). Zurich, Switzerland: IEEE, 2012: 200-210.

[7] HOANG T, DAM H K, KAMEI Y, et al. DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction [C]// Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). Montreal, QC, Canada: IEEE, 2019: 34-45.

[8] 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. Seoul, South Korea: Association for Computing Machinery, 2020: 518-529.

[9] ZHOU X, HAN D, LO D. Assessing Generalizability of CodeBERT [C]// Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). Lux-

embourg:IEEE,2021:425-436.
[10] D'AMBROS M,LANZA M,ROBBES R. Evaluating defect prediction approaches:a benchmark and an extensive comparison [J]. Empirical Software Engineering,2012,17(4):531-577.
[11] TURHAN B,MENZIES T,BENER A B,et al. On the relative value of cross-company and within-company data for defect prediction [J]. Empirical Software Engineering,2009,14(5):540-578.
[12] ZHAO Y H,DAMEVSKI K,CHEN H. A Systematic Survey of Just-in-Time Software Defect Prediction[J]. ACM Computing Surveys,2023,55(10):1. 1-1. 35.
[13] KAMEI Y,SHIHAB E,ADAMS B,et al. A large-scale empirical study of just-in-time quality assurance [J]. IEEE Transactions on Software Engineering,2012,39(6):757-773.
[14] SHIVAJI S,WHITEHEAD E J,AKELLA R,et al. Reducing features to improve code change-based bug prediction [J]. IEEE Transactions on Software Engineering,2012,39(4):552-569.
[15] RAJBAHADUR G,WANG S,KAMEI Y,et al. The Impact of Using Regression Modelsto Build Defect Classifiers [C]// Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories(MSR). Buenos Aires,Argentina:IEEE,2017:135-145.
[16] ZENG Z,ZHANG Y,ZHANG H,et al. Deep just-in-time defect prediction:How Far Are We? [C]// Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. Virtual,Denmark:Association for Computing Machinery,2021:427-438.
[17] MATSUMOTO S,KAMEI Y,MONDEN A,et al. An analysis of developer metrics for fault prediction [C]// Proceedings of the 6th International Conference on Predictive Models in Software Engineering. Timişoara,Romania:Association for Computing Machinery,2010:1-9.
[18] JIANG T,TAN L,KIM S. Personalized defect prediction [C]// Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering(ASE). Silicon Valley,CA,USA:IEEE,2013:279-289.
[19] ZHAO K,XU Z,ZHANG T,et al. Simplified Deep Forest Model Based Just-in-Time Defect Prediction forAndroid Mobile Apps [J]. IEEE Transactions on Reliability,2021,70(2):848-859.
[20] CHEN X,ZHAO Y,WANG Q,et al. MULTI:Multi-objective effort-aware just-in-time software defect prediction [J]. Information and Software Technology,2018,93:1-13.
[21] KONDO M,GERMAN D M,MIZUNO O,et al. The impact of context metrics on just-in-time defect prediction [J]. Empirical Software Engineering,2020,25(1):890-939.
[22] YANG X,LO D,XIA X,et al. Deep Learning for Just-in-Time Defect Prediction [C]// Proceedings of the 2015 IEEE International Conference on Software Quality,Reliability and Security. Vancouver,BC,Canada:IEEE,2015:17-26.
[23] QIAO L,WANG Y. Effort-aware and just-in-time defect prediction with neural network [J]. PLoS One,2019,14(2):1-19.
[24] ZHU K,YING S,ZHANG N,et al. Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network [J]. Journal of Systems and Software,2021,180:111026.
[25] FENG Z,GUO D,TANG D,et al. CodeBERT:A Pre-Trained Model for Programming and Natural Languages [C]// Proceedings of the Association for Computational Linguistics:EMNLP 2020. Online:Association for Computational Linguistics,2020:1536-1547.
[26] VASWANI A,SHAZEER N,PARMAR N,et al. Attention Is All You Need [J]. Advances in Nutrition,2017,30:1-11.
[27] HUANG T,ZHANG Z,ZHANG J. FiBiNET:combining feature importance and bilinear feature interaction for click-through rate prediction [C]// Proceedings of the 13th ACM Conference on Recommender Systems. Copenhagen,Denmark:Association for Computing Machinery,2019:169-177.
[28] KAMEI Y,FUKUSHIMA T,MCINTOSH S,et al. Studying just-in-time defect prediction using cross-project models [J]. Empirical Software Engineering,2016,21(5):2072-2106.
[29] YANG X,LO D,XIA X,et al. TLEL:A two-layer ensemble learning approach for just-in-time defect prediction [J]. Information and Software Technology,2017,87:206-220.
[30] KESHAVARZ H,NAGAPPAN M. ApacheJIT:a large dataset for just-in-time defect prediction [C]// Proceedings of the 19th International Conference on Mining Software Repositories. Pittsburgh,Pennsylvania:Association for Computing Machinery,2022:191-195.
[31] SPADINI D,ANICHE M,BACCHELLI A. PyDriller:Python framework for mining software repositories [C]// Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Lake Buena Vista,FL,USA:Association for Computing Machinery,2018:908-911.
[32] WANG Y,WANG W,JOTY S,et al. CodeT5:Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation [J]. arXiv. 2109. 00859,2021.
[33] LOSHCHILOV I,HUTTER F. Decoupled Weight Decay Regularization [C]// Proceedings ofthe 7th International Conference on Learning Representations. New Orleans,LA,USA:OpenReview. net,2019.
[34] ZHOU X,HAN D,LO D. Simple or Complex? Together for a More Accurate Just-In-Time Defect Predictor [C]// Proceedings of the 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC). Pittsburgh, PA, USA: IEEE, 2022:229-240.
[35] SAITO T,REHMSMEIER M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets [J]. PLoS One,2015,10(3):1-21.
[36] GARCIA H V,SHIHAB E. Characterizing and predicting blocking bugs in open source projects [C]// Proceedings of the 11th Working Conference on Mining Software Repositories. Hyderabad,India:Association for Computing Machinery,2014:72-81.

**ZHU Xiaoyan**, born in 1982, Ph.D, associate professor, Ph. D supervisor, is a member of CCF (No. 73027M). Her main research interests include machine learning and data mining.

(责任编辑:杨雪敏)