

基于多维代码提交表示的即时缺陷预测研究

徐 洋¹, 李英玲^{1,2*}

(1. 西南民族大学计算机与人工智能学院, 四川 成都 610041; 2. 西南民族大学计算机系统国家民委重点实验室, 四川 成都 610041)

摘 要: 即时缺陷预测(JIT-DP)是软件缺陷预测领域的一个重要分支,旨在代码提交阶段及时识别潜在缺陷,从而降低软件维护成本并提升产品质量。然而,现有方法多侧重于结合专家特征与语义特征,未能充分利用代码语法结构中的深层约束与依赖信息,尤其是普遍忽视了代码变更在时间维度上的连续演化规律,导致对复杂缺陷模式的表征能力有限。因此,提出一种基于多维代码提交表示的即时缺陷预测方法(EvoTSS-JIT):首先,对原始提交数据进行预处理,构建时序提交序列并采用不同Git命令提取完整的代码上下文与差异信息;其次,利用LSTM-Transformer混合模型与多尺度融合机制提取时序演化规律,并分别结合微调的CodeBERT模型与GraphCodeBERT模型,以提取当前提交的意图与语法结构特征,从而构建多维特征表示;最后,将融合后的多维特征输入XGBoost分类器进行训练与预测,并通过在验证集上搜索最优阈值来缓解类别不平衡问题。实验结果表明,在5个项目数据集上,EvoTSS-JIT相较于3种当前最优方法,在F1值、AUC-ROC、AUC-PR和召回率四项评价指标上分别平均提升5.35%~30.13%、1.67%~10.23%、5.48%~14.87%和12.02%~83.68%,验证了该方法的有效性。

关键词: 即时缺陷预测;多维特征表示;时序特征;LSTM-Transformer混合模型

中图分类号:TP311.5

文献标志码:A

文章编号:2095-4271(2026)05-0560-09

Just-in-time defect prediction based on multi-dimensional representation of code commits

Xu Yang¹, Li Yingling^{1,2*}

(1.College of Computer Science and Artificial Intelligence, Southwest Minzu University, Chengdu 610041, China; 2. Key Laboratory for Computer Systems of State Ethnic Affairs Commission, Southwest Minzu University, Chengdu 610041, China)

Abstract: Just-in-time defect prediction (JIT-DP) is an important branch of software defect prediction that aims to identify potential defects at the code commit stage, thus reducing software maintenance costs and improving product quality. However, existing approaches have mainly focused on combining expert-handcrafted features with semantic features and have not fully exploited the constraints and dependency information embedded in code syntactic structures. More importantly, the continuous temporal evolutionary patterns of code changes were often overlooked, which limited the ability to represent complex defect patterns. Therefore, a multi-dimensional code commit representation-based Just-in-time defect prediction approach, EvoTSS-JIT, was proposed. First, the raw commit data were extended and preprocessed to construct temporal commit sequences, and different Git commands were employed to extract complete code contexts and difference information. Second, a hybrid LSTM - Transformer model with a multi-scale fusion mechanism was used to extract temporal evolution patterns, while a fine-tuned

收稿日期:2026-01-14

*通信作者, E-mail: liyingling@swun.edu.cn

基金项目:西南民族大学中央高校基本科研业务费专项资金项目(ZYN2025212)

引用格式:徐洋,李英玲.基于多维代码提交表示的即时缺陷预测研究[J].西南民族大学学报(自然科学版),2026,52(5):560-568.

Xu Yang, Li Yingling. Just-in-time defect prediction based on multi-dimensional representation of code commits [J]. Journal of Southwest Minzu University (Natural Science Edition), 2026, 52(5): 560-568.

CodeBERT model and a GraphCodeBERT model were respectively adopted to extract the intent features and syntactic structural features of the current commit, thus constructing a multi-dimensional feature representation. Finally, the fused multi-dimensional features were fed into an XGBoost classifier for training and prediction, and the optimal threshold was selected on the validation set to mitigate class imbalance. Experimental results on five project datasets demonstrated that, compared with three state-of-the-art methods, EvoTSS-JIT achieved average improvements ranging from 5.35% to 30.13% in F1-score, 1.67% to 10.23% in AUC-ROC, 5.48% to 14.87% in AUC-PR, and 12.02% to 83.68% in Recall, respectively, which validated the effectiveness of the proposed approach.

Keywords: just-in-time defect prediction; multi-dimensional feature representation; temporal feature; hybrid LSTM-Transformer model

随着软件系统复杂度和开发规模呈指数级增长,持续集成与协作开发已成为主流的软件开发模式^[1-2].然而,高频率的代码集成往往导致缺陷被引入项目主线,这些缺陷在开发初期难以被及时发现,从而不可避免地导致软件在测试或生产阶段出现故障,引发经济损失、安全漏洞乃至危及生命安全等严重后果^[3-5].即时缺陷预测(Just-in-time defect prediction, JIT-DP)应运而生,旨在代码提交的第一时间识别潜在缺陷,从而优先审查高危变更,显著降低后期修复成本并提升软件质量,对保障软件可靠性具有关键意义.

早期, Kamei 等人^[6]开创地研究即时缺陷预测,他们从扩散、规模、历史等五个维度定义了14个手工特征,并采用逻辑回归构建预测模型,奠定了变更级别缺陷预测的基础.为进一步提升特征表达能力,研究者开始利用深度学习技术自动学习语义特征. Hoang 等人^[7]提出的 DeepJIT 模型将变更代码和提交信息作为文本序列,利用卷积神经网络自动提取特征,实现了从“手工设计”到“自动学习”的转变,并进一步提出了 CC2Vec 模型^[8],使用分层注意力网络从两个输入之间提取层次关系,改进了 DeepJIT 的性能.近年来,研究趋势转向融合多种特征源以构建更全面的表示,例如 Chen 等人^[9]构建的多目标优化框架,以及 Jiang 等人^[10]基于双模态变化预训练模型的方法,均通过结合专家特征与深度学习特征来提升模型性能.

尽管融合专家特征与语义特征的方法已展现出优势,但现有研究在缺陷表征的完整性上仍存在明显局限.具体而言,大多数方法未能充分挖掘代码的深层结构信息,同时普遍将每次提交视为独立事件,忽略了提交历史轨迹中蕴含的时序演化模式,导致对缺陷的感知不够充分.

针对上述不足,本文提出一种基于多维代码提交表示的即时缺陷预测方法(EvoTSS-JIT).该方法通过融合时序特征、提交语义特征与代码结构特征,实现了从当前提交语义到动态演进模式与深层结构特征相结合的多维表征.该模型主要包含三个阶段:首先,对原始 Git 数据进行扩展与预处理,构建时序提交序列并提取完整的变更上下文信息;接着,分别利用混合模型、预训练代码模型提取三类特征,并对代码结构特征进行提交级聚合;最后,将多维特征拼接后输入 XGBoost 分类器中进行训练,并采用自适应阈值优化策略以应对类别不平衡问题,最终实现精准的缺陷风险识别.

本文提出的方法在开源数据集上,与当前3个state-of-the-art方法进行比较,实验结果表明,在F1值、AUC-ROC、AUC-PR和召回率四个评价指标上,分别平均高出基线方法5.35%~30.13%、1.67%~10.23%、5.48%~14.87%和12.02%~83.68%.

1 相关工作

1.1 即时缺陷预测

即时缺陷预测是软件缺陷预测领域的一个重要分支,其核心作用在于预测开发人员提交的每一次代码更改(即 Commit)是否存在缺陷,从而实现缺陷的早期、精准定位^[11].

早期的 JIT-DP 方法主要依赖于专家手工设计的变更级别指标,假设软件度量(专家特征)和缺陷之间存在固定关系,将代码更改特征定义为宏观指标,如添加的代码行数(LA)、修改的子系统数量(NS)和开发人员经验(EXP)等. Mockus 等人^[12]开创性地提出利用提交中的历史信息构建模型来预测新提交的风险,

奠定了 JIT-DP 的基础;Kamei 等人^[6]则从五个维度定义了 14 个变更级别的人工指标,并采用逻辑回归构建了一个 effort-aware 预测模型,具有较强可解释性,但过于简化了软件缺陷的原因,忽视了代码变更内在的语义因素.随着深度学习技术在自然语言处理领域的兴起,许多研究者开始将深度学习应用于 JIT-DP, CODEJIT^[13]同时从代码结构和程序依赖关系两个方面表示代码变更的语义,显著提升了特征表达能力,然而其性能受限于标注数据的规模.为兼顾可解释性与表示能力,最新的研究趋势转向融合多类特征的混合方法,Chen 等人^[9]构建的多目标优化框架,以及 JIT-BiCC^[10]设计的替换消息标识预训练目标,通过特征层面的融合,显著提升了预测模型的准确性和鲁棒性.

尽管与依赖单一专家特征或语义特征的传统方法相比,基于混合特征的方法展现出强大优势,但仍存在一些不足.具体而言,其限制性在于:1)多数方法未充分挖掘代码结构(如抽象语法树、数据流)中蕴含的语法与依赖信息.2)忽略代码演化的连续性,未能捕捉缺陷引入模式的长期演变规律.为此,本文提出融合时序信息与多维特征的即时缺陷预测框架,以实现更精准的缺陷风险感知.

1.2 代码变更表示

代码变更表示是软件工程中的一项关键任务,其目标是将代码的增、删、改等变更操作转化为机器可理解的数值化向量,为下游任务(如变更质量评估^[14]、自动代码审查^[14]和代码总结生成^[15]等)提供支撑.

早期的研究主要依赖人工设计规则,例如 Fluri 等人^[16]通过定义抽象语法树的匹配规则识别变更类型,Kamei 等人^[6]从扩散、规模、历史等维度设计特征评估缺陷倾向.这些方法有一定实用性,但通常仅能提取浅层特征,难以捕获深层语义信息,泛化能力有限^[17].随着深度学习的发展,基于神经网络的代码变更表示学习逐渐成为主流,其通过预训练语言模型等技术将代码变更映射为低维稠密向量^[18].根据对代码形式的不同处理,可分为基于序列的方法和基于图的方法两类.基于序列的方法将代码变更视为 token 序列,例如,Feng 等人^[19]提出了一个基于 Transformer 的预训练模型 CodeBERT,专门针对代码和自然语言的双模态任务进行优化,虽然并非专为变更场景设

计,但成功启发了后续一系列工作^[20].这类方法简单直接,但往往忽略代码上下文与结构化信息,在复杂场景中受限.为了更好地捕捉代码的结构与语义,基于图的表示方法逐渐兴起.Allamanis 等人^[21]提出的结合代码语义与结构的 GNN 模型,以及 Wang 等人^[22]提出的 GraphSPD 方法,均通过图神经网络捕捉代码的语法和深层次语义特征,在多项任务中展现出优越性.

本文受现有代码变更表示方法优缺点的启发,提出一种融合人工特征、语义特征与结构特征的代码提交多维表示框架,从而构建表征更全面的即时缺陷预测模型.

2 基于多维代码提交表示的即时缺陷预测模型

本文提出基于多维代码提交表示的即时缺陷预测模型(如图 1 所示),该模型一共分为 3 个阶段:数据准备、特征提取以及集成学习与预测.具体过程如下:1)数据准备阶段,对原始数据进行扩展字段和预处理,构建时序提交序列、提取提交前完整代码及差异对比,为多维特征提取提供结构化输入.2)特征提取阶段,针对时序特征,使用 LSTM-Transformer 混合模型获得时序嵌入表示;针对提交语义特征,采用 CodeBERT 微调模型提取语义向量;针对代码结构特征,利用 GraphCodeBERT 提取方法级结构表示并进行加权聚合,最终获得三类特征表示.3)集成学习与预测阶段,将三类特征拼接输入至 XGBoost 分类器中训练,并通过在验证集上搜索最优分类阈值以缓解类别不平衡,最终完成提交级即时缺陷预测.

2.1 数据准备

现有研究公开可用的数据集大多只包含由专家构建的人工特征、提交说明以及增加删除的代码行,为支持多维特征的联合提取,本研究首先扩展原始数据集字段,引入 lang(开发语言)、old_file(修改前完整代码)、old_code_diff(diff 格式的变更代码)等字段信息,用于特征提取阶段代码提交级别的语义、结构和上下文变更细节的提取.具体而言,基于现有数据集字段执行三步预处理:1)首先构建变更序列,依据 commit 时间对每个项目提交排序,形成用于时序特征

提取的历史提交轨迹;2)然后,遍历所有 git 仓库路径,根据 commit id 找到包含该 commit 的代码仓库;3)最后,执行不同的 git 命令依次获取所有变更文件、提交前完整内容,生成每个文件的差异对比,最终得到

扩展的数据集. 该阶段实现了从原始 Git 数据到可用的时序变更序列与结构化输入的转换,为三类特征的提取提供数据基础.

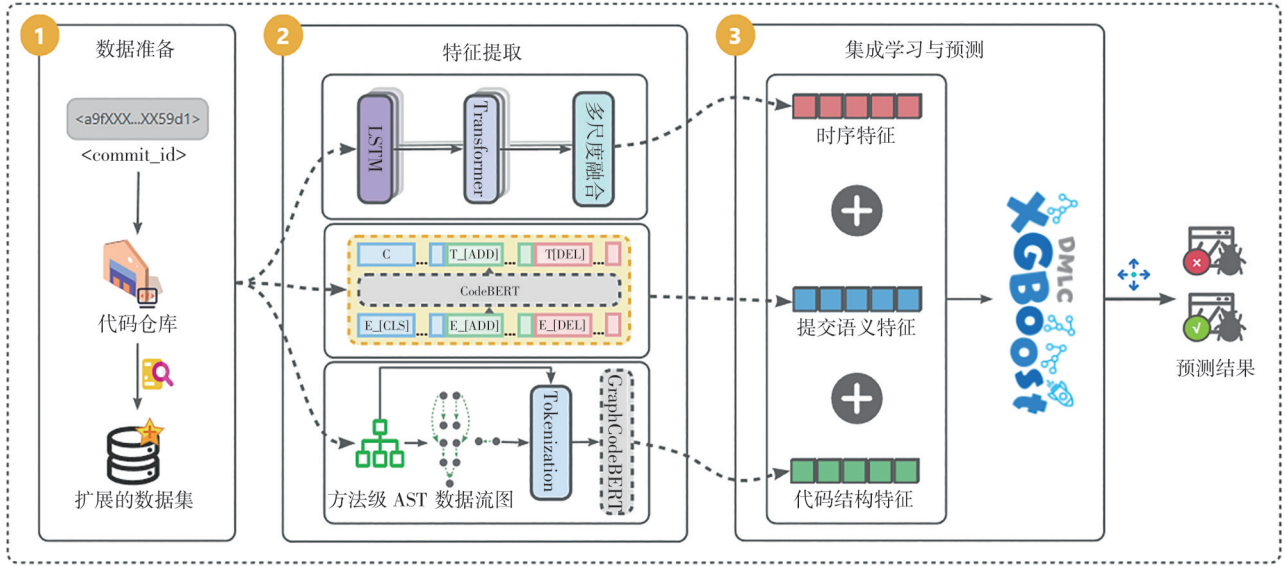


图1 基于多维代码提交表示的即时缺陷预测方法

Fig. 1 Just-in-time defect prediction method based on multi-dimensional representation of code commits

2.2 特征提取

通过构建一个融合时序特征、提交语义特征与代码结构特征的多维特征提取框架,并采用 LSTM-Transformer 混合模型对人工特征序列进行分层编码,以同时捕捉缺陷引入的短期波动与长期演化规律.

2.2.1 时序特征提取

人工特征是由专家根据其缺陷诱导提交的理解以及他们的专业知识和经验定义的描述性特征,用于衡量代码更改的质量. 为学习缺陷在时间维度中的演化模式,本研究选取 Kamei 等人^[6]提出的与缺陷相关度较高的 14 维人工特征(ns, nd, lt, age 等),并基于三个提交窗口构建三个时间序列 $X_t^{(w)} = \{X_{t-(w-1)}, X_{t-(w-2)} \dots X_t\}$, 其中 $w \in \{1, 3, 5, 10, 15, 20, 30\}$ 为窗口大小. 首先,对 14 维人工特征与 commit 时间间隔 Δt 进行标准化. 特别地,本研究使用 Time2Vec 方法对标准化后的时间间隔进行编码,对于时间步 t ,其 Time2Vec 表示定义为:

$$e_t^{time} = \begin{bmatrix} \omega_0 \Delta_t + b_0, \sin(\omega_1 \Delta_t + b_1), \dots, \\ \sin(\omega_{d-1} \Delta_t + b_{d-1}) \end{bmatrix}, \quad (1)$$

其中, Δ_t 为标准化后的时间间隔, d 表示时间嵌入维度.

然后,将时间编码与人工特征拼接,采用 LSTM-Transformer Encoder 混合模型学习短期与长期时序依赖:

$$H^{(w)} = \text{LSTM}(\text{concat}(X_t^{(w)}, e_t^{time})). \quad (2)$$

$$U^{(w)} = \text{TransformerEncoder}(H^{(w)}). \quad (3)$$

为突出窗口内对缺陷预测贡献较大的关键提交,本文采用线性注意力机制进行窗口内时间步加权:

$$\alpha_i^{(w)} = \frac{\exp(u_i^{(w)} W_a)}{\sum_{j=1}^w \exp(u_j^{(w)} W_a)}. \quad (4)$$

$$e_w = \sum_{i=1}^w \alpha_i^{(w)} u_i^{(w)}. \quad (5)$$

最后,考虑不同历史窗口对缺陷预测的贡献存在差异,引入可学习的窗口级权重,对多窗口表示进行融合,形成针对时间维度的时序变更嵌入表示 $E_{temporal}$:

$$\beta_w = \frac{\exp(\lambda_w)}{\sum_w \exp(\lambda_w)}. \quad (6)$$

$$E_{temporal} = \sum_w \beta_w e_w. \quad (7)$$

为使时序嵌入对缺陷预测任务具有判别性,使用提交级缺陷标签对时序模型进行端到端监督训练. 考虑即时缺陷预测任务中类别分布高度不平衡的问

题,采用加权二元交叉熵损失函数进行优化:

$$\mathcal{L}_{temporal} = -\sum_i \left[\frac{N^-}{N^+} y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log (1 - \sigma(\hat{y}_i)) \right], \quad (8)$$

其中, $\frac{N^-}{N^+}$ 为正样本权重, $y_i \in \{0, 1\}$ 表示提交是否为缺陷提交, $\hat{y}_i$ 表示模型的预测, $\sigma(\cdot)$ 为 Sigmoid 函数。

时序特征提取在保持人工特征可解释性的同时,有效增强了模型对缺陷演化趋势、节奏变化与长期依赖关系的学习能力。

2.2.2 提交语义特征提取

语义特征能够捕捉到代码更改中的细微差别和上下文信息,对于 JIT-DP 至关重要。为了强化模型对开发者提交意图与代码语义信息的理解,本研究遵循已有研究^[23]的做法,基于 CodeBERT 微调模型,采用“Input = [CLS]msg[ADD]added_code[DEL]removed_code”的统一模板,将提交消息与增删代码片段共同编码,使模型同时感知提交意图(msg),变更位置及内容(added_code/removed_code)。最终取得 768 维语义特征向量 $E_{semantic}$,用于表征语义层面的缺陷风险。

2.2.3 代码结构特征提取

代码结构特征旨在捕获代码修改本身的语法语义信息与程序内部的数据依赖关系。为深度挖掘代码变更中的缺陷诱导模式,本研究基于 GraphCodeBERT 预训练模型^[24]进行结构表示学习,并设计了跨方法的加权聚合机制以形成提交级的代码结构嵌入。

首先,通过解析 diff code,精确定位受影响的代码行,并基于 Tree-sitter 生成修改前完整代码的抽象语法树,寻找包含变更行的最内层函数或方法体,这一步的目的在于识别出与代码修改相关的具体方法上下文。接着,解析获得的方法级代码,移除注释和文档字符串,利用 Tree-sitter 解析器构建抽象语法树,通过遍历其叶子节点,从中提取出反映代码语法结构的 token 序列 T_{seq} ,每个 token 由其对应的抽象语法树 (AST) 节点在源文件中的位置索引唯一确定。同时,分析变量间的定义-使用关系,生成数据流图 G_{dfg} 。最后,将 (T_{seq}, G_{dfg}) 编码为 GraphCodeBERT 模型的输入格式,通过位置索引 P_{pos} 和注意力掩码 M_{attn} 机制,模型能够理解代码 token 与数据流节点之间的复杂关系:

$$P_{pos}[p] = \begin{cases} p + pad_{id} + 1, & 0 \leq p \leq n + 1 \\ 0, & n + 1 \leq p \leq n + m + 1 \end{cases} \quad (9)$$

$$M_{attn}[p, q] = \begin{cases} 1, & \text{允许位置 } p \text{ 关注位置 } q \\ 0, & \text{不允许位置 } p \text{ 关注位置 } q \end{cases}, \quad (10)$$

其中,允许的关注关系包括:代码 token 间全连接、特殊 token ([CLS], [SEP]) 关注所有代码 token、数据流节点与其对应的代码 token 范围互相关注,以及存在数据流边的节点间互相关注。

模型输出的序列向量经过特殊处理,提取 CLS 向量为方法级表示。针对包含 $k (k > 1)$ 个方法的代码提交,采用基于相似度的加权聚合方法,根据各方法与整体语义的相似度自动分配权重。具体而言,首先计算所有方法向量的均值作为锚点向量:

$$v_{anchor} = \frac{1}{k} \sum_{p=1}^k v_{method}^{(p)}. \quad (11)$$

然后,计算每个方法向量 $v_{method}^{(p)}$ 与锚点向量的点积相似度,并通过 Softmax 函数归一化,得到其贡献权重:

$$\gamma_p = \frac{\exp(\text{sim}(v_{method}^{(p)} \cdot v_{anchor}))}{\sum_{q=1}^k \exp(\text{sim}(v_{method}^{(q)} \cdot v_{anchor}))}. \quad (12)$$

最终,通过加权求和得到提交级的代码结构嵌入表示 $E_{structural}$:

$$E_{structural} = \sum_{p=1}^k \gamma_p \cdot v_{method}^{(p)}. \quad (13)$$

2.3 集成学习与预测

将三类特征拼接为统一的代码提交表示 $(E_{temporal}, E_{semantic}, E_{structural})$, 并采用 XGBoost^[25] 作为最终预测模型,对上述特征进行集成学习,由其学习后的特征权重完成提交级即时缺陷预测:

$$\hat{P}_t = \sigma \left(\sum_x f_x(\text{concat}(E_{temporal}, E_{semantic}, E_{structural})) \right), \quad (14)$$

其中, f_x 表示第 x 棵回归树, $\hat{P}_t$ 表示提交为缺陷提交的预测概率。

在训练阶段, XGBoost 通过最小化二元交叉熵损失函数进行优化:

$$\mathcal{L}_{XGB} = -\sum_i [y_i \log \hat{P}_t + (1 - y_i) \log (1 - \hat{P}_t)]. \quad (15)$$

同时,为了缓解即时缺陷预测任务中类别严重不平衡带来的性能退化问题,本研究在验证集上对分类阈值 θ 进行搜索,具体做法为:在区间 [0.01, 0.99] 上

以步长 0.01 均匀采样,共得到 99 个候选阈值 $\theta_{candidate}$, 根据每个候选阈值产生预测标签,将当前阈值得到的 F1 与历史 best_f1 比较,最终选取能够在验证集上最大化 F1 值的阈值。

3 实验设置与结果分析

3.1 数据集

本文使用 Zeng 等人^[26]的数据集,对其中 5 个项目数据进行 Git 字段补全,表 1 展示了五个研究数据集的统计信息,包括项目名称、Clean 和 Defect 中提交的数量以及提交总数、缺陷率。表格中的“Defect”列表示数据集中存在缺陷的提交数量,“Clean”列表示没有缺陷的提交数量。

表 1 数据集介绍
Table 1 Dataset description

项目名称	Clean/个	Defect/个	总数	缺陷率
OpenStack	16 830	5 927	22 757	26%
JDT	1 677	1 602	3 279	49%
Platform	6 978	4 056	11 034	37%
Gerrit	13 146	1 781	14 927	12%
GO	10 864	8145	19 009	43%

总体来看,这些数据集的缺陷率差异较大,JDT 和 GO 的缺陷率较高(分别为 49% 和 43%),数据集相对平衡,而 Gerrit 的缺陷率较低,数据集较为不平衡。

3.2 评价指标

本文参考了相关文献,选取了即时缺陷预测研究中常用的 4 个评价指标:Recall、F1 值、AUC-ROC、AUC-PR。

① Recall

Recall(召回率)是评估分类模型性能的重要指标之一,被计算为在所有真实为正类的样本中,被模型成功捕捉到的比例,主要用于衡量模型正确识别出正类样本的能力。其取值范围在 0 到 1 之间,数值越接近 1,说明模型漏报正类样本的情况越少。

② F1 值

F1 值是衡量 JIT-DP 模型整体准确性的指标,被计算为精确度和召回率的调和平均值,范围在 0 到 1 之间,1 表示最佳。

③ AUC-ROC

AUC-ROC 是一种广泛使用的度量标准,用于衡量模型在不同阈值设置下区分有缺陷和无缺陷提交的能力。ROC 曲线是通过绘制不同分类阈值下的真阳性率与假阳性率的关系图而创建的,AUC 的范围在 0 到 1 之间,值越大表示模型区分正负样本的能力越强,1 表示完美的分类器,具体公式如下:

$$AUC - ROC = \int_0^1 TPR(t) dFPR(t), \quad (16)$$

其中, t 为分类阈值, $TPR(t)$ 为在阈值 t 下的真阳性率, $FPR(t)$ 为在阈值 t 下的假阳性率。

④ AUC-PR

AUC-PR 是 PR 曲线(以召回率 Recall 为横轴、精确率 Precision 为纵轴)下的面积,反映模型在正样本稀缺场景下的性能。AUC-PR 适用于高度类别不平衡数据,而在多数即时缺陷预测数据集中,包含缺陷的提交远低于干净的代码提交个数,因此,本研究使用 AUC-PR 作为评价指标:

$$AUC - PR = \int_0^1 Precision(t) dRecall(t). \quad (17)$$

3.3 基线方法

本文通过对相关文献的系统调研,选取了即时缺陷预测领域中已被广泛验证且普遍采用的三个代表性方法作为实验基线。这些方法在多项研究中作为标准参照,为本文方法的评估提供客观依据。

① LAPredict^[26]:该方法在逻辑回归模型的基础上,利用从提交信息和代码变更中提取的手工特征进行预测。其模型结构简单,特征工程有效,是一个性能稳定的基线模型。

② JITLine^[27]:该方法通过提取代码提交的词袋特征,采用经差分进化算法优化的 SMOTE 技术处理类别不平衡,构建随机森林分类模型,增强了缺陷识别能力,在软件缺陷预测中表现出色。

③ JIT-Fine^[23]:该方法是对 JITLine 的进一步扩展,将专家定义的特征与基于 CodeBERT 提取的语义特征相结合,通过使用多层全连接网络和注意力机制,进一步提升了缺陷预测性能。

3.4 实验结果及分析

为评估本文提出的方法的有效性,本章节首先在 5 个项目数据集上,将本模型与基线方法在召回率、F1 值、AUC-ROC 和 AUC-PR 四项评价指标上进行对

比. 其次,通过消融实验以验证多维代码提交表示中各个组成部分的具体贡献.

3.4.1 对比实验

表 2 展示了本文提出的方法 EvoTSS-JIT 和基线方法在不同项目上的性能. 总体来看, EvoTSS-JIT 在各项评估指标上普遍优于基线方法. 实验结果表明, 该方法在 F1 值、AUC-ROC、AUC-PR 和 Recall 四个关键指标上, 平均提升幅度分别为 5.35%~30.13%、1.67%~10.23%、5.48%~14.87% 和 12.02%~83.68%.

表 2 EvoTSS-JIT 与基线方法对比

Table 2 Comparison of EvoTSS-JIT with baseline methods

项目数据集	方法	F1 值	AUC-ROC	AUC-PR	Recall
Gerrit	LApredict(2021)	0.176 9	0.637 9	0.158 5	0.122 3
	JIT-Line(2021)	0.141 0	0.719 9	0.140 2	0.117 0
	JIT-Fine(2022)	0.212 6	0.693 3	0.160 6	0.313 8
	EvoTSS-JIT(Ours)	0.246 1	0.731 9	0.142 9	0.417 1
OpenStack	LApredict(2021)	0.270 6	0.705 7	0.399 8	0.174 9
	JIT-Line(2021)	0.480 6	0.764 6	0.425 7	0.531 4
	JIT-Fine(2022)	0.466 0	0.751 7	0.442 4	0.637 7
	EvoTSS-JIT(Ours)	0.492 6	0.773 3	0.467 7	0.626 6
Platform	LApredict(2021)	0.490 5	0.754 2	0.549 6	0.422 1
	JIT-Line(2021)	0.627 3	0.812 9	0.628 9	0.706 8
	JIT-Fine(2022)	0.616 2	0.787 0	0.584 0	0.784 7
	EvoTSS-JIT(Ours)	0.650 8	0.825 2	0.672 6	0.777 8
Go	LApredict(2021)	0.515 3	0.673 3	0.546 4	0.498 0
	JIT-Line(2021)	0.615 5	0.744 5	0.633 5	0.647 1
	JIT-Fine(2022)	0.648 6	0.778 5	0.689 4	0.699 6
	EvoTSS-JIT(Ours)	0.651 2	0.777 2	0.676 1	0.754 9
JDT	LApredict(2021)	0.640 9	0.680 5	0.658 7	0.697 8
	JIT-Line(2021)	0.624 6	0.700 3	0.690 5	0.616 8
	JIT-Fine(2022)	0.643 0	0.654 9	0.625 8	0.704 0
	EvoTSS-JIT(Ours)	0.684 1	0.696 7	0.697 5	0.941 2
平均值	LApredict(2021)	0.418 8	0.690 3	0.462 6	0.383 0
	JIT-Line(2021)	0.497 8	0.748 4	0.503 8	0.523 8
	JIT-Fine(2022)	0.517 3	0.733 1	0.500 4	0.628 0
	EvoTSS-JIT(Ours)	0.545 0	0.760 9	0.531 4	0.703 5

在部分特定项目上,该方法展现出尤为突出的性能. 例如,在 OpenStack 项目数据集上, EvoTSS-JIT 相比于 LApredict 方法,在 F1 值上提高了 82.04%,在 AUC-ROC 指标上提高了 9.58%,在 AUC-PR 指标上提

高了 16.98%,在 Recall 指标上提高了 258.26%. 此外,在缺陷率最低(12%)的 Gerrit 项目中,其 Recall 指标相较 JIT-Line 方法提升了 256.50%,验证了模型在数据集类别严重不平衡场景下的性能;而在正负样本相对平衡的 JDT 项目(缺陷率为 49%)上,其 Recall 指标达到 0.941 2,接近于 1,相较于基线中 Recall 指标最高的 JIT-Fine 方法提升了 33.69%,进一步验证了该方法在不同场景下对缺陷提交的识别能力.

然而,仍然存在部分项目下,一些指标低于基线方法. 例如,在 JDT 项目上,其 AUC-ROC 指标相较于 JIT-Line 方法降低了 0.51%. 经分析,主要原因在于考虑到不同软件项目的时序演化模式具有差异性,本文采用可配置的多窗口融合机制,当前实验时序窗口为人为选定的窗口组合,包括 $\{1, 3, 5\}$ 、 $\{1, 5, 10\}$ 、 $\{1, 5, 15\}$ 、 $\{1, 10, 15\}$ 、 $\{1, 10, 20\}$ 、 $\{1, 20, 30\}$ 六种窗口设计,窗口大小作为模型超参数,可根据具体项目的数据分布与迭代周期进行调整,从而实现对项目特性的最佳适配与预测性能的优化.

值得注意的是,在 Platform 项目数据集上, EvoTSS-JIT 相较于最佳基线方法(JIT-Fine),在 Recall 指标上虽然降低了 0.88%,但在 F1 值、AUC-ROC 和 AUC-PR 三个指标上分别提升了 5.62%、4.85% 和 15.17%,表明本文方法虽然在极端追求检出全部缺陷时稍显保守,但 F1 值的大幅提升证明了模型在精确率与召回率之间取得了更优的平衡;AUC-ROC 的提高说明模型能够更好地区分缺陷与非缺陷提交;而 AUC-PR 的大幅提升反映出模型对正样本(缺陷)的识别能力与置信度均显著加强. 因此,综合性能的全面提升表明,模型在保持较好召回能力的同时,大幅降低了误报率,从而在实际开发中能够更精准地定位高风险提交,提升代码审查效率.

总的来说, EvoTSS-JIT 对比基线方法 LApredict、JIT-Line 和 JIT-Fine,在所有评价指标上普遍优于基线方法,原因在于本文提出的方法能够从时序、语义和结构三个方面全面表征代码提交,而基线方法将每次提交视为独立事件,并且忽略了部分特征. 因此,本文提出的方法在缺陷感知方面更具优势,模型效果更佳.

3.4.2 消融实验

本文通过消融实验旨在评估时序特征(TF)、提交

语义特征(SF)与代码结构特征(StF)在即时缺陷预测中的相对重要性. 实验在五个开源项目上,分别构建了仅包含两种特征的模型(SF+StF、TF+StF、TF+SF)与融合三种特征的完整模型 EvoTSS-JIT (TF+SF+StF),以系统分析各类特征的贡献,结果如表 3 所示.

表 3 消融实验结果
Table 3 Results of ablation experiments

项目数据集	实验设置	F1 值	AUC-ROC	AUC-PR	Recall
Gerrit	SF + StF	0.199 7	0.718 1	0.128 2	0.319 1
	TF + StF	0.189 0	0.675 3	0.126 0	0.267 4
	TF + SF	0.239 6	0.720 0	0.134 6	0.385 0
	EvoTSS-JIT (TF+ SF+ StF)	0.246 1	0.731 9	0.142 9	0.417 1
OpenStack	SF + StF	0.449 9	0.738 7	0.385 5	0.684 6
	TF + StF	0.464 5	0.721 6	0.423 8	0.531 5
	TF + SF	0.486 7	0.767 6	0.464 2	0.576 2
	EvoTSS-JIT (TF+ SF+ StF)	0.492 6	0.773 3	0.467 7	0.626 6
Platform	SF + StF	0.555 9	0.717 0	0.484 3	0.701 1
	TF + StF	0.385 9	0.571 1	0.441 6	0.443 0
	TF + SF	0.622 8	0.806 4	0.645 8	0.789 2
	EvoTSS-JIT (TF+ SF+ StF)	0.650 8	0.825 2	0.672 6	0.777 8
Go	SF + StF	0.584 7	0.670 0	0.540 9	0.811 3
	TF + StF	0.560 6	0.581 1	0.439 1	0.993 2
	TF + SF	0.642 6	0.766 7	0.670 3	0.699 7
	EvoTSS-JIT (TF+ SF+ StF)	0.651 2	0.777 2	0.676 1	0.754 9
JDT	SF + StF	0.669 9	0.664 0	0.620 2	0.856 7
	TF + StF	0.655 2	0.675 0	0.662 1	0.996 7
	TF + SF	0.673 9	0.671 8	0.627 5	0.915 0
	EvoTSS-JIT (TF+ SF+ StF)	0.684 1	0.696 7	0.697 5	0.941 2
平均值	SF + StF	0.492 0	0.701 6	0.431 8	0.674 6
	TF + StF	0.451 0	0.644 8	0.418 5	0.646 4
	TF + SF	0.533 1	0.746 5	0.508 5	0.673 0
	EvoTSS-JIT (TF+ SF+ StF)	0.545 0	0.760 9	0.531 4	0.703 5

实验结果显示,EvoTSS-JIT 在多数关键指标上均取得最优性能,例如当去掉时序特征时,相比于使用三种特征,在 F1 值上平均降低 9.72%,在 AUC-ROC 上降低 7.79%,在 AUC-PR 上降低 18.74%,在 Recall 上降低 4.11%. 在平均性能上,融合三种特征的完整模型

在 F1 值、AUC-ROC、AUC-PR 和 Recall 四个指标上,分别高出双特征组合 2.23%~20.84%、1.93%~18.01%、4.50%~26.98% 和 4.28%~8.83%,表明三类特征具有明显的互补效应,融合后能更全面地刻画缺陷引入模式,从而提升模型的整体判别能力.

特别地,在 Recall 指标上,EvoTSS-JIT 在部分项目中低于某些双特征组合,但结合 F1、AUC-ROC 和 AUC-PR 的综合表现,完整模型仍具备显著优势. 例如,在 GO 项目数据集中,使用三类特征相比于仅使用时序专家特征和代码结构特征,其 Recall 指标降低了 23.99%,但 AUC-PR 指标提高了 53.97%. 这一现象源于:双特征组合在过度捕捉正类样本的同时也引入了大量误报,导致 Recall 虚高而 Precision 偏低,而完整模型通过多特征融合,在 Precision 与 Recall 之间取得了更优的平衡,从而在整体判别性能上表现更好.

总的来说,本消融实验证实了时序特征、提交语义特征与代码结构特征均是缺陷预测的有效信息来源,任何单一特征的缺失都会导致性能下降. 尽管在特定项目的个别指标上完整模型未全面超越双特征组合,但其在综合性能上的显著提升验证了多特征融合的必要性,三类特征共同促进了模型性能的提升.

4 总结

随着软件规模和迭代速度的不断提升,精准识别代码提交中的缺陷已成为保障软件质量的重要环节. 针对现有即时缺陷预测方法依赖特征单一、难以全面捕捉缺陷模式的问题,本文提出了一种基于多维代码提交表示的即时缺陷预测方法,通过融合时序特征、提交语义特征与代码结构特征,构建了表征更全面的预测模型. 实验表明,在五个开源项目上,本方法相较最优基线在 F1 值、AUC-ROC、AUC-PR 和 Recall 上分别平均提升 5.35%、3.79%、6.20% 与 12.02%,有助于更准确地识别高风险变更.

尽管本方法在整体性能上表现良好,但在不同项目间的稳定性仍有提升空间,部分指标尚有优化余地. 未来工作将聚焦于改进时间窗口与特征融合机制,增强模型鲁棒性与泛化能力,进一步平衡各项指标.

参考文献

- [1] Saidani I, Ouni A, Mkaouer M W, et al. On the impact of Continuous Integration on refactoring practice: An exploratory study on TravisTorrent[J]. Information and Software Technology, 2021, 138: 106618.
- [2] 杨岚心, 张贺, 徐近伟, 等. 基于多标签学习的代码评审意见质量评价[J]. 软件学报, 2024, 35(6): 2775-2794.
- [3] Chen X P, Xu F R, Huang Y, et al. JIT-smart: A multi-task learning framework for just-in-time defect prediction and localization[J]. Proceedings of the ACM on Software Engineering, 2024, 1: 1-23.
- [4] Pornprasit C, Tantithamthavorn C K. DeepLineDP: Towards a deep learning approach for line-level defect prediction[J]. IEEE Transactions on Software Engineering, 2023, 49(1): 84-98.
- [5] Zain Z M, Sakri S, Ismail N H A. Application of deep learning in software defect prediction: Systematic literature review and meta-analysis[J]. Information and Software Technology, 2023, 158: 107175.
- [6] 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, 2013, 39(6): 757-773.
- [7] Hoang T, Khanh Dam H, Kamei Y, et al. DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction[C]//2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). Montreal: 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. ACM, 2020: 518-529.
- [9] Chen X, Xia H L, Pei W L, et al. Boosting multi-objective just-in-time software defect prediction by fusing expert metrics and semantic metrics[J]. Journal of Systems and Software, 2023, 206: 111853.
- [10] Jiang Y Z, Shen B J, Gu X D. Just-in-time software defect prediction via bi-modal change representation learning[J]. Journal of Systems and Software, 2025, 219: 112253.
- [11] Ju X L, Cao Y, Chen X, et al. JIT-CF: Integrating contrastive learning with feature fusion for enhanced just-in-time defect prediction[J]. Information and Software Technology, 2025, 182: 107706.
- [12] Mockus A, Weiss D M. Predicting risk of software changes[J]. Bell Labs Technical Journal, 2002, 5(2): 169-180.
- [13] Nguyen S, Nguyen T T, Vu T T, et al. Code-centric learning-based just-in-time vulnerability detection[J]. Journal of Systems and Software, 2024, 214: 112014.
- [14] Li Z Y, Lu S, Guo D Y, et al. Automating code review activities by large-scale pre-training[PP/OL]. V2. arXiv (2022-10-11) [2026-01-03]. <https://doi.org/10.48550/arXiv.2203.09095>.
- [15] Shi E S, Wang Y L, Du L, et al. On the evaluation of neural code summarization[C]//Proceedings of the 44th International Conference on Software Engineering. ACM, 2022: 1597-1608.
- [16] Fluri B, Wursch M, Pinzger M, et al. Change distilling: Tree differencing for fine-grained source code change extraction[J]. IEEE Transactions on Software Engineering, 2007, 33(11): 725-743.
- [17] 朱晓燕, 王文格, 王嘉寅, 等. 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.
- [18] Pravilov M, Bogomolov E, Golubev Y, et al. Unsupervised learning of general-purpose embeddings for code changes[C]//Proceedings of the 5th International Workshop on Machine Learning Techniques for Software Quality Evolution. ACM, 2021: 7-12.
- [19] Feng Z Y, Guo D Y, Tang D Y, et al. CodeBERT: A pre-trained model for programming and natural languages[PP/OL]. V4. arXiv (2020-09-18) [2026-01-03]. <https://doi.org/10.48550/arXiv.2002.08155>.
- [20] Zhou J Y, Pacheco M, Chen J F, et al. CoLeFunDa: Explainable silent vulnerability fix identification[C]//2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). Melbourne: IEEE, 2023: 2565-2577.
- [21] Allamanis M, Brockschmidt M, Khademi M. Learning to represent programs with graphs[PP/OL]. V3. arXiv (2018-05-04) [2026-01-03]. <https://doi.org/10.48550/arXiv.1711.00740>.
- [22] Wang S, Wang X D, Sun K, et al. GraphSPD: Graph-based security patch detection with enriched code semantics[C]//2023 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE, 2023: 2409-2426.
- [23] Ni C, Wang W, Yang K W, et al. The best of both worlds: Integrating semantic features with expert features for defect prediction and localization[C]//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2022: 672-683.
- [24] Guo D Y, Ren S, Lu S, et al. GraphCodeBERT: Pre-training code representations with data flow[PP/OL]. V4. arXiv (2021-09-13) [2026-01-03]. <https://doi.org/10.48550/arXiv.2009.08366>.
- [25] Chen T. XGBoost: a scalable tree boosting system[J]. Cornell University, 2016.
- [26] Zeng Z R, Zhang Y Q, Zhang H T, 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. ACM, 2021: 427-438.
- [27] Pornprasit C, Tantithamthavorn C K. JITLine: A simpler, better, faster, finer-grained just-In-time defect prediction[C]//2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). Spain: IEEE, 2021: 369-379.

(责任编辑:张阳,殷锋,付强,和力新,肖丽;英文编辑:郑玉才,周序林)