基于混合提示调优的函数级即时缺陷预测方法

吴迪[1]，周伽芮[2]，李文珂[2]，罗圣美[3]，周勇[4]，董振江[2*]

（1. 南京邮电大学物联网学院，江苏 南京 210023；2. 南京邮电大学计算机学院、软件学院、网络空间安全学院，江苏 南京 210023；3. 无锡学院，江苏 无锡 214105；4. 浩鲸云计算科技股份有限公司，江苏 南京 211153）

摘 要：即时缺陷预测在代码提交时判断修改是否引入缺陷，为物联网平台和智能家居软件的代码审查和测试提供参考。现有即时缺陷预测方法通常直接训练分类模型，难以充分利用预训练模型中的代码知识。提示调优虽能弥补这一不足，但是其模板通常仅包含单个代码片段，难以直接描述涉及修改前后版本的函数变更。为此，提出了一种基于混合提示调优的函数级即时缺陷预测方法。该方法首先根据修改前后的函数生成代码差异，以修改后函数和代码差异共同表示函数变更。然后，构建包含固定文字和可学习提示向量的混合提示模板，训练 CodeBERT 在掩码位置预测标签词。最后，汇总各函数变更的预测结果，判定提交是否有缺陷。该方法在 20976 条函数级样本上进行验证。结果表明，该方法的 F1 分数达到 95.14%，相比对比方法至少提高了 2.43%。

关键词：即时缺陷预测；预训练模型；代码差异；混合提示

中图分类号：TP311.5

文献标志码：A

doi：10.11959/j.issn.2096−3750.WLW26212

Function-level just-in-time defect prediction method based on mixed prompt tuning

WU Di[1], ZHOU Jiarui[2], LI Wenke[2], LUO Shengmei[3], ZHOU Yong[4], DONG Zhenjiang[2*]

1. School of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing 210023, China

2. School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China

3. Wuxi University, Wuxi 214105, China

4. Whale Cloud Technology Co., Ltd., Nanjing 211153, China

Abstract: Just-in-time defect prediction determines whether code changes introduce defects at commit time, supporting code review and testing for Internet of Things platforms and smart home software. Existing just-in-time defect prediction methods usually train classification models directly, making it difficult to fully exploit the code knowledge in pre-trained models. Although prompt tuning can alleviate this limitation, its templates usually contain only a single code snippet and cannot directly represent a function change involving both the pre-change and post-change versions. To address this issue, a function-level just-in-time defect prediction method based on mixed prompt tuning was proposed. The method first generated code diffs from the pre-change and post-change functions and represented each function change with the post-change function and its code diff. Then, a mixed prompt template consisting of fixed text and learnable prompt vectors was constructed, and CodeBERT was trained to predict label words at the masked position. Finally, the predictions of function changes were aggregated to determine whether a commit was defective. The method was evaluated on 20,976 function-level samples. The results showed that the method achieved an F1-score of 95.14%, outperforming the comparison methods by at least 2.43% in F1-score.

Key words: just-in-time defect prediction, pre-trained model, code diff, mixed prompt

收稿日期：XXXX−XX−XX；**修回日期：**XXXX−XX−XX

通信作者：董振江，dongzhenjiang@njupt.edu.cn

基金项目：国家自然科学基金（No. 62402239）；江苏省重点研发计划（No. BE2023025）；南京邮电大学引进人才科研启动基金项目（自然科学）（No. NY224064）。

0 引言

即时缺陷预测（Just-In-Time Defect Prediction，JIT-DP）在代码提交时判断本次提交是否可能引入缺陷，为代码审查和测试提供参考[1-3]。在物联网平台和智能家居软件中，设备接入、协议集成和自动化控制等功能需要随设备类型和应用需求持续更新[4-5]。这些更新通常需要远程部署到网关和边缘节点，一旦引入缺陷，还需要再次远程更新或回滚恢复[6]。因此，在代码提交阶段判断各个被修改函数是否引入缺陷，有助于在发布前确定需要重点审查和测试的代码。

近年来，研究者提出了多种即时缺陷预测方法[1-2,7-10]。早期方法主要依赖人工设计的变更度量，例如变更规模、修改历史和开发人员经验[1]。随着深度学习的发展，研究者开始从代码变更中自动学习缺陷特征。例如，DeepJIT 利用卷积神经网络从提交消息和代码变更中提取特征[7]，CC2Vec 利用分层注意力网络学习代码变更表示[8]。然而，这类方法仅从特定任务的标注数据中学习，对代码语义的理解受限于训练数据的规模。为利用大规模代码语料中学习的通用知识，研究者进一步将预训练模型引入即时缺陷预测[9-10]。例如，Ni 等提出 COMPDEFECT，采用 GraphCodeBERT 编码修改前后的函数，通过微调完成函数级缺陷预测[9]。在软件工程的其他任务中，预训练模型也与检索增强生成结合[11]。现有基于预训练模型的即时缺陷预测方法通常在预训练模型上添加分类头进行微调，其训练目标与预训练任务存在差异，难以充分利用预训练模型中学习的代码知识[12]，导致方法的性能受限。

为缩小下游任务与预训练任务之间的差异，研究者提出了提示调优。提示调优通过模板将分类任务转化为完形填空任务，由预训练模型预测掩码位置的标签词，再将标签词映射为相应类别。这种预测形式与掩码语言建模（Masked Language Modeling，MLM）预训练任务相衔接，能够利用预训练模型已有的词预测能力完成分类[13]。近年来，提示调优已广泛用于缺陷检测[14]、漏洞检测[15-16]、漏洞评估[17]和方法名推荐[18]等代码智能任务。然而，将提示调优用于函数级即时缺陷预测仍面临以下两个难点。首先，现有提示调优方法通常将单个代码片段填入模板，而函数级即时缺陷预测的对象是一次函数变更，涉及修改前后两个函数版本。若仅填入代码差异，模型缺少修改所在的函数上下文。若仅填入修改后函数，本次修改已融入完整函数，模型难以区分具体改变的语句（即函数变更表示问题）。其次，若将修改后函数和代码差异同时填入模板，两类代码在缺陷判断中的作用不同，前者提供函数上下文，后者标出具体修改，模型需要区分两类代码的含义（即输入区分问题）。

为此，本文提出一种基于混合提示调优的函数级即时缺陷预测方法（Mixed Prompt tuning for function-level Just-In-Time defect prediction，MPJIT）。针对函数变更表示问题，该方法根据修改前函数和修改后函数生成代码差异，以修改后函数和代码差异共同表示一次函数变更，其中修改后函数提供变更后的函数上下文，代码差异显式标出本次新增和删除的语句。针对输入区分问题，该方法构建混合提示模板，以固定文字分别说明修改后函数和代码差异的含义，并在固定文字前加入可学习提示向量，使提示表示能够根据训练样本进行调整。最后，采用 CodeBERT[19]在掩码位置预测标签词，将其映射为有缺陷或无缺陷，并汇总各函数变更的预测结果，判定该提交是否有缺陷。

为验证 MPJIT 的有效性，在 20976 条函数级样本上开展实验。实验结果表明，MPJIT 的 F1 分数和曲线下面积分别达到 95.14% 和 99.35%，F1 分数相比对比方法至少提高了 2.43%。此外，修改后函数和代码差异共同输入的 F1 分数，相比仅使用代码差异和仅使用修改后函数分别提高了 28.26% 和 2.03%。在相同代码输入下，混合提示模板的 F1 分数相比仅使用固定文字和仅使用可学习提示向量的模板分别提高了 1.75% 和 1.92%。

本文的主要贡献如下：

（1）提出一种基于混合提示调优的函数级即时缺陷预测方法，该方法能够结合修改后函数提供的函数上下文和代码差异标出的具体修改，判断函数变更是否引入缺陷。

（2）设计一种面向函数变更的混合提示模板，能够解决函数变更表示问题和输入区分问题。

（3）实验结果表明，MPJIT 能够有效识别引入缺陷的函数变更，并优于现有的即时缺陷预测方法。

1 相关工作

1.1 即时缺陷预测

近年来，即时缺陷预测受到研究者的广泛关注[20-23]。早期即时缺陷预测研究主要依赖人工设计的变更度量[1,24]。Kamei等从变更规模、修改历史和开发者经验等方面构建特征，识别可能引入缺陷的提交[1]。Zeng等研究深度即时缺陷预测方法的效果，并提出仅以新增代码行数为特征的逻辑回归方法，考察简单变更特征的预测能力[2]。

随着深度学习的发展，研究者开始直接从提交内容中学习代码变更表示。DeepJIT以提交消息及新增、删除的代码行为输入，通过卷积神经网络提取特征，并联合学习用于缺陷预测的提交表示[7]。CC2Vec以新增和删除的代码片段为输入，利用分层注意力网络编码代码结构，再比较两部分表示，学习代码变更的分布式向量，其表示学习以提交消息作为监督信号[8]。这两种方法能够从变更数据中学习与缺陷有关的特征，但是主要围绕新增、删除的代码建模，修改所在的完整函数上下文未被直接作为独立输入。其代码语义知识也主要来自用于训练的变更数据，对需要结合完整函数逻辑和接口行为判断的修改，仍有进一步利用上下文及预训练代码知识的空间。

为利用大规模代码语料中的语义知识，研究者将预训练模型引入即时缺陷预测。COMPDEFECT采用GraphCodeBERT编码修改前函数和修改后函数，通过多任务学习联合完成函数级缺陷预测、类别识别和修复[9]。Guo等采用CodeBERT等预训练模型编码提交消息及新增、删除的代码，再结合卷积网络和分类层完成提交级预测[10]。Liu等提出CCRep，利用预训练模型编码修改前后的代码，使变更片段与其所在的代码上下文交互，学习代码变更表示[20]。Lin等提出CCT5，在代码变更和提交消息语料上设计面向变更的预训练任务，再通过微调完成即时缺陷预测等任务[21]。Nam等[22]采用微调方式训练代码语言模型，比较了仅修改后函数、修改前后函数和代码差异等输入形式下的缺陷预测效果。不同于该工作，MPJIT将修改后函数和代码差异共同作为输入，并采用混合提示调优将缺陷预测转化为完形填空任务，而非通过微调完成分类。刘子扬等提出基于个性化联邦学习的跨项目软件缺陷预测方法[25]。Huang等提出JIT-LSM，通过两层大模型结构获取代码上下文，并融合大模型和小模型的预测概率，完成缺陷预测和定位[26]。其中，Guo等的实验和COMPDEFECT均采用微调，将预训练模型学习的代码表示用于缺陷分类。

本文方法与DeepJIT、CC2Vec、COMPDEFECT及采用附加分类头的预训练模型微调研究密切相关，主要区别体现在两个方面。第一，现有方法难以同时利用函数上下文和显式变更信息，MPJIT将修改后函数和代码差异共同作为输入。第二，现有微调方法未充分利用模型的预训练形式，MPJIT通过混合提示将缺陷预测转化为标签词预测。

1.2 代码智能中的提示调优

提示调优是使预训练模型适应下游任务的重要方法，已应用于多种代码智能任务。在代码分析任务中，已有研究将其用于缺陷预测[14,27]、漏洞检测[15-16]和漏洞严重程度评估[17]。在代码生成及软件维护任务中，提示调优还用于方法名推荐和一致性检查[18]、提交消息生成[27]以及问题报告与提交之间的关联恢复[28]。按照提示的构造方式，提示可分为硬提示、软提示和混合提示。硬提示采用固定文字描述输入和任务，软提示通过可学习的连续向量形成任务提示[29]，混合提示则结合固定文字和可学习向量，为模型提供任务说明，并通过训练调整提示表示[14,16-17]。例如，Wang等系统评价了提示调优在多种代码智能任务中的表现，在缺陷预测任务中比较硬提示、软提示和混合提示，输入原始代码片段[14]。PSCVFinder采用硬提示进行智能合约漏洞检测，其输入为经过关键智能合约漏洞切片处理的代码片段[15]。AUMENA采用硬提示完成方法名推荐和方法名一致性检查，将方法体标识符、方法签名及相关上下文填入模板，在一致性检查中还加入待检查的方法名[18]。ProRLearn采用混合提示进行漏洞检测，以经过预处理的源代码片段为输入，并利用强化学习奖励机制优化训练[16]。MPLinker将问题报告文本和提交信息填入三个硬提示模板，分别计算预测概率，再对各模板的概率取平均，判断问题报告与提交之间是否存在关联[28]。PT-SVA采用混合提示评估漏洞严重程度，将源代码片段和漏洞描述共同输入提示模板[17]。此外，Liu等在不同预训练模型上比较适配器调优、低秩适应、提示调优和

前缀调优，考察不同参数更新方式在即时缺陷预测和提交消息生成中的表现[27]。与提示调优相关的提示方法也被用于代码变更分析和软件质量保障。徐圣斌等研究上下文学习和模型微调在提交日志生成中的应用，通过示例检索为模型提供相关代码变更样本[30]。Zhu 等提出 VulInstruct，从历史漏洞及修复中提取安全规范，并检索相关案例和规范，引导模型分析漏洞根因[31]。本文与上述提示调优方法的主要区别体现在输入信息的组织方式上。现有研究尚未针对即时缺陷预测中的代码变更特点组织提示输入。为此，MPJIT 将修改后函数和代码差异共同填入混合提示模板。

2　基于混合提示调优的函数级即时缺陷预测方法

如图 1 所示，MPJIT 以一次代码提交中各个被修改函数的修改后函数和代码差异为输入，输出各函数变更及该提交的缺陷预测结果。该方法包括函数变更表示、即时缺陷预测模型训练和即时缺陷预测三个阶段。在函数变更表示阶段，根据修改前函数和修改后函数生成代码差异，以修改后函数和代码差异共同表示一次函数变更。在即时缺陷预测模型训练阶段，通过混合提示将缺陷预测组织为标签词预测任务，共同更新 CodeBERT 参数和可学习提示向量。在即时缺陷预测阶段，提取待预测代码提交中各个被修改的函数，采用训练好的模型分别进行预测，并汇总判定该提交为有缺陷或无缺陷。

图中，X_{Before}、X_{After}和X_{Diff}分别表示修改前函数、修改后函数和代码差异。X_{Before}仅用于生成X_{Diff}，训练和预测均以X_{After}和X_{Diff}作为代码输入。Θ表示 CodeBERT 参数，Φ表示可学习提示向量，训练后的参数分别记为Θ^*和Φ^*。图中省略样本下标。

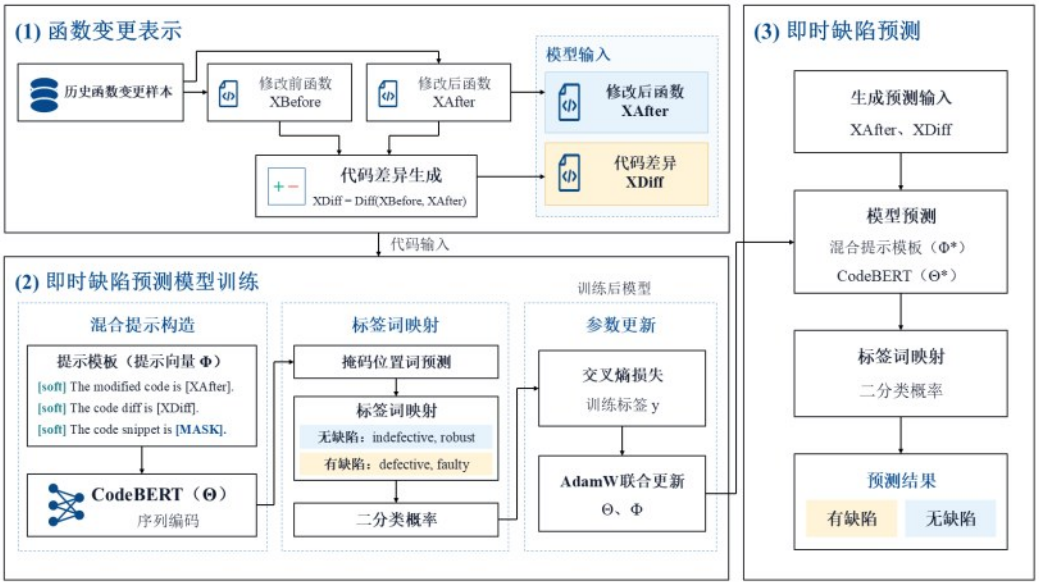


图 1　基于混合提示调优的函数级即时缺陷预测方法流程

2.1　函数变更表示

一次代码提交可能包含多个函数的修改，MPJIT 将该提交对每个函数的修改称为一次函数变更，并分别作为一个预测样本。对于第i条样本，$X_{\text{Before},i}$和$X_{\text{After},i}$分别表示修改前函数和修改后函数，由两者生成的代码差异记为$X_{\text{Diff},i}$。标签y_i取 1 表示有缺陷，取 0 表示无缺陷。模型以$X_{\text{After},i}$和$X_{\text{Diff},i}$为输入，函数变更的表示形式见式（1）。

$$\begin{aligned} X_{\text{Diff},i} &= Diff\left(X_{\text{Before},i}, X_{\text{After},i}\right) \\ X_i &= \left(X_{\text{After},i}, X_{\text{Diff},i}\right) \\ y_i &\in \{0,1\} \end{aligned} \tag{1}$$

式（1）中，X_i表示由第 i 个函数变更构造的模型输入样本，由修改后函数$X_{\text{After},i}$和代码差异$X_{\text{Diff},i}$组成。

为生成$X_{\text{Diff},i}$，比较$X_{\text{Before},i}$和$X_{\text{After},i}$，将删除行和新增行分别标为“−”和“+”[32]。每个变更块保留两侧各最多 3 行邻近上下文[33]，行数按源代码计算。

同一函数包含多个变更块时，按其在函数中的顺序拼接。修改后函数则保留数据集提供的代码提交后的函数版本。图2以Deeplearning4j项目的getAllUpdateTimes()函数为例，展示两类输入。

```
public long[] getAllUpdateTimes(String sessionID, String typeID, String workerID) {
  ...
  LongArrayList list = new LongArrayList();
  while (rs.next()) {
    list.add(rs.getLong(1));
  }
  return list.elements();
  ...
}
```

(a)　修改后函数

```
    while (rs.next()) {
      list.add(rs.getLong(1));
    }
-   return list.toArray();
+   return list.elements();
  } catch (SQLException e) {
    throw new RuntimeException(e);
  }
```

(b)　代码差异

图2　getAllUpdateTimes()函数变更示例

图2(a)展示getAllUpdateTimes()的修改后函数节选，其中保留列表初始化、时间戳收集和数组返回过程，省略号表示未展示的代码。图2(b)则突出toArray()被替换为elements()的具体位置，并保留调用附近的代码。

2.2　即时缺陷预测模型训练

（1）混合提示构造。混合提示将修改后函数和代码差异组织为掩码位置的类别预测任务。固定文字说明两类输入和待预测类别，可学习提示向量随训练调整任务表示。模板包含三个固定语句，具体写为“[soft] The modified code is [X_{After}]. [soft] The code diff is [X_{Diff}]. [soft] The code snippet is [MASK].”。其中，[X_{After}]和[X_{Diff}]分别填入修改后函数和代码差异，[soft]表示可学习提示向量，[MASK]表示类别预测位置。前两个语句区分两类输入，第三个语句说明待预测类别。

参考Wang等在固定文字模板中插入可学习token的混合提示构造方式[14]，本文在说明修改后函数、代码差异和待预测类别的三句固定文字前，分别插入1个可学习token。固定文字用于说明两类代码的含义和预测目标，可学习token则随训练调整提示表示。前两处软提示对应函数上下文和具体修改，第三处软提示对应掩码位置的类别预测。三个位置分别使用独立的向量参数，每个向量的维度与CodeBERT的输入嵌入维度一致。根据已有工作的设置[14,34]，提示向量采用嵌入层的默认随机初始化，初始值服从标准正态分布$N(0,1)$，随后根据训练样本更新。

代码和固定文字经分词后转换为词嵌入，三个提示向量插入模板指定的位置，共同构成CodeBERT的输入序列。CodeBERT对整个序列进行编码，使修改后函数中的上下文和代码差异中的变化语句共同参与掩码位置的预测。

记混合提示构造过程为f_ϕ，其中Φ表示可学习提示向量，Θ表示CodeBERT参数。模型对填入代码的提示序列进行编码，并取[MASK]位置的隐藏状态h_i，如式（2）所示。该状态汇集代码输入、固定文字和提示向量形成的表示，随后用于计算标签词分数。

$$h_i = \text{CodeBERT}_\Theta\left(f_\Phi\left(X_i\right)\right)_{[\text{MASK}]} \tag{2}$$

（2）标签词映射。标签词映射将掩码位置的词预测结果转换为有缺陷和无缺陷两类概率。参考Wang等[14]和Ren等[16]的标签词映射设置，本文采用一对多映射，即每个类别对应多个标签词。结合缺陷二分类任务，选取描述缺陷状态和代码可靠性的词语构建标签词集合。

记标签词集合为W，其中无缺陷类W_0={indefective, robust}，有缺陷类W_1={defective, faulty}。映射函数V将标签词归入相应类别，如式（3）所示。

$$\mathcal{V}:\mathcal{W} \to \{0,1\} \tag{3}$$

CodeBERT的掩码语言模型输出头将掩码位置的隐藏状态$\boldsymbol{h}_i$转换为词表分数向量$\boldsymbol{z}_i$。这些分数由OpenPrompt的ManualVerbalizer处理。标签词添加前缀空格后进行分词。若一个标签词被拆分为多个token，采用默认的first策略，取第一个token的词表分数作为该标签词的分数$s_{i,v}$。

对标签词集合W中的分数进行softmax归一化，得到标签词概率$q_{i,v}$。随后对概率取对数，并在同一类别内求平均，得到类别分数$g_{i,c}$。最后对两个类别的分数进行softmax归一化，得到类别概率，具体的计算过程见式（4）。

$$q_{i,v}=\frac{\exp\left(s_{i,v}\right)}{\sum_{w\in W}\exp\left(s_{i,w}\right)}$$
$$g_{i,c}=\frac{1}{|W_c|}\sum_{v\in W_c}\log\left(q_{i,v}+\varepsilon\right) \tag{4}$$
$$P\left(y_i=c\,|\,X_i\right)=\frac{\exp\left(g_{i,c}\right)}{\sum_{k\in\{0,1\}}\exp\left(g_{i,k}\right)}$$

式（4）中，$c\in\{0,1\}$，W_c为类别c对应的标签词集合。$\varepsilon=10^{-15}$取自OpenPrompt[34]的ManualVerbalizer实现，用于避免标签词概率接近0时出现log(0)，不参与模型训练，也不作为可调超参数。$P(y_i=1|X_i)$和$P(y_i=0|X_i)$分别表示有缺陷类和无缺陷类概率。

（3）参数更新。参数更新利用训练样本的真实标签共同调整CodeBERT参数和提示向量，使模型学习函数变更的类别判别。根据式（4）得到的二分类概率，采用交叉熵损失进行训练。对于训练集中的N条函数变更，对真实类别预测概率的负对数求平均，得到式（5）所示的损失函数。其中，y_i为第i条样本的真实标签，Θ和Φ分别表示CodeBERT参数和可学习提示向量。

$$\mathcal{L}(\Theta,\Phi)=-\frac{1}{N}\sum_{i=1}^{N}\log P\left(y_i|\,X_i;\,\Theta,\Phi\right) \tag{5}$$

在每个训练批次中，将Φ中的三个可学习提示向量（即[soft]）插入输入序列，再由参数为Θ的CodeBERT对该序列进行编码，并根据掩码位置的标签词分数计算类别概率。因此，类别概率同时依赖于Θ和Φ，可记为$P(y_i|X_i;\Theta,\Phi)$。模型根据类别概率计算式（5）的交叉熵损失。反向传播时，梯度同时传至CodeBERT的参数Θ和提示向量Φ，AdamW根据两组参数各自的梯度在同一训练步骤中共同更新Θ和Φ。训练过程中，保存验证集F1分数最高时的检查点，将其中的CodeBERT参数和提示向量固定，分别记为Θ*和Φ*，用于后续代码提交的预测。训练超参数和输入长度设置见第3.4节。

①https://github.com/thingsboard/thingsboard

②https://github.com/openhab/openhab1-addons

2.3　即时缺陷预测

对于待预测代码提交中的每个函数变更，MPJIT首先读取修改前函数和修改后函数，按2.1节的规则生成代码差异，随后将修改后函数和代码差异依次填入混合提示模板。待预测样本的输入顺序和掩码位置与训练阶段保持一致。

模型计算时，模板使用训练得到的提示向量Φ^*，CodeBERT采用最佳检查点中的参数Θ^*。模型对提示序列进行编码，并通过掩码语言模型输出头获得[MASK]位置的词表分数。

随后，按照2.2节的标签词映射过程，提取标签词分数并归一化，计算同一类别的平均对数概率。再通过softmax得到有缺陷和无缺陷两个类别的概率，具体计算见式（4）。

最后，根据有缺陷类概率判定类别，规则见式（6）。参考Wang等[14]的做法，MPJIT将预测概率较大的类别作为判定结果。由于有缺陷类和无缺陷类的概率之和为1，式（6）将判定阈值设为0.5，等价于选择概率较大的类别。此外，当两类概率相等时，判为有缺陷。对于一次待预测的代码提交，若任一被修改函数被预测为有缺陷，则将该提交判为有缺陷。若全部被修改函数均被预测为无缺陷，则将该提交判为无缺陷。同时保留各函数的预测结果，为代码审查和测试提供参考。

$$\hat{y}_i=\begin{cases}1(\textit{有缺陷}), & P(y_i=1|X_i)\geqslant\tau\\ 0(\textit{无缺陷}), & P(y_i=1|X_i)<\tau\end{cases} \tag{6}$$
$$\tau=0.5$$

3　实验设置

本节介绍数据集、评价指标、对比方法和实验配置。

3.1　数据集

MPJIT采用广泛使用的即时缺陷预测数据集Function-SStuBs4J开展实验[9]。该数据集由Ni等在ManySStuBs4J[35-36]的基础上构建，收集了来自Java开源项目的函数变更，包含多个物联网相关项目。例如，ThingsBoard① 用于物联网平台的设备接入与管理，openHAB② 用于智能家居的自动化控制，

Eclipse SmartHome[①]用于异构设备的协议集成。

每条函数变更记录包含修改前函数、修改后函数、修复后函数和类别标签。修改前函数表示该函数变更发生前的代码状态，修改后函数表示该函数变更完成后的代码状态。如果该函数变更引入缺陷，则将修改后函数标记为有缺陷，否则标记为无缺陷。修复后函数表示缺陷修复后的代码状态。本文关注代码提交时的函数级即时缺陷预测，使用每条函数变更记录中的修改前函数和修改后函数生成代码差异，以修改后函数和代码差异作为模型输入。

为减少重复样本对评价结果的影响[37]，在已去除注释、空行和行尾空格的数据上，对修改前函数和修改后函数组成的有序代码对进行精确匹配，最终发现71条重复记录。这可能是因为同一函数变更对应多个缺陷子类型，在原数据集中被重复记录。去除重复样本后，最终得到20976条函数级样本，记为DFunction-SStuBs4J，其中有缺陷样本10454条，无缺陷样本10522条。

3.2 评价指标

实验以代码提交为单位计算各项评价指标，各组实验均采用相同的提交结果汇总规则。以有缺陷提交为正类，采用准确率（Accuracy，ACC）、精确率（Precision，P）、召回率（Recall，R）、F1分数（F1 Score，F1）和曲线下面积（Area Under the Curve，AUC）评价预测效果。TP表示正确预测为有缺陷的提交数，FP表示被预测为有缺陷的无缺陷提交数。TN表示正确预测为无缺陷的提交数，FN表示被预测为无缺陷的有缺陷提交数。准确率衡量全部代码提交中预测正确的比例，计算式为ACC=(TP+TN)/(TP+TN+FP+FN)。

精确率衡量被模型预测为有缺陷的提交中实际有缺陷提交的比例，计算式为P=TP/(TP+FP)。召回率衡量实际有缺陷提交中被模型正确识别的比例，计算式为R=TP/(TP+FN)。F1分数是精确率和召回率的调和平均值，用于综合评价有缺陷提交的识别效果，计算式为F1=2PR/(P+R)。曲线下面积是受试者工作特征（Receiver Operating Characteristic，ROC）曲线下的面积，根据各代码提交的有缺陷预测分数计算。ROC曲线以假正率为横轴、真正率为纵轴，通过改变概率判定阈值反映模型对两类提交的区分能力。曲线下面积的取值范围为0～1，数值越大表示区分能力越强。

此外，本文采用双侧Wilcoxon符号秩检验比较不同方法的预测结果。针对每项评价指标，将两种方法在对应实验条件下的运行结果进行配对检验，以$p<0.05$作为差异具有统计显著性的判定标准。同时，采用效应量r衡量差异大小，以0.1、0.3和0.5分别作为小效应、中等效应和大效应的判定阈值，低于0.1时认为效应可忽略。

3.3 对比方法

实验选取DeepJIT、CC2Vec和COMPDEFECT作为对比方法，采用其公开实现。DeepJIT使用卷积神经网络，从提交消息和代码变更中学习特征。对于代码差异，分别编码删除和新增的代码行[7]。CC2Vec采用分层注意力网络表示删除和新增的代码，再通过比较两部分表示形成代码变更向量[8]。这两种方法以代码提交为预测对象。COMPDEFECT以修改前后函数为输入，联合学习缺陷预测、类别识别和修复任务[9]，与MPJIT同为函数级预测方法。参考COMPDEFECT的对比设置，以代码提交为单位比较各方法的缺陷预测效果[9]。本文关注函数级即时缺陷预测，以每个被修改函数为预测对象。为了公平比较，按DeepJIT[7]和CC2Vec[8]的变更编码方式，将每个函数变更中的新增行和删除行作为两种方法的输入，在函数级样本上进行训练和预测。对于DeepJIT[7]和CC2Vec[8]，均采用Adam优化器，学习率为0.00001，批量大小和嵌入维度均为64，dropout均为0.5，最多分别训练25轮和50轮。COMPDEFECT按其公开训练代码采用AdamW优化器，学习率为0.0001，批量大小为32，dropout为0.1，最多训练50轮[9]。CodeBERT分类头微调采用与MPJIT相同的训练设置。各方法均选择验证集上F1分数最高的模型进行测试。在结果汇总时，将COMPDEFECT预测的无缺陷类记为0，其余缺陷类别统一记为1。对于MPJIT、COMPDEFECT和CodeBERT分类头微调，若同一提交中任一函数变更被预测为有缺陷，则将该提交判为有缺陷，否则判为无缺陷。提交的真实标签由对应函数变更的真实标签确定，包含有缺陷函数变更的提交标为有缺陷，其余提交标为无缺陷。

①https://github.com/eclipse-archived/smarthome

此外，实验设置CodeBERT分类头微调作为对比方法。该方法将修改后函数和代码差异共同输入CodeBERT，利用[CLS]位置的表示，通过附加分类头输出有缺陷和无缺陷两类概率，训练时共同更新CodeBERT参数和分类头参数。该方法采用与MPJIT相同的数据划分和提交级结果汇总规则。

3.4 实验配置

本文沿用COMPDEFECT[9]的数据划分设置，以函数变更样本为单位，按8∶1∶1随机划分训练集、验证集和测试集，并使各集合中有缺陷样本与无缺陷样本的比例与原数据集保持一致。训练集用于学习模型参数和提示向量，验证集用于选择训练检查点，测试集用于评价最终预测效果。各方法使用相同的样本划分进行比较。

实验采用CodeBERT作为掩码语言模型，通过OpenPrompt[34]的MixedTemplate构造混合提示输入，并用ManualVerbalizer完成标签词映射。最大输入序列长度设为512个token，代码、固定文字、可学习提示向量、掩码及特殊标记共同占用输入位置。参考现有工作的多部分输入组织与长度控制方法[17,34]。本文预留提示模板和特殊标记所需的位置，其余位置用于容纳修改后函数和代码差异。若两部分代码的总长度超过可用位置数，则按照修改后函数token数与代码差异token数之比分配需要删除的token数量，取整后调整，使两部分删除的总量等于超出的长度，再分别从末尾截断，保留各自的前缀。该分配比例随每条样本中两类代码的实际长度变化，本文根据验证集上的比较结果选用这一动态分配策略。批量大小为32，最多训练20轮。优化器采用AdamW，学习率为2×10^{-5}。训练期间共同更新CodeBERT参数和提示向量，并使用梯度裁剪。根据验证集表现提前停止训练，选择验证集F1分数最高的检查点进行测试。

实验在配备NVIDIA A30图形处理器（Graphics Processing Unit，GPU）和AMD EPYC 7402处理器的Linux服务器上运行。训练和测试采用同一代码填入顺序及标签词集合，以保持模型输入和类别定义一致。各模型均独立运行10次，每次运行均以代码提交为单位计算评价指标，以各项指标的10次运行平均值作为最终实验结果。

4 实验结果与分析

4.1 即时缺陷预测的有效性

为评价MPJIT在代码提交时识别缺陷的效果，实验在相同的数据集划分上与DeepJIT、CC2Vec、COMPDEFECT和CodeBERT分类头微调进行比较。各方法的测试结果见表1。下文所述的指标提升均按对应指标的百分比直接相减计算。

表1　各方法的预测结果

%

方法	准确率	精确率	召回率	F1分数	曲线下面积
DeepJIT	52.41	53.21	37.18	43.77	53.38
CC2Vec	50.46	50.39	37.42	42.95	52.28
COMPDEFECT	70.95	76.10	62.75	68.78	79.38
CodeBERT分类头微调	92.73	92.70	92.72	92.71	97.64
MPJIT	**95.16**	**95.25**	**95.02**	**95.14**	**99.35**

表1显示，MPJIT在五项评价指标上均取得最高值，其中准确率、F1分数、召回率和曲线下面积分别为95.16%、95.14%、95.02%和99.35%。与DeepJIT相比，准确率和F1分数分别提高了42.75%和51.37%。与CC2Vec相比，分别提高了44.70%和52.19%。上述优势的可能原因是，DeepJIT和CC2Vec均以新增、删除代码作为变更建模的主要依据，侧重于学习代码差异中的局部变更特征。然而，判断一次函数变更是否引入缺陷，还需要结合修改所在的函数上下文及相应的代码语义。MPJIT同时使用修改后函数和代码差异，使模型既能定位具体修改，又能结合修改后函数中的控制流程、变量使用和接口调用分析其影响。同时，CodeBERT提供了大规模预训练中学习的代码语义知识，有助于识别仅从局部变更形式较难判断的行为变化。

COMPDEFECT的准确率和F1分数分别为70.95%和68.78%。与COMPDEFECT相比，MPJIT的准确率和F1分数分别提高了24.21%和26.36%。可能的原因是，COMPDEFECT使用修改前后的完整函数进行编码，未将代码差异作为独立输入，模型需要从两个函数版本中学习具体变化。MPJIT保留修改后函数，同时通过代码差异显式标出新增和删除的语句，使该函数变更涉及的修改语句在修改后函数的上下文中更加突出。此外，

COMPDEFECT采用附加分类头完成预测，混合提示调优则通过在掩码位置预测标签词，再映射为二元类别。这种完形填空式预测与CodeBERT的掩码语言建模目标相衔接，可能有助于将预训练阶段学习的代码语义知识用于缺陷判断。

与CodeBERT分类头微调相比，MPJIT的准确率、F1分数和曲线下面积分别提高了2.43%、2.43%和1.71%。两种方法均使用CodeBERT，并以修改后函数和代码差异共同作为代码输入。分类头微调根据输入序列的表示直接预测类别，MPJIT则通过混合提示说明两类输入及预测目标，在掩码位置预测标签词，再映射为相应类别。上述性能提升的可能原因是，混合提示将缺陷分类组织为完形填空任务，使下游预测形式与CodeBERT预训练中的掩码预测形式相近，有助于利用预训练阶段学习的代码语义知识。同时，固定文字区分函数上下文和具体修改，可学习提示向量根据训练样本调整提示表示，使模型能够结合两类信息进行缺陷判断。

Wilcoxon符号秩检验结果显示，MPJIT与DeepJIT、CC2Vec、COMPDEFECT和CodeBERT分类头微调在五项评价指标上的差异均具有统计显著性（$p<0.05$），对应效应量均大于0.5，达到大效应水平。结合表1的结果，MPJIT在本实验设置下取得了优于四个对比方法的提交级预测效果。

4.2　不同代码输入的有效性

为考察修改后函数和代码差异共同输入的作用，实验设置仅输入修改后函数、仅输入代码差异及两者共同输入三种配置，均采用CodeBERT和混合提示。各配置按其输入内容设置代码填入位置，固定文字相应说明函数内容或代码差异。各配置均按相同规则汇总函数预测结果，以代码提交为单位计算评价指标，结果见表2。

表2显示，修改后函数和代码差异共同输入时，五项评价指标均取得最高值，准确率、召回率和F1分数分别为95.16%、95.02%和95.14%。与仅使用代码差异相比，共同输入的准确率、召回率和F1分数分别提高了30.81%、23.17%和28.26%。可能的原因是，代码差异主要呈现新增行、删除行及邻近上下文，而判断修改的影响还需要了解这些语句在函数中的作用。例如，函数调用替换可能影响返回值的后续使用，条件判断修改可能改变分支中执行的操作。修改后函数保留了相关的调用、变量使用和控制流程，使模型能够结合更完整的函数上下文分析局部修改。

表2　不同代码输入下的预测结果

%

代码输入	准确率	精确率	召回率	F1分数	曲线下面积
仅修改后函数	93.18	93.05	93.17	93.11	98.26
仅代码差异	64.35	62.55	71.85	66.88	80.43
修改后函数和代码差异	**95.16**	**95.25**	**95.02**	**95.14**	**99.35**

与仅使用修改后函数相比，共同输入的准确率、召回率和F1分数分别提高了1.98%、1.85%和2.03%。可能的原因是，本次修改已经融入修改后的函数代码，而代码差异能够直接标明新增和删除的语句。其中，删除行保留了修改后函数中已经不存在的代码，为理解替换或删除操作提供依据。共同输入因此既包含变更后的函数内容，也保留本次函数变更的修改信息，使模型能够将具体变化同其所在的函数逻辑联系起来。

共同输入相较于两种单独输入配置的指标提升均通过显著性检验（$p<0.05$），对应效应量均达到大效应水平（$r>0.5$）。这一结果支持了同时使用修改后函数和代码差异的输入设计。

4.3　不同提示类型的有效性

实验在相同的修改后函数和代码差异输入下比较软提示、硬提示和混合提示，均使用CodeBERT及相同的标签词映射。每个候选模板均选择验证集F1分数最高的模型检查点。对于每种提示类型，在验证集上比较候选模板，选择验证集F1分数最高的模板，再报告该模板在测试集上的提交级评价结果。具体模板如下。

1）软提示模板：“[X_{After}] [soft] [X_{Diff}] [soft] [MASK]”。两段代码按修改后函数、代码差异的顺序输入，可学习提示向量分别位于修改后函数和代码差异之后。

2）硬提示模板：“The modified code is [X_{After}]. The code diff is [X_{Diff}]. The code snippet is [MASK].”。三句固定文字分别说明修改后函数、代码差异和待预测类别。

3）混合提示模板：“[soft] The modified code is [X_{After}]. [soft] The code diff is [X_{Diff}]. [soft] The code snippet is [MASK].”。在硬提示的三句固定文字前

分别加入可学习提示向量，与2.2节的模板一致。

表3　不同提示类型的预测结果

%

提示类型	准确率	精确率	召回率	F1分数	曲线下面积
软提示	93.22	93.10	93.35	93.22	97.98
硬提示	93.39	93.43	93.35	93.39	98.38
混合提示	**95.16**	**95.25**	**95.02**	**95.14**	**99.35**

表3显示，在修改后函数和代码差异共同输入时，混合提示的五项指标均高于所比较的软提示和硬提示模板。其中，混合提示的准确率、F1分数和曲线下面积分别为95.16%、95.14%和99.35%。软提示和硬提示的F1分数分别为93.22%和93.39%。

与软提示相比，混合提示的准确率和F1分数分别提高了1.94%和1.92%。可能的原因是，混合提示中的固定文字分别说明修改后函数、代码差异和待预测类别，使两段代码以不同含义参与预测。软提示通过可学习向量提供任务提示，混合提示在此基础上保留明确的文字说明，将函数上下文、具体修改和类别预测组织在同一模板中。这些文字有助于模型区分两类输入，并根据代码内容预测掩码位置的标签词。

与硬提示相比，混合提示的准确率和F1分数分别提高了1.77%和1.75%。两种模板使用相同的固定文字，混合提示在三句文字之前分别加入可学习提示向量。可能的原因是，固定文字说明了输入内容和预测任务，可学习向量则根据训练样本调整提示表示，使同一模板适应函数变更的缺陷判断。相较于仅使用固定文字，这些向量为模型提供了可随训练更新的任务提示。

混合提示与软提示、硬提示的两组比较在五项评价指标上的差异均达到统计显著水平（$p<0.05$），效应量均大于0.5。

4.4　案例分析

（1）正确预测案例。Deeplearning4j项目[①]提供面向Java环境的深度学习框架，可用于物联网设备上的模型部署和数据分析。本例中的getAllUpdateTimes()函数用于返回指定会话、类型和工作节点的全部更新时间戳。如图3所示，本次修改将数组返回调用由toArray()替换为elements()。函数将数据库查询得到的时间戳逐条加入LongArrayList，随后返回数组。修改前，toArray()返回列表中的实际元素。修改后，elements()直接返回底层存储数组。当底层数组的容量大于实际元素数量时，未使用位置的零值也会进入返回结果，使返回数组包含实际查询记录之外的值。因此，数据集将该函数变更标为有缺陷。

```
public long[] getAllUpdateTimes(String sessionID, String typeID, String workerID) {
  ...
  LongArrayList list = new LongArrayList();
  while (rs.next()) {
    list.add(rs.getLong(1));
  }
-  return list.toArray();
+  return list.elements();
  ...
}
```

图3　getAllUpdateTimes()函数的正确预测案例

```
public Builder memoryCachePercent(int memoryCachePercent) {
    ...
-     int capacity = (1024 * 1024 * (memClass * memoryCachePercent)) / 100;
+     int capacity = (int) ((Runtime.getRuntime().maxMemory()
+             * (memoryCachePercent / 100)));
    ...
}
```

图4　memoryCachePercent()函数的错误预测案例

DeepJIT和CC2Vec均将该函数变更所属的提交误判为无缺陷。可能的原因是，这两种方法主要从提交消息及新增、删除代码行中学习变更特征，而本次修改仅替换了一处数组返回调用。两个接口在代码形式上均用于返回数组，但是返回内容不同，分别为实际元素和底层存储数组。这种接口行为差异需要相应的代码语义知识，仅从标注样本中的局部变更特征学习，较难判断替换后是否会引入额外元素。COMPDEFECT也将该变更误判为无缺陷。该方法使用GraphCodeBERT编码修改前后的函数，但未显式标出本次修改的语句。两个函数版本仅有一处调用不同，模型需要从两段相近的代码中自行发现toArray()到elements()的替换。MPJIT通过代码差异直接标出这处替换，使模型能够结合函数上下文分析其影响。

在输入配置实验中，仅使用修改后函数和仅使

①Deeplearning4j项目代码仓库：https://github.com/deeplearning4j/deeplearning4j

用代码差异的模型也将该变更误判为无缺陷。对于仅使用修改后函数的模型，查询和列表构建过程提供了完整上下文，但是缺少显式的接口替换信息，模型容易忽略本次函数变更改变了数组的返回方式。对于仅使用代码差异的模型，接口替换清晰可见，但是局部上下文较难完整呈现时间戳的查询和添加过程，因而不易将接口的返回语义与列表的实际使用方式相关联。

相比之下，MPJIT将该变更正确预测为有缺陷。可能的原因是，代码差异明确指出发生变化的是数组返回接口，修改后函数则展示了数据库查询、列表初始化和时间戳逐条添加的过程。结合两类输入可以看出，函数需要返回查询得到的全部时间戳，而替换后的接口返回底层存储数组，其长度可能超过实际时间戳数量。混合提示将这处接口替换及其所在的函数上下文组织到同一输入中，使模型能够结合具体修改分析返回结果的变化。此外，通过掩码位置预测标签词，将缺陷预测组织为完形填空任务。这一预测形式与CodeBERT的掩码语言建模目标相衔接，有助于利用预训练阶段学习的代码语义，判断接口替换对函数行为的影响。

（2）错误预测案例。Android-Universal-Image-Loader项目[①]提供Android应用中的图像加载和缓存功能，可用于物联网移动应用中的监测图像展示。其中，memoryCachePercent()函数用于按指定百分比设置内存缓存容量。图4展示了该函数的关键变更，“+”表示新增行，“-”表示删除行。本次函数变更将基于memClass的容量计算改为使用Runtime. getRuntime(). maxMemory()， 并将memoryCachePercent / 100放入乘法表达式内部。数据集将该函数变更标为有缺陷，MPJIT却将其预测为无缺陷。

该函数将百分比参数memoryCachePercent的取值限定在1到80之间。由于该参数和除数100均为整数，memoryCachePercent/100执行整数除法，结果始终为0，导致计算得到的缓存容量也为0。后续if(capacity>0)分支被跳过，本次设置因而未能更新memoryCacheSize。修复版本将100改为100f，使百分比计算采用浮点除法，从而保留小数部分。

MPJIT将该变更误判为无缺陷，可能的原因是模型未能结合参数类型和取值范围，识别整数除法对计算结果的影响。修改前后的表达式都采用乘法和除法计算缓存容量，但是修改后先计算括号内的memoryCachePercent/100，使整数除法产生的零值影响了整个容量计算。代码差异呈现了表达式的变化，判断其是否引入缺陷还需要应用相应的数值运算规则。针对这类涉及数值运算的变更，后续可根据代码中的变量类型和运算表达式，检索相关的语言规则及缺陷知识，并作为补充信息加入提示模板[31]。例如，本例可补充整数除法的计算规则，帮助模型结合参数取值范围推断表达式的结果，进一步判断其对缓存容量更新的影响。

5 结束语

针对现有提示调优方法难以完整描述函数变更的问题，本文提出一种基于混合提示调优的函数级即时缺陷预测方法。该方法将修改后函数和代码差异共同填入混合提示模板，以固定文字说明两类代码的含义，并通过可学习提示向量调整提示表示，再利用CodeBERT在掩码位置预测标签词，判定函数变更及其所属提交是否有缺陷。在DFunction-SStuBs4J数据集上的实验结果表明，MPJIT的F1分数和曲线下面积分别达到95.14%和99.35%，F1分数相比对比方法至少提高了2.43%。两类代码共同输入优于单独使用其中一种代码，混合提示模板也优于仅使用固定文字或仅使用可学习提示向量的模板。在未来工作中，将按时间顺序划分数据并开展跨项目实验，在更多物联网平台和智能家居项目上验证MPJIT的适用性。

参考文献：

[1] 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.

[2] ZENG Z, ZHANG Y, ZHANG H, et al. Deep just-in-time defect prediction: How far are we?[C]//International Symposium on Software Testing and Analysis, 2021: 427-438.

[3] YANG Z, GAO C, GUO Z, et al. A roadmap for modern code review: Challenges and opportunities[J]. ACM Transactions on Software Engineering and Methodology, 2026, 35(9): 271:1-42.

[4] 束方磊，刘佳伟，李博睿，等. 物联网应用开发与生成：从低代码到智能原生[J]. 物联网学报，2026，10(1)：112-124.
SHU F L, LIU J W, LI B R, et al. IoT application development

①Android-Universal-Image-Loader项目代码仓库：https://github.com/nostra13/Android-Universal-Image-Loader

and generation: From low-code to AI-native[J]. Chinese Journal on Internet of Things, 2026, 10(1): 112-124.

[5] 宋航，才建，袁运栋，等. 意图驱动的物联网双时间尺度资源分配方法[J]. 物联网学报，2026，10(1)：216-225.
SONG H, CAI J, YUAN Y D, et al. Intent-driven dual-time-scale resource allocation approach for Internet of things[J]. Chinese Journal on Internet of Things, 2026, 10(1): 216-225.

[6] RADOVICI A, CULIC I, ROSNER D, et al. A model for the remote deployment, update, and safe recovery for commercial sensor-based IoT systems[J]. Sensors, 2020, 20(16): 4393.

[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]//International Conference on Mining Software Repositories, 2019: 34-45.

[8] HOANG T, KANG H J, LO D, et al. CC2Vec: Distributed representations of code changes[C]//International Conference on Software Engineering, 2020: 518-529.

[9] NI C, YANG K, ZHU Y, et al. Unifying defect prediction, categorization, and repair by multi-task deep learning[C]//International Conference on Automated Software Engineering, 2023: 1980-1992.

[10] GUO Y X, GAO X P, ZHANG Z Y, et al. A study on the impact of pre-trained model on just-in-time defect prediction[C]//International Conference on Software Quality, Reliability and Security, 2023: 105-116.

[11] 张犬俊，谢杨，房春荣，等. 检索增强生成在软件工程中的应用综述[J]. 软件学报，2026，37(3)：1316-1339.
ZHANG Q J, XIE Y, FANG C R, et al. Survey on application of retrieval-augmented generation in software engineering[J]. Journal of Software, 2026, 37(3): 1316-1339.

[12] XUE J, CHEN X, WANG J, et al. Towards prompt tuning-based software vulnerability assessment with continual learning[J]. Computers & Security, 2025, 150: 104184.

[13] SCHICK T, SCHÜTZE H. Exploiting cloze-questions for few-shot text classification and natural language inference[C]//Conference of the European Chapter of the Association for Computational Linguistics, 2021: 255-269.

[14] WANG C, YANG Y, GAO C, et al. Prompt tuning in code intelligence: An experimental evaluation[J]. IEEE Transactions on Software Engineering, 2023, 49(11): 4869-4885.

[15] YU L, LU J Y, LIU X L, et al. PSCVFinder: A prompt-tuning based framework for smart contract vulnerability detection[C]//International Symposium on Software Reliability Engineering, 2023: 556-567.

[16] REN Z, JU X, CHEN X, et al. ProRLearn: Boosting prompt tuning-based vulnerability detection by reinforcement learning[J]. Automated Software Engineering, 2024, 31(2): 38.

[17] WANG J, CHEN X, PEI W, et al. Improving prompt tuning-based software vulnerability assessment by fusing source code and vulnerability description[J]. Automated Software Engineering, 2025, 32(2): 45.

[18] ZHU J, LI L W, YANG L, et al. Automating method naming with context-aware prompt-tuning[C]//International Conference on Program Comprehension, 2023: 203-214.

[19] FENG Z, GUO D, TANG D, et al. CodeBERT: A pre-trained model for programming and natural languages[C]//Findings of the Association for Computational Linguistics: EMNLP 2020, 2020: 1536-1547.

[20] LIU Z, TANG Z, XIA X, et al. CCRep: Learning code change representations via pre-trained code model and query back[C]//International Conference on Software Engineering, 2023: 17-29.

[21] LIN B, WANG S, LIU Z, et al. CCT5: A code-change-oriented pre-trained model[C]//Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2023: 1509-1521.

[22] NAM D, KIM T, RYU D, et al. ReDef: Do code language models truly understand code changes for just-in-time software defect prediction?[C]//International Conference on the Foundations of Software Engineering, 2026: 3909-3930.

[23] WU D, LI W, LV G, et al. KnowJIT: Knowledge-augmented just-in-time defect prediction and repair via in-context learning[J]. IEEE Transactions on Reliability, 2026, 75: 3861-3875.

[24] YANG Y, ZHOU Y, LIU J, et al. Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models[C]//International Symposium on Foundations of Software Engineering, 2016: 157-168.

[25] 刘子扬，祝义，周湘，等. 基于个性化联邦学习的跨项目软件缺陷预测方法[J]. 软件学报，2026，37(7)：2911-2935.
LIU Z Y, ZHU Y, ZHOU X, et al. Cross-project software defect prediction method based on personalized federated learning[J]. Journal of Software, 2026, 37(7): 2911-2935.

[26] HUANG Y, CHEN M, CHEN X, et al. Towards integrating large model with small model for just-in-time defect prediction and localization[J]. Information and Software Technology, 2026, 198: 108244.

[27] LIU S, KEUNG J, JIN Z, et al. An empirical study of parameter-efficient fine-tuning in code change learning and beyond[J]. IEEE Transactions on Software Engineering, 2026, 52(1): 3-21.

[28] WANG B, DENG Y, LUO R, et al. MPLinker: Multi-template prompt-tuning with adversarial training for issue-commit link recovery[J]. Journal of Systems and Software, 2025, 223: 112351.

[29] LESTER B, AL-RFOU R, CONSTANT N. The power of scale for parameter-efficient prompt tuning[C]//Conference on Empirical Methods in Natural Language Processing, 2021: 3045-3059.

[30] 徐圣斌，贾林杰，许汤智，等. 基于大语言模型的提交日志生成[J/OL]. 软件学报，2026：1-19.
XU S B, JIA L J, XU T Z, et al. Commit message generation based on large language models[J/OL]. Journal of Software, 2026: 1-19.

[31] ZHU H, LI J, GAO C, et al. VulInstruct: Teaching LLMs root-

cause reasoning for vulnerability detection via security specifications[C]//International Conference on the Foundations of Software Engineering, 2026: 2813-2834.

[32] LI Z, LU S, GUO D, et al. Automating code review activities by large-scale pre-training[C]//Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022: 1035-1047.

[33] CIBOROWSKA A, DAMEVSKI K. Fast changeset-based bug localization with BERT[C]//International Conference on Software Engineering, 2022: 946-957.

[34] DING N, HU S, ZHAO W, et al. OpenPrompt: An open-source framework for prompt-learning[C]//Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2022: 105-113.

[35] KARAMPATSIS R M, SUTTON C. How often do single-statement bugs occur? The ManySStuBs4J dataset[C]//International Conference on Mining Software Repositories, 2020: 573-577.

[36] FU M, PASUKSMIT J, TANTITHAMTHAVORN C. AI for DevSecOps: A landscape and future opportunities[J]. ACM Transactions on Software Engineering and Methodology, 2025, 34(4): 115:1-61.

[37] DING Y, FU Y, IBRAHIM O, et al. Vulnerability detection with code language models: How far are we?[C]//International Conference on Software Engineering, 2025: 1729-1741.