文章编号:1671-4598(2025)08-0086-08 DOI:10.16526/j.cnki.11-4762/tp.2025.08.012 中图分类号:TP311.5 文献标识码:A

基于特征融合的语句级别软件故障定位

傅珍蕾[1,2]，沈祖雄[1,2]，樊 鑫[1,2]
(1. 南昌航空大学 软件学院，南昌 330038；2. 南昌航空大学 软件测评中心，南昌 330038)

摘要：针对软件故障定位任务中提取的特征不全面、将不同特征对故障的贡献无差化的问题，提出了一种基于特征融合的语句级别软件故障定位方法；对每条语句进行语义上的扩展，采用Doc2Vec技术提取扩展后语句的语义信息；选用6种基于频谱的故障定位公式来获取频谱信息，选用两种基于变异的故障定位公式来获取变异信息；采用注意力机制对3种来自不同信息源的特征进行融合处理，自动学习对故障最有效的特征；在Defects4J数据集的5个真实项目进行了实验，采用基于注意力机制的多特征融合在Top-K（K=1，3，5）上能够多定位11～16个故障，在MRR上提高了5.17%，实验结果表明，所提方法能够有效提高模型的定位性能。

关键词：多特征融合；软件故障定位；注意力机制；程序频谱；变异测试

Statement-level Software Fault Localization Based on Feature Fusion

FU Zhenlei[1,2], SHEN Zuxiong[1,2], FAN Xin[1,2]
(1. School of Software, Nanchang Hangkong University, Nanchang 330038, China;
2. School of Testing and Evaluation Center, Nanchang Hangkong University, Nanchang 330038, China)

Abstract: The software fault location has incomplete feature extraction, and different features contribute equally to the fault. To solve this problem, a statement-level software fault localization method based on feature fusion is proposed. Extend each sentence semantically, use Doc2Vec technology to extract the semantic information of the extended sentence, select six spectrum-based fault location formulas to obtain spectrum information, adopt two mutation-based fault location formulas to obtain mutation information, use attention mechanism to fuse features from three different information sources, and automatically learn the most effective features for faults. Experiments are carried out on five real projects of Defects4J dataset. The multi-feature fusion method based on attention mechanism can locate 11 to 16 faults on Top-K (K=1, 3, 5), and improve the mean reciprocal rank (MRR) by 5.17%. Experimental results show that the proposed method can effectively improve the localization performance of the model.

Keywords: multi-feature fusion; software fault localization; attention mechanism; program spectrum; mutation testing

0 引言

近年来，随着信息技术的飞速发展，计算机软件已经渗透到社会的每一个角落，成为推动各行各业进步与创新的关键力量，同时也改变了人们的生活方式。如今的软件呈现出功能多样化、规模复杂化的特点，其中不可避免地会存在一些漏洞或错误。据估计，软件中的错误每年给美国经济造成595亿美元的损失[1]。然而，发现错误与找到错误是完全不同的任务，故障定位侧重于后者，在历史上其通常是一项人工任务，被认为是极其昂贵和耗时的活动。据统计，对于一些项目而言，调试软件故障的费用可能占软件总成本的80%[2]。对此，许多研究人员提出了多种自动化故障定位技术来减少人员的参与，并提高定位的效率。

当前研究最为广泛的自动化故障定位技术是基于频谱的故障定位（SBFL，spectrum-based fault localization），这是一种轻量级的技术，并且能够直观地反映程序的运行信息[3]。程序的运行信息即为程序执行频谱，通过频谱来计算元素可疑值并排序定位。目前已经提出诸如Dstar[4]、Ochiai[5]、Ochiai2[6]、Tarantula[7]、Jaccard[8]和OP[6]等各种不同的可疑值计算公式，然而，它们只依据程序执行频谱来确定可疑元素，而忽略了程序的上下文信息。文献[9]提出一种利用故障传播上下文对可疑元素进行加权，进而重新计算元

收稿日期:2025-03-14； **修回日期**:2025-04-02。

作者简介:傅珍蕾(1998-),女,硕士。

引用格式:傅珍蕾,沈祖雄,樊 鑫.基于特征融合的语句级别软件故障定位[J].计算机测量与控制,2025,33(8):86-93.

素可疑值，以此提高 SBFL 的定位性能。文献［10］使用 PageRank 算法分析测试用例与被测程序的关系，从而考虑不同测试用例的贡献并重新计算频谱。文献［11］通过引入软件实体的局部影响计算，提出一种增强频谱的故障定位的方法，能够充分利用软件实体之间的内部交互信息，还提出了一种新的可疑值度量方法来对可疑元素进行综合评价。基于变异的故障定位（MBFL，mutation-based fault localization）是另一种自动化定位技术，MUSE[12] 和 Metallaxis[13] 是典型的 MBFL 技术，是通过引入人为的程序缺陷（即变异体）来检查每个元素对测试结果的影响[2]。文献［14］提出根据项目历史记录中的早期版本进行变异分析，采用统计推理方法，对变异体的测试结果与其位置进行建模，并推断出故障的位置。为了解决 MBFL 的平局问题[15]，文献［15］通过将变异体计算出来的可疑值和 PageRank 算法计算的不完美度分数进行加权求和，以获得程序元素的可疑值。

尽管传统的 SBFL 技术和 MBFL 技术在一定程度上能获得良好的定位性能，但 SBFL 主要关注单个错误的定位，并且比较依赖测试用例的质量；而 MBFL 通过在程序中注入大量的缺陷会导致额外的运行开销，从而影响故障定位的效率。因此部分研究人员将机器学习或深度学习与传统方法进行结合。文献［16］通过预训练得到了语言模型，其中学习到的表示包含了大量关于语句可疑性的知识，利用这些知识来查找故障，但该方法忽略了动态分析的重要性。文献［17］主张应该明确考虑流的性质，通过设计一个沿着执行路径传输语句语义的基于流的门控循环单元（GRU，gate recurrent unit）来从控制流图（CFG，control flow graph）中学习特征，在路径收敛于同一语句时合并路径。文献［18］通过构建程序依赖图（PDG，program dependence graph）来表示故障上下文，再利用图卷积神经网络（GCN，graph convolutional neural network）来分析上下文并将其合并到可疑性评估中。上述两个方法分别构建 CFG 和 PDG 来完成定位，但这两个图表示并不包含类与类之间的依赖关系。文献［19］将抽象语法树（AST，abstract syntax tree）和语句序列相结合，并通过对比学习来提取错误代码的特征，采用基于注意力的长短期记忆递归神经网络来定位错误元素，该方法只提取了静态特征，而忽略了动态特征。文献［20］对软件源代码进行分析并测试，获得软件网络和程序频谱，建立网络频谱并对其进行特征降维和类不平衡处理，该方法将程序频谱作为节点特征，忽略了程序的语义信息。

针对上述所提的类与类之间的依赖关系，我们在之前的研究中已经证实了它的重要性[21]。在文献［21］中虽然同时考虑了语句的语义特征和频谱特征，但仍然存在两个问题：1）特征的覆盖面太窄，仅从两个维度来获取的故障诊断信息不够全面；2）将它们对故障的贡献视为同等的，而事实上，不同类型的特征对故障的贡献是参差不齐的。对此，本文提出了基于特征融合的语句级别软件故障定位，该方法是针对上述研究中阶段二的进一步改进，从而实现更高的定位性能。具体来说，对于一个存在故障的方法，主要提取了方法内每条语句的 3 种特征信息，分别是语义信息、频谱信息和变异信息。其中语义信息属于静态特征，它包含了当前语句及其执行上下文的信息。频谱信息和变异信息属于动态特征，分别从 6 种不同的传统 SBFL 技术和两种不同的 MBFL 技术中收集而来，它们都包含了语句与软件故障之间的相关程度。采用注意力机制对以上 3 种特征进行融合，在融合过程中突出重要特征并抑制无关特征。然后构建一个以孪生 MLP 为主要框架结构的 RankNet 模型对语句进行两两组合比较可疑值大小。

1　方法总体框架

本文方法的整体框架结构如图 1 所示，整个方法主要包括两个核心模块：特征提取与融合和排序模型。

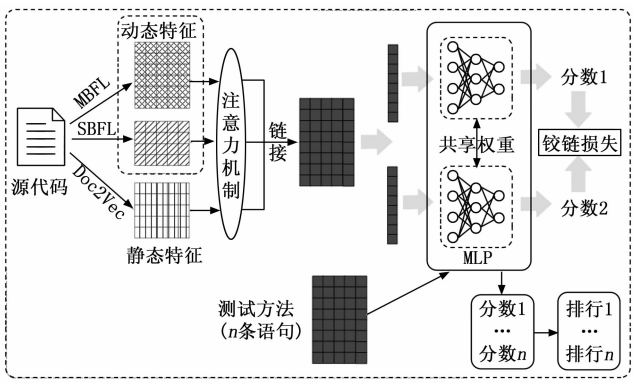

图 1　本文方法框架

在特征提取与融合模块，本章重点提取语句的两类特征：静态特征和动态特征。其中静态特征主要为语句的语义信息，通过 Doc2Vec 技术来提取；动态特征主要包括语句的频谱信息和突变信息，分别从 SBFL 技术和 MBFL 技术中获得。再采用注意力机制对 3 种特征进行融合处理，作为后续排序模型的输入。

在排序模型的构建模块，将一个方法中的所有代码行进行两两配对，每行代码只配对一次，将它们成对地输入到共享权重的孪生多层感知器（MLP，multi-layer perceptron）中。两个 MLP 为每对输入分别输出两个预测分数，然后将这两个预测分数通过铰链损失函数连接在一起来训练模型。

2 研究方法

2.1 特征提取

2.1.1 语义信息

由于单条语句所含有的语义信息有限，并且在语句可疑值排序列表中，程序员很难根据单条语句来确定其是否为故障所在，尤其是在对程序不熟悉的情况下，因此本文需要对原始语句进行扩展。如果本文根据原始语句的相邻上下文来扩充，那么在语义信息中可能会忽略程序的执行顺序，因为代码的执行并不总是自上而下的(例如循环语句和条件判断语句)，且相邻上下文并不能从程序执行的角度来提供有关故障的语义故障诊断信息。因此本文选择执行上下文来对每条语句进行扩展，按照程序的执行顺序将每条语句扩展为执行前语句－原始语句－执行后语句，然后采用 Doc2Vec 技术将扩展后的语句转换为低维空间向量表示。Doc2Vec[22]是一种用于文本表示学习的技术，是在 Word2Vec 的基础上进行了扩展。相比 Word2Vec 技术，Doc2Vec 能够处理可变长度的文本（如句子、段落或整个文档）。通过 Doc2Vec 模型，可以将扩展后的语句表示为一个固定长度的向量，这个向量捕捉了代码的语义信息和语境，其过程如下。

将每一个扩展后的代码看作一个句子，并用唯一的向量 $\boldsymbol{d}_i$ 来表示，所有句子中的每个词（即 token）都被映射为唯一的向量表示，即 $\boldsymbol{W}=\{w_1,w_2,\cdots,w_m\}$。每次训练从一句话中滑动采样固定长度的词，取其中一个词 $\boldsymbol{w}_i$ 作预测词，其他的词和整个句子作为输入词，得到预测词 $\boldsymbol{w}_i$ 的目标函数为最大化平均对数概率，如公式(1)所示：

$$\frac{1}{m}\sum_{i=k}^{m-k}\log p(\boldsymbol{w}_i \mid \boldsymbol{w}_{i-k},\cdots,\boldsymbol{w}_{i+k}) \tag{1}$$

其中：$\boldsymbol{w}_{i-k},\cdots,\boldsymbol{w}_{i+k}$ 为采样滑动窗口大小为 $2k+1$ 的词向量，而预测词 w_i 的预测概率通常是通过多分类(如 softmax 分类器)完成的，如公式(2)所示：

$$p(\boldsymbol{w}_i \mid \boldsymbol{w}_{i-k},\cdots,\boldsymbol{w}_{i+k})=\frac{\exp(y_{w_i})}{\sum_j \exp(y_j)} \tag{2}$$

其中：y_j 表示第 j 个输出的词在未归一化的情况下的对数概率值，计算如公式(3)所示：

$$y=\boldsymbol{U}\boldsymbol{h}(l_{i-k},\cdots,l_{i+k},\boldsymbol{d}_s)+b \tag{3}$$

其中：$\boldsymbol{U}$ 和 b 为 softmax 参数，$h(\cdot)$ 为拼接滑动窗口中采样的词向量和句子向量 $\boldsymbol{d}_s$。然后采用随机梯度下降对词向量和句子向量不断迭代训练并自动更新，在对同一个句子的多次训练中句子向量是共享的，因此随着滑动窗口的移动，句子向量都会作为输入的一部分来训练，该句子表达的语义会越来越准确。

2.1.2 频谱信息

程序频谱信息也称为覆盖信息，是指执行测试用例期间程序元素的动态运行时信息。通过分析程序频谱可以得到 4 个参数：ep、ef、np 和 nf，如表 1 所示。当一条语句被失败测试用例执行的数量（ef）越多，并且被通过测试用例执行的数量（ep）越少，则该语句被认为最有可能存在故障，因此从这 4 个参数中可以分析出程序的不稳定或异常行为模式。

表 1 SBFL 技术中使用的参数

参数	描述
ep	执行语句通过测试用例数量
ef	执行语句失败测试用例数量
np	未执行语句通过测试用例数量
nf	未执行语句失败测试用例数量

在过去几十年间设计了各种不同的可疑值计算公式，文献[23]提出的 Multric 根据 25 种传统的 SBFL 技术计算出来的可疑值来学习故障的位置。本文选择 Dstar[4]、Ochiai[5]、Ochiai2[6]、Tarantula[7]、Jaccard[8]和 OP[6]共 6 种公式来收集基于频谱的特征，如表 2 所示。

表 2 基于频谱的计算公式

名称	公式
Dstar	$\frac{ef^2}{np+ef}$
Ochiai	$\frac{ef}{\sqrt{(nf+ef)\times(np+ep)}}$
Ochiai2	$\frac{ef\cdot ep}{\sqrt{(ef+ep)\times(nf+np)\times(ef+np)\times(nf+ep)}}$
Tarantula	$\frac{\frac{ef}{ef+nf}}{\frac{ef}{ef+nf}+\frac{ep}{ep+np}}$
Jaccard	$\frac{ef}{ef+nf+ep}$
OP	$ef-\frac{ep}{ep+np+1}$

2.1.3 变异信息

变异信息是指利用 MBFL 方法对源代码中的运算符或变量名等进行变异，并通过执行测试用例来收集运行时信息。MBFL 技术基于的思想是：错误语句容易通过某种变异方式被修复，变异被修复的比例越大，则该语句为错误语句的概率越高。对于一条语句 e，采用 MBFL 技术产生了一组变异体 $M(e)=\{M_1(e),M_2(e),\cdots,M_n(e)\}$，在每个变异体上执行全部的测试用例，并记录每个变异体在执行测试用例时的行为，包括成功测试用例与失败测试用例的数量。然后计算每一个变异体的可疑分数，公式(4)是使用 MUSE[12]进行

计算：

$$sus_{\mathrm{MUSE}}[\mathrm{M}_i(e)] = \frac{ef \cap ep_{\mathrm{M}_i(e)}}{Total_p} - \alpha \cdot \frac{ep \cap ef_{\mathrm{M}_i(e)}}{Total_f} \tag{4}$$

其中：$ep_{\mathrm{M}_i(e)}$ 和 $ef_{\mathrm{M}_i(e)}$ 分别表示语句 e 的第 i 个变异体上的通过测试用例和失败测试用例的数量，$Total_p$ 和 $Total_f$ 分别表示通过测试用例和失败测试用例总数，α 表示一个度量权重，其定义如公式（5）所示：

$$\alpha = \frac{f2p}{ep_{\mathrm{M}_i(e)}} \cdot \frac{ef_{\mathrm{M}_i(e)}}{p2f} \tag{5}$$

其中：$f2p$ 表示在一个方法内语句进行变异之前和之后，测试用例从失败到通过的测试结果变化的数量，$p2f$ 则相反。

除了 MUSE，本文还选择了 Metallaxis[13]，在 Metallaxis 中，将对测试用例的执行结果有影响的变异体视为原始语句被测试用例覆盖，反之，则未覆盖。其计算变异体可疑值是使用传统的基于频谱的公式，公式（6）即 Ochiai 公式，因此本节采用 6 种基于频谱的公式来实现 6 个 Metallaxis 变异体的计算：

$$sus_{\mathrm{Metallaxis}}[\mathrm{M}_i(e)] = \frac{ef_{\mathrm{M}_i(e)}}{Total_f \cdot [ef_{\mathrm{M}_i(e)} + ep_{\mathrm{M}_i(e)}]} \tag{6}$$

2.2　特征融合

上一节中，已经获得了语句的 3 种特征信息：代码语义信息、频谱覆盖信息和变异信息，为了能充分利用这些特征信息，需要将这 3 种特征进行融合，获得一个含有更丰富信息的特征。一些常见的融合方法是进行简单的求和或串联，这种方式虽然简单直观，但可能并不是最佳选择。本文采用基于自注意力机制的特征融合方法，自注意力机制更加关注输入特征的内部相关性，通过学习权重分布，对特征进行加权处理，以从不同特征中提炼最有用的故障诊断信息。融合过程如图 2 所示。

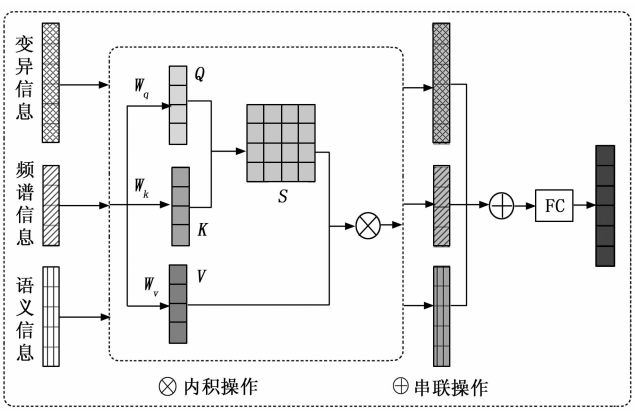

图 2　特征融合过程

图 3 为从特征中提炼故障诊断信息的过程。对于一条语句的每种特征 $\boldsymbol{a} = \{a_1, a_2, \cdots, a_N\}$，首先将其分别乘以 3 个可训练权重矩阵 $\boldsymbol{W}_q$，$\boldsymbol{W}_k$，$\boldsymbol{W}_v$，可以得到 $Q = \{q_1, q_2, \cdots, q_N\}$、$K = \{k_1, k_2, \cdots, k_N\}$、$V = \{v_1, v_2, \cdots, v_N\}$，如公式（7）～（9）所示：

$$\boldsymbol{Q} = \boldsymbol{a} \cdot \boldsymbol{W}_q \tag{7}$$

$$K = \boldsymbol{a} \cdot \boldsymbol{W}_k \tag{8}$$

$$V = \boldsymbol{a} \cdot \boldsymbol{W}_v \tag{9}$$

其中：Q 表示向量 $\boldsymbol{a}$ 中的元素与其他元素之间进行匹配，K 表示向量 $\boldsymbol{a}$ 中元素的关键信息，V 表示向量 $\boldsymbol{a}$ 中元素的重要信息表示。

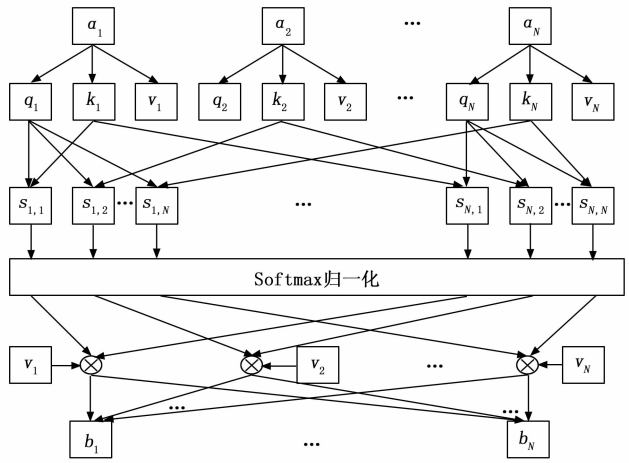

图 3　故障诊断信息的提炼过程

为了计算特征中任意一个元素 a_i 与其他元素 a_j 之间的关联性，需要将 q_i 和 k_j 进行匹配计算，计算公式如（10）所示：

$$s_{i,j} = \mathrm{Attention}(q_i, k_j) = \frac{q_i \cdot k_j^{\mathrm{T}}}{\sqrt{d_K}} \tag{10}$$

其中：d 为 Q 或 K 的矩阵维度，在这里除以 $\sqrt{d_K}$ 是为了防止 q_i 和 k_j 的点乘结果较大。$s_{i,j}$ 即为元素 a_i 和 a_j 的注意力分数。再使用 softmax 函数对注意力分数进行归一化处理，如公式（11）表示，得到所有权重系数之和为 1 的概率分布，这一步操作利用 softmax 函数的特性突出重要元素的权重。根据注意力权重对特征进行加权求和，得到加权后的表示，如公式（12）所示：

$$\alpha_{i,j} = \mathrm{softmax}(s_{i,j}) = \frac{\exp(s_{i,j})}{\sum_{n=1}^{N} \exp(s_{i,n})} \tag{11}$$

$$b_i = \sum_{j=1}^{N} \alpha_{i,j} \cdot v_j \tag{12}$$

最后，将来自不同特征维度的故障诊断信息进行串联，为了防止特征维度过高，采用一个全连接层 FC 对融合后的特征进行降维处理，如公式（13）所示：

$$\boldsymbol{h}_{\mathrm{fuse}} = FC(\boldsymbol{b}_{\mathrm{text}} \| \boldsymbol{b}_{\mathrm{spec}} \| \boldsymbol{b}_{\mathrm{mut}}) \tag{13}$$

2.3　模型构建

为了对每条语句的可疑值进行排序，本文采用

RankNet 排序模型，RankNet 模型在排序前并不在意文档的具体值，而是更加关注它们的相对关系。选用一对共享权重的 MLP 作为 RankNet 的主要框架结构。

对于方法内语句可疑值的目标排序，事实上，在进行实验之前已经在每个方法中对故障代码行进行了标记，但是如果只是简单地将故障代码行标记为 1，其他正确的代码行标记为 0，那么当两行正确代码或两行故障代码作为一个样本对时，模型无法正常训练并学习它们相对排序。在以往的研究中，SBFL 技术因其具有轻量和高效的特点，已被绝大多数研究人员认可[24-26]。因此，本章借助 SBFL 技术所计算出来的每条语句的可疑值来对所有样本进行排序。在这里，并不关注具体值，而是利用它们的相对大小。由于 SBFL 技术存在可疑值并列排名问题，对于可疑值相等且不包含故障的代码行，将对它们随机排序，因为这并不会影响到对故障代码的排序。为了使模型在训练的时候能够识别出故障语句，将故障代码行列于最前端。

此外，在损失函数的选择上，传统 RankNet 技术通常采用交叉熵损失函数。然而，对于 2.1 节中提取的 3 种特征，可以发现位于同一基本块中的不同语句，其频谱覆盖信息是相同的，变异信息是通过对语句进行变异后，记录测试用例的执行情况，本质上也是一种频谱覆盖信息，而语句的语义信息在一定程度上是相似的，尤其是对于包含相似功能或逻辑的代码行。当输入的一对语句的特征之间差异很小时，RankNet 的预测概率将会非常接近，使用交叉熵损失函数可能并不能很好的区分这些语句的相对顺序，因此本文选择铰链损失函数。假设一个方法中有 N 条代码行，那么总共需要处理 $(N^2-N)/2$ 次输入，则铰链损失函数的定义如公式 (14) 所示：

$$Loss = \frac{2}{N^2-N}\sum_{i=1}^{N}\sum_{\substack{j=1\\ e_i \triangleright e_j}}^{N} max[0, margin + MLP(\boldsymbol{h}_{e_j}) - MLP(\boldsymbol{h}_{e_i})] \tag{14}$$

其中：$e_i \triangleright e_j$ 表示语句 e_i 的可疑值大于语句 e_j，$margin$ 表示铰链损失函数的安全系数，$MLP(\cdot)$ 表示一对语句的特征作为样本对分别输入到孪生 MLP 中获得的可疑分数。这个损失函数意味着当 $MLP(\boldsymbol{h}_{e_i}) - MLP(\boldsymbol{h}_{e_j})$ 的值大于 $margin$ 时，损失为 0，这表明只有当 $MLP(\boldsymbol{h}_{e_i})$ 与 $MLP(\boldsymbol{h}_{e_j})$ 的间隔不足或模型分类错误的样本才会贡献损失。通过这种设计，使得模型在训练时会更加关注同一基本块中的不同语句，以区分不同样本对的相对顺序。

在经过充分的训练后，对于测试数据集的输入不再采用成对的形式，而是单个输入到孪生 MLP 中的任意一个，因为它们共享相同的参数，最后根据 MLP 输出的得分对方法内的所有代码行进行排序。

3 实验设置

3.1 实验环境和数据集

实验是在处理器为 Intel Core i5-12490F、显卡为 NVIDIA GeForce RTX4060 8G 和 32G * 2 双通道内存的硬件环境下进行的，采用 JDK1.8、Python3.8 和 PyTorch2.0 等软件环境下进行编码实现。

实验所采用的参数设置为：初始学习率设置为 0.001，dropout 设置 0.1，batch size 设置为 16，模型的默认训练周期 epoch 设置为 100，采用 Adam 优化器。在提取代码的语义特征时，使用嵌入向量大小为 100 的单词来训练 Doc2Vec 模型。

为了便于和其他学者的研究进行评估比较，本文选用了 Defects4J（版本 2.0.1）基准中的 5 个常用的开源 Java 项目，分别是 JFreeChart、Closure Compiler、Commons Lang、Commons Math 和 Mockito，共包含 405 个真实错误，上述 5 个项目的所有错误版本均可以在 Defects4J 资源库中下载。对于每一个错误版本，都提供了对应的问题追踪记录，并且都经过了最小化处理，剔除了重构或功能添加等不相关变更，确保了故障的核心清晰可见。此外，每个错误都有一个测试案例能够展示修复前后的行为差异。本文考虑了其中 354 个错误，因为其余 51 个错误没有被定位在任何方法中或是已经被弃用的错误。表 3 统计了每个项目的故障版本数、平均方法数和平均代码行数。

表 3 Defects4J 数据集

标识	项目名	故障数/个	方法数/个	代码行/k
Chart	JFreeChart	22	8 811	96
Closure	Closure-Compiler	151	5 328	90
Lang	Commons-Lang	57	4 690	59
Math	Commons-Math	98	11 504	170
Mockito	Mockito	26	1 427	10

3.2 评价指标

本文使用两种在软件故障定位领域上广泛使用的评价指标来评估本文的提出的方法，即 Top-K 和平均倒数排名（MRR，mean reciprocal rank）。

3.2.1 Top-K

Top-K 主要关注排序列表中的前 K 个元素是否包含了故障信息，包含了多少个故障，它用于评估故障元素的绝对排名。对于 K 值的选择，较小的 K 值更关注故障定位技术的准确性，而较大的 K 值则更加关注整体的性能。在软件故障定位领域，当一个程序存在故障，并通过故障定位技术获得了元素可疑值排序列表

时，为了减少程序员逐个检查的排序列表中的代码，因此通常会将 K 值设置的较小，以此来确保定位技术的准确性。在确定了 K 值后，Top-K 越高，则表明在前 K 个位置中包含了更多的故障元素，也就是说故障定位技术的性能越好。本文遵循大多数的研究[2,10,18]，将 K 值设置为 1，3 和 5。

3.2.2　*MRR*

倒数排名（Reciprocal Rank）是指将排名的倒数作为分数，例如故障元素的排名为 3，则它的排名分数为 1/3，而 *MRR* 表示每个故障元素在所有元素中排名的倒数之和的平均值，它是用于评估定位技术的整体性能。*MRR* 的取值范围为 0～1，*MRR* 的值越接近 1，表明排序效果越好，定位技术的模型性能越高。此外，*MRR* 对排序的稳定性较为敏感，如果故障元素的排名发生微小变化，可能会导致 *MRR* 值的显著变化。*MRR* 的计算方法如公式（15）所示，其中 N 表示每个项目故障版本的数量，$rank_i$ 表示第 i 个故障元素可疑值排名：

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{rank_i} \tag{15}$$

3.3　基线方法

为了验证本章方法在定位性能方面的有效性，选用 Two-RGCNFL[21]、SupConFL[19]、LLMAO[16]、Ochiai[5] 和 Dstar[4] 作为基线方法进行对比，以下是各个方法的概述。

1）Two-RGCNFL 是一种采用两阶段，基于 RGCN 的故障定位技术，阶段一通过构建方法调用依赖图来获取类内和类之间的依赖关系，从而实现方法级别的定位；阶段二通过提取方法内语句的动态与静态特征，构建 RankNet 模型得到语句可疑值并排序。本文方法即为针对阶段二进行的改进。

2）SupConFL 是一种基于神经网络的故障定位技术，将抽象语法树和语句序列相结合，并通过对比学习来提取错误代码的特征，采用基于注意力的长短期记忆递归神经网络来定位错误元素。

3）LLMAO 是一种基于语言模型的定位技术，通过对 left-to-right 语言模型进行预训练，使其包含了关于代码行可疑性的丰富知识，该方法能够在不执行任何测试用例的情况下对故障元素进行定位。

4）Dstar 是一种传统的 SBFL 技术，基于相似系数分析，可以自动识别可疑位置以进行故障定位，而不需要关于程序结构或语义的任何先验信息。

5）Ochiai 同样是一种传统的 SBFL 技术，相比 Dstar，它采用的是分子生物学领域中的 Ochiai 系数。Ochiai 在以往的研究工作中被广泛使用。

4　实验结果与分析

4.1　对比实验

为了探讨本章方法与基线方法在不同项目上的性能表现，给出了在 Top-K（K=1，3，5）和 *MRR* 两个评价指标上软件故障定位性能的对比实验，实验结果如表 4 所示。表中由于篇幅问题，将 Two-RGCNFL 缩写为 T-RGCNFL。

表 4　不同技术在每个项目中的实验结果

项目	技术	Top-1	Top-3	Top-5	*MRR*
Chart	Ochiai	4	7	10	0.273 2
	Dstar	3	7	11	0.258 6
	SupConFL	8	12	13	0.446 2
	LLMAO	6	10	12	0.365 1
	T-RGCNFL	9	13	17	0.515 1
	Ours	11	15	16	0.536 4
Closure	Ochiai	8	21	36	0.131 7
	Dstar	9	23	37	0.139 2
	SupConFL	43	57	67	0.342 9
	LLMAO	36	49	64	0.304 6
	T-RGCNFL	55	80	91	0.438 7
	Ours	60	85	94	0.462 9
Lang	Ochiai	7	14	18	0.209 7
	Dstar	8	12	19	0.212 5
	SupConFL	16	27	32	0.366 7
	LLMAO	12	23	29	0.308 2
	T-RGCNFL	20	32	36	0.454 6
	Ours	24	32	39	0.483 3
Math	Ochiai	13	25	27	0.207 5
	Dstar	12	23	27	0.193 0
	SupConFL	27	42	51	0.359 2
	LLMAO	24	39	48	0.334 1
	T-RGCNFL	35	52	60	0.442 7
	Ours	38	56	63	0.468 3
Mockito	Ochiai	3	8	10	0.219 6
	Dstar	3	9	11	0.221 3
	SupConFL	7	11	16	0.379 8
	LLMAO	5	9	14	0.297 5
	T-RGCNFL	8	12	17	0.412 4
	Ours	10	13	18	0.429 3

分析表 4 可以发现，传统的 SBFL 技术（Ochiai 和 Dstar）的性能是最差的，其中一个关键的原因在于它们只考虑测试用例对代码的覆盖信息，这非常依赖于测试用例的质量，并且对于同一基本块中的语句给出了相同的可疑值分数，这会导致排序时它们的先后顺序的不定性。这一点针对较为复杂的项目（如 Closure）尤为明显，相比表现较好的 Chart 项目在 MRR 上分别降低了 51.79％和 46.17％。其次，LLMAO 在所有项目中的性能都优于传统 SBFL 技术，例如，在 Math 项目中，

LLMAO 在 Top-K（K=1，3，5）的准确率上平均多定位了 11.5～21 个故障。其原因是它通过预训练得到的语言模型学习到的表示已经包含了大量关于语句可疑性的知识，这使得它能够在不运行任何测试用例的情况下来定位故障，也正是这一点导致它的性能不能达到最优。本文方法与 SupConFL 相比，在所有的项目中，Top-K（K=1，3，5）的准确率多定位了 2～28 个故障，在 *MRR* 上提高了 8.58%～37.47%。SupConFL 在基线方法中效果较好的原因是它能够有针对性地学习到故障元素更丰富的特征，然而该特征中只包含了代码的静态信息。最后，与 Two-RGCNFL 相比，本文方法仍然可以占据一定优势，在 Top-K（K=1，3，5）的准确率上共多定位了 11～16 个故障，在 *MRR* 上平均提高了 5.17%，说明采用基于注意力机制的特征融合方法能够提升模型的定位性能。

4.2 消融实验

为了验证考虑语句语义信息、频谱信息和突变信息对本章方法故障定位信息的影响，实验分别对不考虑语句语义信息（Text-）、不考虑语句频谱信息（Spectrum-）、不考虑语句突变信息（Mutant-）和完整的本文方法（Ours）进行性能对比，实验结果如图 4～7 所示。

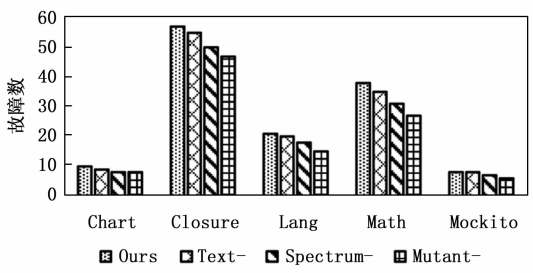

图 4 Top-1 上的消融实验结果

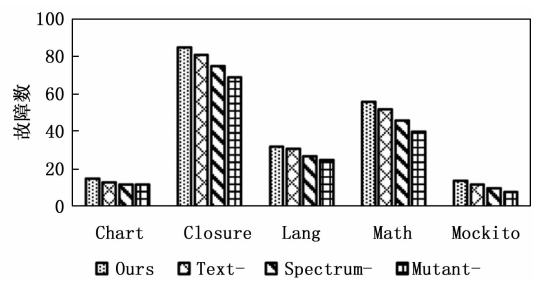

图 5 Top-3 上的消融实验结果

由图 4～7 可知，当同时考虑语句的语义信息、频谱信息和突变信息时，故障定位性能达到最佳，这表明这 3 种特征都为 RankNet 模型提供了有效信息。具体来说，与完整的本文方法相比，在不考虑语义信息的情况下，Top-K（K=1，3，5）的准确率上减少了 0～4 个故障数，*MRR* 的整体定位性能降低了 1.63%～8.43%；在不考虑频谱信息的情况下，Top-K（K=1，3，5）的准确率上减少了 3～10 个故障数，*MRR* 的整体定位性能降低了 9.9%～17.26%；在不考虑突变信息的情况下，Top-K（K=1，3，5）的准确率上减少了 2～18 个故障数，*MRR* 的整体定位性能降低了 18.37%～23.22%。上述表明在不考虑语义信息的情况下，模型在 4 个评估指标上的性能下降幅度最小，这一现象说明测试失败并不总是能够通过语义特征充分反映出来。相比之下，当不考虑频谱信息或突变信息时，故障定位性能出现显著下降，其中突变信息的缺失对模型性能影响尤为明显。这一发现进一步证实了动态分析技术在软件故障定位中的重要性，表明通过程序执行过程中的动态信息能够更有效地识别和定位软件缺陷。

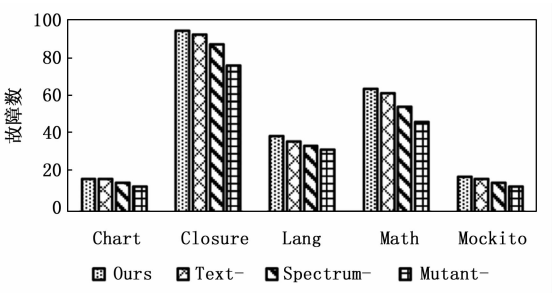

图 6 Top-5 上的消融实验结果

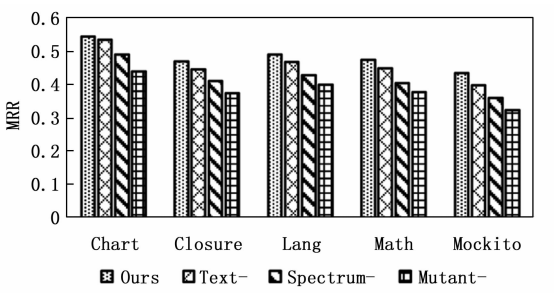

图 7 *MRR* 上的消融实验结果

5 结束语

本文针对软件故障定位工作展开研究，主要是对 Two-RGCNFL 的第二阶段进行改进，提出了一种基于特征融合的语句级别软件故障定位方法。该方法主张将动态特征和静态特征进行结合，主要提取了语句的 3 种特征信息。其中语义信息属于静态特征，它包含了当前语句及其执行上下文的信息，在扩展单条语句的语义的同时，还能捕捉故障传播上下文的信息。频谱信息和变异信息属于动态特征，频谱信息是从 6 种不同的传统 SBFL 技术中收集而来，变异信息则是从两种不同的 MBFL 技术中收集而来，它们都包含了语句与软件故障之间的相关程度。然后采用自注意力机制分别训练不同特征维度的权重，从而获取来自不同信息源中最有用故

障诊断信息，最终将它们链接在一起。采用 Defects4J 基准进行实验，Top-*K* 和 *MRR* 评价指标进行评估，通过实验验证了本文方法的有效性，并且证实了 3 种特征的必要性。

在未来的研究中，将挖掘更多类型的特征，例如代码的结构特征、历史故障信息或切片等，以丰富特征集的多样性。此外，本文方法主要采用 Gzoltar 工具来收集语句的覆盖信息，而 Gzoltar 是一个用于自动调试 Java 程序的库，因此本文方法会存在编程语言的限制。在未来，将选择多种具有代表性的编程语言，如 C++、Python 等。针对这些编程语言，研究与语言无关的覆盖信息采集框架，从而更全面地评估所提出方法的普遍性和适用性。

参考文献：

[1] WONG W, GAO R, LI Y, et al. A survey on software fault localization [J]. IEEE Transactions on Software Engineering, 2016, 42 (8): 707 - 740.

[2] LI X, LI W, ZHANG Y, et al. DeepFL: integrating multiple fault diagnosis dimensions for deep fault localization [C] // New York: Association for Computing Machinery, 2019: 169 - 180.

[3] WIDYASARI R, PRANA G, HARYONO S, et al. Real world projects, real faults: evaluating spectrum based fault localization techniques on Python projects [J]. Empirical Software Engineering, 2022, 27 (6): 147.

[4] WONG W, DEBROY V, GAO R, et al. The DStar method for effective software fault localization [J]. IEEE Transactions on Reliability, 2014, 63 (1): 290 - 308.

[5] ABREU R, ZOETEWEIJ P, GOLSTEIJN R, et al. A practical evaluation of spectrum-based fault localization [J]. Journal of Systems and Software, 2009, 82 (11): 1780 - 1792.

[6] NAISH L, LEE H, RAMAMOHANARAO K. A model for spectra-based software diagnosis [J]. ACM Transactions on Software Engineering and Methodology, 2011, 20 (3): 1 - 32.

[7] JONES J, HARROLD M, STASKO J. Visualization of test information to assist fault localization [C] //New York: Association for Computing Machinery, 2002: 467 - 477.

[8] CHEN M, KICIMAN E, FRATKIN E, et al. Pinpoint: problem determination in large, dynamic Internet services [C] //Piscataway: IEEE, 2002: 595 - 604.

[9] YAN Y, JIANG S, ZHANG Y, et al. A fault localization approach based on fault propagation context [J]. Information and Software Technology, 2023, 160: 107245.

[10] ZHANG M, LI Y, LI X, et al. An empirical study of boosting spectrum-based fault localization via pagerank [J]. IEEE Transactions on Software Engineering, 2021, 47 (6): 1089 - 1113.

[11] ZHAO G, HE H, HUANG Y. Fault centrality: boosting spectrum-based fault localization via local influence calculation [J]. Applied Intelligence, 2022, 52 (7): 7113 - 7135.

[12] MOON S, KIM Y, KIM M, et al. Ask the Mutants: mutating faulty programs for fault localization [C] // Piscataway: IEEE, 2014: 153 - 162.

[13] PAPADAKIS M, LETRAON Y. Metallaxis-FL: mutation-based fault localization [J]. Software Testing, Verification and Reliability, 2015, 25 (5/6/7): 605 - 628.

[14] KIM J, AN G, FELDT R, et al. Learning test-mutant relationship for accurate fault localization [J]. Information and Software Technology, 2023, 162: 107272.

[15] YAN Y, JIANG S, ZHANG Y, et al. An effective fault localization approach based on PageRank and mutation analysis [J]. Journal of Systems and Software, 2023, 204: 111799.

[16] YANG A, LEGOUES C, MARTINS R, et al. Large language models for test-free fault localization [C] //New York: Association for Computing Machinery, 2024: 1 - 12.

[17] MA Y, LI M. The flowing nature matters: feature learning from the control flow graph of source code for bug localization [J]. Machine Learning, 2022, 111 (3): 853 - 870.

[18] ZHANG Z, LEI Y, MAO X, et al. Context-aware neural fault localization [J]. IEEE Transactions on Software Engineering, 2023, 49 (7): 3939 - 3954.

[19] CHEN W, CHEN W, LIU J, et al. SupConFL: fault localization with supervised contrastive learning [C] // New York: Association for Computing Machinery, 2023: 44 - 54.

[20] GOU X, ZHANG A, WANG C, et al. Software fault localization based on network spectrum and graph neural network [J]. IEEE Transactions on Reliability, 2024, 73 (4): 1819 - 1833.

[21] FAN X, FU Z, SHU J, et al. Two-phase software fault localization based on relational graph convolutional neural networks [J]. Computers, Materials and Continua, 2025, 82 (2): 2583 - 2607.

[22] QUOC L, TOMAS M. Distributed representations of sentences and documents [C] // New York: Association for Computing Machinery 2014: 1188 - 1196.

[23] XUAN J, MONPERRUS M. Learning to combine multiple ranking metrics for fault localization [C] // Piscataway: IEEE, 2014: 191 - 200.