基于深度学习的软件自动测试方法研究

王晓宇[1] 孔 杰[1]
WANG Xiaoyu KONG Jie

摘 要 面对复杂软件系统时，传统规则引擎难以对软件需求文档中隐含的语义依赖关系以及接口协议中动态变化的参数约束条件进行精准建模与深度解析，造成缺陷发现率偏低、误检与漏检问题突出。为此，文章提出了一种基于深度学习的软件自动测试方法。先基于深度学习解析需求文档等多源信息，自动提取特征并生成候选测试用例，经优先级排序后输出高质量用例集；随后，构建用例依赖图谱，通过拓扑排序与动态负载均衡实现自动化调度与执行；最后，比对执行结果与预期差异，利用加权余弦相似度匹配缺陷知识库，完成智能识别、分级与报告生成。实验结果表明，该方法的缺陷发现率（92.3%）与缺陷类型识别准确率（91.7%）均显著高于对比方法，同时误检率（3.2%）与重复缺陷检出率（2.1%）得到有效控制，验证了其在复杂场景下兼具优异的测试全面性与缺陷判定精准性。

关键词 深度学习；软件自动测试；测试用例；执行调度；缺陷判定

doi：10.3969/j.issn.1672-9528.2026.04.006

0 引言

软件自动测试是提升测试效率的核心技术，然而当前自动测试面临用例生成与实际需求适配度不足、缺陷判定精准度低等问题，亟须通过智能化技术突破瓶颈，因此开展软件自动测试方法研究具有重要的工程应用价值[1]。针对上述需求，研究者们提出了多种软件测试方法。杨梅芳[2]提出的方法主要关注测试对象状态和属性，缺乏对需求文档中深层语义依赖的挖掘机制；杨洋等[3]以软件数据与用例距离为依据生成用例，忽视了需求中的语义信息，导致难以覆盖复杂执行路径；严文昊等[4]的方法未深入分析需求语义，仅关注表面功能，可能遗漏依赖关系引发的数据异常。传统测试方法如等价类划分和边界值分析在简单系统中有效，但在复杂系统中因缺乏语义分析与动态参数处理能力，同样面临隐性缺陷挖掘不足的挑战。针对这些不足，本文提出基于深度学习的软件自动测试方法。

1 基于深度学习的软件自动测试设计

1.1 基于深度学习的测试用例集生成

本研究引入深度学习技术，开展测试用例集生成设计，具体如下：

先明确深度学习模型所处理的输入信息维度。基于深度学习模型对这些输入数据展开深度语义解析，精准提取核心功能点、业务逻辑关联关系、约束条件以及潜在风险场景[5]。将提取的信息转化为机器可识别的结构化特征向量 $\boldsymbol{F}=[\boldsymbol{f}_1,\boldsymbol{f}_2,\cdots,\boldsymbol{f}_m]$，其中 $\boldsymbol{f}_i$ 代表单个特征项。通过特征映射机制与预存储的测试场景知识库进行匹配[6]。为精准筛选适配当前软件需求的场景模板，计算结构化特征向量与各场景模板特征向量的匹配度，用公式表示为：

$$S(\boldsymbol{F},\boldsymbol{T}_k)=\frac{\boldsymbol{F}\cdot\boldsymbol{T}_k}{\|\boldsymbol{F}\|\cdot\|\boldsymbol{T}_k\|}=\frac{\sum_{i=1}^{m}\boldsymbol{f}_i\cdot t_{ki}}{\sqrt{\sum_{i=1}^{m}\boldsymbol{f}_i^2}\cdot\sqrt{\sum_{i=1}^{m}t_{ki}^2}} \tag{1}$$

式中：$\boldsymbol{T}_k$ 代表第 k 个测试场景模板的特征向量；t_{ki} 代表模板 $\boldsymbol{T}_k$ 中第 i 个特征项的权重值。$S(\boldsymbol{F},\boldsymbol{T}_k)$ 的取值范围为 [0,1]，数值越接近 1，表明场景模板与当前需求的适配度越高。模型筛选出匹配度排名前 K 个场景模板，基于这些模板自动生成候选测试用例集。为提升测试执行效率与用例质量，模型对候选用例集进行两轮优化。一是通过计算候选用例间的特征相似度，剔除重复或高度相似的用例；二是基于用例对应的功能重要度、风险发生概率、执行复杂度等关键指标，通过加权求和计算优先级得分 P，用公式表示为：

$$P=q_1\times I_f+q_2\times R_p+q_3\times(1-C_e) \tag{2}$$

式中：I_f 为功能重要度系数（取值范围 [0,1]，越核心功能系数越接近 1）；R_p 为风险发生概率系数（取值范围 [0,1]，风险越高系数越接近 1）；C_e 为执行复杂度系数（取值范围 [0,1]，

1. 郑州西亚斯学院计算机与软件工程学院 河南新郑 451150
[基金项目] 河南省 2021 年民办普通高等学校学科专业建设资助项目（软件工程专业）（豫财教〔2021〕16 号）；郑州西亚斯学院 2025 年度教育与教学改革研究项目（一般项目）“基于生成式 AI 的编程类课程个性化教学改革与实践”（2025JGYB42）

复杂度越低系数越接近 1）；q_1、q_2、q_3 为归一化权重系数，用于平衡各指标对优先级的影响。优先级得分 P 越高的用例，在后续执行调度中越优先被触发。最终，以标准化格式输出经过优化筛选后的测试用例集。

为验证深度学习驱动的测试用例生成方法的有效性，设计对比测试，从路径覆盖度、语义匹配度和冗余率三个维度评估用例质量。测试以基于规则模板（方法 A）、遗传算法（方法 B）和随机搜索（方法 C）的方法作为基线，针对包含 5 个业务模块的 Web 系统进行测试，每个模块预设 10 条核心路径与 3 类边界条件。各方法生成 200 条测试用例，路径覆盖度反映用例覆盖预设路径的比例，语义匹配度通过 BERT 计算用例与需求文档的一致性，冗余率衡量重复或高度相似用例的占比。测试结果如表 1 所示。

表 1 测试用例生成质量对比结果

单位：%

方法	路径覆盖度	语义匹配度	冗余率
本文方法	94.2	91.5	3.8
方法 A	78.6	72.3	12.4
方法 B	82.1	79.8	9.7
方法 C	65.3	61.2	18.9

测试结果表明，本文方法在路径覆盖度、语义匹配度与冗余控制方面均显著优于传统方法，验证了深度学习在理解复杂语义及生成高质量用例方面的优势。

1.2 测试用例的自动化执行调度策略

在完成测试用例智能生成后，获得的标准化用例集仍处于待执行状态。若缺乏合理的自动化调度策略，大规模测试易出现资源竞争、依赖关系混乱等问题，影响测试准确性。为此，需建立自动化执行调度机制，通过解析用例集、构建依赖图谱并运用拓扑排序确定无冲突执行序列，同时搭建包含资源调度、执行引擎与过程监控等模块的分布式自动化执行框架，为测试工作提供高效可靠的体系化支撑。

对标准化测试用例集进行结构化解析，精准提取关键执行属性。基于对这些依赖关系的解析结果，构建用例依赖图谱，并运用拓扑排序明确无依赖冲突的执行序列基础，为后续的调度工作提供有序的执行框架。为了实现执行资源的最优分配与高效调度，在用例优先级 P 的基础上融合资源负载状态与依赖完成度，构建调度优先级得分 $S=\alpha P+\beta(1-L)+\gamma Q$，其中 α、β、γ 分别表征用例固有优先级、资源负载影响及依赖关系权重；L 为目标执行节点的实时负载系数，该系数越接近 0，表示其负载越低，越适合执行新用例；Q 为用例依赖完成系数，若用例无前置依赖则 Q=1，存在依赖时则 Q 等于所有前置用例的执行完成率。调度引擎基于该公式计算所有待执行用例的调度优先级 S，采用动态负载均衡算法将高 S 值用例分配至低负载节点，实现资源利用率与执行效率的最大化 [7]。在测试用例执行过程中实时捕获执行节点的系统指标（CPU、内存、网络）与用例执行状态（初始化、执行中、成功、失败、超时），为了避免无效执行消耗，设定动态超时阈值，计算公式为：

$$T=T_0\times K^{''} \tag{3}$$

式中：T_0 为用例预设基础执行时长；$K^{''}$ 为环境复杂度调节系数，环境越复杂则 $K^{''}$ 取值越大。针对执行过程中出现的网络波动、连接中断、节点故障等异常情况，采用分级处理策略。所有测试用例按照调度序列执行完成后，结果采集模块按照标准化格式提取用例执行状态、实际响应时间、返回状态码、输出数据特征等核心数据，并将这些数据整合为结构化执行结果向量 $\boldsymbol{R}=[r_1, r_2,\cdots, r_n]$（其中 r_i 对应单项采集指标），为后续的测试结果分析和处理提供数据支持。

1.3 基于测试结果的缺陷智能判定与报告生成

在完成测试用例自动化执行调度后，开展基于测试结果的缺陷智能判定与报告生成，以解决传统方法在缺陷识别与分析环节的不足，达成从测试执行到质量评估的完整闭环。此环节着重解决三个核心问题：传统方法依赖人工经验解析测试结果，难以从海量数据中精准识别潜在缺陷；对已识别缺陷缺乏系统分类与定级机制，影响修复优先级判断；测试报告生成效率低、标准化不足，难以及时有效支持决策。

为实现目标，采用多步骤处理流程：先进行测试结果预处理与特征对齐，提取每个测试用例对应的预期结果向量 $\boldsymbol{E}=[e_1, e_2,\cdots, e_n]$（$e_i$ 与执行结果向量 $\boldsymbol{R}$ 的 r_i 一一对应），逐维度比对生成差异特征向量 $\boldsymbol{D}=[d_1, d_2,\cdots, d_n]$。差异特征计算公式为：

$$d_i=\begin{cases}0 & ,|r_i-e_i|\le\theta_i\\ \dfrac{|r_i-e_i|}{\max(e_i,\eta_i)} & ,|r_i-e_i|>\theta_i\end{cases} \tag{4}$$

式中：θ_i 为第 i 个指标的允许偏差阈值；η_i 为第 i 个指标的最小有效阈值。d_i 的取值范围为 $[0,+\infty)$，数值越大表明该维度结果与预期差异越显著，当 d_i=0 时表示该维度无差异。通过该计算，将原始执行结果转化为聚焦“差异”的高维特征向量 [8]。接下来使用深度学习模型调用预存储的缺陷特征知识库，该知识库包含功能缺陷、性能缺陷、接口兼容缺陷、数据一致性缺陷等多类型缺陷的标准特征向量。采用加权余弦相似度算法计算差异特征向量 $\boldsymbol{D}$ 与各标准缺陷特征向量 $\boldsymbol{B}_j$ 的匹配度 M，用公式表示为：

$$M(\boldsymbol{D},\boldsymbol{B}_j)=\frac{\sum_{i=1}^{n}\omega_i\cdot d_i\cdot b_{ji}}{\sqrt{\sum_{i=1}^{n}\omega_i\cdot d_i^2}\cdot\sqrt{\sum_{i=1}^{n}\omega_i\cdot b_{ji}^2}} \tag{5}$$

式中：ω_i 为第 i 个差异指标的权重系数，其取值与功能重要性正相关，核心功能对应指标赋予更高权重，次要功能指标则赋予较低权重；b_{ji} 为第 j 类缺陷标准特征向量的第 i 个维

度值。$M(\boldsymbol{D}, \boldsymbol{B}_j)\in[0,1]$，数值越接近 1，表明匹配度越高。设定匹配度阈值 τ，当某类缺陷的匹配度 $M \geq \tau$ 时，判定该用例触发对应类型缺陷；若所有缺陷类型的匹配度均小于 τ，则结合模型的异常模式识别能力判定为“未知类型缺陷”[9]。为提升报告决策价值，融合缺陷影响范围、发生概率及修复成本三大核心指标计算缺陷严重程度得分 S_{sev}，依据 S_{sev} 得分将缺陷划分为致命、严重、一般、轻微四级。最后，整合全流程核心数据生成结构化测试报告，以标准化格式输出，完成整个软件自动测试流程。

2 实验

2.1 实验环境与测试对象

为了验证本文提出的基于深度学习的软件自动测试方法（简称“本文方法”）的有效性，搭建统一实验环境：硬件层面采用 Intel Xeon Gold 6330 处理器（24 核 48 线程）、128 GB DDR4 3 200 MHz 内存、2 TB NVMe 固态硬盘及千兆以太网网卡；软件层面部署 Ubuntu 22.04 LTS 操作系统，深度学习模型基于 PyTorch 1.18 框架部署，自动化执行依赖 Selenium 4.10、Appium 2.0 及 JMeter 5.6 等工具。测试对象选取 3 类典型软件系统（Web 业务系统、移动端应用、分布式接口服务），软件代码规模为 5 万 ~20 万行，包含预设的功能缺陷、性能缺陷、接口兼容缺陷等多类型已知缺陷，同时保留隐性缺陷以验证方法的全面性。

2.2 实验参数设置

实验参数设置以方法设计中的核心公式与关键机制为依据，通过预实验验证确定最优取值，具体参数设置如表 2 所示。

表 2 核心实验参数设置表

序号	参数名称	取值
1	场景模板匹配阈值	0.75
2	功能重要度权重	0.4
3	风险发生概率权重	0.35
4	执行复杂度权重	0.25
5	动态超时调节系数	1.5
6	差异度阈值	0.15
7	缺陷匹配度阈值	0.7
8	核心差异指标权重系数	0.6

基于上述参数设置开展对比实验，将本文方法与三种对照方法（方法 A：基于云计算的分布式软件测试方法、方法 B：基于 RRT 技术的应用软件集成测试方法、方法 C：基于 SDL 技术的软件自动化测试方法）部署于同一实验环境，针对相同测试对象，在相同测试时间与资源限制条件下分别执行自动化测试，通过多维度对比验证本文方法的优越性。

2.3 软件测试结果对比分析

实验按“统一测试范围 → 同步启动测试 → 实时数据采集 → 结果汇总统计”的流程执行，针对 3 类测试对象的核心功能模块与非核心功能模块，累计执行测试用例 1 200 组，覆盖预设缺陷与隐性缺陷。四种方法的缺陷发现率对比如图 1 所示。

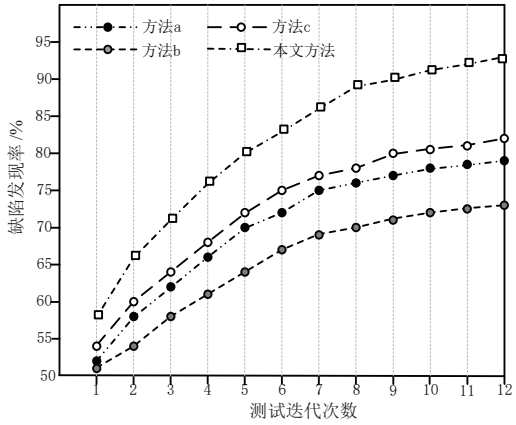

图 1 四种测试方法缺陷发现率对比折线图

缺陷发现率反映测试方法对缺陷的覆盖能力与识别灵敏度，其值越高代表测试全面性越强。结果显示，随着测试迭代次数增加，各方法的缺陷发现率均逐步上升并趋于稳定，但本文方法在所有迭代中均保持最高水平，最终达到 92.3%，显著优于方法 A（78.5%）、方法 B（72.8%）和方法 C（81.2%）。本文方法在测试初期即表现出更强的缺陷识别能力，且优势随迭代持续扩大，说明基于深度学习的语义解析能有效生成覆盖复杂语义路径和边界条件的测试用例，尤其擅长捕捉传统方法难以建模的隐性缺陷。实验结果表明，本文方法在测试全面性方面具备明显优势，尤其适用于高复杂度场景下的缺陷挖掘。

为进一步验证本文方法的测试精准度，实验针对检出的缺陷，统计四类方法的缺陷类型识别准确率、缺陷定位精度及误检率，结果如表 3 所示。

表 3 四种测试方法精准性指标对比表

单位：%

方法	缺陷类型识别准确率	缺陷误检率	重复缺陷检出率	隐性缺陷检出占比	严重缺陷漏检率
本文方法	91.7	3.2	2.1	42.6	1.8
方法 A	76.3	8.7	7.5	28.3	6.2
方法 B	70.5	11.4	9.8	23.7	8.5
方法 C	80.2	6.9	5.3	31.5	4.7

由表 3 可以看出，本文方法的缺陷类型识别准确率达到 91.7%，显著高于方法 A（76.3%）、方法 B（70.5%）和方法 C（80.2%）；同时，缺陷误检率与重复缺陷检出率分别仅为 3.2% 和 2.1%，在所有方法中均为最低。这一结果表明，本文方法在缺陷判定环节具有更高的可靠性与一致性。其优势主要源于基于深度学习模型对多源信息进行语义解析与特征提取所生

成的高质量结构化特征，这些特征能够更准确地表征软件行为，从而在差异特征比对与缺陷匹配过程中有效降低误判与重复。实验证明，本文方法通过深度学习驱动的语义解析确保了测试用例的高质量与高覆盖度，为后续缺陷判定奠定了坚实基础；而基于深层特征表达的智能判定则在此基础上实现了高精准识别，共同保障了最终测试效果的全面提升。

3 结语

本文针对复杂软件系统中语义理解不足与隐性缺陷难以挖掘的挑战，构建了一种基于深度学习的全流程自动化测试框架。核心创新在于三个关键步骤：首先，设计了基于深度学习的语义解析与特征提取机制，能够从多源非结构化信息中精准捕捉语义依赖与动态参数约束，为生成高覆盖度测试用例奠定基础；其次，提出了融合动态负载与依赖关系的用例调度策略，通过量化优先级与实时资源感知，保障测试执行的高效稳定；最后，建立了基于加权余弦相似度的缺陷智能判定模型，利用深层特征实现缺陷的精准识别与分级。实验结果表明，本方法在缺陷发现率、类型识别准确率等关键指标上显著优于传统方法，尤其在挖掘隐性缺陷与控制误检漏检方面优势明显，验证了深度语义理解驱动测试全流程的有效性。

参考文献：

[1] 肖英剑，陈婷．基于深度学习的人工智能软件测试优化分析 [J]. 集成电路应用 ,2025,42(6):146-147.

[2] 杨梅芳．基于云计算的分布式软件测试方法 [J]. 信息与电脑 (理论版),2023,35(18):4-6.

[3] 杨洋，杨庆婷，彭培．基于 RRT 技术的应用软件集成测试方法 [J]. 长江信息通信 ,2023,36(6):63-65.

[4] 严文昊，缪慧宇，龚健，等．基于 SDL 技术的软件自动化测试方法研究 [J]. 无线互联科技 ,2024,21(21):100-102.

[5] 刘阳，寇力，杨基宏，等．基于语义匹配模型的软件测试用例共享和复用方法分析 [J]. 电子技术 ,2025,54(2):83-85.

[6] 吕虹，谢琳．基于深度学习的软件错误定位与修复方法研究 [J]. 信息记录材料 ,2024,25(9):129-131.

[7] 胡明，蔡传军，童绪军．基于深度神经网络的信息管理软件自动化测试方法研究 [J]. 佳木斯大学学报 (自然科学版),2024,42(5):30-33.

[8] 马玲玲．基于强化学习的自动化软件测试用例生成方法探索 [J]. 电脑编程技巧与维护 ,2025(9):56-58.

[9] 崔靖．基于深度学习的计算机应用软件自动化测试方法 [J]. 中阿科技论坛 (中英文),2025(6):97-100.

【作者简介】

王晓宇（1986—），女，河南淮阳人，硕士，助教，研究方向：计算机应用技术。

孔杰（1989—），男，河南焦作人，硕士，助教，研究方向：计算机视觉。

（收稿日期：2025-12-16 修回日期：2026-04-13）

（上接第 27 页）

[15]ARAR Ö F, AYAN K. Software defect prediction using cost-sensitive neural network[J]. Applied soft computing, 2015, 33: 263-277.

[16]AGRAWAL A, MENZIES T. Is "better data" better than "better data miners" ? on the benefits of tuning SMOTE for defect prediction[C]//ICSE'18: Proceedings of the 40th International Conference on Software Engineering. NewYork: ACM, 2018: 1050-1061.

[17]YANG X L, LO D, XIA X, et al. Deep learning for just-in-time defect prediction[C]//2015 IEEE International conference on software quality, reliability and security. Piscataway: IEEE, 2015: 17-26.

[18]HUANG Q, XIA X, LO D. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction[J]. Empirical software engineering, 2019, 24: 2823-2862.

[19] 陈翔，王莉萍，顾庆，等．跨项目软件缺陷预测方法研究综述 [J]. 计算机学报 , 2018, 41(1): 254-274.

[20] 任泽裕，王振超，柯尊旺，等．多模态数据融合综述 [J]. 计算机工程与应用 , 2021,57(18):49-64.

[21] 化盈盈，张岱墀，葛仕明．深度学习模型可解释性的研究进展 [J]. 信息安全学报 , 2020,5(3):1-12.

[22] 纪晨辉，李英梅．一种邻域合成的软件缺陷预测过采样方法 [J]. 软件工程与应用 , 2023, 12: 930-939.

[23] 纪晨辉．软件缺陷预测中数据高维性和不平衡性问题处理研究 [D]. 哈尔滨 : 哈尔滨师范大学 ,2024.

【作者简介】

姚永芳（1975—），女，湖南常德人，硕士，助理研究员，研究方向：模式识别、人工智能、故障诊断、软件工程。

丁永昌（1997—），男，浙江金华人，博士研究生，研究方向：软件缺陷检测、人工智能。

陈林君（1996—），男，江西九江人，博士研究生，研究方向：类不平衡、模式识别。

荆晓远（1971—），通信作者（email:jingxy_2000@126.com），男，江苏南京人，博士，教授，研究方向：模式识别、人工智能、故障诊断、软件工程。

（收稿日期：2025-08-28 修回日期：2026-04-02）