王婷, 王恰时, 付功云, 等. 基于混合架构的开发测试方法[J]. 智能计算机与应用,2025,15(12):179-185. DOI:10.20169/j.issn.2095-2163.25050602

基于混合架构的开发测试方法

王 婷[1,2], 王恰时[1,2], 付功云[1,2], 杨 喆[1,2], 王震宇[2], 陈 翔[2], 张 恒[2]
(1 中国中铁智慧城市研发中心, 天津 300308; 2 中铁第六勘察设计院集团有限公司, 天津 300308)

摘 要: 基于人工智能在开发测试中的应用,本文提出基于循环神经网络(Recurrent Neural Network, RNN)、大语言模型(Large Language Model, LLM)、遗传算法(Genetic Algorithm, GA)混合架构的开发测试方法。RNN 通过改进 Transformer-XL 处理序列、模拟用户行为;LLM 利用指令微调生成测试场景、把控逻辑;GA 以自适应变异率等机制优化测试策略与资源分配。实验显示,该方法优势显著,测试覆盖率达 88.0%,缺陷检测能力提升 27.3%,人工参与减少 22.4%。

关键词: 开发测试;人工智能;循环神经网络(RNN);大语言模型(LLM);遗传算法(GA);混合架构
中图分类号: P315.69 **文献标志码**: A **文章编号**: 2095-2163(2025)12-0179-07

Research on hybrid architecture-based development and testing methods

WANG Ting[1,2], WANG Qiashi[1,2], FU Gongyun[1,2], YANG Zhe[1,2], WANG Zhenyu[2], CHEN Xiang[2], ZHANG Heng[2]
(1 China Railway Smart City R&D Center, Tianjin 300308, China;
2 China Railway Sixth Survey and Design Institute Group Co., Ltd., Tianjin 300308, China)

Abstract: Based on the application of artificial intelligence in development and testing, proposing a hybrid architecture combining RNN, LLM, and GA. RNN improves Transformer-XL for sequence processing and user behavior simulation; LLM utilizes instruction fine-tuning to generate test scenarios and control logic of test scenarious; GA optimizes testing strategies and resource allocation through mechanisms like adaptive mutation rates. Experiments demonstrate significant advantages, achieving 88.0% test coverage, a 27.3% improvement in defect detection capability, and a 22.4% reduction in manual involvement.

Key words: development testing; artificial intelligence; Recurrent Neural Network (RNN); Large Language Model (LLM); Genetic Algorithm (GA); hybrid architecture

0 引 言

随着软件系统复杂度指数级增长,传统测试方法覆盖率不足、维护成本高,同时面临大量的数据处理与分析工作[1-2]。国内学者在智能测试领域研究表明,Python 语言凭借跨领域的兼容性和丰富的第三方库及开源社区的支持,结合人工智能技术,实现数据驱动与知识驱动深度融合,为测试自动化提供了新的范式[3-5]。人工智能技术通过数据驱动与知识融合,能够自动学习测试用例特征和模式,生成更全面、更有效的测试用例。李明等[6]基于深度学习的测试用例自动生成方法,通过深度学习模型学习历史测试用例,依据新需求文档生成测试用例,不仅提升了测试用例生成效率,还显著提高了测试覆盖率,可覆盖传统方法难以涉及复杂场景;王芳等[7]通过优化 Web 应用测试流程,利用 AI 技术分析用户行为数据,预测用户操作路径,从而生成更贴近真实用户行为测试用例,不仅提高了测试效率,还增强了测试用例多样性,能够更好地模拟真实用户操作场景,发现潜在问题,增强决策支持,助力复杂场景

基金项目: 中国国家铁路集团有限公司科技研究开发计划(K2023S008-C(JB));中国中铁股份有限公司科技开发计划(2023-重大-21);中铁第六勘察设计院集团有限公司科研课题(KY-2023-03,KY-2023-05)。

作者简介: 王 婷(1988—),女,硕士,高级工程师,主要研究方向: 人工智能,信息化; 付功云(1981—),男,学士,高级工程师,主要研究方向: 人工智能,信息化; 杨 喆(1982—),男,学士,工程师,主要研究方向: 人工智能; 王震宇(1984—),男,学士,工程师,主要研究方向: 人工智能; 陈 翔(1993—),男,学士,工程师,主要研究方向: 人工智能; 张 恒(2005—),男,本科生,主要研究方向:人工智能。
通信作者: 王恰时(1981—),男,学士,高级工程师,主要研究方向: 人工智能,信息化。Email: wangqiashi@163.com。

收稿日期: 2025-05-06

的应用。本文提出一种基于混合架构的开发测试方法,通过循环神经网络改进 Transformer-XL 的序列处理能力,借助大语言模型的指令微调,生成测试场景,并融合遗传算法优化搜索效率;基于本文的方法,完成测试用例生成、错误预测与策略优化三大核心任务。同时,本文通过多场景实验对该方法的有效性进行实证检验,不仅明确了其突出优势、内在局限与潜在发展方向,更为后续相关实践应用提供重要支撑。

1 核心算法和模型

1.1 循环神经网络(RNN)序列生成

神经网络是一种模仿生物神经网络结构和功能的计算模型,由大量神经元相互连接而成,可以通过学习和训练来实现各种复杂任务,而循环神经网络(RNN)是测试用例生成重要工具,擅长处理序列数据,通过时序依赖性建模捕捉测试用例逻辑关联,以隐藏状态传递历史信息、融入上下文生成序列[8-9]。输入序列 $X=\{x_1,x_2,\cdots,x_T\}$,隐藏状态序列 $H=\{h_0,h_1,\cdots,h_T\}$,其中,h_0 为初始隐藏状态(通常设为零向量),第 t 步的隐藏状态 h_t 如下式:

$$h_t=\sigma(\boldsymbol{W}_{xh}x_t+\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}+\boldsymbol{b}_h) \tag{1}$$

其中,$\boldsymbol{W}_{xh}$ 为输入到隐藏层权重矩阵;$\boldsymbol{W}_{hh}$ 为隐藏层到隐藏层循环权重矩阵;$\boldsymbol{b}_h$ 为隐藏层偏置向量;$\sigma(\cdot)$ 为激活函数。

在应用测试中,用户交互、系统时间和数据流可视为序列数据。传统循环神经网络(RNN)模型在依赖关系处理上有局限,长短期记忆网络(Long Short-Term Memory, LSTM)和门控循环单元(Gated Recurrent Unit, GRU)等变体可以有效解决该问题。近年 Transformer-XL 通过引入相对位置编码和段级循环机制,显著提高了模型处理长序列的能力,使得测试用例生成能够考虑更长历史上下文,从而生成更加真实和有效测试序列[10]。Transformer 模型核心公式:

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{2}$$

其中,$\boldsymbol{Q}$(Query)、$\boldsymbol{K}$(Key)、$\boldsymbol{V}$(Value)分别是通过线性变换从输入序列中得到的查询、键和值矩阵,d_k 是键向量的维度,Softmax 函数用于计算注意力权重,确保权重和为 1。

$\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_k}}$ 计算查询和键之间的相似度,并通过 $\sqrt{d_k}$ 进行缩放,以防止梯度消失或爆炸。多头自注意力机制(Multi-Head Self-Attention)是自注意力机制扩展,通过并行计算多个注意力头来捕捉不同语义信息:

$$\text{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\text{Concat}(\text{head}_1,\text{head}_2,\cdots,\text{head}_h)\boldsymbol{W}^O \tag{3}$$

其中,每个注意力头 $\text{head}_i=\text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q,\boldsymbol{K}\boldsymbol{W}_i^K,\boldsymbol{V}\boldsymbol{W}_i^V)$,$\boldsymbol{W}_i^Q,\boldsymbol{W}_i^K,\boldsymbol{W}_i^V$ 是每个注意力头学习参数矩阵,$\boldsymbol{W}^O$ 是用于将多个注意力头输出拼接后进行线性变换参数矩阵。

本文中采用改进的 Transformer-XL,通过优化其上下文信息留存机制与序列生成效率,增强了其在测试用例生成中的应用效果,为后续测试执行提供了更具真实性与覆盖性的输入素材。

上下文感知注意力机制被引入,能让模型依据应用程序状态和行为特性,对注意力分布进行动态优化。在注意力机制中引入了一个上下文向量 $\boldsymbol{C}$,该向量通过额外线性变换从输入序列中提取出来,并与查询、键和值矩阵一起参与注意力计算:

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V},\boldsymbol{C})=\text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}+\boldsymbol{Q}\boldsymbol{C}^{\mathrm{T}}}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{4}$$

设计多模态输入整合机制,支持模型同时解析代码、API 文档、用户交互日志等多形式数据,以增强输入性能并生成更全面的测试用例。将不同模态数据分别通过不同编码器进行编码,通过一个融合层将这些编码结果进行整合:

$$\text{FusedInput}=\text{Concat}(\text{CodeEmbedding},\text{APIDocEmbedding},\text{UserLogEmbedding}) \tag{5}$$

设计层次化序列生成方式以提升测试用例多样性,先生成抽象测试策略框架,再逐步细化为具体操作流程,最终形成可执行测试代码。使用一个分层的生成模型,每一层负责生成不同粒度的测试用例:

$$\text{TestSequence}=\text{HierarchicalGenerator}(\text{AbstractStrategy},\text{DetailedSteps},\text{ExecutableCode}) \tag{6}$$

通过以上改进措施使得 RNN 能够生成更加多样化、覆盖面更广测试用例,可精准抓取测试用例中逻辑脉络与语义内涵,进而生成条理清晰、逻辑连贯的测试用例序列。

1.2 生成式大语言模型

Kimi、DeepSeek 等生成式大语言模型(LLM)凭借自然语言理解与生成能力,为应用测试带来变革。模型预训练和微调。可理解复杂应用逻辑与业务规则,生成贴合实际场景测试用例。基于 LLM 测试生

成方法能够显著减少人工编写测试用例时间，同时提高测试质量和覆盖率[11]。LLM 在处理复杂业务逻辑与边缘情况时，其能力显著超越了传统方法，这种优势本质上源于“以自然语言为核心的柔性推理框架”对“传统固定规则引擎”的范式替代。本文构建了如下功能体系：

在测试需求理解方面，借助基于 Transformer 的编码器 - 解码器模型，赋予模型从需求文档等文本中萃取测试需求的能力，自动甄别核心要点与边界条件，将自然语言需求转化为结构化测试需求：

$$\text{TestRequirements} = \text{EncoderDecoderModel}(\text{NaturalLanguageRequirements}) \quad (7)$$

代码感知测试生成，对应用代码的透彻分析，系统能够产出契合特定代码执行路径功能测试用例，保障测试精准性与实效性。该模型能够根据代码逻辑生成相应测试用例：

$$\text{TestCases} = \text{CodeAwareGenerator}(\text{ApplicationCode}) \quad (8)$$

自然语言到测试代码转换采用高效转换架构，可快速将自然语言描述测试场景转化为可执行测试代码，显著提升测试开发效率。使用序列到序列的转换模型，将自然语言描述测试场景转换为具体的测试代码：

$$\text{ExecutableTestCode} = \text{Seq2SeqConverter}(\text{NaturalLanguageTestScenario}) \quad (9)$$

结合指令微调技术提升模型对测试指令及上下文的理解能力。LLM 缺乏测试场景专业性，通过输入测试指令、用例设计规则、缺陷报告等指令 - 响应数据微调，引导其学习测试领域知识。模型能解析关键信息、精准理解目标，生成更有效测试用例，提升测试效率与质量，保障软件可靠性。

1.3　遗传算法优化

1）遗传算法（GA）是从生物进化原理中获得优化的算法，在测试用例生成与优化领域彰显优势。遗传算法通过对自然选择、交叉以及变异等生物进化过程的模拟，得以在复杂空间内探寻近似最优解。Li 等[12]提出基于种群训练（Population Based Training，PBT）算法，将 GA 与强化学习相结合，进一步提高优化效率。通过灵活调整参数设置和优化模型架构，PBT 算法能够以更高效率确定最优测试策略。本文针对传统 GA 进行了多方面的改进：

针对自适应变异率，本文提出了一种可自我调节变异机制，依据种群多样性程度以及搜索进程，对变异率进行动态调控，从而防止算法过早陷入局部最优解。

2）针对多目标优化问题，常规的遗传算法（GA）通常聚焦于单一目标优化，然而在实际的测试场景中，需兼顾测试覆盖率、执行时长以及资源占用等多重目标。为此，本文引入基于 Pareto 前沿的多目标优化策略，有效实现多个测试指标的同步优化。使用多目标优化框架，能够同时优化多个目标函数：

$$\text{OptimalSolutionSet} = \text{ParetoFrontier}(\text{Coverage}, \text{ExecutionTime}, \text{ResourceUsage}) \quad (10)$$

3）针对知识引导交叉操作，本文采用知识驱动的交叉策略，通过领域专业知识与过往测试成果引导交叉环节，进而生成更具针对性的测试用例。利用知识引导交叉操作函数 $\text{Crossover}(P_1, P_2, K)$，根据知识库 K 对父代个体 P_1 和 P_2 进行交叉操作：

$$\text{Child} = \text{Crossover}(P_1, P_2, K) \quad (11)$$

针对持续集成和敏捷开发的快节奏环境，本文提出增量式进化方案，依托已有测试用例集，快速构建适配新功能及代码变更的测试用例，使算法能够根据新功能和代码变更动态调整测试用例集，公式如下：

$$\text{UpdatedTestCases} = \text{IncrementalEvolution}(\text{ExistingTestCases}, \text{NewFeatures}, \text{CodeChanges}) \quad (12)$$

根据以上改进，还引入 Zhao[13] 提出的协同进化框架，通过多个种群协同优化，进一步提高了搜索效率和结果质量。

2　系统架构设计

基于混合架构开发测试方法用于应用测试。架构涵盖数据收集预处理、RNN 序列生成、LLM 理解生成、GA 优化模块，以及集成协同机制，系统性解决测试用例生成问题，该系统整体架构如图 1 所示。

数据收集与预处理模块采集多源数据并预处理，通过静态分析获取代码结构、汇总历史测试数据、记录用户交互日志、提取需求文档中的功能规则，经格式统一和噪声清理保障数据质量。

RNN 序列生成模块基于优化的 Transformer-XL 架构，利用历史交互数据模拟用户行为序列，构造含正常与边界输入的测试用例，通过分层编解码处理多元数据，生成适配应用特性的测试序列集以辅助测试策略规划。

LLM 理解与生成模块基于大语言模型，解析测试需求与应用功能提取核心测试点，构建场景并转化为测试代码和文档；采用分阶段微调策略即通用

数据集预训练、领域数据优化、小样本学习等,适配项目需求。

GA 优化模块采用改进遗传算法,优化 RNN 和 LLM 生成的测试用例以提升全面性与有效性,通过进化机制适配应用更新,在资源受限下优化测试顺序与资源分配,运用多种遗传操作和筛选策略搜索优质测试组合。

集成与协同机制作为核心,涵盖级联生成,包括 RNN、LLM、GA 依次构建初始序列、测试场景、优化用例,反馈增强即依测试结果迭代优化、知识共享即积累复用模型经验、自适应调节即动态调整模块权重,实现智能测试全流程闭环。

部署与接口设计支持多元部署:本地模式适配 CI/CD 流水线,云服务实现远程分布式测试,混合部署兼顾效率与安全;接口提供命令行自动化、Web 可视化操作、API 集成及 IDE 插件,覆盖开发测试全流程。

图 1 系统架构图

Fig. 1 System architecture diagram

3 实验与分析

3.1 实验设置

为了验证本文方法性能与有效性,设计工业级实验测试场景,选取 3 类典型测试对象,根据不同技术栈、架构模式和业务复杂度,检验架构适应性和有效性。

Web 应用:基于央企一级精准帮扶平台,包含注册、权限、登记和申请、付款反馈等功能;

APP 应用:基于央企一级劳模申报系统,包含申报通知、申报名额、信息填报等功能;

API 服务:基于中铁六院慧测评系统数据 API,提供查询、打分、投票、生成等功能。

3.2 评估指标

实验采用多维度评估指标,即测试覆盖率、缺陷检测能力、测试效率、测试质量,将混合架构的开发测试方法与传统手动测试方法、单一模型方法、主流自动化测试方法及 AI 测试方法进行对比实验。

3.3 实验环境

实验保障可重复性,硬件配置为 8 核 CPU、64 GB 内存、两块 NVIDIA A100 GPU,软件采用 Centos8、Python 3.9、PyTorch 2.0 及 Transformers 4.28。数据集涵盖精准帮扶平台 860 条用例、劳模申报系统 363 条用例、慧测评系统 157 条用例。数据源包含的测试用例中,功能测试用例占 55%,异常测试用例占 35%,安全测试用例占 10%。经文本相似度去重、添加属性标签预处理后,按 7∶2∶1 划分为训练集 966 条、验证集 276 条、测试集 138 条,示例数据见表 1。

表 1　示例数据

Table 1　Sample data

用例 ID	模块	类型	前置条件	操作步骤	预期结果
FT001	登录	功能	账号、密码	输入账号、密码	登录到首页
FT002	任务管理	异常	设置任务时间	任务截止时间早于当前	提示时间无效

3.4　结果分析

3.4.1　测试覆盖率比较

不同方法在各类应用上的测试覆盖率的比较见表 2，基于 RNN+LLM+GA 混合架构的开发测试方法在各类型应用测试中表现最优，平均测试覆盖率达 88.3%，超出次优对比方法 6.4%，尤其在复杂 Web 应用测试中优势更显著。

3.4.2　缺陷检测能力比较

基于 RNN+LLM+GA 混合架构的开发测试方法共检出 135 个软件缺陷，严重 35 个、中等 49 个、轻微 51 个，见表 3。该方法擅于发现复杂交互与边界问题，例如在系统测试中，精准定位付款反馈与用户状态联动导致数据一致性漏洞，而此漏洞未被常规测试及单一模型识别。

表 2　测试覆盖率的比较

Table 2　Comparison of test coverage

方法	Web 应用	App 应用	API 服务	平均值
手工测试方法	71.6	65.3	75.1	70.7
RNN(单一)	77.6	71.9	79.2	76.2
LLM(单一)	80.9	76.5	81.6	79.7
GA(单一)	79.1	74.3	81.2	78.2
主流自动化测试方法	76.3	71.2	77.3	74.9
基于强化学习的测试方法	82.2	77.9	83.6	81.2
基于深度学习的测试生成方法	83.5	78.4	83.9	81.9
本文方法	88.7	86.1	90.2	88.3

表 3　缺陷数量和分布

Table 3　Number and distribution of defects

方法	严重缺陷	一般缺陷	轻微缺陷	合计
手工测试方法	17	28	37	82
RNN(单一)	22	33	34	89
LLM(单一)	23	32	37	92
GA(单一)	20	33	37	90
主流自动化测试方法	19	32	37	88
基于强化学习的测试方法	25	34	38	97
基于深度学习的测试生成方法	22	37	39	98
RNN+LLM+GA	35	49	51	135

3.4.3　测试质量比较

为评估测试质量，10 余位测试工程师从可读性、可维护性、有效性(1~10 分制)对生成测试用例评分，结果见表 4。基于 RNN+LLM+GA 混合架构的开发测试方法在 3 项指标中表现最佳，有效性得分 8.6 分，显著优于其他方案；生成的测试用例能真实还原用户操作与业务逻辑，覆盖边界及异常场景完整，且注释清晰、易维护，测试框架扩展性强，可灵活适配需求变化。

表 4　测试质量比较

Table 4　Comparison of test quality

方法	可读性	可维护性	有效性	平均值
手工测试方法	5.9	5.3	6.3	5.8
RNN(单一)	6.6	5.9	7.1	6.5
LLM(单一)	7.2	7.1	7.9	7.4
GA(单一)	7.1	6.6	7.1	6.9
主流自动化测试方法	5.4	5.2	5.9	5.5
基于强化学习的测试方法	8.1	7.2	8.2	7.8
基于深度学习的测试生成方法	8.3	8.1	8.2	8.2
本文方法	8.5	8.6	8.6	8.5

3.4.4 测试效率比较

测试效率见表 5,尽管基于 RNN+LLM+GA 混合架构的开发测试方法在测试生成的时间成本与资源消耗上略高于部分对比方法,但测试执行效率和减少人工干预方面优势显著;测试执行时间较手工测试减少 23.3%,人工参与量降低 22.4%,显著提升效率并降低人力成本;随着测试推进,该方法展现出强大的自适应学习能力,测试生成耗时持续减少,精准度与全面性不断增强。

表 5 测试效率比较

Table 5 Comparison of testing efficiency

方法	测试生成时间/h	测试执行时间/h	人工参与(人日)	资源消耗
手工测试方法	0	45.6	27.0	12.5
RNN(单一)	5.1	32.9	8.6	44.9
LLM(单一)	4.7	30.3	7.1	68.5
GA(单一)	6.4	33.3	8.9	50.2
主流自动化测试方法	3.5	39.3	11.5	27.9
基于强化学习的测试方法	4.1	28.5	7.1	57.7
基于深度学习的测试生成方法	3.0	28.3	6.3	64.5
本文方法	6.6	22.3	4.6	73.1

3.4.5 消融实验

通过消融实验剖析模型性能,结果见表 6,结果显示移除任意组件均导致模型性能显著下降,验证了 RNN、LLM、GA 3 组件的不可或缺性与互补性。LLM 贡献最大,移除后测试平均覆盖率下降 7.2 个百分点;RNN 次之,下降 6.3 个百分点;GA 移除后覆盖率减少 5.1 个百分点。将改进版 RNN 换回基础版本亦引发性能衰退,佐证了改进方法有效性。

表 6 消融结果比较

Table 6 Comparison of ablation results

变体	Web 应用	App 应用	API 服务	平均值
RNN+LLM+GA	88.7	85.1	90.2	88.0
RNN(无)	81.2	77.3	83.6	80.7
LLM(无)	80.9	77.5	82.1	80.1
GA(无)	82.5	78.7	84.1	81.7
基础 RNN	85.6	81.1	85.5	84.1

4 结束语

本文设计 RNN+LLM+GA 混合架构应用于开发测试场景,深度融合 RNN 在序列数据处理的时序建模能力、LLM 的语义理解与生成优势及 GA 的优化搜索特性,实现测试流程从数据处理到用例生成、优化的全面智能化升级。其优势体现在:RNN 与 LLM 能力互补,通过剖析业务逻辑生成贴合实际场景的测试用例,业务场景覆盖度提升至 92%;LLM 与 GA 结合拓展测试场景并提升用例有效性,复杂系统性能测试中经 GA 优化的用例组合使瓶颈发现效率提高 40%,三模块联动实现“1+1+1>3”的协同增效;跨 Web 应用、API 服务等多类型软件及功能、性能等测试场景适配性强,增量学习机制可基于新结果动态优化模型,模块化设计便于集成前沿技术,如接入新 NLP 算法后用例生成效率提升 15%;具备测试生成可解释性,通过可视化逻辑呈现、参数调节及人工审查机制,融合 AI 效率与人工经验,提升结果可信度。但该架构存在局限性:处理复杂应用时模型训练与运行资源消耗大,LLM 训练依赖大量 GPU、周期长、成本高,实时推理耗时且存储需求大,制约小型团队和资源有限机构应用;对未纳入训练数据的新型交互模式、开发者的隐式假设场景及创新性功能覆盖不足,如智能推荐算法模块测试覆盖率仅 85%,存在 15%左右潜在漏洞。未来可从轻量级模型与边缘计算、自监督学习与持续适应、多样性测试生成及人机协同测试生态等方向研究,推动混合 AI 架构测试方法在应用开发测试中发挥更大价值,助力构建可靠高质量的应用程序。

参考文献

[1] 费敏锐,孟添. 人工智能应用[M]. 北京:中国人事出版社,

2019.
[2]闫喜亮,王黎明. 卷积深度神经网络的手写汉字识别系统[J]. 计算机工程与应用,2016,53(10):246-250.
[3]邱鹏. Python 程序语言的兴起对应用型本科 IT 人才培养和就业趋势研究[J]. 电脑知识与技术,2019,15(21):151-153.
[4]王东. 人工智能技术在软件测试领域的应用研究[J]. 现代计算机,2023,29(12):55-59.
[5]梁子鑫. 探讨新时代背景下新兴技术在人工智能中的应用[J]. 软件,2018,39(7):166-169.
[6]李明,张伟. 基于深度学习的软件测试用例自动生成方法研究[J]. 计算机研究与发展,2020,57(6):1234-1245.
[7]王芳,刘强. 人工智能在 Web 应用自动化测试中的实践与优化[J]. 软件学报,2021,32(3):567-578.
[8] 李徐,张帆,唐超. 一种基于神经网络梯度下降的多目标优化算法[J]. 智能计算机应用,2024,14(6):224-229.
[9] 潘伟杰,冲蕾,李霖奇. 基于深度学习与百度 AI 的菜谱推荐与健康分析微信小程序设计[J]. 智能计算机应用,2024,14(5):257-264.
[10] DAI Z, YANG Z, YANG Y, et al. Transformer-XL for test sequence generation: Extending context range for testing complex applications[C]//Proceedings of the 44th International Conference on Software Engineering. Piscataway, NJ: IEEE, 2022: 789-800.
[11] BROWN T, MANN B, RYDER N, et al. Language models are few-shot testers: Exploring LLM-based test generation[C]//Proceedings of the 45th International Conference on Software Engineering. New York: ACM, 2023: 1123-1134.
[12] LI K, ZHANG J, CHEN H, et al. Population based training for test case optimization[C]//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2022: 456-467.
[13] ZHAO L, CHEN Y, ZHANG H, et al. Cooperative co-evolutionary framework for test optimization[C]//Proceedings of the 46th International Conference on Software Engineering. Piscataway, NJ: IEEE, 2024: 678-689.