

基于扰动和冻结预训练模型的程序自动修复

张李政, 杨秋辉, 代声馨

引用本文

张李政, 杨秋辉, 代声馨. [基于扰动和冻结预训练模型的程序自动修复](#)[J]. 计算机科学, 2025, 52(12): 18-23.

ZHANG Lizheng, YANG Qihui, DAI Shengxin. [Automated Program Repair Based on Perturbing and Freezing Pre-trained Model](#) [J]. Computer Science, 2025, 52(12): 18-23.

相似文章推荐 (请使用火狐或 IE 浏览器查看文章)

Similar articles recommended (Please use Firefox or IE to view the article)

[基于改进ModernTCN的光伏发电中短期预测](#)

Prediction of Short-and-Medium Term Photovoltaic Power Generation Based on Improved ModernTCN
计算机科学, 2025, 52(11A): 241000164-7. <https://doi.org/10.11896/jsjcx.241000164>

[基于消除语义特征的图像篡改定位模型对抗攻击](#)

Attacking Image Manipulation Localization Model by Eliminating Semantic Features
计算机科学, 2025, 52(11A): 241100104-7. <https://doi.org/10.11896/jsjcx.241100104>

[自动化软件缺陷定位技术研究](#)

Advances in Automatic Software Defect Location Techniques
计算机科学, 2025, 52(11A): 250200024-14. <https://doi.org/10.11896/jsjcx.250200024>

[层次时间序列预测方法与应用综述](#)

Comprehensive Review of Hierarchical Time Series Forecasting Methods and Applications
计算机科学, 2025, 52(11A): 241000139-7. <https://doi.org/10.11896/jsjcx.241000139>

[融合自适应优化与多维聚焦的点云配准网络](#)

Point Cloud Registration Network Integrating Adaptive Optimization and Multi-dimensional Focusing
计算机科学, 2025, 52(11A): 250100019-7. <https://doi.org/10.11896/jsjcx.250100019>

基于扰动和冻结预训练模型的程序自动修复

张李政 杨秋辉 代声馨

四川大学计算机学院 成都 610065

(zleo0824@gmail.com)

摘要 随着软件复杂性的增加,程序缺陷的规模和复杂度也随之增加,程序缺陷不仅消耗大量开发成本,还会导致现实世界中的安全问题。现有的程序修复方法普遍存在修复效果不佳、训练成本高的问题。针对这些问题,提出了基于扰动和冻结预训练模型的程序自动修复方法。该方法通过基于矩阵的扰动方法对模型参数增加噪声,缓解了微调过程中预训练模型在程序修复任务上的过拟合问题;冻结预训练模型中的编码器,缩短了模型的训练时间和减少了计算资源的消耗。此外,通过检查点集成策略,增强了模型的修复效果。在 QuixBugs 数据集中的 40 个 Python 程序上进行实验,结果表明,所提方法在缩短模型训练时间和降低计算资源消耗方面以及修复效果方面都具有显著优势,它仅需要训练原始模型 41.62% 的参数量,训练时间缩短了 39.16%,能修复数据集中 70% 的缺陷,修复的缺陷类型具有多样性。

关键词: 程序自动修复;深度学习;预训练模型;微调;检查点集成

中图分类号 TP311.5

Automated Program Repair Based on Perturbing and Freezing Pre-trained Model

ZHANG Lizheng, YANG Qiuhui and DAI Shengxin

College of Computer Science, Sichuan University, Chengdu 610065, China

Abstract With the increasing complexity of software, the scale and complexity of program defects are also increasing. Program defects not only consume a large amount of development costs but also lead to real-world security issues. Existing program repair methods generally suffer from poor repair effectiveness and high training costs. To address these issues, this paper proposes an automatic program repair method based on perturbation and freezing of pre-trained models. By adding noise to the model parameters through a matrix-based perturbation method, it alleviates the overfitting problem of pre-trained models on the program repair task during fine-tuning. Furthermore, freezing the encoder in the pre-trained model reduces the model's training time and computational resource consumption. Additionally, the checkpoint ensemble strategy is adopted to enhance the model's repair effectiveness. Experiments on 40 Python programs in the QuixBugs dataset demonstrate that the proposed method has significant advantages in reducing model training time and computational resource consumption, as well as in repair effectiveness. It only requires training 41.62% of the parameters of the original model, reduces training time by 39.16%, and can repair 70% of the defects in the dataset, demonstrating the diversity of the repaired defect types.

Keywords Automated program repair, Deep learning, Pre-trained model, Fine-tuning, Checkpoint ensemble

1 引言

程序缺陷是指程序中的语法错误、语义错误、安全漏洞等问题。在日益复杂的软件系统中,程序中的缺陷数量与日俱增^[1]。据统计,缺陷程序的调试和修复占据了软件开发过程中超过 50% 的成本^[2],美国软件厂商在 2020 年花费近 1.56 万亿美元用于处理程序中的缺陷^[3]。程序缺陷不仅消耗时间和人力成本,还会导致现实世界中的安全事故。对于开发人员来说,缺陷修复既耗时伤神,又充满挑战且极易出错,

因此自动化修复程序中的缺陷成为软件开发中的重要课题。

利用深度学习技术实现程序缺陷修复,能够通过模型自动生成推荐补丁,通过验证的补丁可以直接修复缺陷,或者作为开发者修复时的参考,减少了开发人员调试代码所用的时间。

目前基于神经网络翻译(Neural Machine Translation, NMT)的研究被广泛应用到缺陷修复任务中,这些模型通过收集项目中过往的缺陷修复记录,并训练基于循环神经网络(Recurrent Neural Network, RNN)^[4-6]、长短期记忆网络

到稿日期:2024-11-28 返修日期:2025-03-07

基金项目:国家自然科学基金(62302323);四川省科技计划(2023NSFSC1413,2023YFG0117)

This work was supported by the National Natural Science Foundation of China (62302323) and Sichuan Science and Technology Program (2023NSFSC1413,2023YFG0117).

通信作者:杨秋辉(yangqiuhui@scu.edu.cn)

(Long Short-Term Memory, LSTM)^[7-8]、门控循环单元(Gated Recurrent Unit, GRU)^[9]等的 NMT 模型,能学习代码缺陷和修复后程序之间的映射关系,从而将缺陷代码“翻译”成正确的代码。在这些研究中,受限于训练数据的稀缺以及模型自身的规模,修复的效果不佳。

最近,CodeT5^[10], CodeT5 +^[11], CodeGen^[12], InCoder^[13], PLBART^[14]等预训练代码语言模型在代码生成、代码理解任务上取得了显著的效果。这些预训练模型在大规模、多样化的数据集上进行预训练,支持代码修复等下游任务。这些预训练模型能够充分理解代码中的语义信息,通过简单的微调过程,就能够超越传统基于 NMT 架构的模型在修复效果上的表现。

由于预训练模型训练时使用的数据集与实际需要修复的程序存在差异,仅使用缺陷数据集对预训练模型简单地微调会存在过拟合问题,仅能达到次优的效果^[15],并且预训练模型通常参数巨大,微调过程非常消耗时间和资源。

针对以上问题,提出了基于扰动和冻结预训练模型的程序缺陷修复方案。本文的贡献如下:

1)在模型训练阶段扰动模型参数缓解过拟合问题。通过基于矩阵的扰动方法在 CodeT5 预训练模型的参数中增加噪声,缓解预训练模型与下游任务的过拟合问题,提升模型的修复性能。

2)在模型训练阶段通过冻结编码器网络,降低微调时需要训练的参数量,缩短模型训练需要消耗的时间和减少计算资源。

3)应用检查点集成策略增强修复效果。在模型训练过程中保存多个检查点,在推理阶段使用多个检查点生成补丁并验证,多个检查点能够更好地捕获缺陷修复数据中的多样性。

2 相关工作

2.1 基于神经机器翻译的程序自动修复方法

NMT 在自然语言处理领域的文本翻译任务中取得了优秀的效果,缺陷修复任务也大量使用 NMT 架构,从缺陷代码“翻译”为修复后的正确代码。Chen 等^[16]提出的 SequenceR,基于 NMT 和序列到序列学习方法构建修复模型,并增加注意力机制提供更具体的上下文向量,使用复制机制解决词汇问题,实现了 Java 项目中程序缺陷的自动修复。Cao 等^[17]在 SequenceR 的基础上设计了 MNRRepair,该方法采用带注意力机制的编码器-解码器 NMT 框架,利用覆盖机制记录历史翻译信息,以解决过翻译和欠翻译问题。Lutellier 等^[18]提出的 CoCoNuT 使用 CNN 网络构建了具有上下文感知能力的 NMT 架构的修复模型,使用束搜索生成并排序候选补丁,完成了多语言的缺陷修复。

2.2 基于预训练模型的程序自动修复方法

最近,随着预训练模型在代码相关任务上取得了出色效果,涌现了大量使用预训练模型和微调的程序修复方法。

对于 T5 预训练模型,Fu 等^[19]提出 VulRepair,通过收集 C 语言漏洞修复数据集,并通过 T5 预训练模型学习缺陷代码和修复后代码的映射,实现了程序中漏洞的自动修复。Beraibi 等^[20]提出 Tfix,将程序缺陷构建为文本到文本的预测

任务,通过微调 T5 预训练模型,实现了 JavaScript 程序缺陷的自动修复。

对于 CodeBERT 预训练模型,Wan 等^[21]提出 CEMR,通过完全掩码策略遮蔽缺陷代码的 AST 中的所有标记,并使用 AST 重构技术生成一系列语义等价但语法结构各异的 AST,通过数据增强提升修复率。为了高效准确地生成补丁,Xiao 等^[22]提出了基于 CodeBERT 的 Confix 方法,该方法首先通过构建代码信息树,挖掘节点级的细粒度修复模板,并基于修复模板遮蔽缺陷代码行,以控制补丁的生成范围,最后通过引入修复策略来辅助预训练模型生成正确的补丁。

对于 CodeT5 预训练模型,Gharibi 等^[23]提出 T5APR,使用开源仓库中的缺陷修复历史微调 CodeT5 预训练模型,完成多语言的程序缺陷自动修复。Hao 等^[24]提出基于检索的修复框架 RetypeR,在外部检索阶段基于 AST 检索策略从历史修复中检索修复模式,在内部检索阶段引入整合了稀疏检索模块和密集检索模块的修复成分检索器,从而检索超出模型上下文窗口限制的程序内部信息。RetypeR 结合从两个阶段中检索到的信息构建上下文感知的修复提示,实现了 Python 语言中类型错误的自动修复。

除此之外,Ahmed 等^[25]提出基于 RoBERTa 预训练模型的修复方案 SynShine,该方案利用编译器诊断信息和无监督预训练方法,通过多标签分类方法生成补丁,实现了 Java 程序缺陷的自动修复。Prenner 等^[26]基于 Codex 预训练模型和提示词微调技术验证了 Java 和 Python 语言的修复效果。

随着软件系统的复杂度日益增加,提高代码的可靠性和可维护性是必要的,程序自动修复方法可以帮助开发人员提高代码质量,减少维护成本。以上程序自动修复方法由于模型和微调方法的局限性,在代码的修复能力和训练的成本上都还有提升空间,而本文方法能够用更少的训练成本实现更好的修复效果。

3 问题定义和总体方案

3.1 程序修复任务的定义

给定有缺陷的代码片段 $\mathbf{X}_i = [x_1, \dots, x_n]$,其包含 n 个代码标记,修复后的代码片段 $\mathbf{Y}_i = [y_1, \dots, y_m]$,其包含 m 个代码标记,程序修复的任务是最大化条件概率 $P(\mathbf{Y} | \mathbf{X}) = \prod_{i=1}^m P(y_i | y_1, \dots, y_{i-1}; x_1, \dots, x_n)$ 。

3.2 总体方案

图 1 展示了本文方案的整体流程。在预训练模型 CodeT5 的基础上使用扰动和冻结的方法微调模型,并使用检查点集合的策略,提高代码修复的能力,减少训练成本。

方案主要分为训练和推理两个阶段。在训练阶段:为缓解微调过程中的过拟合问题,对 CodeT5 模型的参数添加了基于矩阵的噪声;为了减少微调的参数量,在微调过程中冻结了模型的 Encoder 部分,只微调 Decoder 部分;使用检查点集成策略,保存训练过程中 k 个检查点。在推理阶段:分别对 k 个检查点进行推理,获得 k 个候选补丁列表,通过补丁在每个检查点上推理得到的排名和置信度对补丁排序,并通过测试套件验证这些补丁。

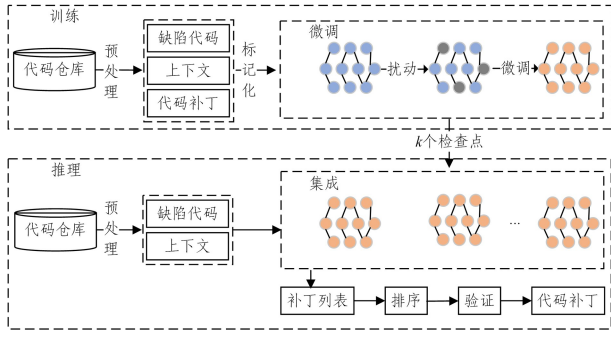


图1 程序修复方案流程

Fig. 1 Flowchart of the program repair scheme

3.3 数据预处理

本文采用 CoCoNuT 数据集训练模型。由于该数据集中有大量重复的数据、修复前和修复后代码相同的数据、修复代码缺失的数据,为了保证模型在高质量的数据集上学习,对数据集进行清洗,过滤掉上述无效数据。

在完成数据集清洗后,从数据集中提取缺陷代码、修复后代码,以及为了让模型充分学习到程序中的语义信息,还将包含缺陷代码的方法体作为上下文信息引入模型,最终模型的输入形式为 $I = \{buggy, fixed, context\}$ 。

3.4 微调 CodeT5 预训练模型

在微调过程中,由于预训练模型与下游任务之间存在差异,加之预训练数据和目标任务数据分布的差异,模型容易陷入局部最优,导致性能欠佳。此外,如果使用全局微调,则需要训练的参数量过大,导致训练成本高。因此,使用有效的微调策略以及正确谨慎地选择需要冻结的模块,能让模型在下游任务上表现更好并且降低训练成本。微调流程如算法 1 所示。

算法 1 预训练模型的微调算法

输入: 预训练模型 M , 数据集 D , 检查点数量 k , 轮数 T

输出: k 个模型检查点

1. 冻结模型的 Encoder, 仅微调 Decoder
2. for $\theta \in M$. Encoder. parameters() do
3. $\theta.requires_grad \leftarrow \text{False}$
4. end for
5. for $t \leftarrow 1$ to T do
6. for batch $\in D$ do
7. $\eta \leftarrow$ 生成与 Decoder 同尺寸的噪声矩阵
8. M . Decoder. parameters() $+= \eta$
9. $\hat{y} \leftarrow M(\text{batch})$
10. $L \leftarrow$ 计算损失(\hat{y}, y)
11. $\nabla L \leftarrow$ 计算梯度(L, M . Decoder. parameters())
12. 更新参数(M . Decoder. parameters(), ∇L)
13. end for
14. if 检查点数量 $< k$ then
15. 保存当前模型检查点
16. else
17. 替换当前最差的检查点
18. end if
19. end for
20. return k 个检查点

3.4.1 CodeT5 预训练模型

本文基于 CodeT5 模型进行微调, CodeT5 是一个编码器-解码器 Transformer 模型, 在 CodeSearchNet 数据集和 Big-Query 数据集上完成了模型的预训练。CodeT5 在对编程语言编码时, 为了捕获更多特定代码的特征, 将编程语言转换成抽象语法树, 并且提取每个代码令牌的节点类型来捕获更多特定代码的特征。在预训练过程中, CodeT5 有效地利用了代码中的标识符和代码注释, 通过完成掩码跨度预测、标识符标记、掩码标识符预测和双峰双生成任务来获得代码感知功能。CodeT5 有效地提升了模型对代码的语法和语义的理解, 并且在代码生成、代码摘要、代码优化等任务上实现了优秀的性能。

3.4.2 基于矩阵的参数扰动方法

通过在模型参数中增加噪声能够缩小预训练模型和下游任务的差异, 提升预训练模型的效果^[27]。由于预训练模型具有不同类型的参数矩阵, 添加统一噪声的效果不佳, 因此根据参数矩阵的方差, 向不同的参数矩阵添加不同强度的噪声, 帮助模型更好地适应下游任务的数据。本文令参数矩阵为 $[W_1, W_2, \dots, W_N]$, 其中 N 是参数矩阵类型的数量。扰动后的参数矩阵 \tilde{W} 如式(1)所示:

$$\tilde{W}_i = W_i + U\left(-\frac{\lambda}{2}, \frac{\lambda}{2}\right) * std(W_i) \quad (1)$$

其中, std 代表标准差, $U(a, b)$ 代表 (a, b) 的均匀分布噪声, λ 是控制噪声强度的超参数。

3.4.3 冻结 Encoder

在代码生成任务上, CodeT5 可以自适应其 Seq2Seq 框架, CodeT5 的编码器对输入的编程语言生成序列的隐藏表示, 解码器实现生成输出的序列。代码修复任务是一种生成类的任务, 为了适应下游任务, 本文将微调重点放在解码器上, 因此冻结了编码器, 保留模型在原始数据集上学习到的语义信息和修复能力, 仅对解码器网络微调, 从而减少了参数量, 加快了训练速度。

对算法 1 进行理论分析可以发现, 冻结编码器层后, 训练阶段不需要计算其梯度, 从而减少了反向传播过程中对编码器层的计算量, 同时因为反向传播不需要存储编码器层的梯度和中间激活值, 能够减少显存的使用。

对于有 N_{total} 个参数的模型, 编码器部分占 $N_{Encoder}$ 个参数, 冻结编码器层后, 节省计算量比例和节省显存比例如式(2)所示:

$$\text{节省计算量比例} = \text{节省显存比例} = \frac{N_{Encoder}}{N_{total}} \quad (2)$$

3.4.4 检查点集成策略

在机器学习中, 使用集成学习整合多个学习器的预测结果, 能有效提升单一学习任务的结果。但是训练多个模型需要大量开销并且不一定能获得好的效果。本文受集成学习策略启发, 将集成策略的思想应用到模型的检查点上, 通过集成多个检查点的结果, 获得了最佳候选补丁。具体而言: 在模型训练阶段, 保存 k 个步骤中的检查点; 在推理阶段, 使用 k 个检查点生成候选补丁。实验证明, 检查点集成策略有助于提高补丁的质量。

3.5 补丁生成和验证

本文使用模型的 k 个检查点进行推理,通过束搜索在每个检查点上生成缺陷程序的 n 个补丁,根据补丁在 k 个检查点上生成的排名和置信度合并补丁列表,得到最终的推荐补丁列表。

对于所有候选补丁,在去重后,首先按照排名进行升序排序,排名相同的,按照模型输出的置信度逆序排序;随后使用测试套件验证补丁是否能通过所有测试用例,对于通过的补丁标记为似真补丁;最后人工验证这些补丁是否与真实的修复程序语义一致,满足条件的标记为正确补丁。

4 实验验证

4.1 实验对象

本文使用 CoCoNuT 数据集对模型进行训练,CoCoNuT 数据集广泛应用于缺陷修复领域,具有权威性和代表性,并且数据质量高,其中包含 48 077 个从开源程序仓库中收集的 Python 程序缺陷以及对应的修复记录;使用 QuixBugs^[28] 数据集对方案进行验证,QuixBugs 包含 40 个单行缺陷的 Python 程序以及对应的测试套件。

4.2 评估指标

本文使用训练参数量和训练时间来评估修复方法在训练成本方面的有效性,使用似真补丁和正确补丁数量来评估模型的修复效果。

在验证过程中,对于模型生成的补丁,运行数据集中所对应的测试套件,当所有测试用例通过后,判定为似真补丁,最后将似真补丁与数据集提供的正确补丁进行对比,如果具有相同的语义则判定为正确补丁。

4.3 实验设计

为了验证本文方案的有效性,选取基于 NMT 架构的经典修复方法 CoCoNuT,以及目前广泛应用于工业领域的 Co-

dex 和修复效果最佳的 T5APR 进行对比实验,回答如下研究问题。

RQ1:本文方法是否能降低训练成本?

RQ2:本文方法中各部分对修复效果有什么影响?

RQ3:本文方法与目前先进的方法相比效果如何?

RQ4:本文方法的修复结果是否具有多样性?

4.3.1 RQ1

从表 1 中可以看出,当使用全参数微调时,模型需要训练的参数量为 60 492 288,此时训练一个 epoch 的时间为 2 053 s;当在微调过程中使用扰动方法时,模型训练时间与原始模型训练时间相差不大;当在微调过程中冻结编码器时,模型仅需训练原始模型 41.62% 的参数量,此时训练一个 epoch 的时间为 1 249 s,模型的训练时间缩短了 39.16%。

总的来说,实验结果表明,扰动模型参数对于模型训练时间几乎没有影响,当在微调过程中冻结编码器后,能够明显降低模型的训练参数量和训练时间,说明在程序修复任务方面,冻结预训练模型的编码器能够有效降低训练成本,缩短训练时间。

4.3.2 RQ2

从表 1 中可以看出,通过扰动为模型参数添加噪声后,模型在生成似真补丁方面得到了提升,共生成 29 个似真补丁,优于所有其他方案,但正确补丁数量仅有 26 个;当冻结编码器进行微调时,模型生成 28 个似真补丁,数量少于扰动后的模型生成的似真补丁数量,但其中共有 27 个正确补丁,超越了使用扰动的方法;当同时使用扰动和冻结参数时,模型的修复效果最好,能够生成 32 个似真补丁以及 28 个正确补丁;当不使用集成策略时,模型效果变差,生成了 26 个似真补丁,其中有 23 个正确补丁;在同时使用参数扰动、冻结编码器、集成策略时,获得了最优效果,模型共生成 32 个似真补丁,其中有 28 个正确补丁。

表 1 消融实验结果

Table 1 Ablation study results

模块	总参数量	冻结参数量	训练参数比例/%	训练时间/s	补丁数量(正确补丁/似真补丁)
CodeT5	60 492 288	0	100.00	2 053	24/28
CodeT5+Noise	60 492 288	0	100.00	2 036	26/29
CodeT5+Freeze	60 492 288	25 175 808	41.62	1 257	27/28
CodeT5+Noise+Freeze/w/o Ensmble	60 492 288	25 175 808	41.62	1 249	23/26
CodeT5+Noise+Freeze	60 492 288	25 175 808	41.62	1 249	28/32

对于检查点集成策略,探索了检查点的数量 k 对似真补丁数量的影响,实验结果如图 2 所示,当 $k=5$ 时,能得到最多的似真补丁。

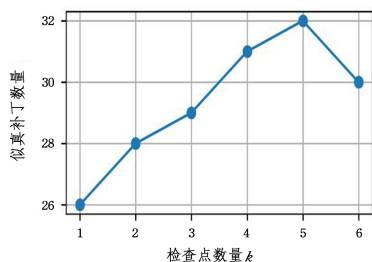


图 2 似真补丁数量随检查点的变化

Fig. 2 Number of plausible patches over checkpoints

总的来说,对模型参数进行扰动调整的策略,能够使模型在生成似真补丁方面具有优势,但是会生成更多的不正确补丁,在冻结参数后,虽然无法提升似真补丁的数量,但此时模型生成的结果更加正确,正确补丁的数量得到了提升。检查点集成策略也能明显提升模型的修复性能,检查点数量 $k=5$ 时效果最佳。

4.3.3 RQ3

表 2 列出了本文方案与当前主流修复方案的修复结果对比,图 3 显示了各个方法修复结果的差异,包括独立修复和重复修复的数量。从结果中可以看出,基于 NMT 架构的修复方法 CoCoNuT 表现最差,因为其模型的训练数据量少,无法充分理解程序的语义;基于 Codex 预训练模型的方法优于使

用 NMT 架构的修复模型,因为预训练模型在大规模数据集上进行训练,能充分理解代码的语义信息,但该方法仅通过简单的提示微调对模型调优,无法明显提升预训练模型在缺陷修复方面的能力,仅能修复数据集中 52% 的缺陷;T5APR 通过缺陷数据集对预训练模型微调,效果超过了 Codex,能修复 65% 缺陷程序;本文方案生成了 32 个似真补丁,其中 28 个为正确补丁,共修复了数据集中 70% 的缺陷,超越了其他主流方案。

表 2 对比实验结果

Table 2 Results of the comparative experiments

模型	补丁数量 (正确补丁/似真补丁)	修复比例/%
CoCoNuT	19/21	47.50
Codex	21/21	52.50
T5APR	26/28	65.00
Ours	28/32	70.00

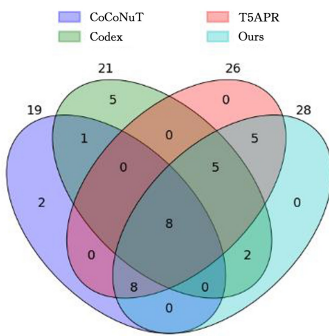


图 3 修复效果对比

Fig. 3 Comparison of repair results

总的来说,本文方案所获效果表明,在缺陷修复任务上,

if b == 0;	+ for x in arr[k:];	+ if len(arr) <= 1;
return a	- for x in arr;	- if len(arr) == 0;
else:	yield heapq.heappushpop(heap, x)	return arr
+ return gcd(b, a % b)		
- return gcd(a % b, b)		
gcd.py		
while n:	kheapsort.py	mergesort.py
+ n &= n - 1	if s[i] == t[j]:	def wrap(text, cols):
- n &= n - 1	+ dp[i, j] = dp[i - 1, j - 1] + 1	+ lines.append(text)
	- dp[i, j] = dp[i - 1, j] + 1	-
bitcount.py	lcs_length.py	wrap.py

图 4 修复结果案例(电子版为彩图)

Fig. 4 Case study of repair results

总的来说,本文方法能够正确修复多种类型的程序缺陷,修复结果具有多样性。

结束语 本文针对目前程序自动修复方法训练成本过高以及修复效果不佳的问题,提出了基于扰动和冻结的方法微调预训练模型,并通过检查点集成策略完成程序修复任务。本文在 QuixBugs 数据集上进行了验证,通过消融实验,研究了本文方法各部分对模型的影响,并且将本文方法与目前主流方法进行对比实验,验证了本文方法的有效性。

然而,所提方法仍然存在不足之处。受限于硬件资源,本文仅针对 CodeT5 预训练模型进行了实验,没有探索更大数量的模型在程序缺陷修复任务上的潜力,未来可以使用参数量更大的预训练模型进行实验;本文方案并不受限于 Python 语言的程序修复,CodeT5 模型在多个编程语言上对代码相关任务进行了预训练,未来可以针对其他语言进行实验,

使用预训练模型的修复效果优于基于 NMT 架构的模型。通过对比 Codex 和 T5APR 这两个基于预训练模型的方法,可知微调方法对于模型提升具有关键影响,本文的参数扰动和冻结的微调策略优于简单的提示微调和直接使用缺陷数据集的微调。

4.3.4 RQ4

为了评估修复结果的多样性,本文统计了数据集中不同缺陷类型及其对应的补丁数量。为了更直观地展示模型的修复效果,选取了部分程序进行了案例研究,具体统计结果如表 3 所列,部分修复案例如图 4 所示(程序已做省略处理),其中删除的代码以红色标出,新增的代码以蓝色标出。模型成功修复了 gcd.py 中的函数参数错误、kheapsort.py 中的循环条件错误、mergesort.py 中的判断条件错误、bitcount.py 中的运算符错误、lcs_length.py 中的数组下标错误以及 wrap.py 中的语句缺失错误。

表 3 缺陷类型与修复补丁统计

Table 3 Defect type and patch statistics

缺陷类型	缺陷数量	正确补丁数量
函数参数	8	6
函数调用	2	1
函数缺失	2	0
循环条件	6	5
判断条件	6	5
运算符	1	1
数组下标	2	1
语句缺失	4	4
返回语句	7	5
变量名	2	0
Total	40	28

以此验证该方案的泛化能力。

参考文献

- [1] LUCA G, DANIELA M, LEONARDO M. Automatic Software Repair: A Survey[J]. IEEE Transactions on Software Engineering, 2019, 45(1): 34-67.
- [2] ZHANG Q, FANG C, MA Y, et al. A survey of learning-based automated program repair [J]. ACM Transactions on Software Engineering and Methodology, 2023, 33(2): 55.
- [3] HUANG K, XU Z, YANG S, et al. A survey on automated program repair techniques [J]. arXiv:2303.18184, 2023.
- [4] CHEN Z, KOMMRUSCH S, TUFANO M, et al. Sequencer: Sequence-to-sequence Learning for End-to-end Program Repair. [J]. IEEE Transactions on Software Engineering, 2019, 47(9): 1943-1959.

- [5] TUFANO M, PANTIUCHINA J, WATSON C, et al. On Learning Meaningful Code Changes Via Neural Machine Translation [C] // IEEE 41th International Conference on Software Engineering. IEEE, 2019; 25-36.
- [6] TUFANO M, WATSON C, BAVOTA G, et al. An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation [J]. ACM Transactions on Software Engineering and Methodology, 2019, 28(4): 19-48.
- [7] CHAKRABORTY S, DING Y, ALLAMANIS M, et al. Codit: Code Editing with Tree-Based Neural Models [J]. IEEE Transactions on Software Engineering, 2022, 48(4): 1385-1399.
- [8] MENG X, WANG X, ZHANG H, et al. Improving Fault Localization and Program Repair with Deep Semantic Features and Transferred Knowledge [C] // Proceedings of the 44th IEEE/ACM International Conference on Software Engineering. 2022; 1169-1180.
- [9] GUPTA R, PAL S, KANADE A, et al. Deepfix: Fixing Common C Language Errors by Deep Learning [C] // Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. 2017; 1345-1351.
- [10] WANG Y, WANG W, JOTY S, et al. Codet5: Identifier-aware Unified Pre-trained Encoder-decoder Models for Code Understanding and Generation [C] // Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 2021; 8696-8708.
- [11] WANG Y, LE H, GOTMARE A, et al. CodeT5+: Open Code Large Language Models for Code Understanding and Generation [C] // Conference on Empirical Methods in Natural Language Processing. 2023; 1069-1088.
- [12] NIJKAMP E, PANG B, HAYASHI H, et al. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis [C] // International Conference on Learning Representations. 2022.
- [13] FRIED D, AGHAJANYAN A, LIN J, et al. InCoder: A generative model for code infilling and synthesis [J]. arXiv; 2204.05999, 2022.
- [14] AHMAD W, CHAKRABORTY S, RAY B, et al. Unified pre-training for program understanding and generation [C] // Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. ACL, 2021; 2655-2668.
- [15] QI W, YAN Y, GONG Y, et al. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training [C] // Findings of the Association for Computational Linguistics: EMNLP. 2020; 2401-2410.
- [16] CHEN Z, KOMM R S, TUFANO M, et al. Sequencer: Sequence-to-sequence learning for end-to-end program repair [J]. IEEE Transactions on Software Engineering, 2019, 47(9): 1943-1959.
- [17] CAO H L, HAN D, CHU Y H, et al. Multi-mechanism neural machine translation framework for automatic program repair [J]. Journal of Intelligent & Fuzzy Systems, 2024, 46: 7859-7873.
- [18] LUTELLIER T, PHAM H V, PANG L, et al. Coconut: Combining context-aware neural translation models using ensemble for program repair [C] // Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020; 101-114.
- [19] FU M, TANTITHAMTHAVORN C, LE T, et al. VulRepair: a T5-based automated software vulnerability repair [C] // Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2022; 935-947.
- [20] BERABI B, HE J, RAYCHEV V, et al. Tfix: Learning to fix coding errors with a text-to-text transformer [C] // International Conference on Machine Learning. PMLR, 2021; 780-791.
- [21] WAN H, LUO H Z, LI M Y, et al. Automated program repair for introductory programming assignments [J]. IEEE Transactions on Learning Technologies, 2024, 17: 1705-1720.
- [22] XIAO J M, XU Z P, CHEN S P, et al. Conflix: Combining node-level fix templates and masked language model for automatic program repair [J]. Journal of Systems and Software, 2024, 216: 112116-112130.
- [23] GHARIBI R, SADREDDINI M H, FAKHRAMAD S M. T5APR: Empowering automated program repair across languages through checkpoint ensemble [J]. Journal of Systems and Software, 2024, 214: 112083.
- [24] HAO S C, SHI X J, LIU H W. RetypeR: Integrated retrieval-based automatic program repair for Python type errors [C] // 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2024; 199-210.
- [25] AHMED T, LEDESMA N R, DEVANBU P. SynShine: Improved fixing of syntax errors [J]. IEEE Transactions on Software Engineering, 2023, 49(4): 2169-2181.
- [26] PRENNER J A, BABII H, ROBBES R. Can OpenAI's codex fix bugs? an evaluation on QuixBugs [C] // Proceedings of the Third International Workshop on Automated Program Repair. New York; ACM, 2022; 69-75.
- [27] WU C, WU F, QI T, et al. NoisyTune: A little noise can help you finetune pretrained language models better [C] // Annual Meeting of the Association for Computational Linguistics. 2022.
- [28] LIN D, KOPPEL J, CHEN A, et al. QuixBugs: a multi-lingual program repair benchmark set based on the quixey challenge [C] // Proceedings of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications; Software for Humanity. New York; ACM, 2017; 55-56.



ZHANG Lizheng, born in 2000, post-graduate, is a member of CCF (No. Q6918G). His main research interests include software quality assurance and testing and automated program repair.



YANG Qihui, born in 1970, Ph.D, associate professor. Her main research interests include software automation testing and software project management.