****

本科毕业设计(论文)

调研报告

|  |  |
| --- | --- |
| 题 目： | 端到端的缺陷自动修复  算法研究 |
| 学生姓名： | 王小娅 |
| 指导教师： | 康松林 罗京 |
| 学 院： | 计算机学院 |
| 专业班级： | 软件工程1802班 |

本科生院制

2022年2月

端到端的缺陷自动修复算法研究

摘要

软件缺陷在开发和维护过程中是不可避免的。随着软件规模的增大和复杂性的增强，完全依赖人工修复不仅给开发人员带来巨大的压力而且修复质量难以保证。软件缺陷自动修复技术有望将开发人员从繁重的调试任务中解脱出来，并提升软件的质量。

本课题旨在设计一个基于机器学习的端到端缺陷修复模型，能够自动化地生成单行补丁修复软件缺陷。该修复模型基于异步的序列到序列架构，将有缺陷的程序作为输入，修复好的程序作为输出。模型的编解码器皆使用引入门控机制的循环神经网络来缓解长程依赖问题。再结合缺陷上下文抽象技术和复制机制解决程序修复中的无限词汇表问题。模型在开源的真实项目中获取程序进行有监督的训练，生成的补丁会经过程序测试并和人工编写的补丁进行语义上的比对。

该报告为论述该课题的前期调研工作，包括该课题的研究背景、设计目标、国内外研究现状、相关技术基础、研究内容及方案、难点及初步解决办法、预期研究成果与时间进度安排等。

**关键词：**端到端 缺陷自动修复 机器学习 编解码器 注意力机制

**An End-to-end Automatic Program Repair Algorithm**

**ABSTRACT**

Software defects are inevitable during development and maintenance. With the increase of software scale and complexity, completely relying on manual repair not only brings great pressure to developers but also makes it difficult to guarantee the quality of repair. Automatic program repair is expected to free developers from heavy debugging tasks and improve the quality of software.

This paper aims to design an end-to-end defect repair model based on machine learning, which can automatically generate a single line of patches to repair software defects. The repair model is based on an asynchronous sequence-to-sequence architecture, with the defective program as input and the fixed program as output. The encoder and decoder of the model both use a recurrent neural network with the gating mechanism to alleviate the problem of long-range dependence. In addition, buggy context abstraction technique and replication mechanism are combined to solve the problem of infinite vocabulary in program repair. The model obtains programs from open source real projects for supervised training, and the generated patches are tested and semantically compared with manually written patches.

The golden model with good training is expected to get a lower degree of perplexity, can be verified on the defect repair benchmark data set Defects4J with high accuracy and recall rates, and will be tested on a large scale against the end-to-end defect repair benchmark model.

Keywords: end-to-end automatic program repair machine learning codec attention mechanism

目录

[第1章 绪论 1](#_Toc96932844)

[1.1 研究背景 1](#_Toc96932845)

[1.2 设计目标 1](#_Toc96932846)

[1.3 国内外研究现状 2](#_Toc96932847)

[1.3.1 缺陷自动修复技术发展现状 2](#_Toc96932848)

[1.3.2 基于机器学习的缺陷修复研究现状 3](#_Toc96932849)

[第2章 相关技术基础 6](#_Toc96932851)

[2.1 缺陷上下文抽象 6](#_Toc96932852)

[2.2 序列生成模型 8](#_Toc96932853)

[2.3 补丁推理 9](#_Toc96932854)

[2.4 补丁准备 9](#_Toc96932855)

[第3章 研究内容及实验方案 10](#_Toc96932856)

[3.1 研究内容 10](#_Toc96932857)

[3.1.1 如何评估算法缺陷修复的质量 10](#_Toc96932858)

[3.1.2 如何评估算法在基准数据集数据集上的有效性 10](#_Toc96932859)

[3.1.3 如何将提出的算法和其他端到端缺陷修复算法进行比较 10](#_Toc96932860)

[3.2 实验方案 10](#_Toc96932861)

[3.2.1 采用困惑度评估模型分布质量 10](#_Toc96932862)

[3.2.2 基于标准数据集中的单行缺陷评估算法有效性 11](#_Toc96932863)

[3.2.3 根据应用场景异同分别比较 12](#_Toc96932864)

[第4章 难点及初步解决办法 13](#_Toc96932865)

[4.1难点 13](#_Toc96932866)

[4.1.1 长程依赖问题 13](#_Toc96932867)

[4.1.2 无限词汇表问题 13](#_Toc96932868)

[4.1.3 召回率和准确率低 13](#_Toc96932869)

[4.2 初步解决办法 14](#_Toc96932870)

[4.2.1 采用缺陷上下文抽象与GRU 14](#_Toc96932871)

[4.2.2 采用复制机制 15](#_Toc96932872)

[4.2.3 采用注意力机制、束搜索和双向循环神经网络 15](#_Toc96932873)

[第5章 预期成果及时间安排 18](#_Toc96932874)

[5.1 具备的条件 18](#_Toc96932875)

[5.1.1 研究基础 18](#_Toc96932876)

[5.1.2 软硬件环境 18](#_Toc96932877)

[5.2 预期研究成果 18](#_Toc96932878)

[5.3 时间进度安排 19](#_Toc96932879)

[第6章 结论 20](#_Toc96932880)

[参考文献 21](#_Toc96932881)

# 第1章 绪论

## 1.1 研究背景

软件缺陷是软件中存在的影响正常运行能力的问题，或者是功能与性能的失效或违背。由于人为因素、软硬件系统的复杂性、进度压力等种种原因，软件缺陷在软件开发和维护过程中不可避免。严重的软件缺陷不仅会给组织带来经济和声誉上极大的伤害，甚至会对人类的生命安全造成重大的威胁。因此，缺陷修复意义重大。传统的缺陷修复由人工完成，然而随着软件规模的增大和复杂性的提升，完全依靠人工进行软件缺陷的修复会给维护人员造成巨大的负担，并且修复的效果难以得到保证。

软件缺陷自动修复指针对软件中存在的缺陷自动生成相应补丁，使得程序满足用户需求。该技术有望为软件开发维护人员分担繁重的修复任务和提升软件的质量做出贡献。近年来，软件缺陷自动修复成为一个热门的研究方向，目前已经融合了软件分析、启发式搜索、程序综合和机器学习在内的多种技术。随着人工智能的飞速发展，一些智能化的技术如神经网络和深度学习也被应用到缺陷自动修复中，这类方法的优点是通用性强，不需要依赖启发式规则和任何人工定义的模板。

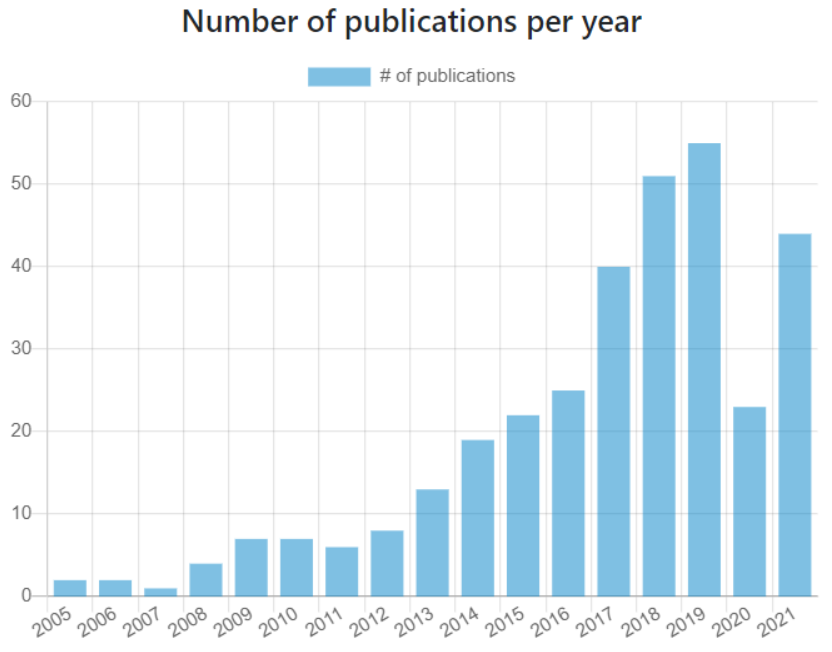
## 1.2 设计目标

设计一个端到端的软件缺陷自动修复算法，能够实现通过替换单行语句修复软件缺陷。该算法应用机器学习中基于循环神经网络的编码器-解码器模型，输入为有缺陷的源程序序列，输出为缺陷修复后的程序序列。算法采用缺陷上下文抽象操作(Buggy Context Abstraction)对输入的源程序首先进行符号化处理(tokenization)，以避免变量名称和常量值等带来的干扰。随后，缺陷上下文抽象会根据预定长度对过长的程序进行截断(truncation)，以实现在获得尽可能丰富的上下文信息的同时避免模型处理过长的输入。算法在解码器中应用全局注意力机制来获取序列中更重要的信息，从而提高神经网络的效率。最后，为了提高准确率和效率，算法采用束搜索(Beam Search)这一启发式搜索算法对修复的预测进行调优。

## 1.3 国内外研究现状

### 1.3.1 缺陷自动修复技术发展现状

过去的十几年中，来自软件工程、程序语言、人工智能、形式化验证等多个领域的学者提出了大量的缺陷自动修复技术[1]。下图是缺陷自动修复的共享主页(<https://program-repair.org/statistics.html>) 提供的统计结果。

  
图1-1 不同年份的论文发表数量分布

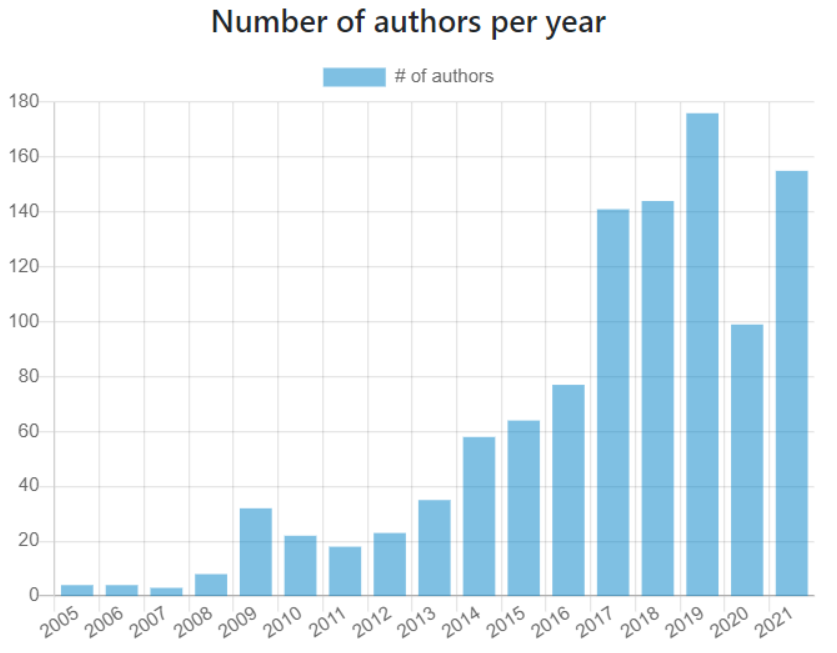
  
图1-2 不同年份的发表论文作者数量分布

图1-1展示了从2005年至2021年每年的论文发表数量统计，从图中数据可以发现软件缺陷自动修复领域论文发表数量呈逐年增加的趋势。仅在2019年就有55篇相关的论文发表。图1-2展示了从2005年至2021年每年发表论文的作者数量统计，数量最多的2019年达到了176位。这些数据表明，软件缺陷自动修复领域的研究热度在不断增加，成为了近年来的热门领域。

### 1.3.2 基于机器学习的缺陷修复研究现状

根据生成补丁的方式，可以将缺陷自动修复方法分为以下4种类型[2]，分别是基于搜索的启发式搜索的(例如GenProg[3]和SimFix[4])，基于语义约束的(例如PAR[5]和SketchFix[6])，基于人工修复模板的(例如Nopol[7]和Angelix[8])以及基于机器学习的(例如Deep-Repair [9]和MiMo[10])。我的毕业设计主要研究的端到端的补丁生成模型属于第四类，因此下面着重介绍基于机器学习的程序缺陷自动修复方法的研究现状。

2017年，Rahul Gupta等人[11]提出了修复常规程序错误的端到端方案DeepFix，其核心是一个序列到序列神经网络模型。该模型的编解码器均采用GRU(Gated Recurrent Unit，门控循环单元网络)，并结合了注意力机制(Attention Mechanism)来提高缺陷预测的准确性。实验结果表明，对于6971个包含常规错误的C语言程序，DeepFix能够完全修复1881个(占比27%)程序，能够部分修复1338个(占比19%)程序。

2018年，Sahil Bhatia等人[12]提出了一种针对学生Python语言作业的修复方式。该方式结合了神经网络和基于约束的推理，首先运用一个RNN(Recurrent Neural Network，循环神经网络)对缺陷程序执行语法上的修复，接着在使用语义约束技术来保证程序在功能上的正确。该技术应用到14500个有错误的学生提交作业中，能够正确修复8689个(占比60%)语法错误，正确完成3455个(占比23.8%)功能性修复。该方法的数据集程序功能相对简单，但是在上一个工作的基础上增加了语义上的修复。

2019年，Vasic M等人[13]针对程序中的变量误用这一类型的错误提出了一个联合预测模型。其主体部分是一个多头指针网络(multi-head pointer netwirk),能够对程序中缺陷的位置及补丁进行联合预测。论文作者在开源网站上的15万条Python代码上对该方法进行实验，结果表明出错位置的预测准确率为71.0%，修复准确率为65.7%。和之前同类型的方法相比，该方法更适用于大型的项目代码，但有着仅针对变量误用类型错误的局限性。

2020年，Thibaud Lutellier等人[14]提出了能够应用于多种编程语言的自动修复模型CoCoNuT，该模型结合了CNNs(convolutional neural networks，卷积神经网络)和一个上下文敏感的NMT(neural machine translation，神经网络机器翻译)架构。其中NMT架构分开表示缺陷源代码及其上下文，CNNs被堆叠从而获取层次化的特征。相较于RNNs，CNNs能够更好地对不同粒度级别地源代码进行建模。在由Java, C, Python以及JavaScript四种语言组成的6个数据集上对CoCoNuT进行实验，共计正确修复了509个缺陷，其中有309个缺陷是其他27中技术都未能正确修复的。

综上，可以看出基于机器学习的修复技术近年来迅速发展。相较于其他类型修复技术，这类模型从数据中学习，优点是通用性更强，不再需要依赖人工精心设计的启发式搜索方法或人工定义的修复模板。虽然大量的开源软件为机器学习模型提供了充足的训练数据，但这些未经过滤的数据噪声很强，距离真正应用还需要耗费人工大量精力去进行标准化和截断。其中端到端的缺陷修复模型主要适用于NLP(Natural Language Processing，自然语言处理)领域，而程序语言和自然语言相比，存在对错误的容忍度更低、词表更庞大、上下文中依赖距离更远的特性，因此这类模型目前的表现并不理想。尽管存在以上困难，基于机器学习的修复技术依然具备很大提升空间，并且有望成为未来的实用化修复方法。

# 第2章 相关技术基础

缺陷修复模型是一个序列到序列的机器学习模型，适用函数体中只有一条缺陷语句的场景。其技术路线如图2-1所示。

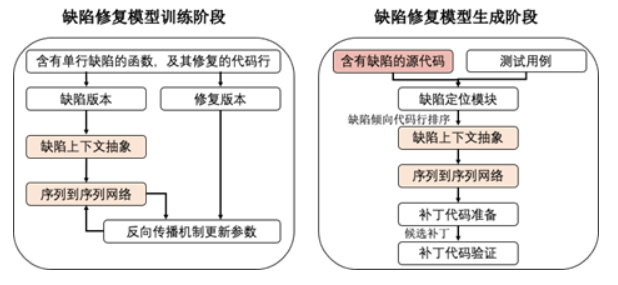


图2-1 缺陷修复模型技术路线图

对于一个含有缺陷的软件系统，首先将采用经典的缺陷定位技术识别有缺陷的方法以及代码行。然后，模型进行缺陷上限文抽象处理操作，将缺陷定位的数据组织成一个简明的、适合深度学习模型进行学习但又不会丢失重要信息的表示。随后，该表示将被送入到一个序列生成模型以进行修复补丁的推理，并能够生成多个单行的补丁作为修复缺陷的候选补丁。最后，模型在补丁准备阶段通过格式化操作和替换缺陷语句操作生成修复补丁。

## 2.1 缺陷上下文抽象

缺陷的上下文内容对于理解缺陷以及推理可能的修复方案具有重要作用。在缺陷修复过程中，开发人员通常需要找到缺陷所在的行，然后分析该行代码如何与方法中的其他代码之间进行交互，并观察其上下文(如变量和其他方法)以便于推理出可能的修复方案，并可能从上下文中选择相关的代码序列进行缺陷修复。

缺陷上下文抽象通过抽象缺陷上下文来模仿这一过程，并将缺陷定位的相关数据组织成简明的但同时能保留重要上下文信息用于预测可能的修复的一个表征。因此，模型需要处理缺陷上下文不可过长也不可过短两个相对冲突的目标。

(1)尽可能减少缺陷上下文的序列长度以构建精简的序列。

(2)尽可能保留更多的上下文信息，用于预测可能的修复。

对于一个给定的缺陷位置集合，对于每一个位置（包括缺陷的类，缺陷的方法和缺陷的代码行,即, 修复模型将执行以下操作：

**缺陷代码行**。在缺陷代码行之前插入标记，并在结尾插入标记，其基本原理是模型想传播缺陷定位技术中提出来的信息，并指示模型哪一个是含有缺陷的行。这可以模仿开发人员在修复缺陷的过程中关注到含有缺陷的代码行的行为。

**缺陷方法**。含有缺陷的代码行中的函数剩余部分将保持原样，用于生成方法的表示。其基本原理是，方法提供了重要的信息表示缺陷代码行所在位置以及缺陷代码行与方法剩余部分的交互信息。

**缺陷类**。对于缺陷类而言，模型保存所有类实例的变量、初始化函数、类的构造器签名以及不含缺陷的、即使没有被缺陷方法调用的方法。对于非怀疑方法的函数体内容将被省略。其基本原理是，修复模型将变量名和方法签名作为可能的构建修复补丁行的素材。

通过这些操作之后，模型将进行分词和截断操作，以便缺陷上下文抽象操作。其中，截断操作主要用于限制缺陷上下文抽象操作的序列长度，使其满足预定义的长度。截断操作可以分为无需处理、截断处理、填充处理三种情况。

(1)无需处理。保障绝大部分的文件不需要进行截断操作的处理。

(2)截断处理。如果缺陷行本身的长度超过了截断长度，那么将按照从左到右的顺序保留。

(3)填充处理。缺陷行本身的长度没有达到截断长度，缺陷的代码行将整个保留，并使得缺陷行前面的内容保留的长度尽可能是缺陷行后面内容保留的长度的两倍。

总体而言，截断操作将删除真实类定义中的内容，但是缺陷行上下文的信息将被保留并辅助缺陷补丁的生成。

## 2.2 序列生成模型

在此阶段，模型将学习对于一个给定的缺陷，如何生成对应的补丁。特征地，该模型是一个序列生成模型，是一个编码器和解码器结合的模型，并使用注意力机制和拷贝机制，如图2-2所示。为了训练这样一个模型，需要收集大量的缺陷数据，并将其划分成训练集和测试集。

在训练阶段，源代码序列经过缺陷上下文抽象后，即输入到模型的编码器中，其中表示缺陷上下文抽象的序列长度。然后，解码器产生目标输出序列，即生成修复补丁序列，其中表示生成的目标补丁序列长度。此外，方向传播和随机梯度下降将用于更新模型的参数。

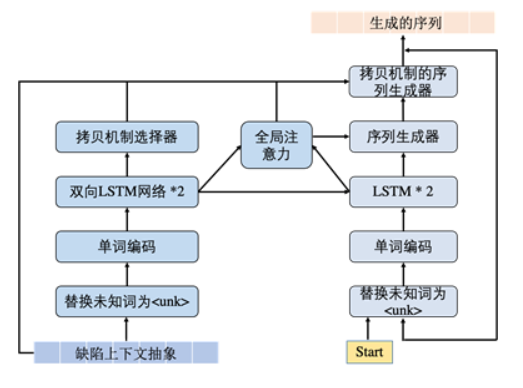


图2-2 序列生成模型示例

## 2.3 补丁推理

一旦序列生成模型训练好了之后，就可以在训练数据之外的数据集上进行测试。在补丁推理阶段，模型针对缺陷仍然生成抽象的缺陷上下文。然后使用beam-search技术生成多个可能的修复补丁。Beam-search技术通过到达当前解码状态的最好的n个序列。这些被成功筛选的是通过他们各自的累加概率计算并排序得到的。随后，这n个最好的序列将被传递到解码器的下一个状态。其中，n表示beam-search的宽度或者大小。

## 2.4 补丁准备

从序列生成模型得到的原始输出不能够直接生成可用的补丁。主要原因有生成的序列包含未知字符和空白两个方面。

1)未知字符。生成的序列中可能包含未知字符，仅管我们使用了拷贝机制，但也不能保证完全处理掉未知的字符

2)空白。生成的序列中的每个字符之间都含有空白，这在很多情况下并不能很好的形成代码的风格(如，空白符是不允许在点操作，调用操作或者取属性操作之后的)。

因此，需要一个补丁准备阶段。该阶段将抛弃所有含有未知字符，的候选补丁，并通过增减空格来格式化生成的序列，使其满足较好的代码展示风格。最后保留的补丁，将替换源程序中含有特殊标记的缺陷代码行，即以开头，以结尾的代码行。

# 第3章 研究内容及实验方案

## 3.1 研究内容

### 3.1.1 如何评估算法缺陷修复的质量

在算法能够实现通过替换单行语句修复程序缺陷的基本要求后，还需要一个评价指标来评估缺陷修复的效果。

### 3.1.2 如何评估算法在基准数据集数据集上的有效性

Defects4J[15]是一个来自多个Java语言开源项目的可复现缺陷数据集[16]。近年来许多热门缺陷自动修复技术采用Defects4J作为基准数据集[17-20]。因此Defects4J的实验结果是证明模型有效性的一项重要依据。

### 3.1.3 如何将提出的算法和其他端到端缺陷修复算法进行比较

近年来出现大量基于机器学期的端到端缺陷修复算法，在算法之外，算法适用的场景也不同：数据集可能来自真实程序、学生作业或基准数据集；数据集的语言是某种编程语言（C/C++，Java，JavaScript，Python）或多种语言（Multilingual）；缺陷修复的方式有替换、删除、插入；修复的缺陷有语法和语义之分。因此如何将本次毕业设计提出的算法与其他算法进行有效比较从而证明有效性是复杂且重要的研究内容。

## 3.2 实验方案

### 3.2.1 采用困惑度评估模型分布质量

困惑度(perplexity)是信息论中的一个概念，可以用来衡量两个分布之间的差异。 对于一个未知的数据分布和一个模型分布，我们从中采样处一组测试样本，模型分布的困惑度为

(3-1)

其中为样本的经验分布和模型分布之间的交叉熵，也是所有样本上的负对数似然函数。

具体到序列生成模型，困惑度的定义为

(3-2)

其中是源程序序列，是正确的目标序列，是第个目标词。

采用困惑度来衡量模型分布和样本经验分布之间的契合程度，困惑度越低说明缺陷修复算法的质量越高[21]。

具体而言，计划用Open-NMT[22]这一开源神经网络机器翻译框架来实现模型的困惑度度量，并计算另外两个先进的端到端修复技术的困惑度，在相同数据集上进行比较。

### 3.2.2 基于标准数据集中的单行缺陷评估算法有效性

本次毕业设计预计构建的模型实现的缺陷修复类型是单行替换修复，而在Defects4J中有75个缺陷能够被人工单行补丁被正确修复。在实验中，为了将缺陷定位和补丁生成区分开，会同时输入缺陷代码文件及缺陷所在行的序号。模型会通过束搜索生成若干个补丁，所有的补丁会通过编译器和编写好的测试代码的测试，最后和人工编写的补丁进行语义上的比对。

生成的补丁可以分为以下4种类型：错误补丁(incorrect patches)、编译正确补丁(compilable patches)、似真补丁(plausible patches)和正确补丁(correct patches)。不能够正确编译的补丁属于错误补丁。能够正确编译但无法经过测试工具结果不正确的属于编译正确补丁。通过了测试工具的验证、但是语义上不符合要求的补丁属于似真补丁。同时满足能够被正确编译、测试结果正确、语义上和人工编写结果等价三个条件的补丁属于正确补丁。

(1)从缺陷角度出发

在Defects4J数据集的75缺陷代码上对提出的算法进行测试，需要收集的数据是：针对多少个缺陷生成了补丁、针对多少个缺陷生成了编译正确补丁、

针对多少个缺陷生成了似真补丁、针对多少个缺陷生成了正确补丁。最后计算正确补丁修复的缺陷数和缺陷总数的比值，关注召回率。

(2)从补丁角度出发

在Defects4J数据集的75缺陷代码上对提出的算法进行测试，需要收集的数据是：生成的补丁总数、生成的编译正确补丁数、生成的似真补丁数、生成的正确补丁数。最后计算正确补丁数和似真补丁数的比值，关注准确率。

### 3.2.3 根据应用场景异同分别比较

（1）和应用场景相似的算法

在同一数据集上(如Bears[23]，bugs.jar[24]，Defects4J)和两个前沿水平的(state-of-the-art)针对Java语言的端到端补丁生成模型比较，计划选取Michele Tufano等人提出的基于神经机器翻译(Neural Machine Translation)的缺陷修复算法[25]和Chen等人提出的SequenceR模型[26]。前者在GitHub中收集了上百万个实际开发者进行缺陷修复的历史，从中提取代表性的例子训练编解码器模型，结果表明在9%-50%的情况下该模型能够预测出开发者的修复补丁。

数据集均需要经过处理，筛选出通过单行替换能够正确修复的缺陷代码，再控制输入的词的数量，确保对于三个模型输入都是合理的。

（2）和应用场景不同的算法

在模型复杂程度、算法分解后应用的策略异同、模型训练和推理时间、测试数据的修复难度等方面进行比较，从而提取本次毕业设计提出算法的优点，并通过向其他模型学习弥补不足。

# 第4章 难点及初步解决办法

## 4.1难点

### 4.1.1 长程依赖问题

端到端的缺陷修复模型中的编解码器均采用循环神经网络，在模型学习的过程中由于梯度消失或爆炸，状态之间长时间间隔(Long Range)的状态之间的依赖关系难以建模。而程序语言相比自然语言，往往存在更多的长时间间隔的依赖关系，如一个变量的使用和该变量的声明可能间隔几十行的代码。如果时刻的输出依赖于时刻的输入，当间隔比较大时，简单的神经网络很难建模这种长距离的依赖关系，称为长程依赖问题(Long-Term Dependencies Problem)[27]。

### 4.1.2 无限词汇表问题

由于开发人员命名习惯的不同和定义标识符的自由性，程序中可能出现的标识符有无限多个。如果把一段源程序看成是来自某一词表的词组成的序列，那这个词表远大于自然语言的词表。这对通过单句替换来解决程序缺陷方法造成了一个难点，即正确的补丁需要的词有可能并不包含在训练词表中。

### 4.1.3 召回率和准确率低

目前广泛使用的反映缺陷修复工具生成补丁能力的评价指标是召回率(recall)，反应补丁质量的评价指标是准确率(precision)。二者的定义如下。

而召回率和准确率是一对矛盾的度量标注，二者在某种程度上相互制约。例如希望提高召回率，就要尽可能提高修复工具的补丁生成能力，这样会导致更多的似真补丁，而降低了修复的准确率。因此，实际的缺陷修复算法必须在两者之间权衡。

## 4.2 初步解决办法

### 4.2.1 采用缺陷上下文抽象与GRU

(1) 缺陷上下文抽象

缺陷上下文抽象技术可以从有缺陷的源代码周围捕获最重要的上下文信息从而降低输入序列的复杂度。

具体而言，对于一个给定了错误位置的序列，会在缺陷所在行的首尾分别插入和的标识。缺陷所在方法的全部代码会被完整地保留下来。对于缺陷所在的类，保存所有类实例的变量、初始化函数、类的构造器签名以及不含缺陷的、即使没有被缺陷方法调用的方法。对于非怀疑方法的函数体内容将被省略。

不同的初始代码在完成上述操作后会得到不同长度的序列。算法的装桶策略即根据预先设定好的不同长度，分别对各种长度的序列进行截断处理。若一段序列超出设定好的长度，会从代码开始处进行保留；若一段序列未达到设定好的长度，会将序列前后无关的代码也包括进来，并且满足前面的代码长度是后面的两倍。

在此之后，还会进行标识化工作。一段抽象上下文代码是来自某一词表的词序列。对于超出词表的内容，用这个标识符来取代。

(2) 采用GRU

改善循环神经网络长程依赖问题的一种比较成熟的解决方案是引入门控机制，通过有选择地加入新的信息和遗忘旧的积累信息来控制信息的积累速度。普通循环神经网络的状态更新公式为

(4-1)

GRU（Gated Recurrent Unit，门控循环单元）网络[28]在公式1的基础上增加了更新门（Updata Gate），能够同时控制当前状态对历史状态的保留比例和从候选状态中接受新信息的比例，即

(4-2)

其中为更新门，

(4-3)

当时，当前状态和前一时刻的状态之间为非线性函数关系，当时，和之间为线性函数关系。下图是GRU网络的循环单元结构

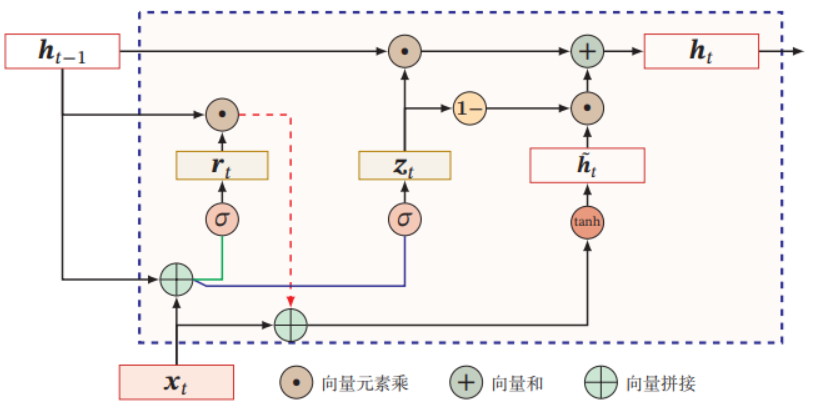


图4-1 GRU网络的循环单元结构

### 4.2.2 采用复制机制

复制机制的原理是一个正确补丁所需要的超出词表的罕见词，极有可能就在输入的词序列中。因此允许算法从输入的抽象上下文中直接获取词，会使得修复的召回率显著提升。具体实现时，计划在编码过程引入一个因子，该因子决定算法是否从输入中获取词。抽象缺陷上下文中的标识符或者被输入序列中的词替换，或者被训练词表中的词替换。如果一个补丁中的违背替换，则这样的补丁会被丢弃。

### 4.2.3 采用注意力机制、束搜索和双向循环神经网络

本次构建的模型更注重解决低召回率问题，具体而言是通过注意力机制和束搜索来提升召回率。

1. 注意力机制

注意力机制能够在计算资源有限的情况下，对资源进行重新分配使得更重要得信息得到处理。其计算过程分为两步：一是在所有输入信息上计算注意力分布，二是根据注意力分布来计算输入信息的加权平均。具体到本次毕设采用的编解码器架构，注意力机制的使用过程如下。

在解码过程的第t步时，先用上一步的隐状态hdect-1作为查询向量，利用注意力机制从所有输入序列的隐状态Henc=[henc1,...,hencS]中选择相关信息。

(4-4)

其中为打分函数。

然后，将从输入序列中选择的信息也作为解码器在第步时的输入，得到第步的隐状态

(4-5)

最后，将输入到分类器中来预测词表中每个词出现的概率。

1. 束搜索

使用自回归模型生成最可能的序列的过程，时从左到右的贪婪式搜索过程。束搜索作为一种常用的减少搜索错误的启发式方法。在每一步中，生成K个最可能的前缀序列，其中超参数K为束的大小（Beam Size）。图4-2是束搜索过程的实例，其中词表V={A,B,C}，束的大小为2。

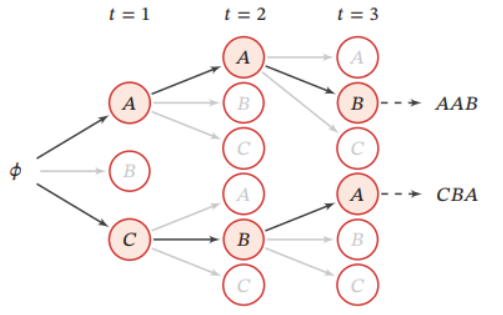


图4-2 束搜索实例

束搜索的过程如下：在第1步时，生成个最可能的词；在后面每一步中，从个候选输出中选择个最可能的序列。束的大小越大，束搜索的复杂度越高，但越有可能生成最优序列。需要通过调整束大小来平衡搜索质量和模型的复杂度。

1. 双向循环神经网络

在缺陷修复任务中，一个缺陷不仅和之前的程序相关，也和之后的程序相关。因此能够增加一个按照时间的逆序来传递信息的网络层，来增强网络的能力。

双向循环神经网络(Bidirectional Recurrent Neural Network，Bi-RNN)由两层循环神经网络组成，它们的输入相同，信息的传递方向相反。

假设第1层按时间顺序传递信息，则第2层按照时间逆序传递，在时刻的因状态定义为和，则

(4-6)

(4-7)

(4-8)

其中为向量拼接操作。下图给出了按时间展开的双向循环神经网络

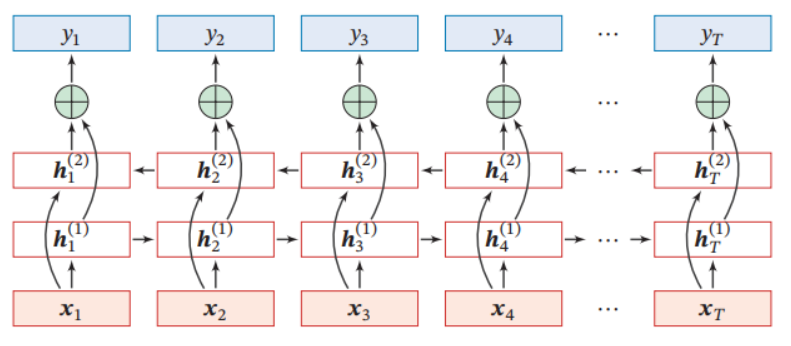


图4-3 按时间展开的双向循环神经网络

# 第5章 预期成果及时间安排

## 5.1 具备的条件

### 5.1.1 研究基础

该课题来源于硕士导师的指导，硕士导师长期从事软件质量的分析研究工作，在缺陷预测，缺陷定位和缺陷自动修复方法具有丰富的研究经验，并在国内外著名期刊和会议上发表了多篇文章：开发语言特性对软件缺陷的影响（TOSEM’2021），函数级程序缺陷自动修复（TOSEM’2021）,跨项目软件缺陷预测中有无监督方法之间的对比（TSE’2020），基于特征聚类的缺陷预测研究（JCST’2017，COMPSAC’2017）,基于多任务模型的缺陷预测研究（JSEP’2019）,基于多目标优化的缺陷预测研究（JSS’2019）,缺陷数研究中有监督和无监督方法的研究（IST’2019），基于特征选择和实例选择的跨项目软件缺陷预测（软件学报’2019），异构的跨项目缺陷预测（IST’2021）等。并且部分工作是基于导师的先期研究基础，具有理论可行性。

### 5.1.2 软硬件环境

此外，当前方法的提出需要大量的计算资源，硕士导师团队是浙江大学计算机学院软件所的老师，拥有几十台PC机，多台高性能PC服务器及配套软件。批准我使用该实验室的软、硬件条件进行实证研究、代码开发和测试，从而为本项目的研究提供充分基础设施保障，在实验设备上具有可行性。

## 5.2 预期研究成果

开发出一个基于Open-NMT框架的端到端缺陷修复模型，能够通过单行替换的方式修复Java语言程序方法内的缺陷。在Defects4J基准缺陷修复数据集上达到良好的召回率和准确率，和前沿水平的同类型同应用场景算法相比有明显的改进。

## 5.3 时间进度安排

对毕业设计目标进行自顶向下的分解，估算每个子任务的工作量并安排相应的时间，绘制出毕业设计进度安排甘特图，如图5-1所示。

ID

任务名称

开始

结束

周期

2022/2/21-2022/6/4

1

完成调研报告、文献翻译

2022/2/21

2022/2/26

1 周

2

掌握OpenNMT-py这一神经网络机器翻译框架，初步实现序列到序列缺陷修复模型

2022/2/28

2022/3/12

2 周

3

将模型的编解码器替换为门控神经单元(GRU)进行消融实验，收集数据

2022/3/14

2022/3/19

1 周

4

采用装桶(bucketing)策略改进模型的缺陷上下文抽象中的截断操作进行实验，收集数据

2022/3/21

2022/3/26

1 周

5

在Defects4J数据集上验证模型的修复效果，计算模型的困惑度(perplexity)

2022/3/28

2022/4/2

1 周

6

复现两个用于对比的基准模型

2022/4/4

2022/4/9

1 周

7

确定合适的数据集，将训练好的最佳模型与基准模型进行比较

2022/4/11

2022/4/16

1 周

8

继续从各方面改进模型，重复之前的实验，整理数据

2022/4/18

2022/4/30

2 周

9

检查实验结果，

撰写毕业论文

2022/5/2

2022/5/14

2 周

10

根据指导老师的修改意见

完善论文

2022/5/16

2022/5/28

2 周

11

整理材料，准备毕业设计答辩

2022/5/30

2022/6/4

1 周

图 5-1 毕业设计进度安排甘特图

# 第6章 结论

软件缺陷在开发和维护过程中由于种种因素不可避免。随着软件规模的增大和复杂性的增强，完全依赖人工修复不仅给开发人员带来巨大的压力并且修复质量难以保证。近年来软件缺陷的自动修复技术蓬勃发展，其中基于机器学习的修复技术有望成为未来实用化的自动修复方式。本次毕业设计计划提出的端到端的缺陷自动修复算法属于此类，能够通过单行替换的方式对真实项目中的缺陷进行有效修复，具有实际的研究意义。预计训练好的最佳模型能够取得较低的困惑度，在缺陷修复基准数据集Defects4J上验证得到较高准确率和召回率，并和端到端缺陷修复基准模型进行大规模的对比试验。相关的技术基础如门控神经单元网络、缺陷上下文抽象、束搜索等已经发展成熟，同时目前具备的研究条件充足，因此本课题的在理论和实验上都具备可行性。

# 参考文献

1. 姜佳君,陈俊洁,熊英飞.软件缺陷自动修复技术综述[J].软件学报,2021,32(09):2665-2690.DOI:10.13328/j.cnki.jos.006274.
2. 曹鹤玲,刘昱,赵晨阳,王玉华.程序缺陷自动修复研究进展及关键问题[J/OL].小型微型计算机系统:1-13[2022-02-23].[http://kns.cnki.net/kcms/detail/21.1106.TP.20211 102.1359.024.html](http://kns.cnki.net/kcms/detail/21.1106.TP.20211102.1359.024.html).
3. Le Goues C, Nguyen T V, Forrest S, et al. Genprog: A generic method for automatic software repair[J]. Ieee transactions on software engineering, 2011, 38(1): 54-72.
4. Le Goues C, Nguyen T V, Forrest S, et al. Genprog: A generic method for automatic software repair[J]. Ieee transactions on software engineering, 2011, 38(1): 54-72.
5. Kim D, Nam J, Song J, et al. Automatic patch generation learned from human-written patches[C]//2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013: 802-811.
6. Hua J, Zhang M, Wang K, et al. Towards practical program repair with on-demand candidate generation[C]//Proceedings of the 40th international conference on software engineering. 2018: 12-23.
7. Xuan J, Martinez M, Demarco F, et al. Nopol: Automatic repair of conditional statement bugs in java programs[J]. IEEE Transactions on Software Engineering, 2016, 43(1): 34-55.
8. Mechtaev S, Yi J, Roychoudhury A. Angelix: Scalable multiline program patch synthesis via symbolic analysis[C]//Proceedings of the 38th international conference on software engineering. 2016: 691-701.
9. White M, Tufano M, Martinez M, et al. Sorting and transforming program repair ingredients via deep learning code similarities[C]//2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2019: 479-490.
10. Zhong H, Mei H. Mining repair model for exception-related bug[J]. Journal of Systems and Software, 2018, 141: 16-31.
11. Gupta, Rahul, Soham Pal, Aditya Kanade and Shirish K. Shevade. “DeepFix: Fixing Common C Language Errors by Deep Learning.” AAAI (2017).
12. Bhatia S, Kohli P, Singh R. Neuro-symbolic program corrector for introductory programming assignments[C]//2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 2018: 60-70.
13. Vasic M, Kanade A, Maniatis P, et al. Neural program repair by jointly learning to localize and repair[J]. arXiv preprint arXiv:1904.01720, 2019.
14. 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.
15. <https://github.com/rjust/defects4j>
16. Martinez M, Durieux T, Sommerard R, et al. Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset[J]. Empirical Software Engineering, 2017, 22(4): 1936-1964.
17. Wen M, Chen J, Wu R, et al. Context-aware patch generation for better automated program repair[C]//2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 2018: 1-11.
18. Xiong Y, Wang J, Yan R, et al. Precise condition synthesis for program repair[C]//2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE, 2017: 416-426.
19. Xiong Y, Wang J, Yan R, et al. Precise condition synthesis for program repair[C]//2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE, 2017: 416-426.
20. Jiang J, Xiong Y, Zhang H, et al. Shaping program repair space with existing patches and similar code[C]//Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis. 2018: 298-309.
21. Vinyals O, Le Q. A neural conversational model[J]. arXiv preprint arXiv:1506.05869, 2015.
22. Klein G, Kim Y, Deng Y, et al. Opennmt: Open-source toolkit for neural machine translation[J]. arXiv preprint arXiv:1701.02810, 2017.
23. [https://bears-bugs.github.io/bears-benchmark](https://bears-bugs.github.io/bears-benchmark/#!/)
24. https://github.com/bugs-dot-jar/bugs-dot-jar
25. 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 (TOSEM), 2019, 28(4): 1-29.
26. Z. Chen, S. Kommrusch, M. Tufano, L. -N. Pouchet, D. Poshyvanyk and M. Monperrus, "SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair," in IEEE Transactions on Software Engineering, vol. 47, no. 9, pp. 1943-1959, 1 Sept. 2021, doi: 10.1109/TSE.2019.2940179.
27. Bengio Y, Frasconi P, Simard P. The problem of learning long-term dependencies in recurrent networks[C]//IEEE international conference on neural networks. IEEE, 1993: 1183-1188.
28. Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.