软件缺陷自动修复挑战赛初赛作品说明

队伍信息

队伍名称： CSTAR

成员1： 赵壮，武汉大学计算机学院， zhao2z@whu.edu.cn

成员2： 刘孝凡，武汉大学计算机学院，xfliu@whu.edu.cn

成员3： 丁正杰，武汉大学计算机学院，zhengjie.ding@whu.edu.cn

成员4： 薛州邑，武汉大学计算机学院，chouyihsueh@whu.edu.cn

初赛提交作品同步Push至Github：https://github.com/Zhao2z/ChinaSoft_RepairCamp_CSTAR

提交文件说明

> 在赛题发布后，参赛学生可自行组队。比赛将提供基准测试集。参赛队伍设计并实现修复工具，并适配所提供的基准测试集。作品提交时，参赛队伍应提交设计文档、执行录像、符合格式要求的补丁文件、执行日志、作品源码等内容。

根据初赛要求，本邮件的附件中包含：

1. 设计文档（本文档）
2. 执行录像.mp4 （包含3部分，缺陷定位+缺陷代码提取+输入模型获取Patch）
3. patch文件夹（补丁文件）
4. log.txt info.txt input.txt（执行日志）
5. src文件夹（作品源码）
6. model_repair 项目所用模型相关代码（训练数据集、模型pkl）

Patch文件夹组织结构说明：

Patch文件采用 `diff -u` 指令生成，针对每个 Project_id 生成一个目录，目录下对各个 bug_id 进行 patch文件的生成，在文件中会标注出修改的对应java代码文件的具体位置。

```
❱ tree ChinaSoft_RepairCamp_CSTAR/diff
ChinaSoft_RepairCamp_CSTAR/diff
├── AaltoXml
│   ├── 1.src.patch  : AaltoXml项目的第1号bug
│   ├── 2.src.patch
│   ├── 4.src.patch
│   ├── 5.src.patch
│   ├── 7.src.patch
│   ├── 8.src.patch
│   └── 9.src.patch
├── Assertj_assertions_generator
│   ├── 1.src.patch
```

```
13 │   ├──
14 │
```

```
// The content of AaltoXml/2.src.patch
--- /tmp/AaltoXml_2_buggy/src/main/java/com/fasterxml/aalto/out/WriterConfig.java 2024-09-09 21:32:45.160228522 +0800
+++ /home/zhengjie/Desktop/ccf_competition/merged/AaltoXml_2_buggy/src/main/java/com/fasterxml/aalto/out/WriterConfig.java  2024-09-10 15:35:28.299886011 +0800
@@ -382,7 +382,7 @@

     public boolean willEscapeCR() {
         // !!! TBI
-        return true;
+return false;
     }

     /*
```

技术说明文档

1. 缺陷定位

在本项目中，我们利用基于谱的缺陷定位工具Ochiai，结合Gzoltar平台，对GrowingBugs数据集中未指定子项目定位器的项目和bug编号进行系统性的缺陷定位。此过程中产生的关键数据文件包括：

- `line.ochiai.ranking.csv`
- `ochiai.ranking.csv`
- `statistics.csv`
- `matrix.txt`
- `sfl`
- `source_code_lines.txt`
- `spectra.csv`
- `tests.csv`

这些文件提供了关于代码文件中“可疑度”分析的详尽数据，所有分析结果均被妥善保存在 `/sflresults` 目录下。

在缺陷定位的深入分析阶段，我们首先对这些文件进行细致审查，从中提取出最可疑的Java文件、方法以及具体的bug所在行号。接下来，通过精确的正则表达式匹配技术，我们精确定位可疑方法的起始和结束行号，并从代码库中提取出相应的代码段。为了便于后续的自动修复阶段，我们在代码中插入了 `<BUGS>` 和 `<BUGE>` 标签。

> 值得一提的是，由于defects4j的版本兼容性问题，我们面临了找不到可直接应用于当前GrowingBugs项目的合适Gzoltar版本的挑战。将Ochiai成功集成到我们的项目中，耗费了团队大量的时间和精力（笑

2. 基于代码模型的缺陷自动修复

本阶段负责针对通过缺陷定位确定的可疑代码展开自动修复。我们将缺陷修复问题视作神经机器翻译问题的一种变形，其中源语言是存在缺陷的代码，目标语言是经过修复的正确代码。基于此框架，我们对预训练的代码模型进行针对性微调，从而修复确定的缺陷。

2.1 代码表示形式

微调模型的输入是缺陷定位阶段确定的可疑代码，这部分代码用 `<BUGS>` 和 `<BUGE>` 标记，并组织成单行形式。这些特殊标记帮助模型利用缺陷定位信息学习从缺陷代码到正确代码的转换。模型的输出是可疑代码的修复补丁，同样使用 `<FIXS>` 和 `<FIXE>` 来标记。

```
private... <BUGS> if (Patterns.WEB_URL.matcher(url).matches()) { <BUGE> webView...
<FIXS> if ((url != null) && (Patterns.WEB_URL.matcher(url).matches())) { <FIXE>
```

图2.1 微调模型输入、输出的代码表示形式示意

2.2 微调过程

我们采用Tufano等人收集的Bug-Fix数据集，对预训练的代码模型进行微调。此数据集选用其中的medium methods部分，共52364行代码，每行代表一个修复实例。

选择的预训练代码模型是UniXcoder，具体为unixcoder-base模型，具有125M的参数量。微调时的训练参数如表2.1所示。

Param	Value
learning_rate	5e-5
batch_size	8
beam_size	5
max_length	512
epochs	30

2.3 修复过程

我们针对缺陷定位阶段确定的可疑度最高的代码行所在函数（可疑函数）进行修复。对于函数中可疑度高于某一阈值的代码行，将其与上下文一同用 `<BUGS>` 和 `<BUGE>` 标记并组织成单行形式，作为微调模型的输入。使用模型输出的修复补丁替换原有的可疑代码行，从而完成修复。若一个函数中存在多个可疑代码行，将这些行的修复补丁综合以得到最终的修复结果。