LLM-based 软件软修

3.30
总结一下目前大家看论文的进度，把大概用的到的信息整理出来

AdaPatcher

prompt表示方法

- 假设有一个有错误的代码文件 c 和一个已修正的代码文件 y，生成diff文件
- 代码一致性指标：r/k;，k表示固定代码中的代码行总数，以及r表示修改后代码中保留的代码行数。（**可作为一个指标用来改善LLM的修复能力**）

DEVLoRe

有用的函数

- MethodRecorder：记录运行失败测试时调用的所有方法签名（但是还是需要我们拥有测试样例和test代码）
- DebugRecorder 获取定位方法内的详细**调试信息**(同上需要我们拥有测试样例和test代码)

prompt表示方法

- 也是使用diff格式的补丁
- 固定模版：General Task Prompt + Input Prompt + Expected Output Prompt
  - Input Prompt: {related_methods} {error_stack} {issue_content} {debugging_info}
  - Expected Output Prompt: Please localize/generate.....

FuseFL

定位技术

- SBFL：利用coverage进行覆盖率分析，然后根据代码的“可疑度”排序选择最可疑的线路作为故障代码

数据库/代码描述

- 模版

```
Faulty Code:
[Faulty Code]

Task Description:
[Task Description]

Test Results:
The provided code is producing incorrect results. For
examples:
  - Running the function as follows `[Input]` generate a
  `[Error name]` in line [i] `[Code content]`.
  - Running the function as follows `[Input]`
  yields`[Output]` instead of the expected `[Expected
  Output]`.

Spectrum-based Fault Localization (SBFL) Techniques
Results:
We have used SBFL techniques to identify potentially
faulty lines. Here are the results of top-[X] suspicious
lines sorted in descending order of the score:
  1. Line [i] `[Code content]`, [Technique name] score:
  [Score]
  2. Line [j] `[Code content]`, [Technique name] score:
  [Score]

Analyze the provided code and utilize task description,
test results, and SBFL techniques results to help in
identifying potentially faulty lines. Provide the results
following this JSON template:
```json
{
  "faultLoc": [
    {
      "faultyLine": (indicating the line number of the
      suspicious code),
      "code": (displaying the actual code),
      "explanation": (step by step reasoning on why this
      location is considered potentially faulty)
    },
    ...
  ]
}
```
Ensure that the objects in t
in descending order of suspi
```

错误代码（Faulty Code）：[ ]
任务描述（Task Description）：[ ]
测试结果（Test Results）：该代码产生了错误的结果。例如：
以下列方式运行函数 [输入] 时，在第 [i] 行发生了 [错误名称]：[代码内容]；
以下列方式运行函数 [输入]，输出为 [输出结果]，但期望的结果是 [期望输出]。
基于频谱的故障定位（SBFL）技术结果：我们使用 SBFL 技术来识别潜在的故障代码行。以下是根据得分降序排序的前 [X] 条可疑代码行结果：
第 [i] 行：[代码内容]，[技术名称] 得分：[得分]
第 [j] 行：[代码内容]，[技术名称] 得分：[得分]
请分析提供的代码、测试结果以及 SBFL 技术的结果，以帮助识别潜在的故障代码行。
请使用以下模板，以 JSON 格式提供结果：

```
{
"faultLoc": [
{
"faultyLine": [可疑代码的行号],
"code": [该行实际代码],
"explanation": [关于为什么该位置被认为是潜在错误的逐步推理说明]
},
...
]
}
```

- 代码描述库：https://github.com/githubhuyang/refactory

ChatRepair

迭代修复，可以考虑，但是先把非迭代的做好吧

修复场景

- 单行修复（优先级最高）
- 单块修复（优先级第二）
- 单函数修复（优先级第三）

prompt表示方法

- 测试名称
- 导致失败的相关代码行
- 产生的错误信息
- 正确情况下的预期输出和函数行为信息

Toggle

- Prompt 1：完整函数替换（不使用任何位置偏置信息）
  - 描述：给出完整的 buggy 函数，要求模型生成完整的修复函数。
- Prompt 2：提供 shared prefix，预测 truncated 修复函数
  - 描述：在输入中提供共享前缀（如函数定义等），buggy 函数中保留前缀部分，模型只需生成从 bug 起始处的修复代码。
- Prompt 3：去掉 shared prefix，引导模型从 bug 开始生成修复代码
  - 描述：输入中省略前缀，buggy 函数从 bug 开始；模型预测修复代码后再拼接前缀得到完整函数。
- Prompt 4：去掉 shared prefix 和 suffix，模型仅需预测 bug 的替换部分
  - 描述：输入中不包含前缀和后缀，只包含 buggy 部分；修复时将替换段生成后再拼接前后缀。

现在最重要的是探究可行性，相信大家看完了论文之后，大概知道了如何结合LLM去进行软件修复。
但是我们看到的基本都是集成好的一整套修复逻辑，而我们现在需要自己搞出一套会比较困难，直接沿用他们的就容易导致重复的工作，老师那边未必过得了关。
所以不妨先简陋些，至少我们需要一套能够自己跑起来的流程，然后我们再慢慢去完善。
我想了想，大概分以下几个阶段

1. 首先要确定好一个**prompt输入方式**，最好是可修改的（因为可能要进行横向对比/消融实验，探究不同组合下哪种prompt最好），如果将已有的buggy数据库转化为prompt输入是我们目前要攻克的第一个大问题，这一步我们**可以借用论文中的方法**（我们应该优先选不需要测试样例/test代码的emm，因为如果这样我们的选择就只能限制在某个数据库了；不过也都试一试吧）。

> 这里需要大家去真实地看一下他们是如何生成prompt的，最好能自己动手模仿他们生成一个自己写的代码的，看看工作量如何（如果工作量不大，我们就可以有自己的buggy代码了）。

2. 确定好prompt输入方式后，需要探究如何**把prompt给到大模型**，这一步也可以进行横向的探究，比如说分步给，迭代着给，一次性给，这一阶段时间充足的话可以写一个接口方便与大模型对接。
3. 当我们把错误信息给大模型之后，大模型会返回给我们一个修复方案，我们要思考**如何去评价/验证（如果可以的话）这个修复方案**，报告的结果应该也就出自于这里了。
4. 整合信息

大家的任务till——4.1周二下课
完成第一步，各自看看自己负责部分的prompt生成，课后讨论一下确定**prompt输入方式**即可。
（不用太急hhh，完不成也没关系，我们还有时间）

4.2至4.5周六晚上
最主要的任务：开题报告，确定内容
开题报告（3分钟）：组队、题目、内容、计划

- 不知道要不要做ppt？（我上课没听，不知道老师说了没用），不过做一个应该也不用很久，我来做一下吧
- 需要**确定好我们的内容和计划**，现有的想法就是对比不同的prompt输入方式，可能内容不够充实，所以还需要**想一些比较创新且有一定工作量的点**

我提一些可能的方向：

1. 用一些领域知识对大模型进行预训练是否会有明显成效？（比如先给它与该项目有关的几篇论文之类的）
2. 在评价体系中，加入对代码风格的评估，目的是为了让代码风格尽量保持不变
3. 现实情况中，往往一次迭代下不能彻底修复bug，所以需要**多轮迭代**，
4. 加入“自我反思机制”，每次LLM生成反馈的时候让它反思一下，是否存在问题，如果存在问题，是否需要进行修改。即它自己的“自我迭代”

欢迎大家思考并且写下来

关于prompt生成，目前保底估计就是用defects4j数据库，从库到prompt的脚本我已经写好了一版，ChatRepair的函数也基本上可以用来直接生成最终的prompt。我可以整合一下

这段时间我去调研一下怎么使用世航的那两个java函数，然后世航可以写一版你那里的生成prompt的脚本，康悦那边丢掉自适应什么的内容，没有源码的话去看看FuseFL里的SBFL（计算故障代码中每一行的可疑分数），因为Defects4j里也有测试样例，如果这两个可以接起来的话，把SBFL的可疑性结果作为prompt的一部分很有可能会提高修复效果，这个应该是可以算比较创新的点（）