南京大學

基于迁移学习的人工智能框架缺陷修复技术探索

姓名 孙立帆　章华鹏　王子搏　　学号 201250181　201250183　201250188　　院系 软件学院

摘　要

近年来，随着人工智能技术在自动驾驶、图像识别等领域不断获得突破性的成果，研究人员对智能软件安全的重视程度也逐步提高。为了降低智能软件开发者的开发难度，更好地完成人工智能任务，推动人工智能发展，人工智能框架不断涌现，比如 Tensorflow、Pytorch 、Caffe 等。由于大量的智能软件使用这些框架，框架中的缺陷可能会带来严重的后果，因此如何快速地、自动化地修复这些框架中的缺陷显得十分重要。尽管目前已经存在一些较为成熟的基于深度学习的程序修复技术，但是由于人工智能框架的特殊性 (主要是: 数据量少, 难以从头训练一个成熟的模型; 往往涉及到多种语言, 而现有的技术往往针对单个语言) 无法直接应用。

针对以上提及的两个难点, 本项目对迁移学习 (transfer learning) 和提示学习 (prompt learning) 在自动程序修复上的应用做了探索, 使用 T5 架构的预训练模型在框架数据集上进行微调, 并结合提示学习技术指示编程语言信息, 构建了一个针对人工智能框架的自动程序修复模型 AutoFix。

目录

1 引言

随着人工智能技术，主要是深度学习技术，在各领域的应用不断取得突破性进展 (如计算机视觉、自动驾驶和自然语言处理等), 对智能软件安全的重视也不断提高。目前的智能软件大多都是使用人工智能框架 (如 TensorFlow、Pytorch、Caffe 等) 提供的编程接口来进行编程, 这些框架中的缺陷对于人们使用的智能软件的影响是巨大的。因此, 快速地、自动化地对人工智能框架中的缺陷进行修复的意义是重大的。尽管目前已经存在一些比较成熟的基于深度学习的自动程序修复技术，但是由于人工智能框架的特殊性，这些已有的修复技术往往难以直接运用或者效果不佳，首先，(1) 人工智能框架数据量较少，而通用缺陷修复技术往往需要大量语料训练一个成熟的模型; 同时，(2) 现有通用缺陷修复技术往往针对单个语言，而人工智能框架往往涉及多个语言 (比如 C++ 和 Python)。

为了解决第一个问题，我们利用了迁移学习 (transfer learning) 技术，在预训练模型上进行微调。此外，受 NLP 文本增强技术的启发，我们设计了若干针对于人工智能框架的数据扩增方法，一定程度上也缓解了数据不足的问题。

为了解决第二个问题, 我们利用了提示学习技术 (prompt learning), 设计了一组提示模板，来为模型指示编程语言相关的信息，以增强模型跨语言的能力。此外，我们构建了不同编程语言的关键字 (比如 C++ 的 nullptr 和 Python 的 None) 之间的映射表，以进一步修正跨语言带来的误差。

我们先在当前 SOTA 的基于深度学习的通用缺陷修复模型上使用框架数据测试集进行了实验，以验证当前的基于深度学习的通用缺陷修复技术在人工智能框架的修复上无法直接应用，接着在本项目构建的模型上使用同样的测试集进行测试，以验证我们的方法的有效性。实验的结果表明: (1) 目前 SOTA 的通用缺陷修复模型在人工智能框架的修复上表现不佳，无法直接应用; (2) AutoFix 在人工智能框架缺陷修复方面展现出良好的能力, 并且 prompt 机制在其中展现出了可观的作用。

总结来说，本项目主要完成了如下工作:

- 人工对原始的数据集进行了筛选，并通过我们设计的数据扩增技术对筛选出的数据集进行了扩增，初步构建了一个可用的数据集。
- 对现有的 SOTA 的基于深度学习的程序修复模型在人工智能框架缺陷修复的能力上进行了评估。
- 提出了一种基于迁移学习和提示学习的针对人工智能框架的缺陷修复技术 AutoFix, 并评估了该方法的有效性。

本文的剩余部分按照如下方式组织: 第二节介绍背景信息和相关的研究工作, 第三节介绍我们提出的方法, 第四节介绍使用的数据集和评价指标，第五节是实验结果和分析, 第六节是总结内容。

2 背景和相关工作

2.1 基于深度学习的自动程序修复

在自动程序修复领域, 目前基于深度学习的方法取得了 SOTA 的效果 [1]。大部分方法都将程序修复问题作为一个翻译任务 (Neural Machine Translation), 使用 encoder-decoder 架构的模型, 通过监督学习的方式在缺陷-修复对上进行学习，尝试学习缺陷修复的隐模式。研究者提出了许多基于深度学习的程序修复模型, 这些模型使用了不同的输入形式和模型架构, 比如 CoCoNut 使用卷积神经网络分别编码缺陷代码和上下文 [2]。近年来，预训练模型也被用于自动程序修复，比如 CodeT5[4] 和 CodeBert[3].

2.2 预训练模型

对于参数规模非常巨大的模型，从头开始使用监督学习进行训练得到一个可用的模型需要的数据量是巨大的。因此人们提出了自监督学习的方法，设计一系列特定的自监督学习任务，使用大量无标注的数据进行训练，得到预训练模型。预训练模型已经广泛用于 NLP、CV 等领域，近年来，也有研究者提出了在编程语言语料上训练的预训练模型，比如 CodeT5, CodeBert 等。

2.3 迁移学习

传统的机器学习方法使用某个领域的数据进行学习，之后在同一个领域上继续发挥作用。但是对于人类来说，我们可以做的更好，人类可以应用在相似的领域学到的知识在一个新的领域解决新的任务。这种方法使得我们面对一个新的领域时可以利用从别的领域习得的知识，而不需要从新开始学习。迁移学习就是应用了这一观点，在已经训练了一定程度的模型上使用新领域的数据进行微调，来适应新领域的任务。具体来说，在本文提出的方法中，我们使用了 CodeT5 预训练模型，在通用缺陷数据集上先进行了微调，接着在此基础上在人工智能框架数据集上进行了进一步的微调。

2.4 提示学习

提示学习是 NLP 领域中近年来开始流行的一种新范式, 它通过把下游任务转化为和预训练任务相似或相同的形式, 弥合下游任务和预训练任务之间的鸿沟, 配合预训练模型使用在某些任务上可以达到 few shot learning 甚至 zero shot 的效果 [5]。一个提示是一小段插入到输入中的词元，以将原任务表示为预训练任务的形式。提示学习是用来填补从预训练任务到下游任务之间的差距，促进微调的过程。在本项目中我们人工设计了一些前缀作为提示拼接到每个输入前面。

3 方法

为了解决第一节中提到的问题和挑战, 我们提出了 AutoFix, 一个针对深度学习框架缺陷的程序修复模型。在本节中我们将对 AutoFix 的设计细节做更详细的介绍。首先，在 3.1 节中我们对整个系统的架构做概述。AutoFix 使用了基于 T5 的模型架构 (3.3 节)，通过提示学习来促进微调，3.2 节介绍了 Prompt-Based 的输入表示, 3.4 节介绍了我们在模型的推理阶段使用的针对跨语言的再修复方法。

3.1 概述

图 1 和图 2 展示了 AutoFix 的系统架构。AutoFix 包括两个阶段，训练和推理。在训练阶段，首先使用预先设计好的 prompt 函数来将原始输入转化为 fill-in-the-blank 的任务的形式，接着把 prompt 之后的数据送入 T5 Base 的程序修复模型中。为了解决 OOV 问题，AutoFix 使用了 subword tokenization. 接着程序修复模型生成一系列候选的修复版本，并计算损失函数，通过最小化优化目标来更新模型。在推理阶段，将输入的待修复程序片段经过 prompt function 处理之后送入程序修复模型，得到输出之后经过我们设计的针对跨语言的 re-repair 方法进行精化，得到最后的 candidate patches, 具体来说，我们通过建立了一张映射表来建立不同语言的语言元素之间的映射，详细细节将在 3.4 节介绍。

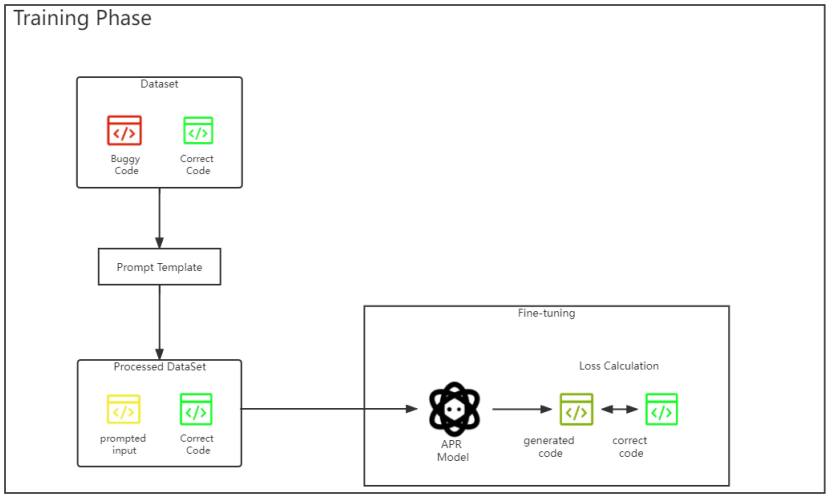

图 1: 系统架构: 训练阶段

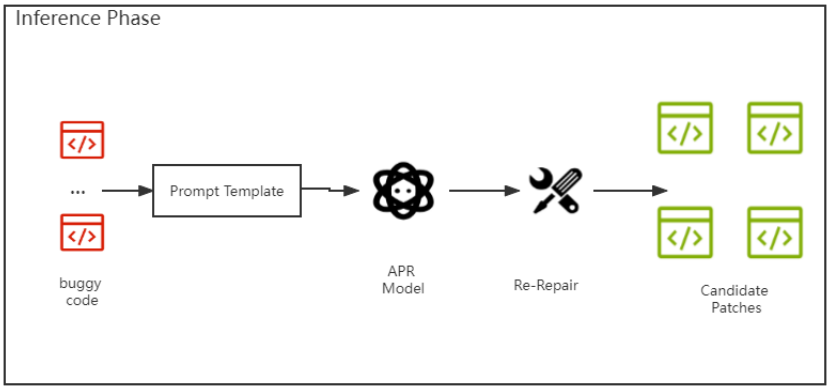

图 2: 系统架构: 推理阶段

3.2 Prompt-Based 的输入表示

由于 CodeT5 的预训练任务中包含对注释信息的理解，我们以注释的形式来把 prompt 拼接到输入上。具体来说，如果用 $b = (w_1, w_2, ..., w_n)$ 来表示 buggy code，用 $f = (z_1, z_2, ..., z_m)$ 来表示 fixed code，对于 C++ 语言的输入，我们使用如下的 prompt: p_1 = // this is a buggy cpp program, p_2 = // below is fixed cpp program; 对于 Python 语言的输入，我们使用如下的 prompt: p_1 = # this is a buggy python program, p_2 = # below is fixed python program 。prompt 之后得到的输入 $b' = (p_1, w_1, w_2, ..., w_n, p_2)$，图 3 是一个具体的示例。

3.3 模型架构

T5 是一种 encoder-decoder 架构的 transformer 模型，在巨大的 (超过 750 GB) 数据集上进行了预训练，在多个 NLP 的任务上取得了 SOTA 的效果。Yue Wang et al. 基于 T5，设计了新的预训练任务，并使用大量 PL 的语料进行了预训练，发布了 CodeT5 模型，在 code summarization, defect detection, code refinement 等 PL 相关的下游任务上取得了 SOTA 的效果。

T5 的 encoder 是一系列 transformer blocks，每个 transformer block 都包含以下的组件: 一个

buggy code

```
int add(int a, int b){
    return a - b;
}
```

prompted input

```
// this is a buggy cpp program
int add(int a, int b){
    return a - b;
}
// below is fixed cpp program
```

图 3: prompt 输入

multi-head self-attention，跟着一个 position-wise feed-forward network. 在以上的每个组件之间还加入了 Layer Normalizarion, residual connectors 和 dropout layer 来提升训练的稳定性。

T5 的 decoder 和 encoder 的结构也类似，不同之处是它应用了 Masked Multi-Head Attention 层。

我们使用了 codet5-base 来作为我们初始微调的起点，在通用程序修复数据集上微调先得到了通用程序修复模型，作为我们框架特定的修复模型的训练起点。

3.4 跨语言的再修复

由于模型需要针对不同语言 (在本项目中是 python 和 C++) 进行修复, 为了进一步提升修复的效果，在测试阶段我们进行了针对跨语言的再修复。具体来说我们构建了一张不同语言之间关键字的映射表，对不属于当前语言的关键字进行替换，以进行修正。图 4 是一个具体的映射表的例子。

C++	Python
NULL	None
nullptr	None
false	False
true	True
...	...

图 4: 关键字映射表

4 实验准备

在本小节中我们将介绍关于实验的细节，包括研究问题，数据集，实验设计和基线模型。

4.1 研究问题

AutoFix 是为了解决跨语言的人工智能框架中的缺陷修复问题而提出的，因此，我们提出了如下的研究问题:

- 研究问题 1: 我们的方法的跨语言修复能力如何?
- 研究问题 2: 各组件在我们的方法中的贡献如何?

4.2 数据集

我们首先在通用缺陷修复数据集上训练了通用自动程序修复模型，使用的是 CodeT5 论文中提到的由 Michele Tufano et al. 发布的 Java 语言的通用程序修复数据集 [6], 数据处理的流程如图 5 所示。

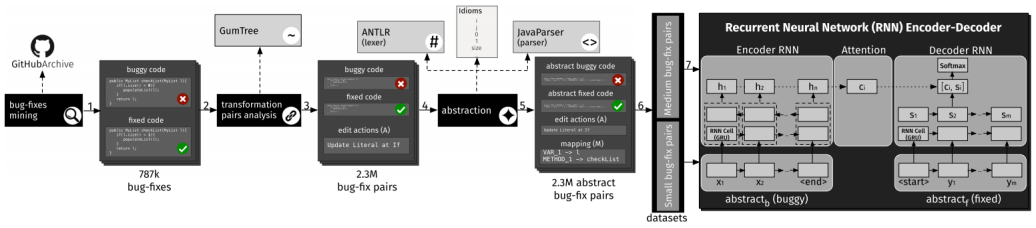

图 5: 通用缺陷数据集处理流程

图 6 是通用缺陷数据集中的 Bug-Fix Pair 按照 Token 长度的分布情况，我们选取了其中的 medium size 部分 (长度在 50 - 100 个 tokens, 有 65k 个 BugFix Pair) 来进行通用程序修复模型的微调。

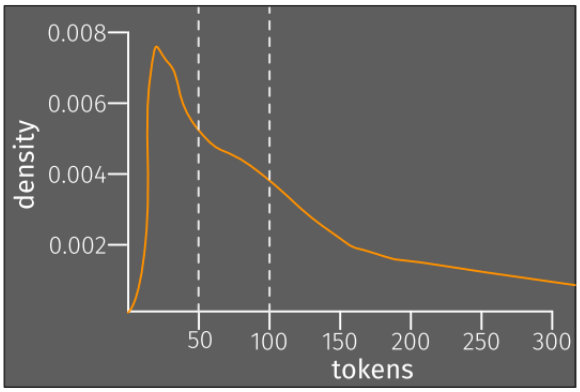

图 6: 通用缺陷数据集: token 长度分布

由于人工智能框架的缺陷修复是一个前沿的研究领域，目前并没有成熟的数据集，因此我们对课程提供的初始数据集进行了处理，得到一个可用性更高的数据集。具体的处理流程如图 7 所示:

训练了通用程序修复模型之后我们在通用修复模型的基础上使用框架数据集进行微调。框架数据集是从各大深度学习框架的 GitHub 仓库中爬取，包含 Python 和 C++ 两种语言。由于初始的数据集中的代码片段非常残破，且质量参差不齐，我们首先人工对其中可用的数据进行了挑选，我们挑选的标准如下:

- 只涉及注释改动的数据行应该丢弃。

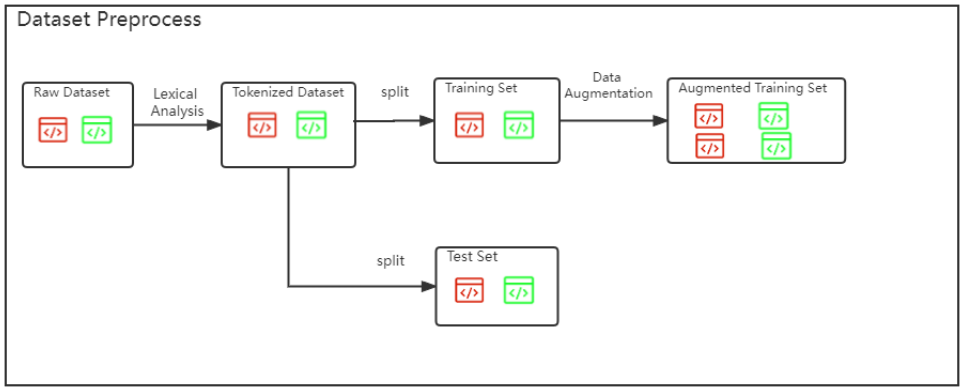

图 7: 框架缺陷数据集处理流程

- 大量改动的数据行应该丢弃。
- 非常明显的语义等价的改动的数据行应该丢弃，一个例子是在 python 语言程序中把双引号字符串改为单引号字符串。
- 截取缺陷代码片段周围带有 1 - 2 行上下文片段，不宜过长，60 个 token 以内最好。
- 不是改动程序代码的数据行应该丢弃，这是因为初始数据集中有相当数量涉及对 shell 脚本的更改。
- 无上下文的残破片段应该丢弃。
- 无改动应该丢弃，即 buggy code 和 fixed code 之间只有空白符的区别。

数据筛选过程由三个具有一定 Python 和 C++ 基础的软件工程专业大三学生进行，按照上文提及的协议进行数据筛选，一共筛选了 12000 条数据，得到 600 条有效数据。由于初始得到的每条数据都是原始的代码文本形式，没有进行 tokenization，我们使用 Antlr 构建了一个简易的 C++ 和 Python 的词法分析器，并对数据集中的代码文本进行了 tokenization。我们对处理后的数据集进行了分析，得出了框架缺陷数据集中 BugFix Pair 在 token 长度上的分布，如图 8 和图 9 所示，95% 以上的 BugFix Pair 都在 80 个 token 以下，78% 以上的 BugFix Pair 都在 60 个 token 以下。

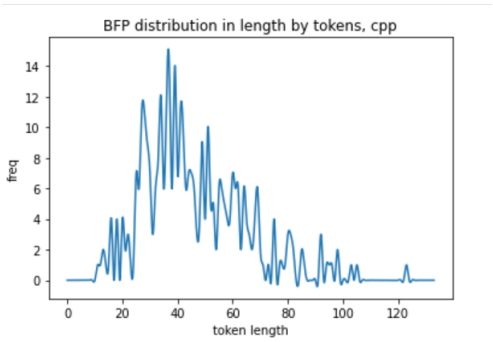

图 8: BFP Distribution: cpp

接着我们按照 7：1：2 左右的比例划分了训练集，验证集和测试集。由于数据集规模过小，我们对训练集进行了数据增强。经过人工对数据的浏览，我们发现了一系列缺陷的模式，并据此设计

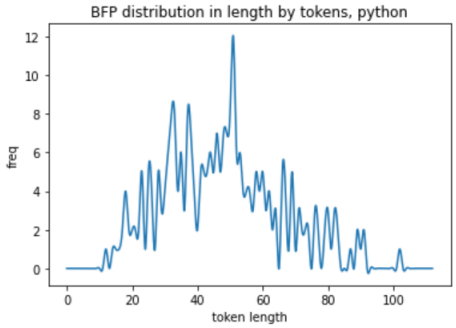

图 9: BFP Distribution: py

了相应的简单的数据增强规则，具体可参见我们仓库中关于数据增强部分的 notebook。最后得到的训练集，验证集，测试集大小分别是: 1.3k, 0.05k, 0.1k.

4.3 基线

为了验证 AutoFix 在修复人工智能框架缺陷方面的优势，我们选择了近几年达到 SOTA 效果的通用缺陷修复模型作为 baseline. 由于算力资源和时间的紧张，我们仅在 CodeT5 和 CodeBert 上进行了实验，使用作者发布的在通用程序修复任务上微调的 checkpoints 在我们的框架缺陷测试集上进行实验，以评估通用缺陷修复模型在框架缺陷修复上的效果。

4.4 实验过程及细节

我们的模型以及其他工具代码都是基于 Pytorch 编写的。在通用修复模型训练阶段，我们使用 huggingface 上发布的 codet5-base 模型作为训练的起点。训练过程使用 AdamW 优化器，使用 5e-5 的学习率，batch size 为 32，最大 buggy token 长度和 fixed token 长度为 240，训练最多 50 个 epoch，采用 early stop, 5 个 epoch bleu 不提升则停止. 在框架缺陷修复模型训练阶段，我们以训练好的通用缺陷修复模型的 best-bleu checkpoint 为训练起点，使用 AdamW 优化器，5e -5 学习率，weight decay 为 1e-3. batch size 为 16, 最大 buggy token 和 fixed token 长度为 240，训练最多 25 个 epoch，采用 early stop, 3 个 epoch bleu 不提升则停止.

在推理阶段，我们使用我们在 3.4 节提到的方法对模型的输出进行了 re-repair. 为了评估 AutoFix 的效果，我们使用 exact match 和 bleu-4 作为 metrics. 以上的训练和评估过程都在一台装有 Ubuntu 操作系统，带有一张 RTX 3090 GPU 的服务器上完成。

5 实验结果和分析

在本小节我们对实验的结果进行分析，并基于实验结果尝试回答 4.1 节中提出的研究问题。

5.1 研究问题 1: 我们的方法的跨语言修复能力如何?

由于人工智能框架往往涉及多种语言 (至少有 Python 和 C++), 我们希望提高 AutoFix 的跨语言修复能力。为此，我们使用了 prompt 来为模型指示编程语言信息，并在推理阶段加上了基于映射表的 re-repair，这是 AutoFix 和其他通用程序程序修复模型 (本文中指直接在 CodeT5 和 CodeBert 上微调得到的模型, 以下用 CodeT5 APR 和 CodeBert APR 指代) 的不同之处。我们使用 AutoFix 和其他通用程序修复模型在测试集上进行了多次评估，在 0.1k 个 BugFix Pairs 中, CodeT5 APR 和

CodeBert 平均修复的 exact match 都为 0 , 而 bleu-4 score 平均分别为 35.04 和 4.05, 而 AutoFix 的修复方案 exact match 平均达到了 23 左右，bleu-4 达到 82.5 左右。

我们对每个模型的预测输出进行了检查，发现在 CodeT5 APR 和 CodeBert APR 中, 由于存在着大量源于它们的训练集的语言的语法特征，导致即使能够生成出语义相近的代码，但是无法准确匹配，因此 exact match 都为 0, 其中 CodeBert APR 在 bleu-4 分数上也远低于其他两者，我们推测这是因为 CodeBert 是在长度较短的输入序列上进行微调，而测试集的的输入相比于它训练时的输入序列要长得多，导致表现不佳。

综上, 和目前现有的通用缺陷修复模型相比, AutoFix 展现出了可观的跨语言修复能力。

5.2 研究问题 2: 各组件在我们的方法中的贡献如何?

我们主要对 prompt 和 re-repair 机制的作用进行了探究。我们分别在使用 prompt 和不使用 prompt, 使用 re-repair 和不使用 re-repair 的四种组合设置上进行了实验, 每种设置进行了 5 轮, 取了其中比较有代表性的绘制了如下的 validation bleu + exact match 随 epoch 变化的趋势图, 如图 10 所示。

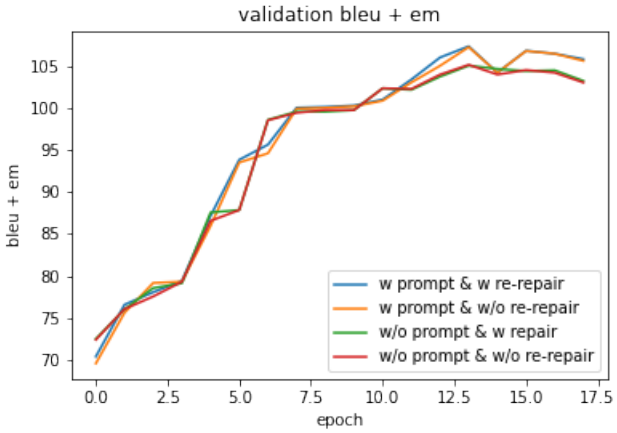

图 10: validation bleu + em 随着 epoch 变化的情况

可以看出总体来说 re-repair 机制并没有起明显的作用, 但是 prompt 使得模型的 validation bleu + em 达到了更高的水平, 一定程度上说明 prompt 对提高模型对框架缺陷的修复能力有促进作用。

em / bleu	with prompt	without prompt
with re-repair	em: 23.53, bleu: 83.21	em: 21.95, bleu: 83.23
without re-repair	em: 23.32, bleu: 82.98	em: 22.01, bleu: 82.99

图 11: 实验结果

图 11 的表格是在测试集上我们在以上四组实验设置上进行的实验得到的平均结果, 进一步佐证了 prompt 的作用。

6 总结

在本文, 我们提出了 AutoFix, 一个针对人工智能框架的缺陷修复模型。具体来说, 这个模型主要有以下三个组件, 一个 prompt-based 的数据表示, 一个基于 CodeT5 的 APR 模型, 一个基于映射表的 re-repair 组件。prompt-based 的数据表示缩小了程序修复这个下游任务和 CodeT5 的预训练任务之间的差距, re-repair 组件使用简单的基于映射表的机制对 APR 模型的输出做修正。

我们进行了一系列的实验来验证 AutoFix 的有效性和其中各组件的作用。实验结果表明, AutoFix 相比于在单语言上微调的通用程序修复模型展现出了良好的跨语言修复能力, 在模型的各组件中 prompt 机制起了重要的作用。

但是由于算力资源和时间的紧张, 尽管我们通过进行多次实验观察平均结果来尽力减少随机性带来的误差, 由于可用的数据集规模太小, 文中得出的结论正确性仍有可能受到威胁, 若要得到更加严密的结论还需要更大规模的数据集和精细的实验设计。

参考文献

[1] Wenkang Zhong, Chuanyi Li, Jidong Ge, and Bin Luo. 2022. Neural Program Repair : Systems, Challenges and Solutions. In Proceedings of the 13th Asia-Pacific Symposium on Internetware (Internetware '22). Association for Computing Machinery, New York, NY, USA, 96–106. https://doi.org/10.1145/3545258.3545268

[2] Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. 2020. CoCoNuT: combining context-aware neural translation models using ensemble for program repair. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2020). Association for Computing Machinery, New York, NY, USA, 101–114. https://doi.org/10.1145/3395363.3397369

[3] Feng Zhangyin , Guo Daya , Tang Duyu , Duan Nan, Feng Xiaocheng, Gong Ming, Shou Linjun, Qin Bing, Liu Ting, Jiang Daxin, Zhou Ming. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. 1536-1547. 10.18653/v1/2020.findings-emnlp.139.

[4] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding andgeneration. arXiv preprint arXiv:2109.00859 (2021).

[5] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2022. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ACM Comput. Surv. Just Accepted (September 2022). https://doi.org/10.1145/3560815

[6] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2019. An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. ACM Trans. Softw. Eng. Methodol. 28, 4, Article 19 (October 2019), 29 pages. https://doi.org/10.1145/3340544