Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

自动化程序修复及其应用研究

博士研究生 张钊

2024年08月11日

问题回溯

- 总结反思：
  - 学术报告要讲解深入，注意整体架构、方法细节和关键技术的讲解
  - 要声音洪亮清晰，提高现场沟通互动频次

- 相关内容：
  - 于浩淼《软件缺陷自动修复方法》
  - 张凌浩《代码异味检测》
  - 孔令迪《源代码漏洞检测》
  - 谢宁《软件漏洞检测及其严重性评估》

内容提要

- 预期收获
- 题目内涵解析
- 研究背景与意义
- 研究历史与现状
- 知识基础
- 算法原理
  - TENURE
  - CAP-Gen
- 特点总结与工作展望
- 参考文献

预期收获

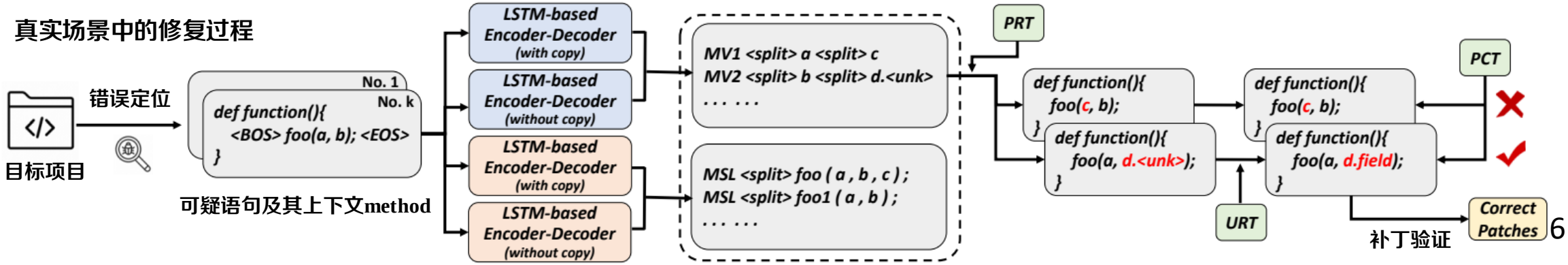
- 预期收获
  - 1. 了解程序缺陷修复的背景和基本概念
  - 2. 熟悉自动化程序修复的历史现状及应用场景
  - 3. 理解自动化程序修复方法的应用
  - 4. 明确自动化程序修复的发展趋势和未来前景

内涵解析与研究目标

- 研究目标
  - 利用自动化技术提高缺陷修复的效率、质量和覆盖范围，增强修复过程的可解释性和验证能力，提升软件的质量和开发效率，减少人工干预，降低软件维护成本
- 内涵解析
  - 软件缺陷：软件系统中存在的导致程序行为不符合预期的缺陷或问题。软件缺陷影响软件的功能、性能或用户体验，可能导致系统崩溃、错误输出或安全漏洞
  - 软件缺陷修复：传统的软件缺陷修复通常由开发人员手动识别、定位和修复缺陷
  - 自动化缺陷修复（APR）：通过工具和算法自动识别、生成和应用修复补丁来修复软件缺陷，修复过程尽可能减少人工干预

背景意义

- 研究背景
  - 缺陷普遍性和复杂性：软件系统复杂性增加，缺陷无所不在导致软件功能失效、性能问题、安全漏洞等，涉及多个模块或组件时，缺陷难以定位与修复
  - 人工缺陷修复局限性：人工修复缺陷耗时且劳动密集，增加软件维护成本，开发人员需仔细分析缺陷原因，并编写修复代码，会延迟软件的发布或更新
- 研究意义
  - 加速开发周期：缩短缺陷从发现到修复的时间，加快软件开发和发布周期
  - 减少重复劳动与节省资源：可以重复执行修复任务，避免人工修复的重复劳动和低效操作，减少人工修复时间和成本，实现更高的资源利用效率



研究历史与现状 自动化程序修复

Jiang等人结合预训练模型和神经机器翻译模型学习代码语法和修复模式，利用有效标识符检查和代码感知束搜索策略生成正确修复。

Ye等人集合交叉熵和程序编译、测试执行信息以提高生成补丁的质量，并提取缺陷类的摘要以丰富缺陷上下文中语义信息。

Meng等人结合基于模板和神经机器翻译构建模型，利用编码器-解码器模型学习深层语义特征，以生成不同模板的补丁中间表示。

Ye等人将故障定位、补丁生成和补丁验证集成在一个迭代循环，改进之前修复尝试中生成的部分补丁，以收敛到最终的正确补丁。

2021 2022 2023 2024

2021

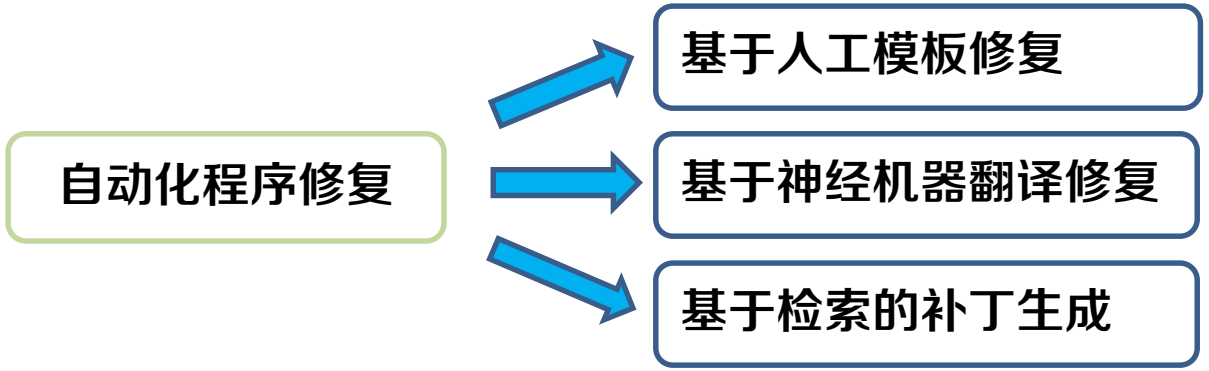
Zhu等人利用语法引导的解码器生成长度较短的编辑指令而非直接生成修复代码，支持生成占位符并将其实例化为项目特定标识符

2022

Xia等人直接利用大型预训练代码模型在零样本学习环境下进行修复，通过上下文信息（即完形填空或文本填充任务）直接预测正确代码。

2023

Wang等人构建混合补丁检索器并基于原始源代码进行词汇和语义匹配，实现不依赖特定功能和语言无关的缺陷代码补丁生成。



知识基础 补丁

- 补丁（patch）
  - 定义：指对源代码进行修改以修复缺陷或错误的代码片段，包括代码变更、修复描述
  - 代码变更：对源代码具体修改，如增加新代码行、删除错误代码行、或修改现有代码
  - 修复描述：通常包含关于补丁目的和修复方法的描述，帮助理解补丁的作用。

删除

增加

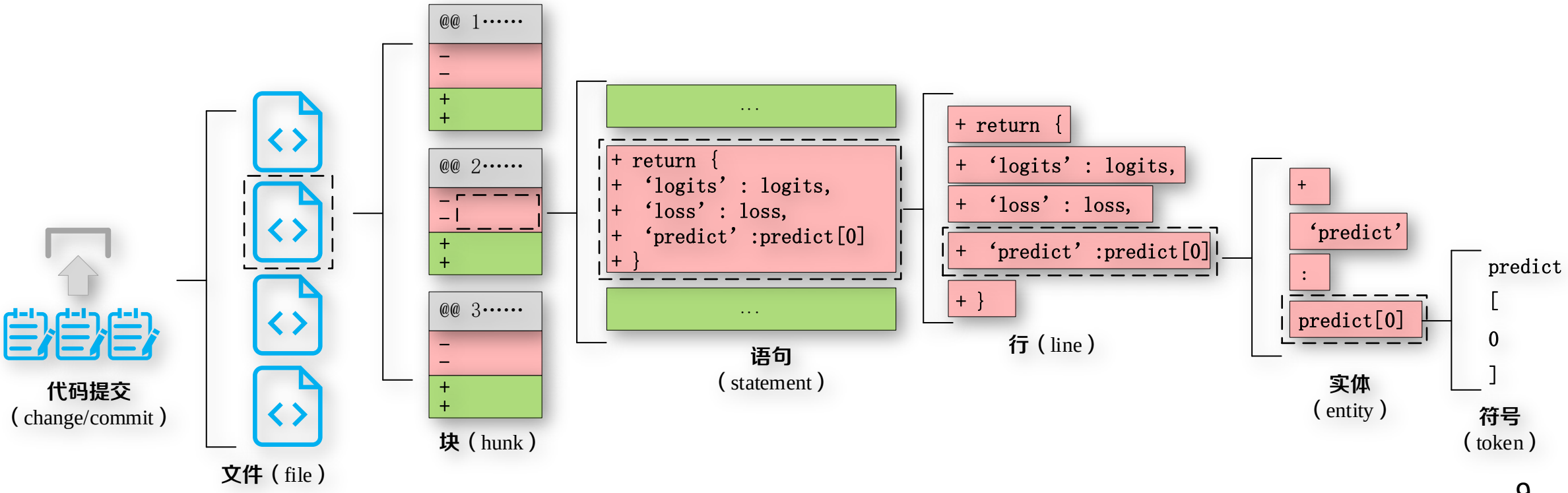
```
@@ -421,7 +421,7 @@ public class TestFormAuthenticator
extends TomcatBaseTest {
-   private class FormAuthClientBase extends
SimpleHttpClient {
+   private abstract class FormAuthClientBase extends
SimpleHttpClient {
        protected static final String LOGIN_PARAM_TAG =
        "action=";
        protected static final String LOGIN_RESOURCE =
        "j_security_check";
```

提交信息

Commit Message:
Make base class **abstract**

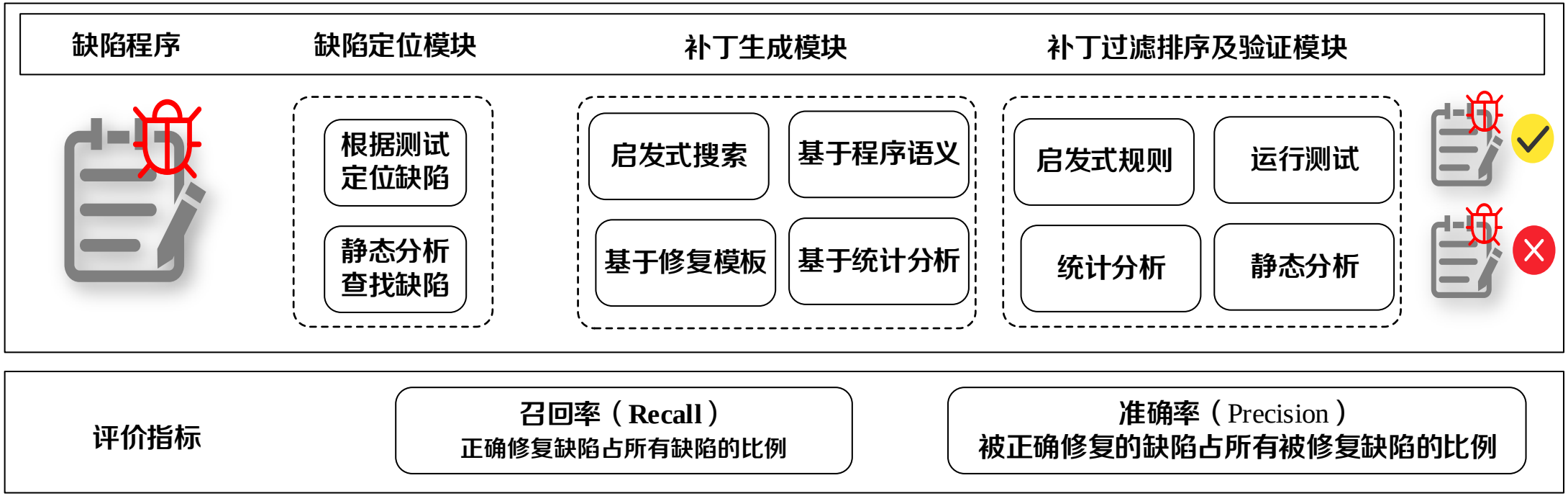
知识基础 代码变更粒度

- 代码变更粒度
  - 代码的结构性使代码变更也具有层次性
  - 代码变更粒度按照从大到小包括: 代码提交(commit/change)、文件 (file)、块 (hunk)、语句 (statement)、行 (line)、实体 (entity) 和符号 (token)



知识基础　缺陷修复过程

- 缺陷修复基本流程
  - 缺陷定位：通过静态分析、动态测试等技术手段确定程序中可能的缺陷代码位置
  - 补丁生成：每次选择一个候选出错位置，使用补丁生成技术尝试生成修复补丁
  - 补丁排序过滤以及验证：验证生成的修复补丁的正确性



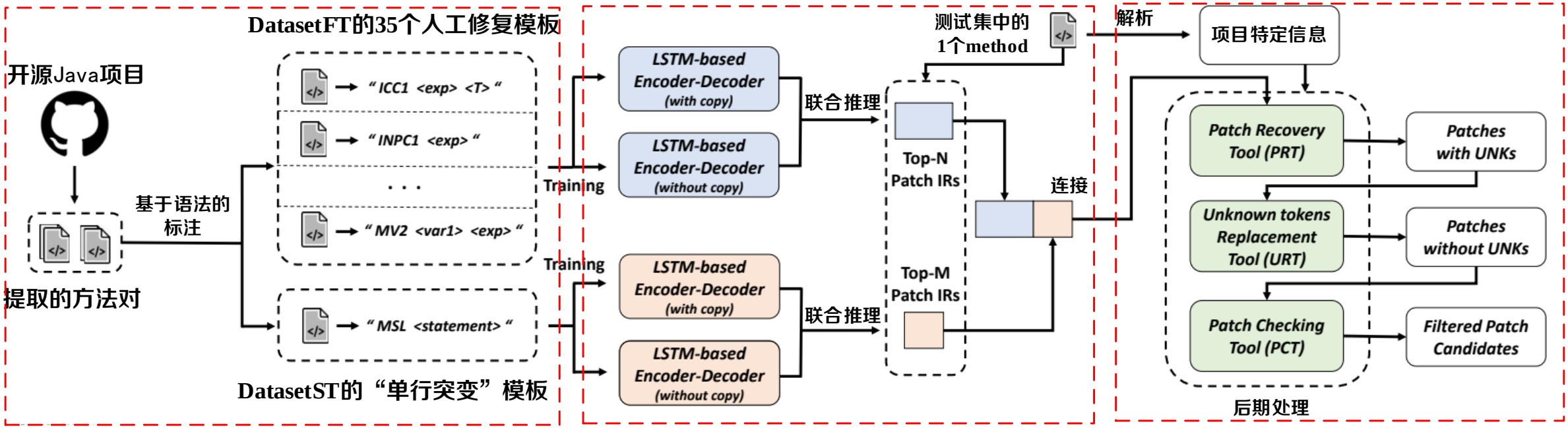
TENURE

TENURE TIPO

T	目标	生成准确的修复补丁
I	输入	含缺陷的项目源码
P	处理	1. 构建2个大规模数据集，分别对应含35个基于模板的修复模板和1个基于神经机器翻译（NMT）的特殊修复模板 2. 利用编码器-解码器模型学习深层语义特征，生成不同模板的补丁中间表示（IRs），通过优化的复制机制缓解Out-of-Vocabulary问题 3. 基于不同模板的组合补丁中间表示，开发3种工具从IRs中恢复真实补丁，替换未知Token，利用项目特定信息过滤编译错误的候选补丁
O	输出	正确的缺陷补丁

P	问题	现有方法利用语法合成补丁，未兼顾语义信息的充分利用，导致补丁范围小（例如仅修复单行代码）且不准确
C	条件	从原始代码中构建人工修复模板和代码语句数据集
D	难点	如何更准确的从补丁中间表示恢复真实补丁
L	水平	ICSE2023 （CCF-A）

- 基本框架
  - 结合35种人工模板和1种单行代码模板构建2个修复数据集（DatasetFT和DatasetST）
  - 利用编码器-解码器模型学习缺陷method深层语义特征，生成补丁中间表示（IRs）
  - 利用补丁恢复（PRT）、未知令牌替换（URT）和补丁检查（PCT）生成真实补丁



- 基于修复模板构建数据集
  - 补丁中间表示（IRs）：由格式化模板和填充的程序元素构成。例如，变量a和b替换格式化模板“MV1 var1 var2”中的“var1”和“var2”，生成相应IRs“MV1 a b”
  - 样本：含缺陷的method及其相应补丁的IRs
  - 数据集：DatasetFT（人工修复模板*35，356629个样本）

    DatasetST（MSL修复模板*1，223599个样本）

修复模板 代码变更操作

No.	Fix Templates	Code Change Actions
1	Insert Null Pointer Checker 1 插入空指针检查器1	+ if (exp != null) { ... exp ... ; + }
2	Move Statement 1 移动语句 1	- statement ; ... + statement ;
3	Remove Buggy Statement 1 删除有缺陷的语句 1	... - statement ; ...
4	Mutate Variable 1 突变变量 1	- ... var_1 ... ; + ... var_2 ... ;

修复模板 格式化模板中间表示

Fix Template	Formatted Template Rep.	Fix Template	Formatted Template Rep.
1. Insert Cast Checker 1	ICC1 ⟨exp⟩ ⟨T⟩	19. Mutate Data Type 1	MDT1 ⟨T1⟩ ⟨T2⟩
2. Insert Null Pointer Checker 1	INPC1 ⟨exp⟩	20. Mutate Data Type 2	MDT2 ⟨T1⟩ ⟨T2⟩
3. Insert Null Pointer Checker 2	INPC2 ⟨exp⟩ ⟨default⟩	21. Mutate Integer Division Op. 1	MIDO1 ⟨divisor⟩
4. Insert Null Pointer Checker 3	INPC3 ⟨exp⟩ ⟨exp1⟩	22. Mutate Integer Division Op. 2	MIDO2 ⟨dividend⟩
5. Insert Null Pointer Checker 4	INPC4 ⟨exp⟩	23. Mutate Integer Division Op. 3	MIDO3 ⟨dividend⟩ ⟨divisor⟩
6. Insert Null Pointer Checker 5	INPC5 ⟨exp⟩	24. Mutate Literal Expression 1	MLE1 ⟨literal1⟩ ⟨literal2⟩
7. Insert Range Checker 1	IRC1 ⟨exp⟩ ⟨index⟩	25. Mutate Literal Expression 2	MLE2 ⟨literal1⟩ ⟨exp⟩
8. Insert Range Checker 2	IRC2 ⟨exp⟩ ⟨index⟩	26. Mutate Method Invocation Exp. 1	MMIE1 ⟨method1⟩ ⟨method2⟩
9. Insert Missed Statement 1	IMS1 ⟨expression_statement⟩	27. Mutate Method Invocation Exp. 2	MMIE2 ⟨buggy_args⟩ ⟨patch_args⟩
10. Insert Missed Statement 2	IMS2 ⟨default⟩	28. Mutate Method Invocation Exp. 3	MMIE3 ⟨deleted_args⟩
11. Insert Missed Statement 3	IMS3	29. Mutate Method Invocation Exp. 4	MMIE4 ⟨point_args⟩ ⟨insert_args⟩
12. Insert Missed Statement 4	IMS4 ⟨conditional_exp⟩	30. Mutate Operators 1	MO1 ⟨op1⟩ ⟨op2⟩
13. Remove Buggy Statement 1	RBS1	31. Mutate Operators 2	MO2 ⟨b_infix_exp⟩ ⟨p_infix_exp⟩
14. Move Statement 1	MS1 ⟨move_step⟩	32. Mutate Operators 3	MO3 ⟨exp⟩ ⟨T⟩
15. Mutate Conditional Exp. 1	MCE1 ⟨cExp1⟩ ⟨cExp2⟩	33. Mutate Return Statement 1	MRS1 ⟨exp1⟩ ⟨exp2⟩
16. Mutate Conditional Exp. 2	MCE2 ⟨op⟩ ⟨cExp2⟩	34. Mutate Variable 1	MV1 ⟨var1⟩ ⟨var2⟩
17. Mutate Conditional Exp. 3	MCE3 ⟨cExp1⟩ ⟨op⟩ ⟨cExp2⟩	35. Mutate Variable 2	MV2 ⟨var1⟩ ⟨exp⟩
18. Mutate Class Instance Creat. 1	MCIC1	***36. Mutate Single Line***	***MSL*** ⟨***statement***⟩

缩写的模板名称 骨干元素

- 基于模板的神经补丁中间表示（IRs）生成
  - 核心思想：给定1个含缺陷源码的method，利用编解码器模型生成补丁IRs。提出联合推理策略增强模型的复制机制，提升输入method中低频和项目特定token预测性能
  - （1）构建生成补丁IRs的编解码模型
    - 假设输入包含n个token的$x_1, \ldots, x_n$，利用BiLSTM的编码器提取代码语义特征，后将其组合为h_t。解码器利用全局注意力机制计算上下文向量v_i以输出token

$$v_i = \sum_{j=1}^{n} a_{ij} h_j, a_{ij} = \frac{\exp(s_{ij})}{\sum_{k=1}^{n} \exp(s_{ik})}, s_{ij} = f(q_{i-1}, h_j)$$

a_{ij}注意力权重，h_j编码器隐藏状态，q_{i-1}之前的译码器隐藏状态，f用mlp算编解码器隐藏状态相似分数

    - 计算每个token在字典V中的概率p_v，g_a为具有激活函数的MLP，y_i为输出token

$$p_v(y_i | y_1, \ldots, y_{i-1}, c) = g_a(y_{i-1}, q_i, v_i)$$

    - 定义损失函数以优化模型：$L(\theta) = -\sum_{i=1}^{N} \sum_{t=1}^{M} logP(y_t^{(i)} | y_1^{(i)}, \ldots, y_{t-1}^{(i)}, c)$

θ可训练的参数，N是训练样本的数量，M每个训练样本的长度

- 基于模板的神经补丁中间表示（IRs）生成
  - （2）提升复制机制的联合推断策略
  - 核心思想：利用复制机制缓解基于NMT修复方法中的OOV问题，将低频token从输入序列复制到相应的补丁IRs
    - 利用注意力权重计算中产生的中间结果，计算复制概率p_{copy}，g_c 为另1个MLP层

$$p_{copy} = g_c(y_{i-1}, q_i, v_i)$$

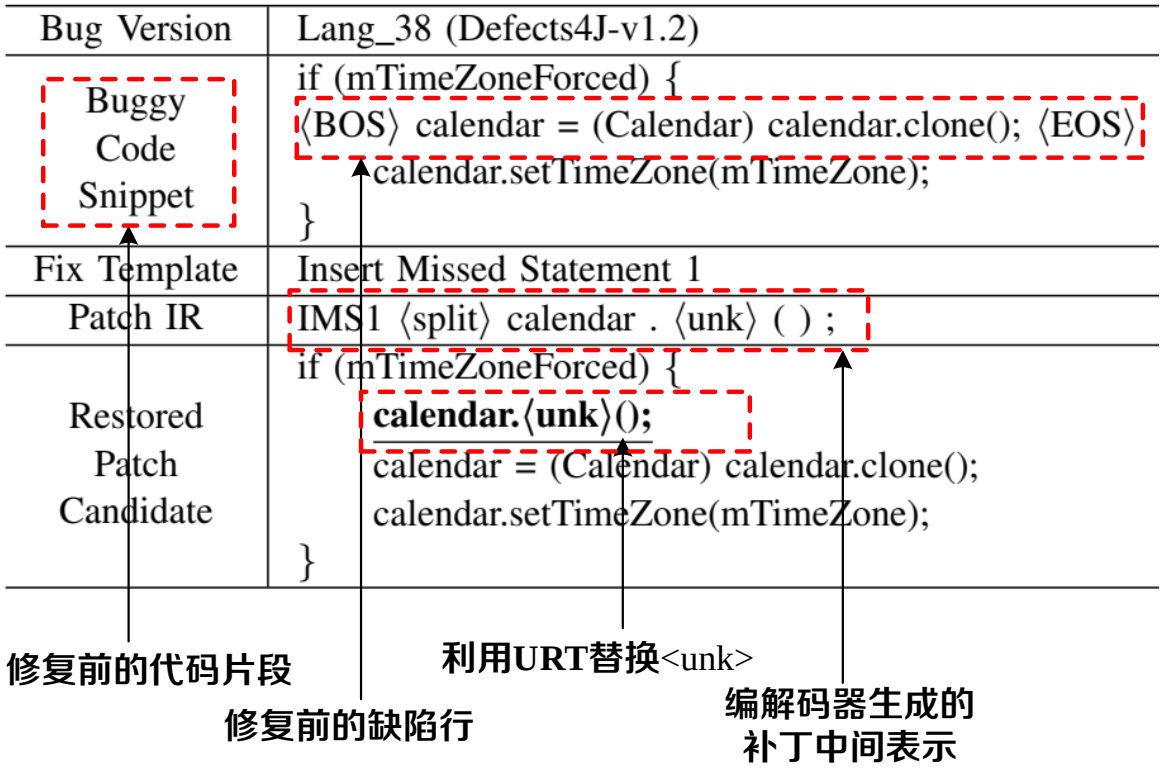
    - 计算token在字典V中的概率：$p_c(y_i) = (1 - p_{copy}) \cdot p_v(y_i) + p_{copy} \cdot \sum_{j:x_j=y_i}^{n} a_{ij}$

其中，$p_c \in V \cup \{x_1, \ldots, x_n\}$, $p_v(y_k) = 0 \; if \; y_k \notin V$

    - 每个数据集训练2个编码器模型：1个模型$model_{(w)}$含复制机制，另1个$model_{(wo)}$不含
    - 计算最终输出token的概率: $p_f(y_i) = \frac{p_c^{(w)}(y_i) + p_v^{(wo)}(y_i)}{2}$

$p_c^{(w)}(y_i)$和$p_v^{(wo)}(y_i)$分别表示$model_{(w)}$和$model_{(wo)}$生成y_i的概率

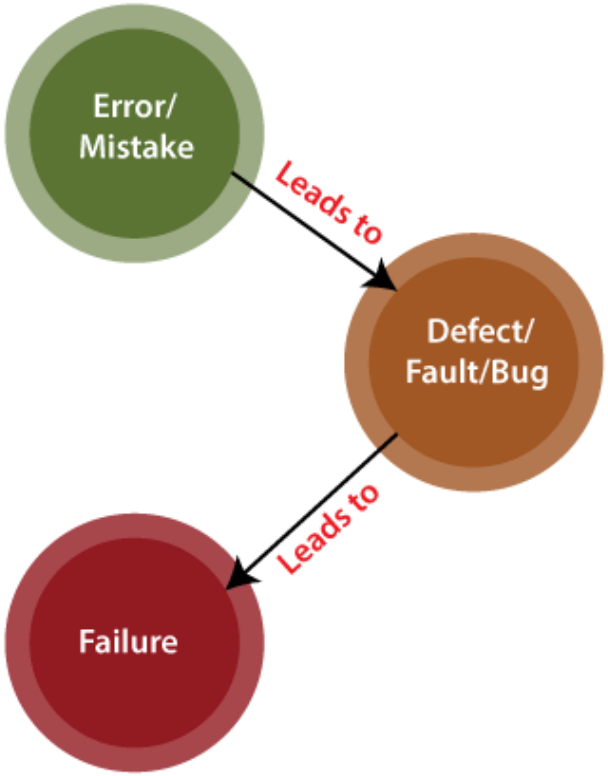
- 具有项目特定信息的真实补丁生成
  - 项目特定信息
  - 从补丁IRs中恢复真实补丁
    - 设计补丁恢复工具PRT
  - 用项目特定信息替换未知令牌
    - 设计未知令牌替换工具URT
    - 分析程序上下文的语法约束，从项目特定信息中搜索语法正确的代码元素
  - 过滤编译错误的候选补丁
    - 设计补丁检查工具PCT
    - 设计不同检查规则，分为12类
    - 过滤常见的编译错误



Insert Missed Statement 1：在缺陷行之前添加一条语句

<unk>是method调用的名称，其约束为：
1）在calendar的声明类型中，
2）目标方法的参数数量为零

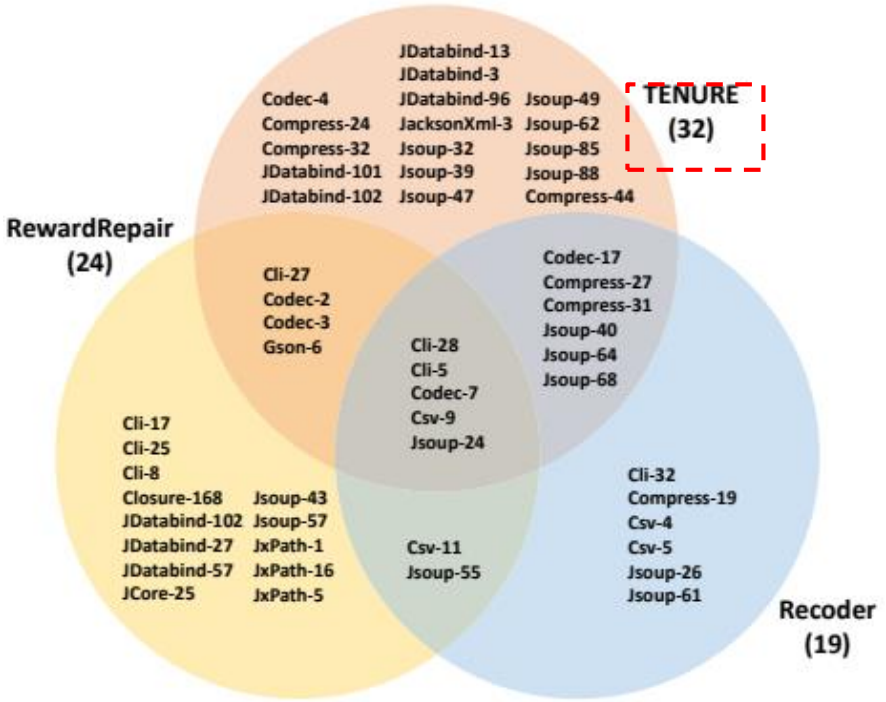
- 数据集
  - 1个基准数据集：DatasetFT（用于训练35个人工修复模板）
    DatasetST（用于训练1个突变单行模板）
  - 测试：Defects4J-v1.2和Defects4J-v2.0
- 对比方法
  - 基于人工修复模板的方法*1
  - 基于神经机器翻译的方法*4：
- 评价指标
  - 缺陷数量：正确修复的缺陷数量



TENURE 实验结果

- APR任务的对比实验
  - 目的：与其他方法相比，TENURE在APR任务中的性能
  - 设计：在完美定位和Ochiai定位下比较程序被正确修复的数量
  - 结论1：融合人工修复模板和基于NMT方法的知识能够提高修复性能
  - 结论2：使用补丁IRs进行模型训练和额外的MSL固定模板可以提升了修复效果

Techniques	Fault Localization Settings			
	Perfect		Ochiai	
	D4J-v1.2	D4J-v2.0	D4J-v1.2	D4J-v2.0
TBar [5]	62	8	41	8
CoCoNuT [18]	44	-	33	-
CURE [19]	57	-	36	-
DEAR [21]	53	-	47	-
TRANSFER-PR [26]	67	-	42	-
RewardRepair [22]	45	45	29	24
Recoder [20][1]	66	-	51	19
TENURE	**79**	**50**	**52**	**32**



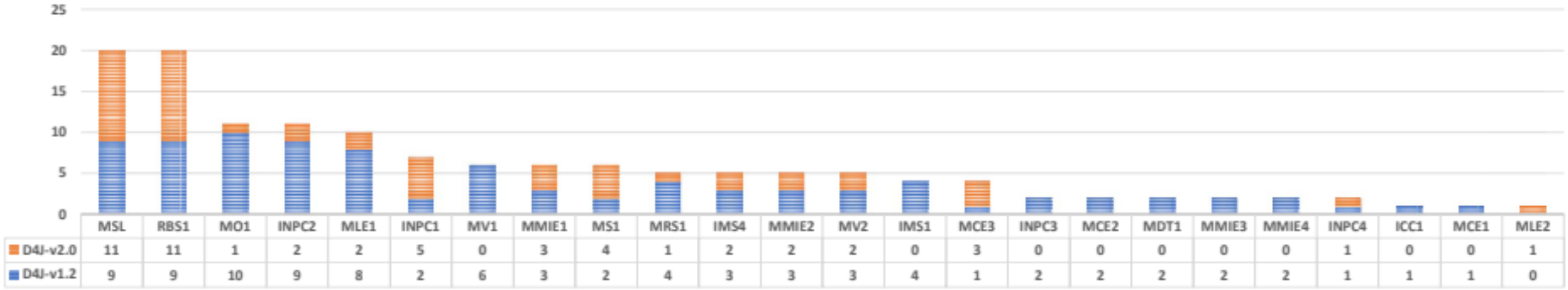
TENURE 实验结果

- TENURE的消融实验
  - 目的：验证TENURE主要组成部分的有效性
  - 设计：分别测试人工修复模板、MSL模板、联合推理策略、URT工具性能
  - 结论1：35个人工修复模板比MSL模板修复的缺陷数量更多
  - 结论2：联合推理策略和URT相结合有利于处理OOV问题
  - 结论3：URT模块有利于缺陷修复，具有unk占位符的候选补丁无法修复任何缺陷

Model Variants	#Bugs
TP_{35} + JOINT + URT + PCT	73
TP_{MSL} + JOINT + URT + PCT	32
TP_{35} + TP_{MSL} + Only copy + URT + PCT	66
TP_{35} + TP_{MSL} + No copy + URT + PCT	56
TP_{35} + TP_{MSL} + JOINT + PCT	63
TP_{35} + TP_{MSL} + JOINT + URT + PCT (TENURE)	79

TENURE 实验结果

- TENURE的消融实验
  - 目的：验证不同修复模板对修复结果的贡献
  - 设计：统计不同模板对修复的补丁个数
  - 结论1：保留了24个修复模板，忽略未生成任何正确补丁的12个修复模板
  - 结论2：MSL模板贡献了20个正确额补丁，对35个人工修复模板有补充效果
  - 结论3：12个修复模板在DatasetsFT数据集中样本量小，导致实验无法充分利用



	MSL	RBS1	MO1	INPC2	MLE1	INPC1	MV1	MMIE1	MS1	MRS1	IMS4	MMIE2	MV2	IMS1	MCE3	INPC3	MCE2	MDT1	MMIE3	MMIE4	INPC4	ICC1	MCE1	MLE2
D4J-v2.0	11	11	1	2	2	5	0	3	4	1	2	2	2	0	3	0	0	0	0	0	1	0	0	1
D4J-v1.2	9	9	10	9	8	2	6	3	2	4	3	3	3	4	1	2	2	2	2	2	1	1	1	0

TENURE 实验结果

- TENURE的消融实验
  - 目的：验证补丁检查工具的有效性
  - 设计：分析PCT工具对修复效果、候选补丁通过率的影响及其运行效率
  - 结论1：PCT可以减少修复过程达到真正缺陷语句的时间，但不影响修复的有效性
  - 结论2：PCT可以提高任意波束参数下的补丁通过率
  - 结论3：javac编译1000个java文件需1201秒，PCT只需30秒

Techniques	Top-30	Top-100	Top-200
SequenceR [15]	33%	-	-
CoCoNuT [18]	24%	15%	6-15%
CURE [19]	39%	28%	14-28%
RewardRepair [22]	45.3%	37.5%	33.1%
TENURE w/o PCT	47.3%	38.0%	33.0%
TENURE w/ PCT	**70.0%**	**65.7%**	**62.8%**

Rap-Gen

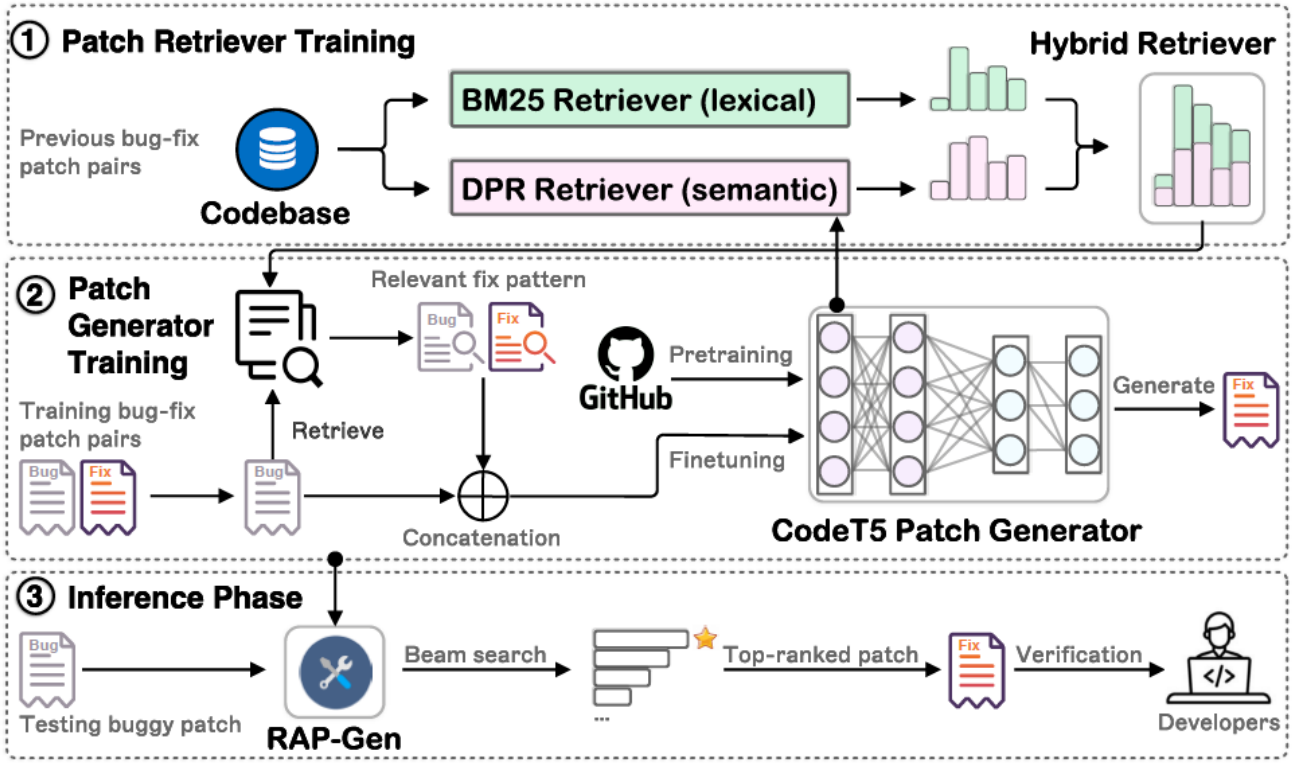
RAP-Gen TIPO

T	从以往缺陷修复对数据库中检索修复模式
I	含缺陷的补丁
P	1. 补丁检索器训练：构建与学习混合检索器，根据词汇和语义相似性找到相关的代码补丁 2. 补丁生成器训练：训练CodeT5模型，基于缺陷输入和检索到的缺陷修复示例生成修复补丁 3. 推理：预测多个修复补丁，开发人员对排名靠前的补丁进行验证
O	生成的修复补丁

P	现有神经机器翻译方法的补丁生成受固定模型参数集的限制，难以学习用于程序修复的高度复杂的修复模式
C	结合预训练模型和基于检索的技术
D	如何构建无关编程语言、代码功能的补丁检索器
L	FSE2023（CCF-A）

RAP-Gen 算法原理

- 基本框架
  - 构建与学习混合检索器，根据词汇和语义相似性找到相关的代码补丁
  - 训练CodeT5模型，基于缺陷输入和检索到的缺陷修复示例生成修复补丁
  - 预测多个修复补丁，通过开发人员对排名靠前的补丁进行验证



- 任务建模
  - 程序修复数据集：$D = \{(X_i, Y_i)\}_{i=1}^{|D|}$， X_i和Y_i分别是第i个缺陷和修复的程序补丁
  - 以往缺陷修复对数据库： $C = \{(B_j, F_j)\}_{j=1}^{|C|}$， (B_j, F_j)表示以前1个缺陷修复对
  - 给定D中的缺陷程序补丁X_i，通过ϕ参数化的相关性评分函数$f_\phi(B_j, F_j)$，检索器会检索到代码库C中最相关的错误修复对(B_j, F_j)
  - 利用检索到的(B_j, F_j)增强原始输入序列X_i，形成新的输入序列$\widehat{X_i} = X_i \oplus B_j \oplus F_j$，其中$\oplus$表示级联操作。
  - 利用seq2seq生成器以自回归方式从$\widehat{X_i}$生成Y_i。形式上讲，目标是使用参数化为θ的补丁seq2seq生成器学习以下概率：

$$P_\theta(Y_i|\widehat{X_i}) = \prod_{k=1}^{n} P_\theta(Y_{i,k}|\widehat{X_i}, Y_{i,1}:Y_{i,k-1})$$

- 混合补丁检索器
  - 核心思想： 结合基于词汇的BM25检索器和基于语义的DPR检索器，以获取词汇和语义信息，稀疏和密集检索器相互补充，实现更稳健的文本检索
  - 补丁相似度：计算查询补丁X_i和候选补丁B_j之间的词汇相似性
    - 基于词汇的检索器：$f_\phi(X_i|B_j) = BM25(X_i|B_j)$
    - 基于语义的检索器：$f_\phi(X_i, B_j) = sim(X_i, B_j) = [CLS_{X_i}]^T [CLS_{B_j}]$
      - 训练过程优化InfoNCE对比损失函数：

$$L_{nce} = \frac{1}{N}\sum_{i=1}^{N} -log \frac{exp(sim((B_j, F_i)))}{\exp\left(sim(B_j, F_i)\right) + \sum_{j\in M, j\neq j} \exp(sim(B_j, F_i))}$$

    - 混合检索器：$f_\phi(X_i, B_j) = sim(X_i, B_j) + \lambda \mathrm{BM25}(X_i, B_j)$

λ是平衡两类检索器的权重

- 检索增强的补丁生成器
  - 核心思想：给定1个有缺陷的补丁X_i，通过在以往数据库搜索1个最相关的修复模式(B_j, F_j)并将其传递给补丁生成器以生成固定代码补丁
  - 扩展的输入：$\widehat{X_i} = X_i \oplus B_j \oplus F_j$

    其中，X_i第i个缺陷，B_j和F_j是在数据库中检索到的Bug-fix pairs
  - 用于生成补丁的预训练模型：**CodeT5**
  - 训练阶段：训练策略最小化所有训练实例上的交叉熵损失L_{ce}，损失函数为

$$L_{ce} = -\sum_{i=1}^{|D|} \log(p_\theta(Y_i|\widehat{X_i}))$$

p_θ：模型参数，Y_i：目标输出，$\widehat{X_i}$ 是扩展的输入

  - 推理阶段：采用波束搜索为输入的有缺陷的补丁生成一个固定补丁候选的排名列表

RAP-Gen 实验设计

- 数据集
  - TFix
  - Code Refinement
  - Defects4J

Table 1: Statistics of three program repair benchmarks.

Benchmark	Version	Train	Valid	Test
TFix	Original	84,846	9,454	10,504
TFix	Deduplicated	84,673	9,423	10,465
Code Refinement	Small	46,680	5,835	5,835
Code Refinement	Medium	52,364	6,545	6,545
Defects4J	v1.2	-	-	388
Defects4J	v2.0	-	-	430

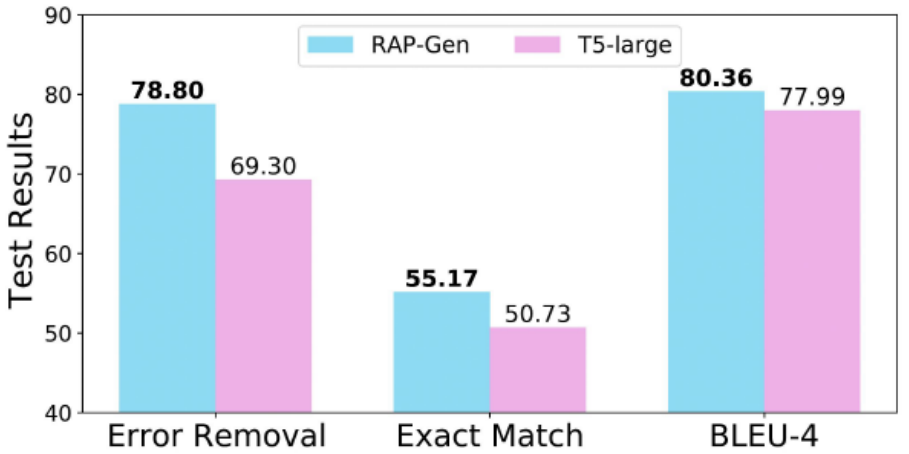
- 对比方法
  - 方法*9：SequenceR、CoCoNuT、CURE、RewardRepair、Recoder、DLFix、DEAR、BugLab、SelfAPR
- 评价指标
  - 缺陷修复数量
  - EM Accuracy：完全匹配（生成的缺陷补丁和真实补丁完全一致）准确率
  - Error Removal：Error被删除且没有引入新的Error
  - BLEU-4：评估生成的文本与参考文本的相似度

RAP-Gen 实验结果

- **RAP-Gen与基于DL的APR模型在TFix数据集上的对比实验**
  - **目的：在数据集TFix中，比较CodeT5和RAP-Gen的补丁生成准确率**
  - **设计：利用加权完全匹配（EM）缓解缺陷类型分布的不平衡**
  - **结论1：利用大规模源代码感知预训练的CodeT5模型对程序有更好的理解力**
  - **结论2：语义信息和词汇信息有利于RAP-Gen检索相关修复模式**

Model	EM w/ spaces		EM w/o spaces	
	Avg.	W. Avg.	Avg.	W. Avg.
Naive Copy	0.00	0.00	0.00	0.00
SequenceR	17.90	-	-	-
CoCoNuT	11.70	-	-	-
T5-small	44.46	44.44	44.52	44.60
T5-base	48.54	47.63	48.72	47.70
T5-large	49.33	**49.65**	49.35	49.70
CodeT5-small	47.14	46.35	50.22	50.31
- error-information	45.97	45.80	49.26	49.70
CodeT5-base	**50.88**	49.42	**54.30**	**53.57**
- error-information	46.88	47.17	50.36	51.25

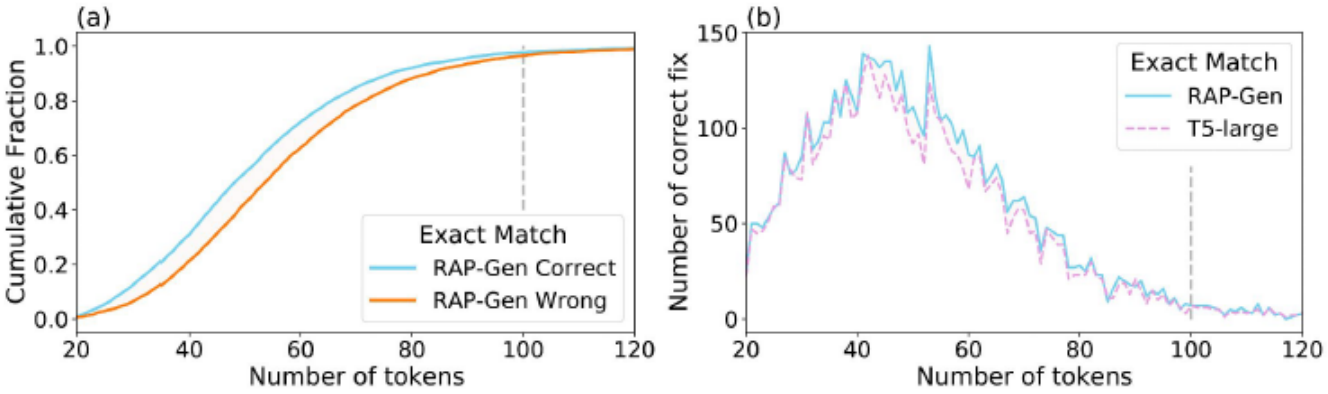
Model	EM	BLEU-4
T5-large (TFix)	49.58	76.96
CodeT5-base	53.46	78.92
RAP-Gen	**54.15**	**79.66**



RAP-Gen 实验结果

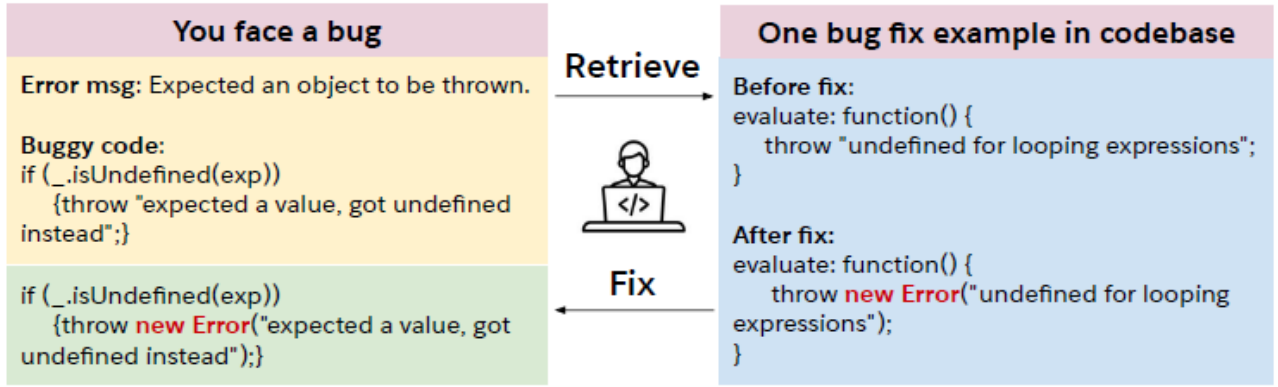
- **RAP-Gen在TFix数据集上的性能分析**
  - **设计：在修复的错误类型、修复操作模式和补丁长度的角度分析RAP-Gen的性能**
  - **结论1：相比于T5-large模型， RAP-Gen修复的error更多**
  - **结论2：RAP-Gen可以学习多种错误修复模式，未过度依赖琐碎的错误行删除模式**
  - **结论3：RAP-Gen成功修复的补丁比失败的补丁短，比T5-large修复更多的缺陷**

	T5-large	CodeT5	RAP-Gen
# Ground-truth EL Removal	2,381	2,381	2,381
# Predicted EL Removal	1,882	1,925	1,922
# Correct EL Removal	1,811	1,858	1,866
# False Positive	71	67	56
Precision (%)	96.23	96.52	**97.09**
Recall (%)	76.06	78.03	**78.37**
F1 (%)	84.96	86.30	**86.73**



RAP-Gen 实验结果

- RAP-Gen与基于DL的APR模型在Code Refinement数据集上的对比实验
  - 设计：在Code Refinement数据集不同体量子集下分析不同模型的修复性能
  - 结论1：与小子集相比，中等子集结果较低，表明较长的错误函数更难修复
  - 结论2： RAP-Gen检索到的修复模式有利于指导程序修复，混合检索器通过使用词汇和语义信息而稳健性更高



Model	Small		Medium	
	EM	BLEU-4	EM	BLEU-4
Naive Copy	0.00	78.06	0.00	90.91
LSTM	10.00	76.76	2.50	72.08
Transformer	14.70	77.21	3.70	89.25
RoBERTa (code)	15.90	77.30	4.10	90.07
CodeBERT	16.40	77.42	5.16	91.07
GraphCodeBERT	17.30	80.02	9.10	91.31
PLBART	19.21	77.02	8.98	88.50
CoTexT	21.58	77.28	13.11	88.40
NSEdit	24.04	71.06	13.87	85.72
CodeT5	21.61	77.43	13.96	87.64
RAP-Gen	**24.80**	78.28	**15.84**	90.01

RAP-Gen 实验结果

- 混合补丁检索器分析
  - 设计：分析不同检索器组件（BM25、CodeBERT、CodeT5）的有效性，检索到补丁的适应性（词汇匹配：Token重叠度BLEU-4、语法匹配：余弦相似度CosSim）
  - 结论1：基于BM25和CodeT5的混合检索器更有效
  - 结论2： 混合检索器平衡词汇和语义信息，并且比基于词汇的检索器更稳健

Retriever	TFix	Refine-Small	Refine-Medium
No Retriever	53.46	21.61	13.96
Random	52.98	21.25	13.53
BM25	53.88	23.82	15.37
CodeBERT	52.96	22.28	15.42
CodeT5	53.93	24.37	15.60
Hybrid (BM25+CodeT5)	54.15	24.80	15.84

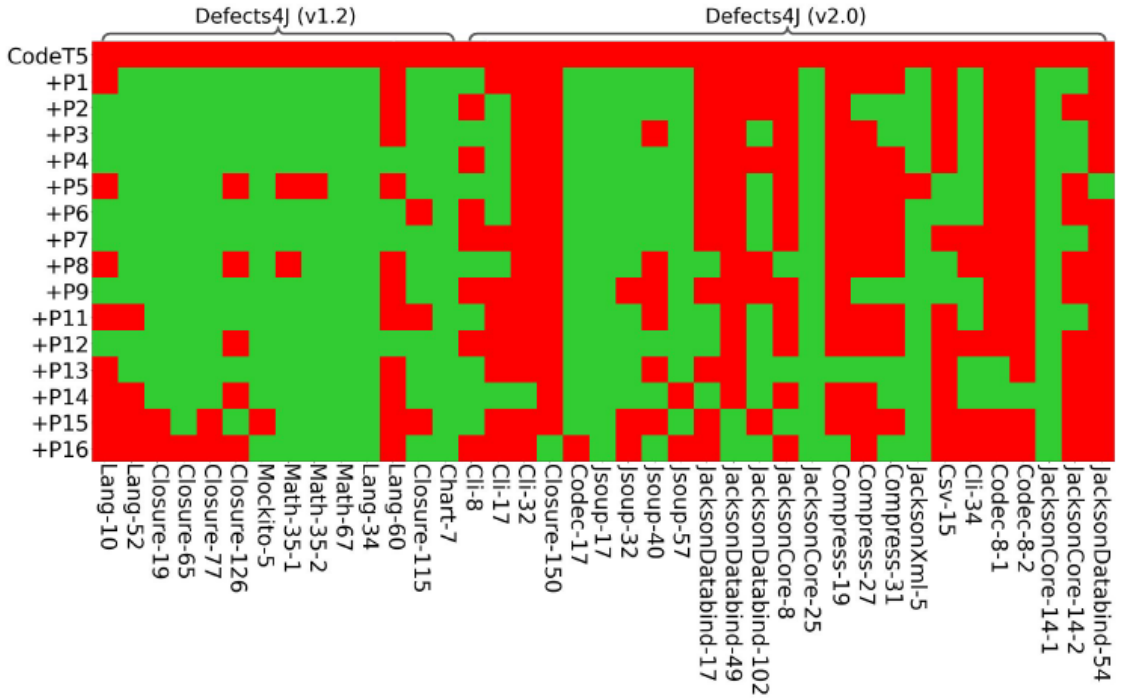
Retriever	TFix		Refine-Small		Refine-Medium	
	BLEU-4	CosSim	BLEU-4	CosSim	BLEU-4	CosSim
Random	0.1	35.5	14.6	35.4	14.6	30.6
BM25	23.7	70.9	41.5	68.5	39.0	66.6
DPR	21.7	**75.4**	54.4	**84.9**	44.3	**81.3**
Hybrid	**24.4**	73.4	**57.4**	84.2	**45.0**	80.9

RAP-Gen 实验结果

- **RAP-Gen与基于DL的APR模型在Defects4J数据集上的对比实验**
  - **设计：频谱定位和完美定位两种缺陷定位方式下，比较RAP-Gen的修复性能**
  - **结论1：与对比方法相比，RAP-Gen修复的缺陷数量更多**
  - **结论2： RAP-Gen从修复模式中检索有助于纠正CodeT5对Defects4J缺陷误预测**

Model	Spectrum-based FL		Perfect FL	
	v1.2	v2.0	v1.2	v2.0
SequenceR [7]	-	-	14	-
BugLab [2]	-	-	17	6
DLFix [31]	30	-	40	-
CoCoNuT [40]	-	-	43	-
RewardRepair [76]	27	24	44	43
DEAR [32]	47	-	53	-
CURE [20]	-	-	55	-
Recoder [79]	**49**	19	64	-
SelfAPR [75]	39	**28**	65	45
CodeT5	27	13	58	28
RAP-Gen	48	26	**72**	**53**

In the table cells, it represents the number of correct patches.
'-' indicates data unavailability.



特点总结与未来展望

- 特点总结

算法	TENURE	RAP-Gen
优势	1. 学习深层语义特征和结构化补丁中间表示，提升对样本的表征能力 2. 未知token替换策略和编译错误补丁检查缓解了OOV问题，提高修复效率	1. 构建了自适应缺陷修复框架，便于替换高性能的预训练模型和检索器 2. 结合稀疏和密集检索器匹配词汇和语义，无关编程语言
劣势	1. 仅测试了java编程语言，未验证35个人工模板在其他语言的有效性	1. 受限于收集的以往数据库的质量； 2. 仅测试了java编程语言，未在其他语言的有效性

- 未来展望
  - 高修复精度和智能化：利用深度学习和强化学习技术，更准确地生成和验证修复补丁，理解复杂的程序上下文和逻辑。
  - 泛化性和可解释性：支持多种编程语言和平台APR工具，可应用于各种开发环境。提高生成修复补丁的可解释性，让开发人员更易理解和审查生成的补丁

参考文献

[1] Meng X, Wang X, Zhang H, et al. Template-based neural program repair[C]//2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023: 1456-1468.

[2] Wang W, Wang Y, Joty S, et al. Rap-gen: Retrieval-augmented patch generation with codet5 for automatic program repair[C]//Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2023: 146-158.

[3] See A, Liu P J, Manning C D. Get to the point: Summarization with pointer-generator networks[J]. arXiv preprint arXiv:1704.04368, 2017.

谢谢！

大成若缺，其用不弊。大盈若冲，其用不穷。大直若屈。大巧若拙。大辩若讷。静胜躁，寒胜热。清静为天下正。