AiDD
2024 AI+研发数字峰会
AI+ Development Digital summit
AI驱动研发变革 促进企业降本增效
北京站 08/16-17

一种基于修复偏好的自动程序修复工具集成策略

李传艺 南京大学

李传艺

南京大学 助理教授

CCF系统软件专委、软件工程专委执行委员。主要研究方向为智能化软件技术，在ACM TOSEM、IEEE TSC、IEEE/ACM TASLP、ACM TKDD、JSS、InS.、FGCS、JPDC、软件学报、计算机学报 和 ICSE、ESEC/FSE、ASE、ACL、AAAI、IJCAI、EMNLP等国内外重要学术期刊和会议发表论文60余篇，获授权专利6项。主持国家自然科学基金项目、CCF-华为胡杨林基金项目，以及华为、腾讯、华夏幸福等企业创新项目10余项，成果研制了一系列智能化软件工具，落地应用于国产编程语言开发环境搭建、国产操作系统生态建设等信创领域。

目录
CONTENTS

PART 01
APR研究现状及集成方法

丰富的APR系统 （1）

- APR(Automated Program Repair)旨在自动修复软件系统中的缺陷
- 目前已经有超过40种的APR工具，包括
  - 传统的基于规则的工具
  - 基于神经机器翻译的工具（NPR）
  - 基于大语言模型（LLM）的工具

1. 基于启发式搜索：人工定义启发式规则,指导修复补丁的生成过程
例子: GenProg[1], 采用遗传算法，以能通过的测试用例的数量为优化目标，不断修改缺陷的代码片段直到产生通过所有测试用例的代码

2. 基于语义约束：通过某种手段推断程序的正确规约,作为约束指导补丁的生成过程
例子: Nopol[2],针对java程序中的条件语句，首先在出错的代码位置搜集所有变量的取值情况,然后根据期望的条件语句取值情况(true 或 false),将程序语义编码成为Z3约束求解器的约束进行求解

3. 基于修复模板的方法：根据开发者、研究人员的经验或者数据挖掘的结果预定义一些补丁模板或者补丁生成策略用于指导修复的过程
例子: TBar[3],基于修复模板的大成之作，集成了大量的补丁模板

[1] Le Goues, C., Nguyen, T., Forrest, S., & Weimer, W. (2011). Genprog: A generic method for automatic software repair. Ieee transactions on software engineering, 38(1), 54-72.
[2] Xuan, Jifeng, et al. "Nopol: Automatic repair of conditional statement bugs in java programs." IEEE Transactions on Software Engineering 43.1 (2016): 34-55.
[3] Liu, K., Koyuncu, A., Kim, D. and Bissyandé, T.F., 2019, July. TBar: Revisiting template-based automated program repair. ISSTA 2019 (pp. 31-42).

丰富的APR系统 （2）

- APR(Automated Program Repair)旨在自动修复软件系统中的缺陷
- 目前已经有超过40种的APR工具，包括
  - 传统的基于规则的工具
  - 基于神经机器翻译的工具（NPR）
  - 基于大语言模型（LLM）的工具

1. Tufano等人[1]将程序修复定义为神经机器翻译(Neural Machine Translation)任务; **CoCoNut**[2]就结合了上下文感知的NMT架构和CNN模型用于程序修复

2.目前NPR的改进角度大致可以分为三类：数据预处理、输入表示(recoder)和输出搜索(decoder)

3. **SequenceR**[3] 能够在类级别的代码上下文中接受最多1,000个令牌作为输入，以确保其在不同场景中的适用性；**DLFix**[4]为有bug的代码实现了重命名抽象，这样可以增强修复模型学习如何修复类似错误的能力。**Cure**[5]在其NMT架构中添加了一个在源代码上预训练的GPT编码模块，而**Circle**[6]和**RewardRepair**[7]则利用一个基于Transformer的预训练语言模型T5作为它们的编码器。目前NPR的解码器架构主要分为两种类型：一类是LSTM或Transformer序列解码器，由**Codit**[8]和**SelfAPR**[9]采用；另一类是**Recoder**[10]和**Tare**[11]自己设计的结构感知解码器，它们将解码阶段建模为AST的修改，而不是生成文本序列。
4. 最近随着各种大语言模型（LLM）的出现，NPR研究人员开始探索将LLM用于程序修复。 **RepairLLaMA**[12]，设计了多种输入/输出的内容格式，并采用Lora方法来微调CodeLLaMA

丰富的APR系统 （2）——引用列表

[1] An Empirical Study on Learning Bug_x0002_Fixing Patches in the Wild via Neural Machine Translation.

[2] CoCoNuT: combining context-aware neural translation models using ensemble for program repair.

[3] SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair.

[4] DLFix: context-based code transformation learning for automated program repair.

[5] CURE: Code-Aware Neural Machine Translation for Automatic Program Repair.

[6] CIRCLE: continual repair across programming languages.

[7] Neural Program Repair with Execution-based Backpropagation.

[8] CODIT: Code Editing With Tree-Based Neural Models.

[9] SelfAPR: Self-supervised Program Repair with Test Execution Diagnostics.

[10] A syntax-guided edit decoder for neural program repair.

[11] Tare: Type-Aware Neural Program Repair.

[12] RepairLLaMA: Efficient Representations and Fine-Tuned Adapters for Program Repair.

丰富的APR系统 （3）

- APR(Automated Program Repair)旨在自动修复软件系统中的缺陷
- 目前已经有超过40种的APR工具，包括
  - 传统的基于规则的工具
  - 基于神经机器翻译的工具（NPR）
  - 基于大语言模型（LLM）的工具

1. **AlphaRepair**[1] 是第一个non-training LLM-based APR工具：将缺陷修复转化为MLM任务，在遮蔽缺陷行后，利用CodeBERT对遮蔽进行预测，从而修复缺陷行。

2. Jiang等人[2] 对于PLBART、CodeT5、CodeGen和InCoder四种LLM进行了实证，实验结果发现，在不微调的情况下，这四种LLM的修复性能在Defects4J v2.0、QuixBugs和HumanEval-Java上优于传统的NPR模型，而在Defects4J v1.2上与最先进的NPR模型修复性能基本持平；

3. Xia等人[3] 也进行了相似的研究，发现GPT-Neo、GPT-J、GPT-NeoX和InCoder的修复性能仍逊于最先进的NPR工具，但Codex超过了现存所有NPR。这对于LLM4APR的未来，给了研究人员们极大的信心。

4. 而随着GPT-3.5的出现， LLM4APR的浪潮达到了顶峰，GPT(3.5/4)4APR成为了目前最主流的修复工具：如利用Bug外部信息与GPT进行对话式修复的**ChatRepair**[4]；有利用往复翻译（RTT）进行修复的**RTTAPR**[5]；有利用GPT-3.5生成额外信息指导开源模型修复的**Srepair**[6]；有利用思维链增强修复性能的**ThinkRepair**[7]；也有很多Agent-based的修复工具，如**FixAgent**[8]、**RepairAgent**[9]。

5. 目前这些GPT(3.5/4)-based APR在诸如Defects4J、QuixBugs(Python/Java)上远超其他任何APR技术的修复性能。

丰富的APR系统 （3）——引用列表

[1] Less Training, More Repairing Please: Revisiting Automated Program Repair via Zero-shot Learning

[2] Impact of Code Language Models on Automated Program Repair

[3] Automated Program Repair in the Era of Large Pre-trained Language Models

[4] Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT

[5] A Novel Approach for Automated Program Repair using Round-Trip Translation with Large Language Models

[6] How Far Can We Go with Practical Function-Level Program Repair?

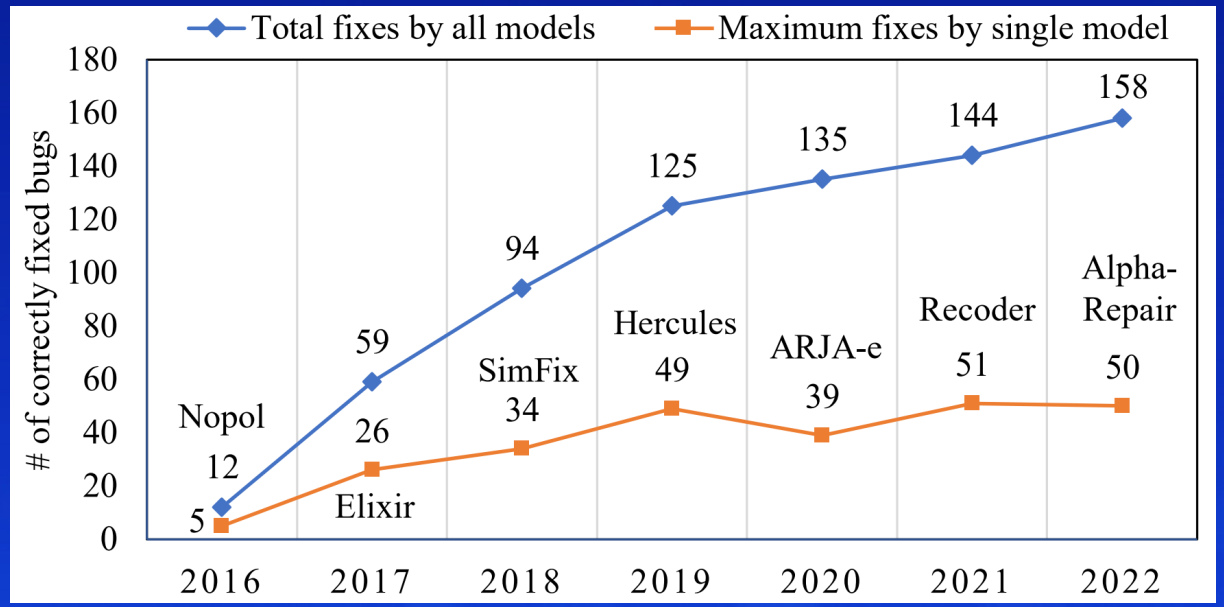
[7] ThinkRepair: Self-Directed Automated Program Repair

[8] A Unified Debugging Approach via LLM-Based Multi-Agent Synergy

[9] RepairAgent: An Autonomous, LLM-Based Agent for Program Repair

交叉的修复性能——不包括非训练方式LLM-based

- 没有一个工具能够超越其他所有工具
  - 对每一个工具而言，都存在自己不能修复但其他工具能修的Bug



单个模型能够修复的Bug数量

巨大差距

所有模型能够修复的Bug数量

对同一个Bug尝试所有工具 *vs.* 首先尝试最有可能修复Bug的工具

基于学习的APR工具集成方法

Aldeida Aleti and Matias Martinez. 2021. **E-APR**: Mapping the effectiveness of automated program repair techniques. *Empirical Software Engineering,* 26, 5 (2021), 99.

- 创新想法：不同于追求单个APR工具的性能，首次尝试为Bug选择工具
  - 收集能够度量Bug特征的角度，用于关联Bug和APR工具；做特征关联分析
  - 搜集和构造<Bug，APR工具>数据对
  - 使用SVM、DT、RFC、MLP等相对传统的算法
- 效果

R3: The Random Forest Classifier is the best performing Machine Learning technique for APRT selection, and can predict the most suitable technique with **88%** precision, **81%** recall and **84%** f1-score.

PART 02
本工作动机

弥补基于学习的集成方案的不足?

- 基于学习的集成方法比较常见：代码检索、代码补全、代码摘要等
- 但基于学习的方法存在一些问题
  - 训练数据构造成本高：依赖大规模实际修复数据，要求对一个Bug尝试多种不同APR工具
  - 特征选取随机性大，和APR工具自身特征的联系松散
  - 增加新的APR工具代价大：一方面需要构造对应的修复数据，另一方面需要重新训练

非基于训练的集成方案? →

- **不需要构建专门的训练数据**
- **可以灵活添加新的APR工具**
- **随着修复经验积累性能增强**

→ **如何直接度量一个APR工具是否能够修复一个Bug?**
通过待修复Bug和APR工具能修复、不能修复的Bug之间的关系度量!

PART 03
基于修复偏好的集成方案

基础概念

- Preference-based Ensemble Program Repair, P-EPR
- 修复偏好：APR工具对各类特征（共4类）的Bug的修复能力（能/不能）

No.	Bug Feature	Pattern*
BF1	Node type of the buggy statement	P6, 11, 12
BF2	Child node types within the buggy statement	P3, 5, 7-9
BF3	BF1 & BF2	P1, 10
BF4	Type of the test error	P4

No.	Repair Pattern	Pre-requirement
P1	Insert Cast Checker	A buggy statement that contains at least one unchecked cast expression
P2	Insert Null Pointer Checker	A buggy statement if, in this statement, a field or an expression (of non-primitive data type) is accessed without a null pointer check
P3	Insert Range Checker	Inserting a range checker for the access of an array or collection if it is unchecked
P4	Throw Exception	The failed test type is throwing an exception
P5	Mutate Class Instance Creation	If the class instance creation is in an overridden clone method
P6	Mutate Conditional Expression	A conditional expression that returns a boolean value
P7	Mutate Data Type	A variable declaration or a cast expression
P8	Mutate Integer Division Operation	An expression that contains integer division operation
P9	Mutate Literal Expression	Boolean, Number, or String in a buggy statement
P10	Mutate Method Invocation Expression	An expression that contains method invocation
	.....	.....

- 工具分成两大类

基于启发式规则的

基于约束的

基于模板的

→ 从APR工具的修复操作推理Bug特征 →

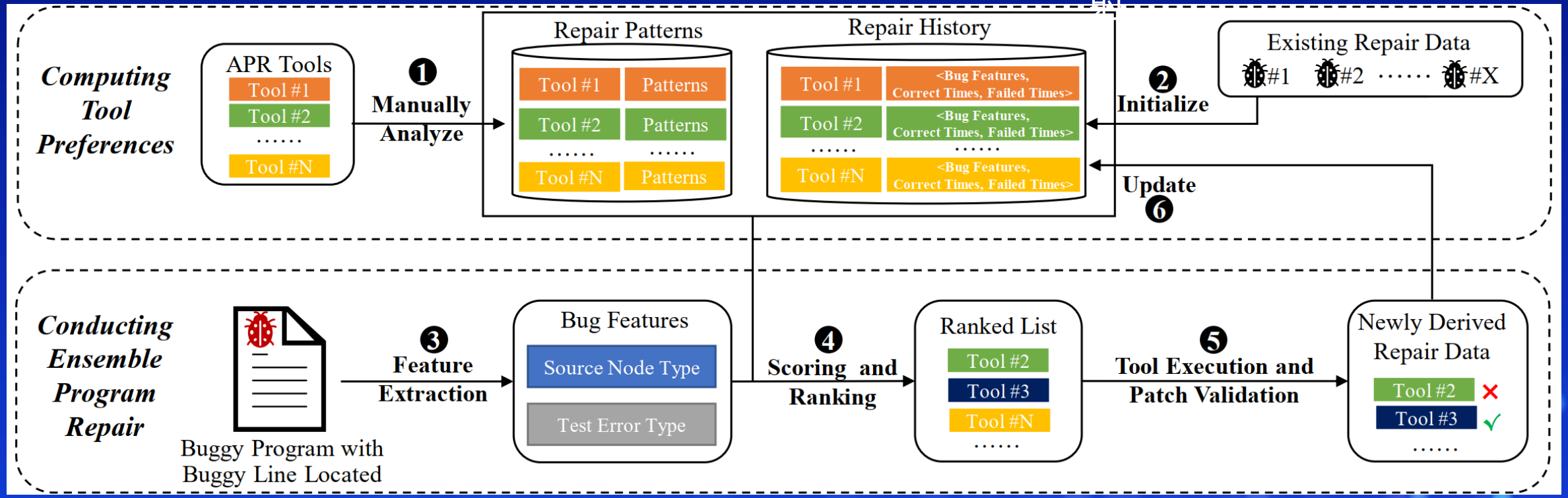
基于学习的（NPR） → 直接以列表形式记录对各种特征的Bug的修复情况

未包含LLM-based self-enhancing APR

集成框架

Step1：为3类传统APR工具整理其修复能力与Bug特征之间的映射关系

Step2/6：根据修复历史数据为所有工具初始化/更新Bug特征与修复能力的映射



Step3：抽取待修复Bug的特征

Step4：为所有工具计算成功修复输入Bug的得分并排序

Step5：按序执行工具并记录结果

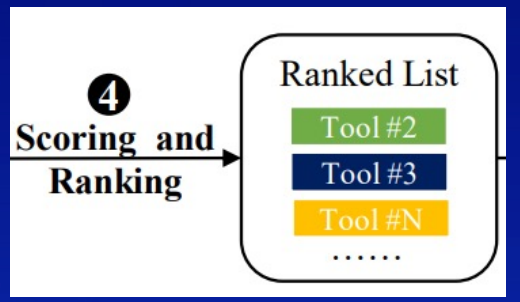
细节1——评分

- 修复历史是基础，修复偏好是加分项

Algorithm 1: Calculate preference scores for APR tools

Input: The faulty line IDs, *allFaultyLineIds*
Input: The faulty class file, *buggyFile*
Input: The bonus coefficient of pattern match, EM_α
Output: The preference scores of all tools, *preferScores*

```
1  preferScores ← ∅ ;
2  EM_α ← 0.5 ;
3  for lineId ∈ allFaultyLines do
4      bugFeatures ← ExtractFeature (lineId, buggyFile);
5      for tool ∈ availableTools do
6          finalScore ← 0;
7          historyScore ← 0;
8          for feature ∈ bugFeatures do
9              historyScore ←historyScore +
                CalculateHistoryScore(tool, feature);
10         end
11         if PatternMatch(tool, bugFeatures) then
12             finalScore ← historyScore ∗ (1 + EM_α);
13         else
14             finalScore ← historyScore;
15         end
16         preferScores.set(tool, finalScore);
17     end
18 end
```



Final Score = Base Score * (1+Pattern Factor)

可配置的参数

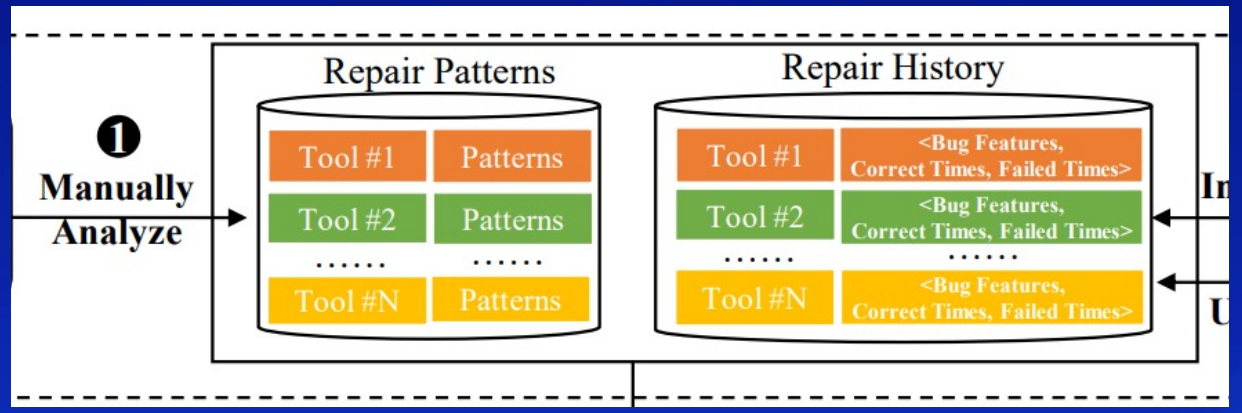
BaseScore: 根据修复历史中记录的修复能力和Bug特征之间的关系就算基础得分；所有类型工具均可计算

PatternFactor: 如果Bug特征符合传统工具的修复偏好，则通过提升影响因子的方式为传统工具加分

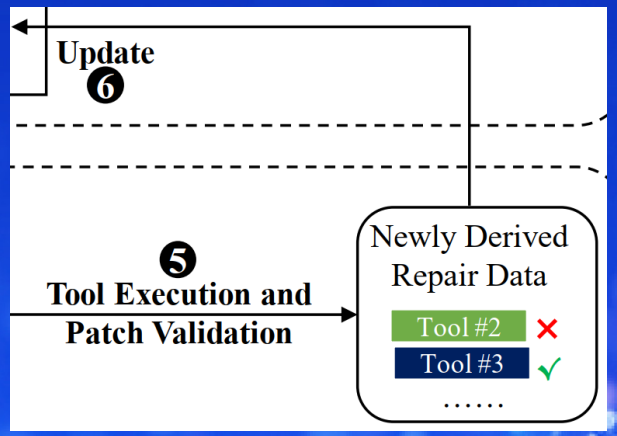
优先选择传统APR工具，可解释、更可信

细节2——更新

- 添加新的工具：直接更新Repair Patterns、Repair History，不影响其他工具



- 持续更新修复历史：所有工具的修复历史都会持续更新；不强制新工具有历史数据



冷启动问题：得分低，
难以被选中

PART 04
实验和结果讨论

实验设置（1）

- 包括21个APR工具的仿真模拟实验
  - 集成框架仅为测试集中的Bug推荐APR工具的执行顺序
  - 实际修复与否、修复代价则直接复用已公开数据
  - 数据集为Defects4J；Repair History设置为空
  - 输入顺序Random，修复偏好影响因子0.5
- 包括4个APR工具的真实场景下的执行与验证实验
  - 集成框架不仅为测试集中的Bug推荐APR工具的执行顺序
  - 也实际执行修复任务，包括补丁人工验证
  - 数据集为Defects4J 和 Bears
  - 生成300个Patch，自动验证时间限制2小时，人工10分钟

	System	# Correct	# Overfit	Source
heuristic-based	jGenProg	6	10	[24]
	GenProg-A	8	21	
	RSRepair-A	9	25	
	ARJA	11	25	
	SimFix	29	21	
	jKali	2	6	
	Kali-A	5	37	
	jMutRepair	5	6	
	TransplantFix	36	33	[50]
constraint	Nopol	2	7	[24]
	ACS	16	5	
	Cardumen	2	14	
	DynaMoth	3	10	
template	kPAR	33	30	[24]
	AVATAR	30	20	
	FixMiner	34	29	
	TBar	54	30	
learning	SequenceR	27	24	[53]
	CodeBERT-ft	29	28	
	RewardRepair	43	22	
	Recoder	56	22	
	Total	122	121(58)	

实验设置（2）

- Baseline
  - E-APR：和原文设置相同，集成10个APR工具
  - E-APR(enhanced)：和本文设置相同，集成21个APR工具
  - Random Selection：为每一个Bug随机选择APR工具
  - Invoking All：执行所有被集成的APR工具——最大开销
  - Optimal Selection：每次Top 1就是能修复Bug的APR工具
- 评估指标
  - 被正确修复的Bug数量、找到Plausible补丁的Bug数量
  - APR工具调用次数、人工检验Plausible补丁的次数
  - 相比执行所有工具而节约的工具调用次数、人工检验似真补丁的次数

- RQ1.总体性能
- RQ2.组件消融
- RQ3.实用性能

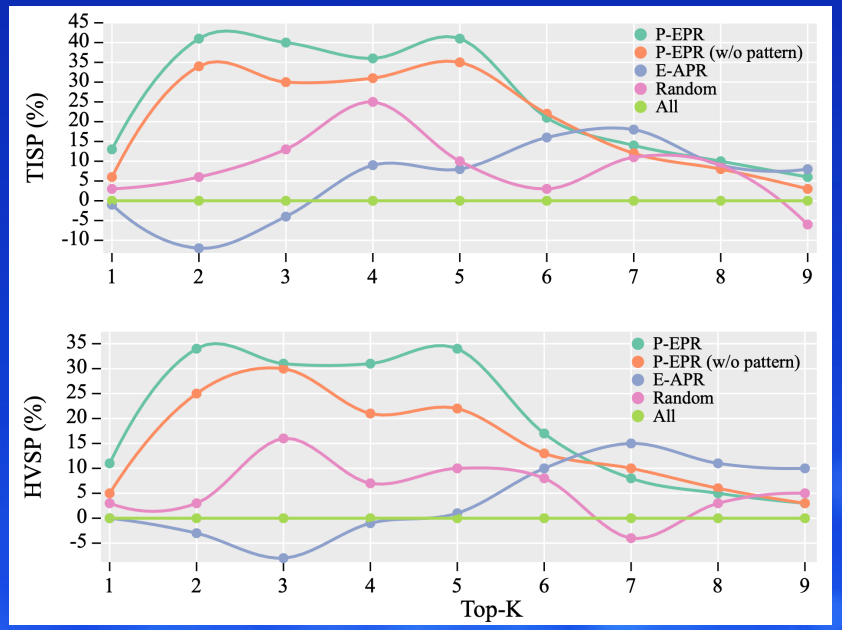
RQ1. 总体性能 & RQ2. 组件消融（1）

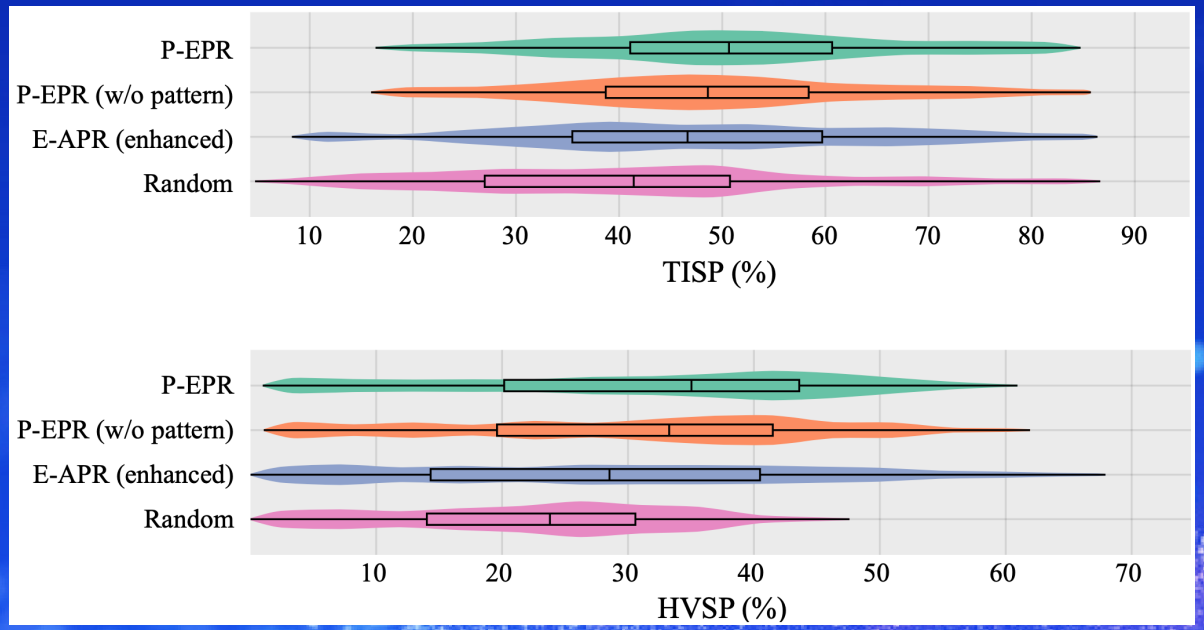
- 与Invoking All、Optimal Selection对比

	Top-1	Top-2	Top-3	Top-4	Top-5	Top-6	Top-7	Top-8	Top-9	Opt	All
# of correctly/plausibly fixed bugs	54/89	68/100	78/113	87/129	86/133	95/138	101/146	108/157	109/160	122/180	122/180
# of plausible patches	108	159	219	278	326	389	453	478	510	180	859
Tool Invocation Times (TISP)	1010 (33%)	1461 (39%)	1925 (41%)	2330 (44%)	2701 (38%)	3102 (41%)	3488 (41%)	3906 (42%)	4282 (38%)	395 (95%)	8295
Human Valdation Times (HVSP)	98 (20%)	117 (26%)	148 (27%)	175 (27%)	191 (22%)	214 (24%)	229 (25%)	257 (24%)	271 (21%)	180 (54%)	393*

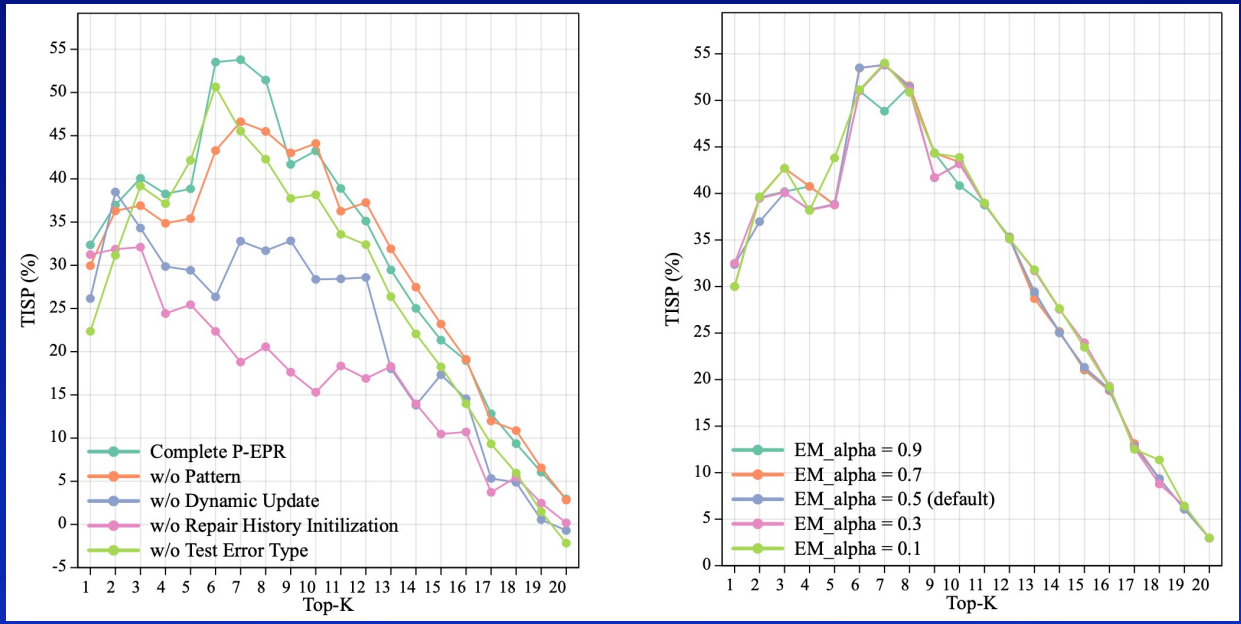
  - 取得接近性能时大幅度降低工具执行、人工检查的开销

- 与E-APR、E-APR(enhanced)对比；Pattern对P-EPR的贡献



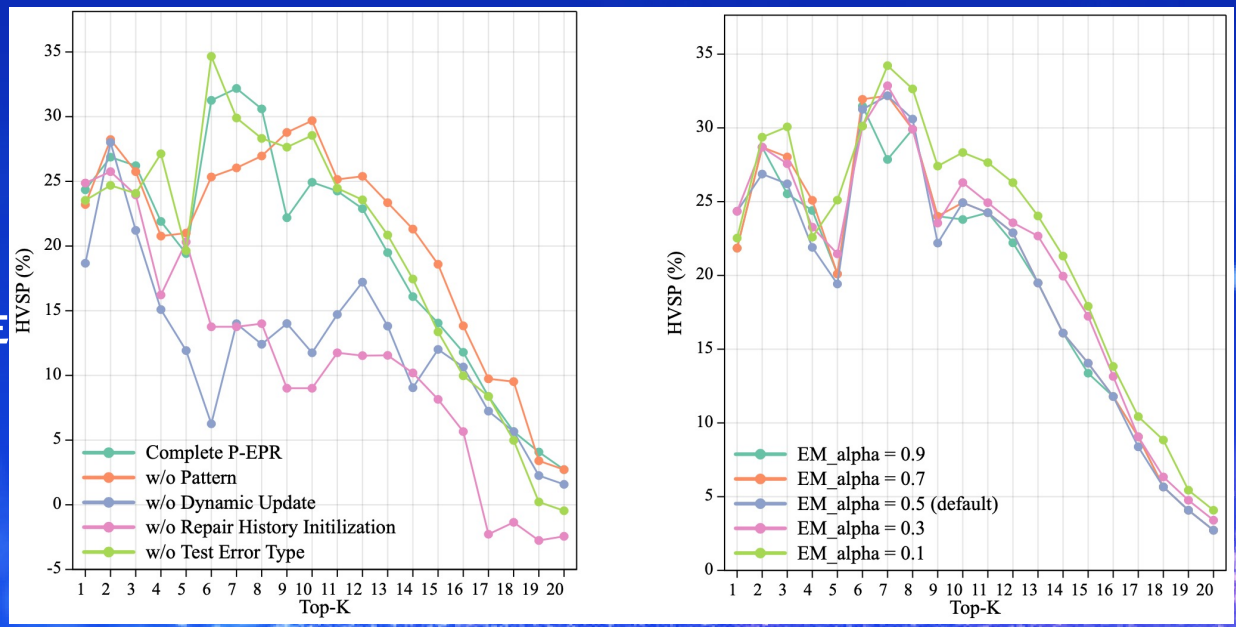


RQ2. 组件消融（2）



- 所有组件都有影响
- 影响较大的
  - 初始修复历史数据
  - 是否动态更新修复历史

- Pattern影响因子取值影响不大
- 工具调用开销波动较规律，人工则不是
- 人工验证开销是更加细粒度的评估



RQ3. 实用性能

- 4个NPR工具在Bears数据集中的83个Single-hunk的Bug上真实执行

Metrics	Single System				Selection Strategy		
	Recoder	CodeBERT-ft	RewardRepair	SequenceR	P-EPR (top-1)	Optimal*	All*
# of correct/plausible patches	10/21	12/25	12/21	14/28	16/28	22/37	22/37
Precision	48%	48%	57%	50%	57%	59%	/
Inference Time (min)	11	2	4	4	7	6	28
Machine Validation Time (hour)	57	54	58	52	52	40	221
Human Validation Time (min)	41	54	36	62	56	84	118
TISP	20%	29%	29%	38%	48%	75%	/
HVSP	5%	7%	15%	10%	20%	29%	/
GPU Memory Usage (GB)	19.1	3.84	7.32	8.17	7.83	/	/

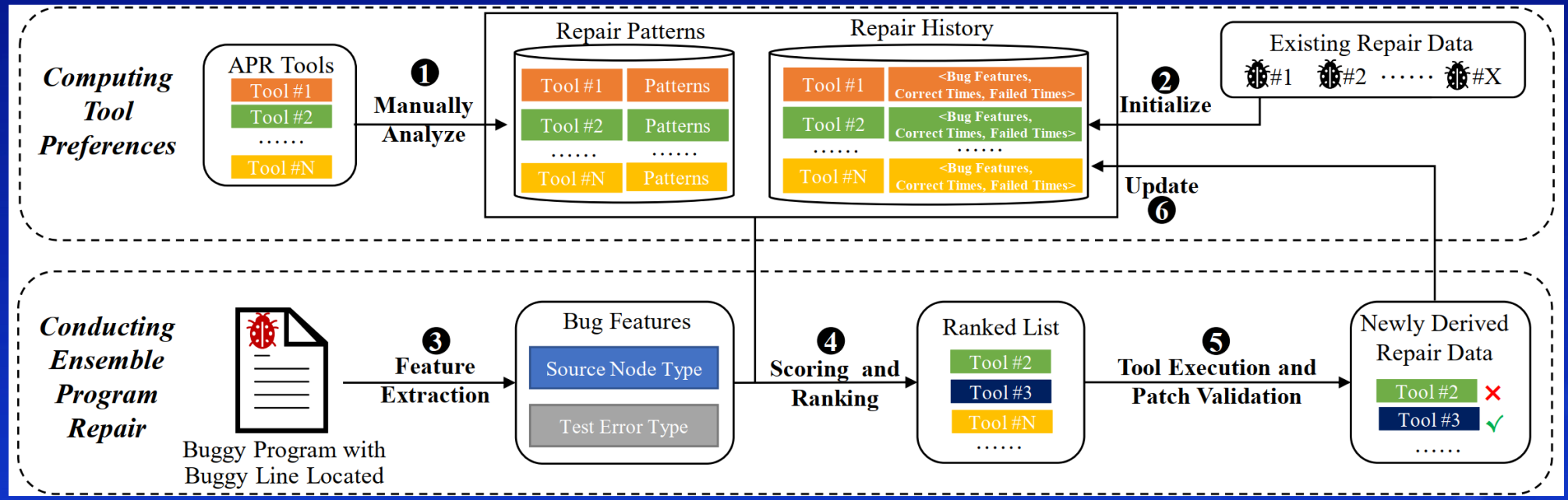
- P-EPR集成策略超越了所有单个NPR工具的修复性能：正确修复数量、精准率
- 单位修复正确的Bug节约的工具执行开销、人工验证开销都比单个工具要高很多
- 相比单独使用任何一个模型更加实用！

PART 05

总结、展望与补充讨论

总结

- 多种APR工具能力互补 ➔ 集成的优势 ➔ 基于学习的集成存在不足 ➔ P-EPR



一种实用的APR集成方案：充分发挥各APR工具的优势，在降低时间、计算资源开销的基础上提升修复性能；容易加入新的APR工具；随着历史数据积累的越多，效果越好。

展望1：集成方案的优化

- 提高Top 1选中正确APR的概率
  - 传统的APR工具根据其修复偏好关联Bug特征，能够有效匹配工具和Bug
  - 但基于学习的NPR工具主要依赖修复历史，类似用Bug之间的相似度替代NPR和Bug的关联
  - 尝试用“基于学习的策略”度量NPR和Bug之间的关联

收集<NPR, Bug, Yes/No>数据集，训练为Bug预测NPR的分类器，替换基于Repair History的计算模块

但是，该集成思路能正确修复的Bug数量，无法超越所有单个模型能修复的Bug并集

能力更强的APR工具；在生成补丁程序Token时集成不同的修复工具。

展望2：集成LLM-based Self-enhancing APR

- 定义：直接用LLM而非Fine-tuning的方式，fine-tuning有时间和费用的成本

C. S. Xia and L. Zhang, Less training, more repairing please: revisiting automated program repair via zero-shot learning. ESEC/FSE, 2022, pp. 959–971.

  - 基于LLM使用Chain-of-Thought、Agents、Additional Generation、Self-correction等技术
  - 仅使用Bug内部信息：上下文代码、错误代码的位置
  - 另使用Bug外部信息：测试用例、报错信息、调试信息等

- 修复能力较传统APR和NPR都更强，基于Repair History集成可能总是Top 1
- 除了Repair History，难以度量其与Bug之间的关联

将其放在固定的位次作为候选，例如尝试Top 3都不行，则尝试这类方法

补充：LLM-based Self-enhancing APR 泛化性 (1)

- 使用了大模型的自增强APR方法表现出非常突出的性能
  - SRepair[1]修复Defects4J数据集上近300个Bug(共500+)；非LLM方案最多能修复不到100个

[1] J. Xiang, X. Xu, F. Kong, M. Wu, H. Zhang, and Y. Zhang, "How far can we go with practical function-level program repair?" arXiv preprint arXiv:2404.12833, 2024.

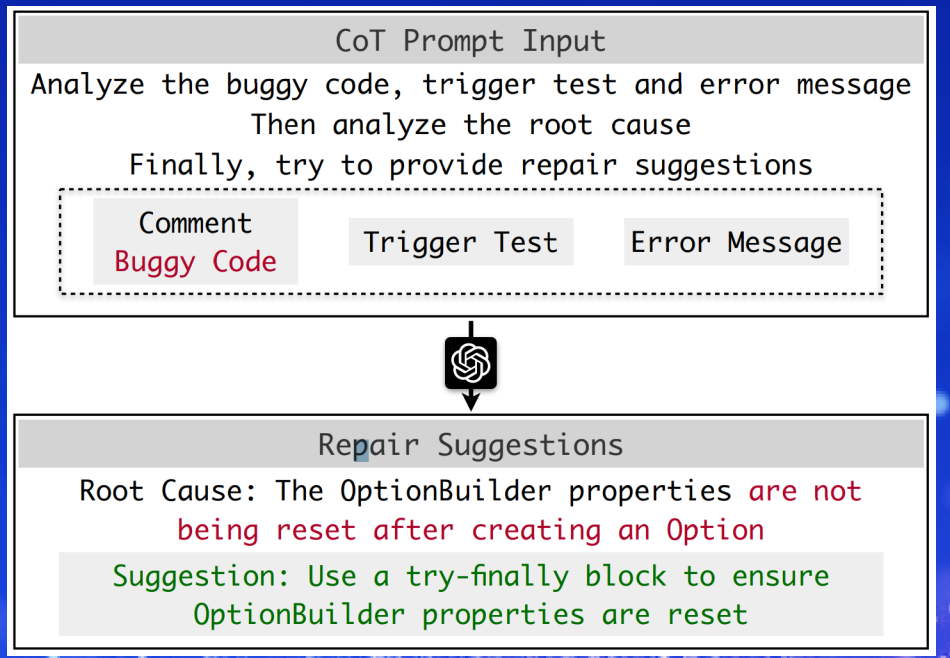
- 使用的额外信息
  - 缺陷报错信息
  - 触发缺陷的测试用例
  - 开发者或用户对缺陷的评论

不是所有缺陷修复场景下这都很容易获取

对不同场景的泛化性受到影响

测试了Defects4J、QuixBugs

SRepair在更多其他数据集上都能表现很好吗?



Magicoder

补充：LLM-based Self-enhancing APR 泛化性 (2)

- 简单扩展对SRepair修复能力的评估；对比相较基础的LLM的提升
  - 完整的Defects4J（835个Bug）
  - HumanEval-Java（163个Bug）
  - BugsInPy（493个中的54个Bug，借鉴TypeFix）

提升表现和另两个数据集有较大差异

Model or Method	**Defects4J** (835 bugs)		**HumanEval-Java** (163 bugs)		**BugsInPy** (54 bugs)	
	Number of Fixed Defects	*Enhancement Ratio*	*Number of Fixed Defects*	*Enhancement Ratio*	*Number of Fixed Defects*	*Enhancement Ratio*
GPT-3.5-Turbo	53	-	97	-	7	-
SRepair	137	258.5%	160	165.0%	7	100.0%
FixAgent	126	237.7%	114	117.5%	9	128.6%
SRepair w/o	84	158.5%	146	150.5%	5	71.4%
FixAgent w/o	104	196.2%	103	106.2%	6	85.7%
Test Report Prompt	62	117.0%	113	116.5%	7	100.0%

LLM本来就知道的才会被激发；本来不知道的没法激发。

是不是GPT-3.5-Turbo对前两个数据集知道的就比后面一个多？

补充：LLM-based Self-enhancing APR 泛化性 (3)

- 如何度量LLM对数据集的掌握情况——记忆程度？
  - 如果LLM对不同数据集的记忆程度不同，且记忆多效果好，记忆少效果差
  - 只能初步说明：记忆程度是一个影响自增强方法泛化能力的潜在因素

LLM补全的代码和真实代码之间的Tpye-1克隆：数量 和 密度

Metric	Benchmark		
	Defects4J	*HumanEval-Java*	*BugsInPy*
ATCC	**0.36**	0.16	0.11
ATCH	**0.03**	0.02	0.02
AP	6.64E+275	**1.48E+38**	1.94E+286
RAP	1.25	**1.10**	1.38

- 增大记忆会提升效果吗？ + 降低记忆会降低效果吗？

构造包含BugsInPy general-level和bug-level知识的数据集，并微调Base LLM；对Srepair第二个LLM也使用微调后的LLM。

Model or Method	Initial		SFT	
	#of Fixed	*PI Ratio*	*#of Fixed*	*PI Ratio*
Base LLM	7	-	**15**	-
SRepair w/o E	5	71.4%	**13**	**86.7%**
FixAgent w/o E	6	85.7%	**20**	**133.3%**
RefinedTRP	7	100.0%	**19**	**126.7%**
SRepair_ft w/o E	-	-	17	113.3%

补充：LLM-based Self-enhancing APR 泛化性 (4)

- 增大记忆会提升效果吗？ + 降低记忆会降低效果吗？

Metric	Initial	Memorization-decayed Defects4J		
		RC_486	*VR_631*	*US_597*
ATCC	**0.34**	0.25	0.21	0.18
ATCH	0.03	0.03	0.02	0.01
AP	**1.71E+153**	3.08E+186	2.71E+198	2.46E+194
RAP	**1.32**	1.36	1.48	1.42

对Defects4J分别做三种类型的变换：降低LLM对新数据集的记忆程度，使用新数据集测试记忆与效果之间的关系。

Model or Method	Initial		ReorderCondition_486		VariableRenaming_631		UnusedStatement_597	
	#of Fixed	*PI Ratio*	*#of Fixed*	*PI Ratio*	*#of Fixed*	*PI Ratio*	*#of Fixed*	*PI Ratio*
GPT-3.5-Turbo	**53**	-	38	-	30	-	23	-
SRepair w/o E	**84**	**158.5%**	51	134.2%	41	136.7%	31	134.8%
FixAgent w/o E	**104**	**196.2%**	40	105.3%	38	126.7%	28	121.7%
RefinedTRP	**62**	**117.0%**	33	86.8%	32	106.7%	26	113.0%

- Base LLM对评测数据集的记忆程度会影响对自增强方法性能提升的评测结果
  - 评测时使用多种不同的评测数据集合、尽可能使用包含新收集数据的评测数据集
  - 使用新的Bug数据持续更新APR的评测数据集

THANKS