AiDD AI$^+$软件研发数字峰会
AI$^+$ software Development Digital summit

AI驱动软件研发
全面进入数字化时代

中国·深圳 11.24-25

AI$^+$
software
Development
Digital
summit

缺陷自动修复的“卡脖子”问题——补丁正确性验证技术

文明 华中科技大学

科技生态圈峰会 + 深度研习

——1000 + 技术团队的共同选择

K+峰会

K+峰会 深圳站

K+全球软件研发行业创新峰会

会议时间：2024.05.24-25

K+峰会 上海站

K+全球软件研发行业创新峰会

会议时间：2024.09.20-21

AiDD峰会

AiDD峰会 深圳站

AI⁺软件研发数字峰会

会议时间：2023.11.24-25

AiDD峰会 北京站

AI⁺软件研发数字峰会

会议时间：2024.07.19-20

AiDD峰会 深圳站

AI⁺软件研发数字峰会

会议时间：2024.11.15-16

演讲嘉宾

文明

华中科技大学 副教授

文明博士主要聚焦软件安全、软件测试与分析、以及代码大模型安全等研究，在软件工程领域累计发表了CCF-A类推荐会议或期刊40余篇，其他高水平论文10余篇。主持国家自然科学基金青年项目、面上项目、以及包括华为胡杨林基金系统软件专项在内的多项企业合作项目，参与湖北省重点研发项目等重要课题，担任了中国计算机学会系统软件、以及软件工程专委会委员。他常年担任TSE，TOSEM，TDSC等CCF-A类国际期刊的审稿人，以及CCF-A/B类会议ASE 2021/2023, ESEC/FSE 2022/2024, SANER 2022, ISSRE 2022/2023的程序委员会委员。同时也荣获了Internetware 2023杰出论文奖、ACM 新星奖 2023（武汉）、以及入选了第七届中国科协青年人才托举工程计划。

目 录
CONTENTS

1. 程序自动修复与补丁正确性验证
2. 基于表示学习的补丁验证
3. 基于缺陷定位的补丁排序
4. 总结与展望

PART 01

程序自动修复与补丁正确性验证

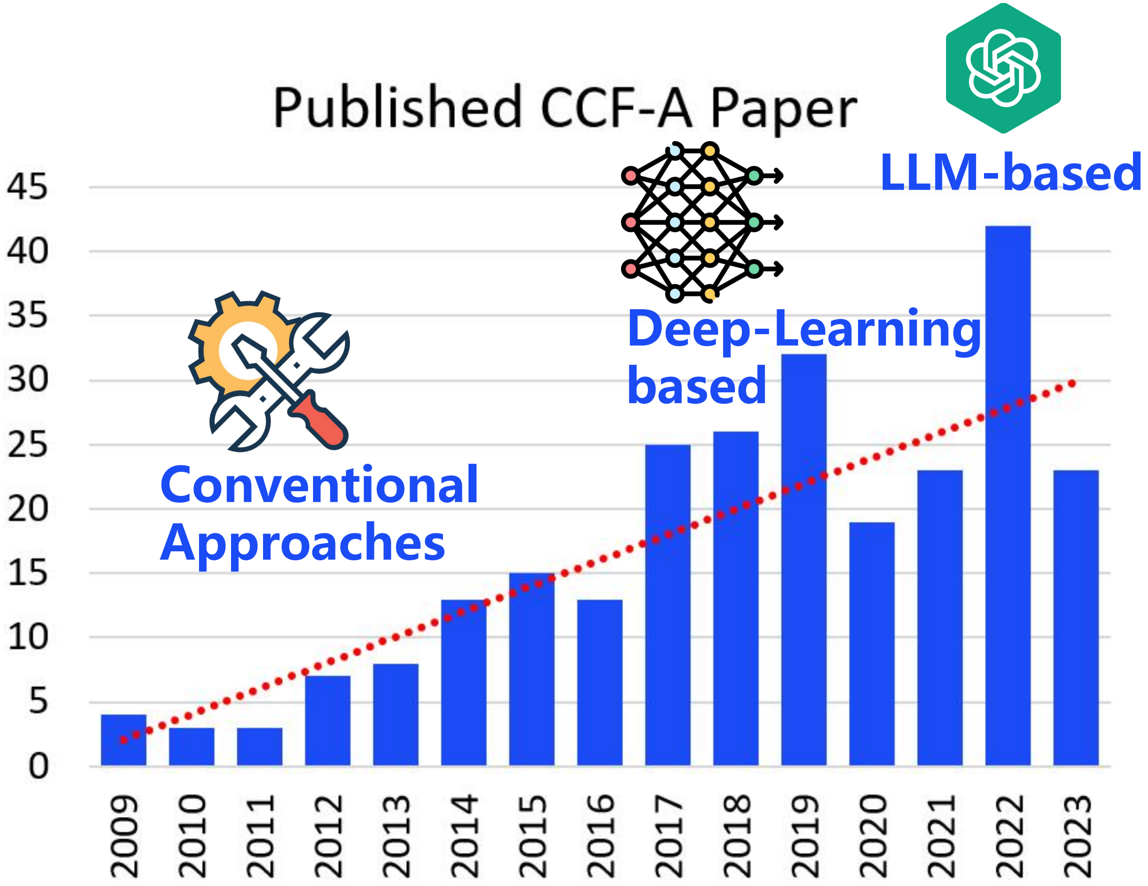
▶ 程序自动修复—发展历程

Background

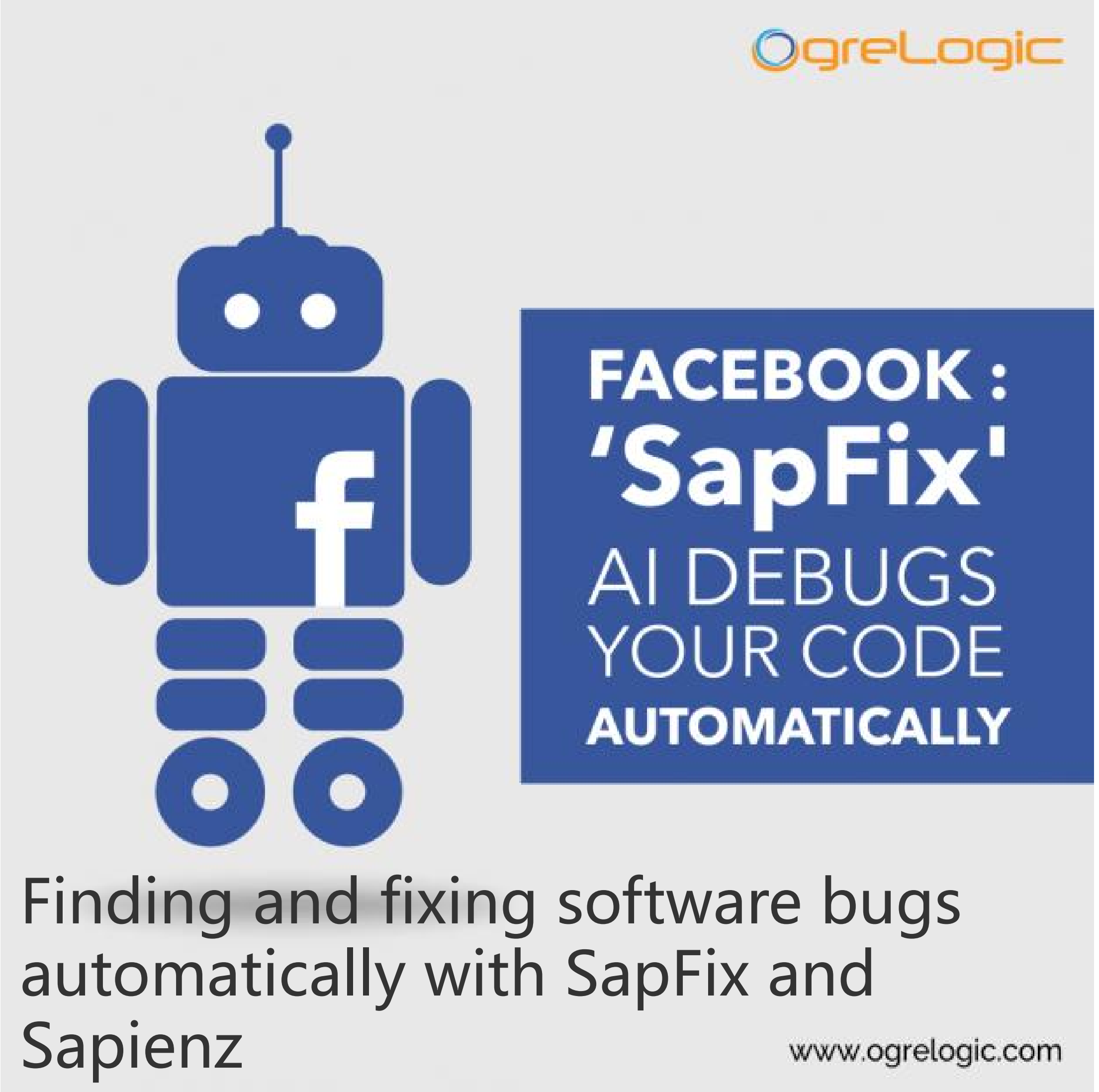
- The problem was first formulated in **2005** [Jobstmann, CAV 05]

- Since **GenProg** [Weimer, ICSE 09] was proposed in 2009, APR has received huge research interests.

https://program-repair.org/bibliography.html



程序自动修复—工业界实践



OgreLogic

FACEBOOK : 'SapFix' AI DEBUGS YOUR CODE AUTOMATICALLY

Finding and fixing software bugs automatically with SapFix and Sapienz

www.ogrelogic.com

MIT Technology Review

Business Impact

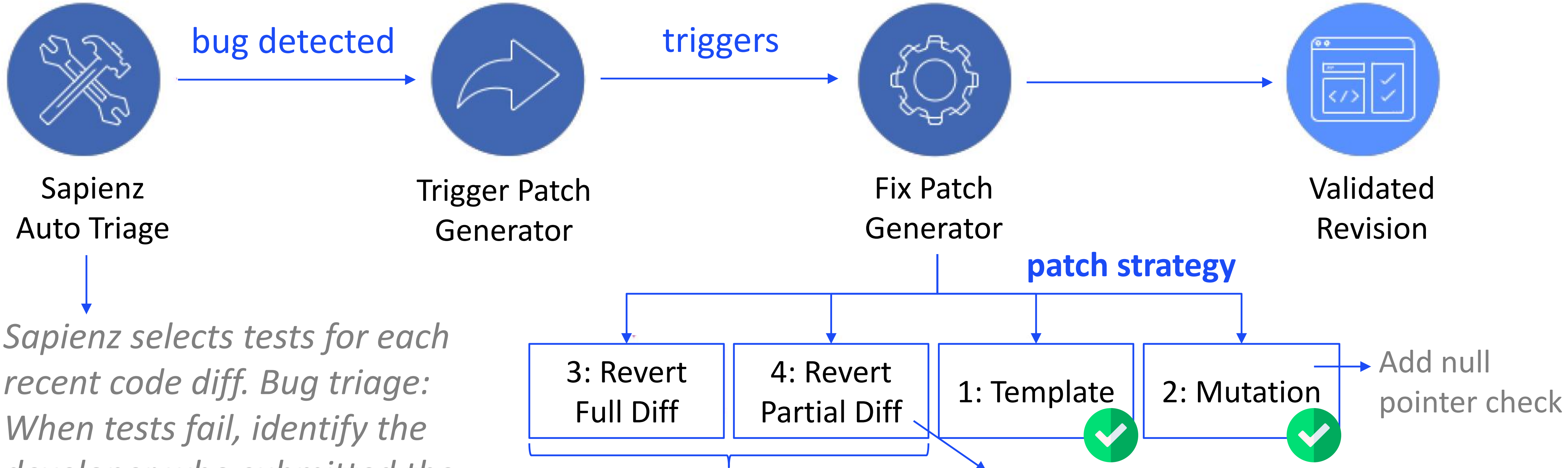
A bot disguised as a human software developer fixes bugs

The automated programmer, called Repairnator, wrote patches good enough to fool actual human engineers.

by Emerging Technology from the arXiv October 23, 2018

"In this world nothing can be said to be certain, except death and taxes," wrote Benjamin Franklin in 1789. Had he lived in the modern era, Franklin may well have added "software bugs" to his list.

程序自动修复— SapFix修复框架



Whether all tests are passed

bug detected

triggers

Sapienz Auto Triage

Trigger Patch Generator

Fix Patch Generator

Validated Revision

patch strategy

3: Revert Full Diff

4: Revert Partial Diff

1: Template ✓

2: Mutation ✓

Add null pointer check

Sapienz selects tests for each recent code diff. Bug triage: When tests fail, identify the developer who submitted the code diff

Able to identify the diffs that possibly induce the failing tests

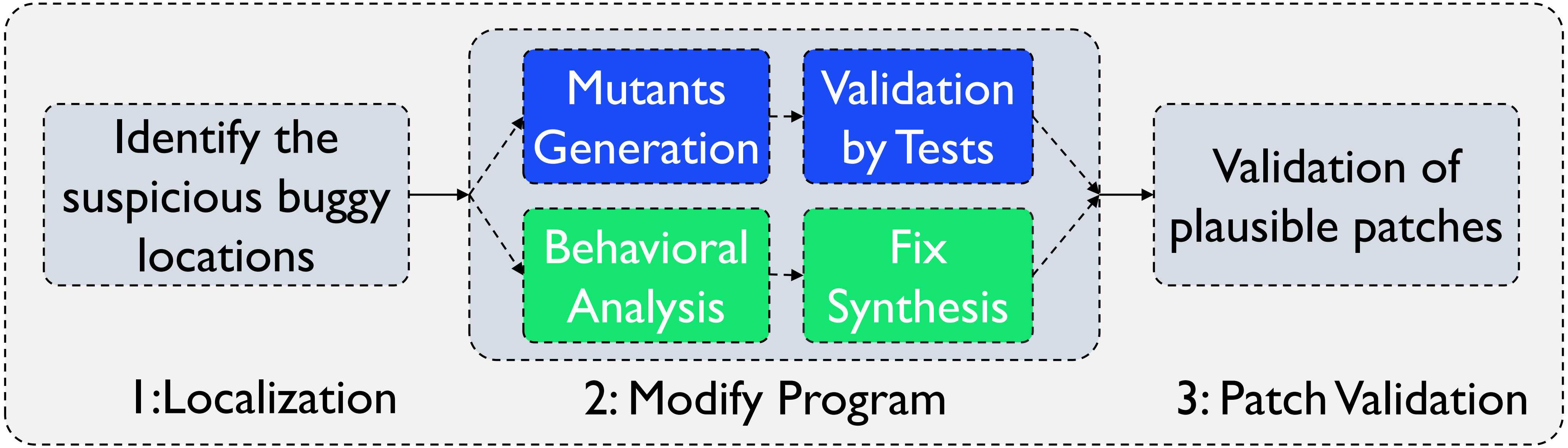
Revert subsets of diffs so that the revised program compiles successfully

Credit: Facebook

程序自动修复—基本流程

General Process

Conventional Automated Program Repair Approaches


Identify the suspicious buggy locations

Mutants Generation

Validation by Tests

Behavioral Analysis

Fix Synthesis

Validation of plausible patches

1:Localization

2: Modify Program

3: Patch Validation

🟦 Search-based (also known as generate-and-validate)

🟩 Semantics-based

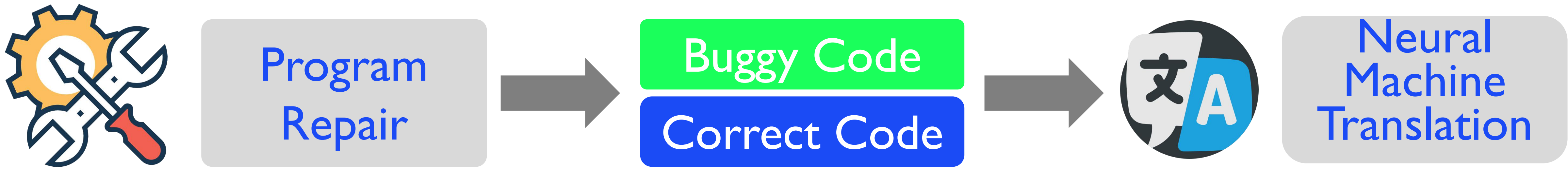
▶ 程序自动修复技术—基本流程

General Process

Deep-Learning Driven
Automated Program Repair

github
SOCIAL CODING

sourceforge

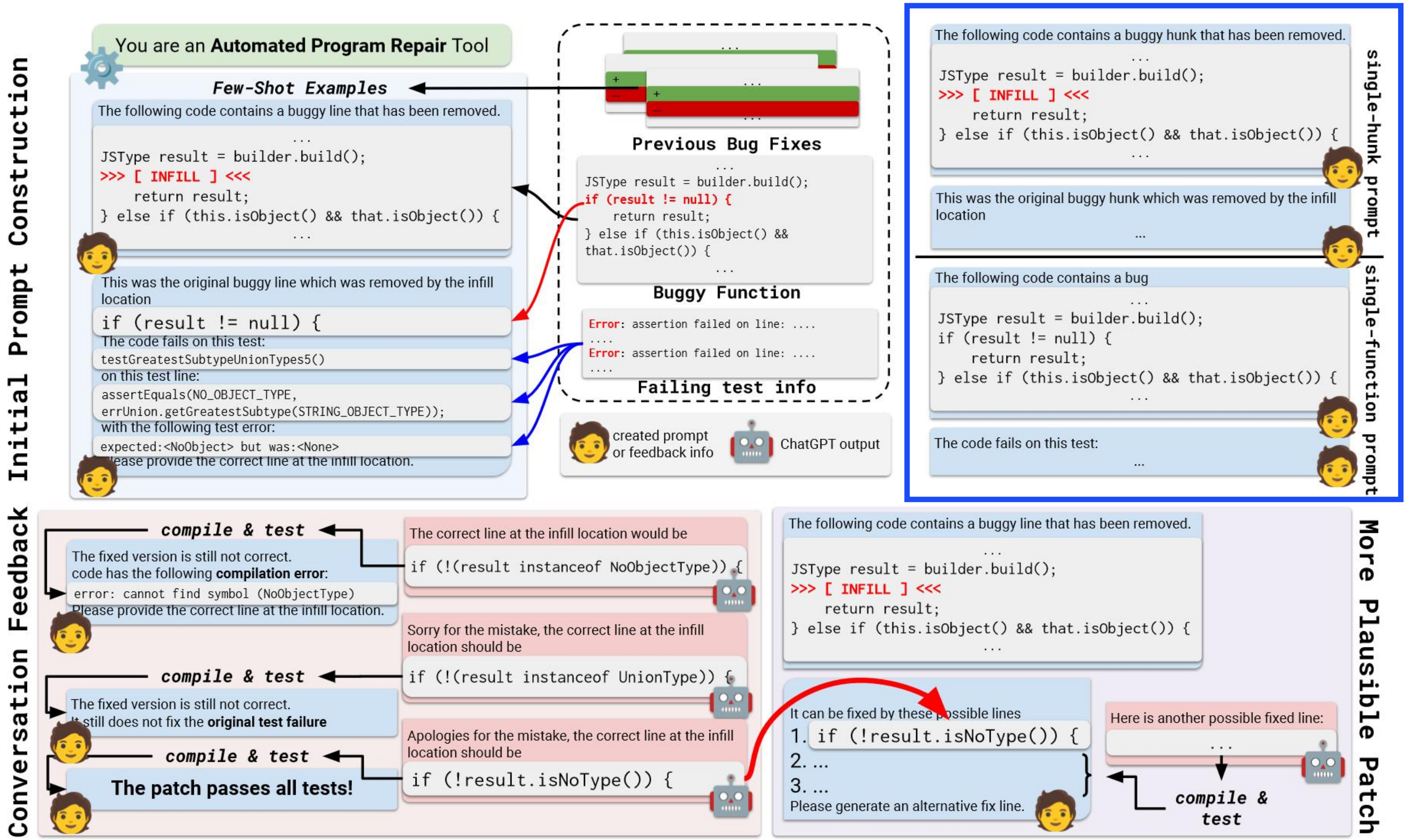


- Lutellier, Thibaud, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. "**Coconut: combining context-aware neural translation models using ensemble for program repair**." *In the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pp. 101-114. 2020.
- Jiang, Nan, Thibaud Lutellier, and Lin Tan. **"CURE: Code-aware neural machine translation for automatic program repair."** *In the 43rd International Conference on Software Engineering (ICSE)*, pp. 1161-1173. IEEE, 2021.

程序自动修复技术—基本流程

General Process

Large-Language Model based
Automated Program Repair



You are an **Automated Program Repair** Tool

The following code contains a buggy line that has been removed.

```
...
JSType result = builder.build();
>>> [ INFILL ] <<<
    return result;
} else if (this.isObject() && that.isObject()) {
...
```

This was the original buggy line which was removed by the infill location

```
if (result != null) {
```

The code fails on this test:

```
testGreatestSubtypeUnionTypes5()
```

on this test line:

```
assertEquals(NO_OBJECT_TYPE,
errUnion.getGreatestSubtype(STRING_OBJECT_TYPE));
```

with the following test error:

```
expected:<NoObject> but was:<None>
```

Please provide the correct line at the infill location.

```
...
JSType result = builder.build();
if (result != null) {
  return result;
} else if (this.isObject() &&
that.isObject()) {
...
```

```
Error: assertion failed on line: ....
....
Error: assertion failed on line: ....
....
```

The following code contains a buggy hunk that has been removed.

```
...
JSType result = builder.build();
>>> [ INFILL ] <<<
    return result;
} else if (this.isObject() && that.isObject()) {
...
```

This was the original buggy hunk which was removed by the infill location

...

The following code contains a bug

```
...
JSType result = builder.build();
if (result != null) {
    return result;
} else if (this.isObject() && that.isObject()) {
...
```

The code fails on this test:

...

The fixed version is still not correct.
code has the following **compilation error**:

```
error: cannot find symbol (NoObjectType)
```

Please provide the correct line at the infill location.

The correct line at the infill location would be

```
if (!(result instanceof NoObjectType)) {
```

Sorry for the mistake, the correct line at the infill location should be

```
if (!(result instanceof UnionType)) {
```

The fixed version is still not correct.
It still does not fix the **original test failure**

Apologies for the mistake, the correct line at the infill location should be

```
if (!result.isNoType()) {
```

The patch passes all tests!

The following code contains a buggy line that has been removed.

```
...
JSType result = builder.build();
>>> [ INFILL ] <<<
    return result;
} else if (this.isObject() && that.isObject()) {
...
```

It can be fixed by these possible lines

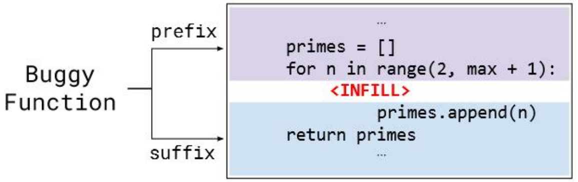
1. if (!result.isNoType()) {
2. ...
3. ...

Please generate an alternative fix line.

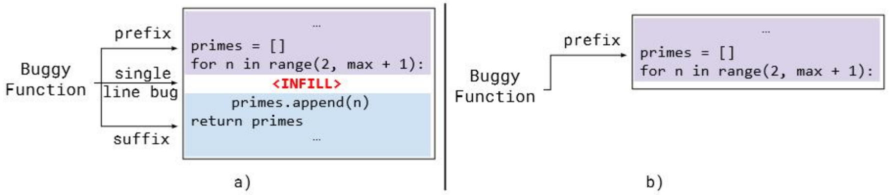
Here is another possible fixed line:

...

Prompt Design



```
...
primes = []
for n in range(2, max + 1):
<INFILL>
    primes.append(n)
return primes
...
```



```
...
primes = []
for n in range(2, max + 1):
<INFILL>
    primes.append(n)
return primes
...
```

```
...
primes = []
for n in range(2, max + 1):
```

- Chunqiu Steven Xia and Lingming Zhang. "**Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT**." ArXiv abs/2304.00385 (2023):

▶ 程序自动修复技术—补丁过拟合

- *Overfitting*

Weak Test Suite

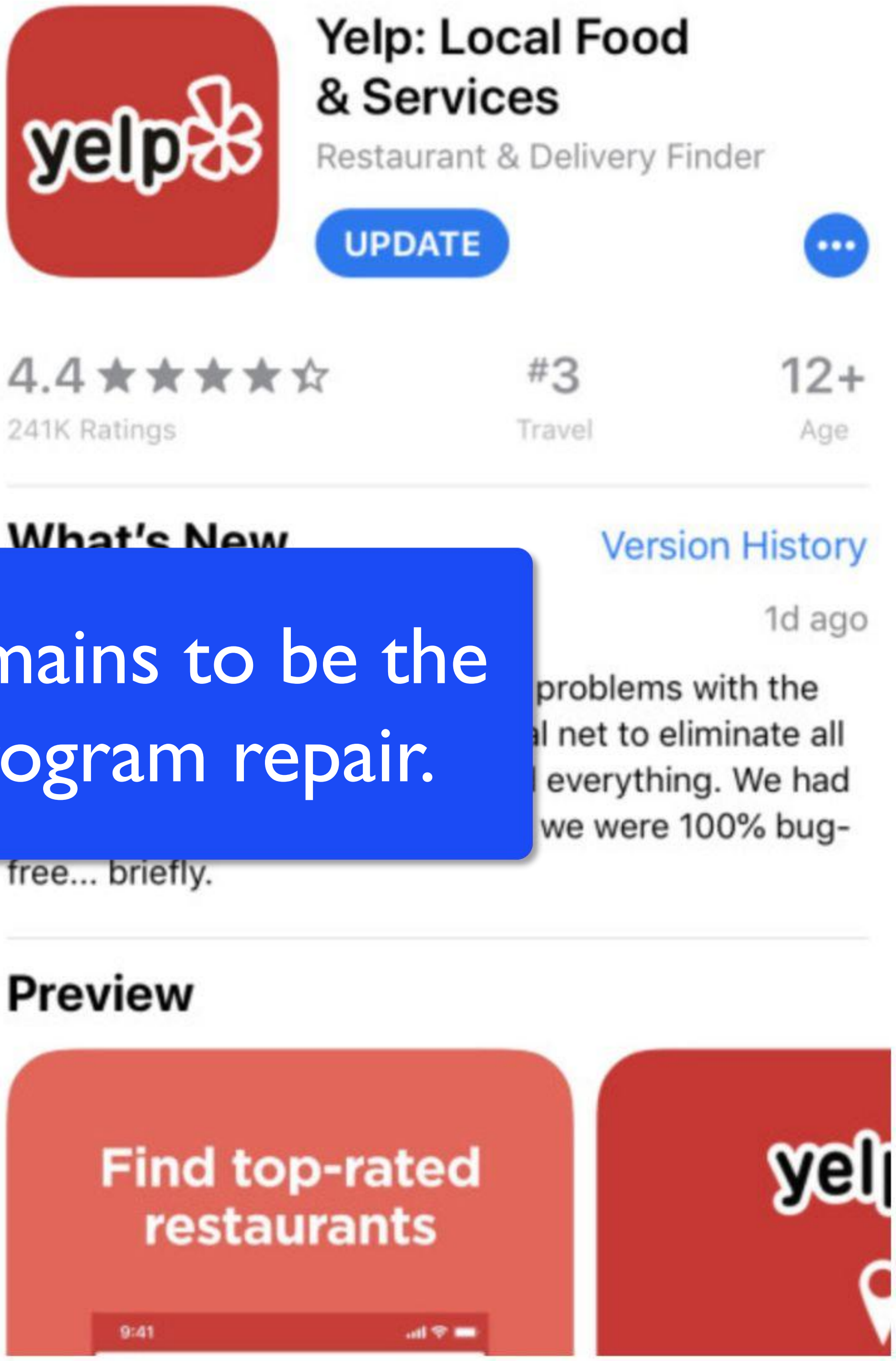
Tests are imperfect metric of program correctness, whi
discriminating between ***correct fixes*** and those patches

Techniques

```
          De
     if  (condition)
            return
```

Patch correctness assessment remains to be the open challenge of automated program repair.

Many of the patches generated by existing approaches ***function deletion.***



Yelp: Local Food & Services

Restaurant & Delivery Finder

UPDATE

4.4 ★★★★☆ 241K Ratings

#3 Travel

12+ Age

What's New

Version History

1d ago

problems with the
l net to eliminate all
everything. We had
we were 100% bug-
free... briefly.

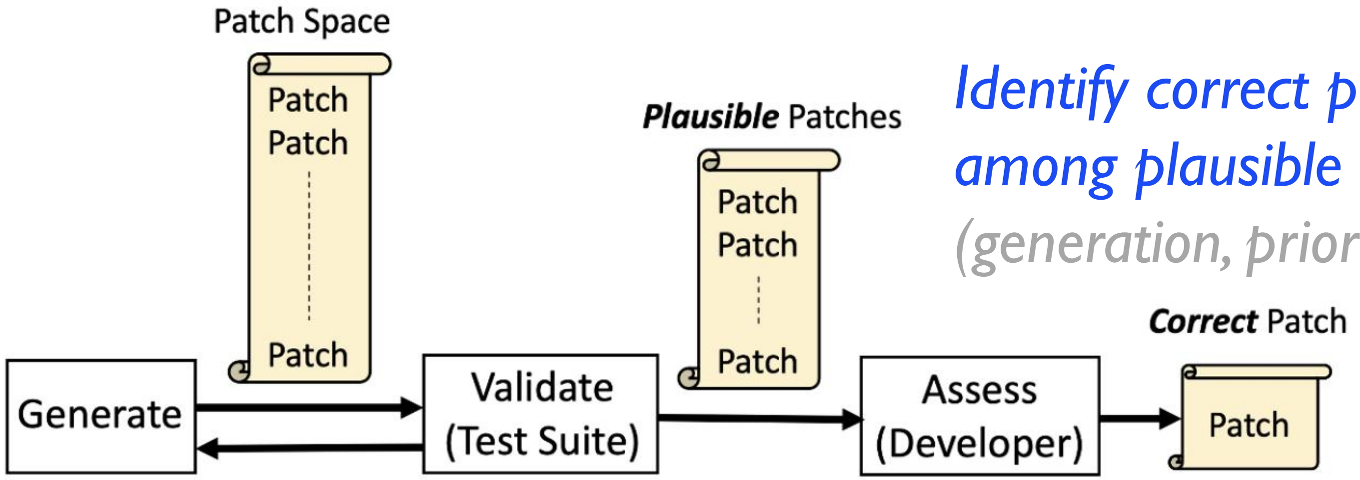
Preview

Find top-rated restaurants

9:41

补丁正确性验证技术—基本定义

APCA:
Automated
Patch
Correctness
Assessment



Identify correct patches among plausible ones.
(generation, prioritization)

Plausible patch: a patch that passes the test suite is a plausible patch;

Correct patch: a plausible patch that indeed fixes the target bug is deemed correct;

Overfitting patch: a plausible patch that actually does not fix the target bug.

> *"APR techniques generate more overfitting patches than correct ones on real bugs" – [1][2]*

> *"Patches overfit to the test suite, often breaking undertested functionality." – [3]*

[1] Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. 2015. **An analysis of patch plausibility and correctness for generate-and-validate patch generation systems**. *In Proceedings of the 24th International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 24–36.
[2] Xuan Bach D.Le, FerdianThung, David Lo,and Claire Le Goues. 2018. **Overfitting in semantics-based automated program repair.** *Empirical Software Engineering* 23, 5 (2018), 3007–3033.
[3] Edward K Smith, Earl T Barr, Claire Le Goues, and Yuriy Brun. 2015. **Is the cure worse than the disease? overfitting in automated program repair**. *In Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*. ACM, 532–543.

补丁正确性验证技术

Our Contribution

CapGen
[ICSE 2018]

Automated program repair based on context information (**patch prioritization**)

Extensive Study
[ASE 2020]

The first large-scale study to investigate the **patch correctness assessment** problem

Acro
[TSE 2021]

A automated repair framework for on-chain smart contract (**patch validation**)

Cache
[TOSEM 2022]

An advanced **patch assessment tool** based on context-aware code change embedding

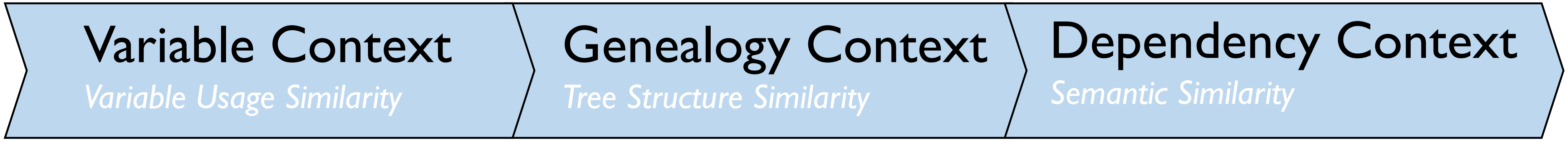
Dataset
[ESEC/FSE 2023]

A large-scale dataset for **patch correctness assessment**

▶ 补丁正确性验证技术—补丁排序

Patch Prioritization

Intuition *A fixing ingredient should be applied to the location with similar contexts compared with the location where it is extracted*



Variable Context

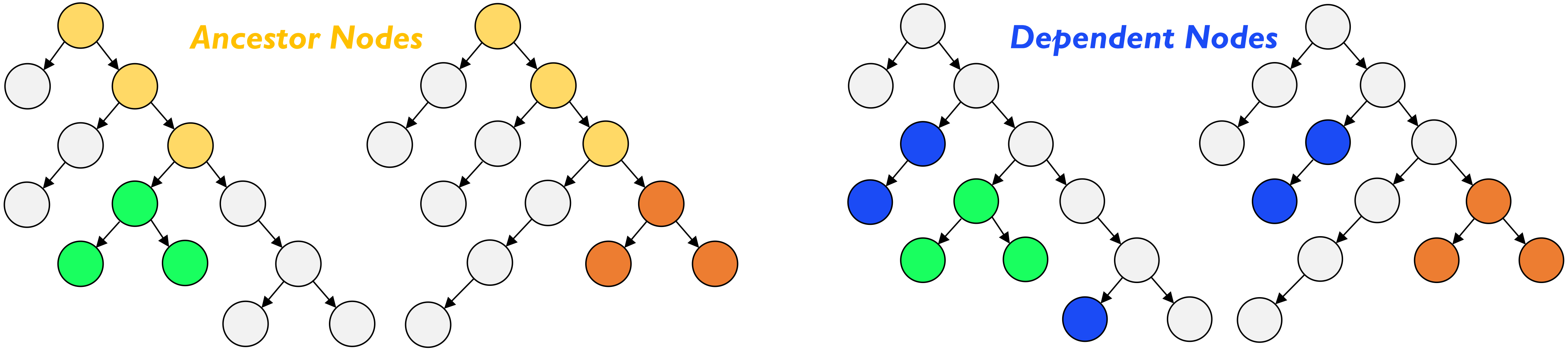
Variable Usage Similarity

Genealogy Context

Tree Structure Similarity

Dependency Context

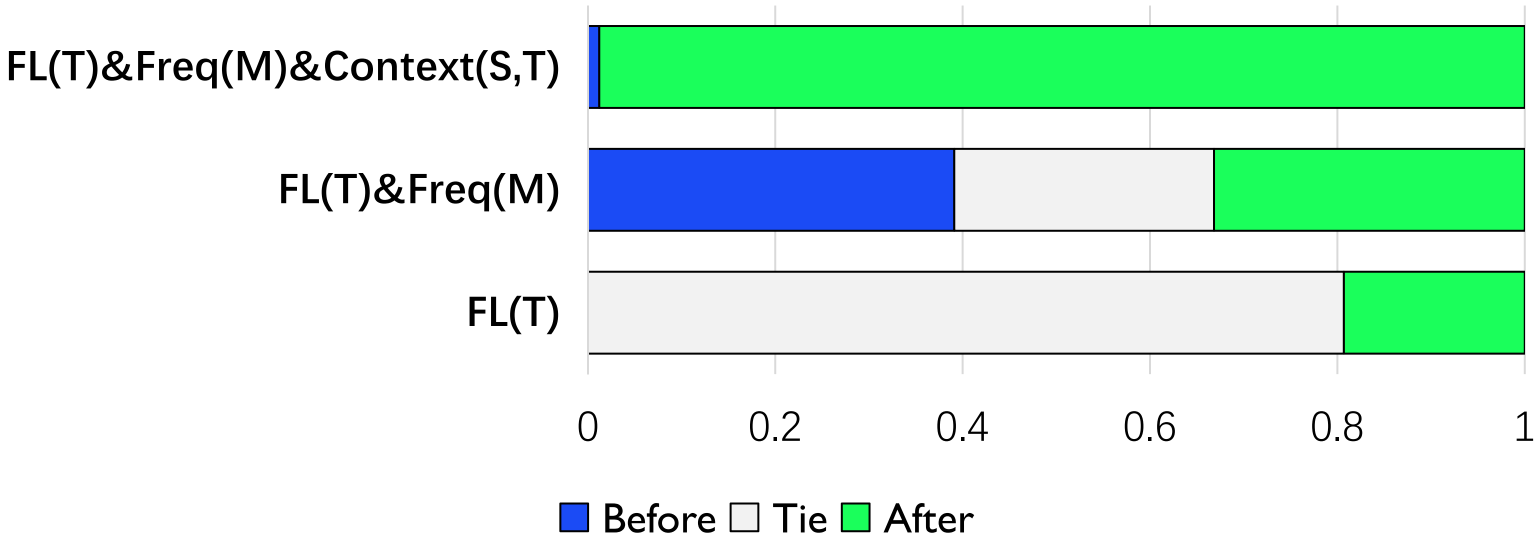
Semantic Similarity



▶ 补丁正确性验证技术—补丁排序

Patch Prioritization

$$Correctness\ Score(<T,M,S>) = FL(T) * Freq(M) * Context(S,T)$$



The relative ranks of the incorrect plausible patches

- Our context-aware prioritization strategies can rank of the correct patches in prior to 98.78% of the incorrect plausible ones

⬇

Avoid Overfitting

补丁正确性验证技术—发展历程

Overview **APCA:** Automated Patch Correctness Assessment

“A correct patch is often syntactically and semantically proximate to the original program”

Static Feature Based

Anti-patterns [FSE 2016]

Check if the code change in the patch violates pre-defined rules.

ssFix [ASE 2017]

- TokenStrct: similarity of structural tokens
- TokenConpt: similarity of conceptual tokens

S3 [FSE-2017]

- ASTDist: number of changed AST nodes
- ASTCosDist: distance of distinct AST node types
- VariableDist: distance of locations of variables and constants

CapGen [ICSE-2018]

- VariableSimi: similarity of variables
- SyntaxSimi: similarity of syntactic structures
- SemanticSimi: similarity of contextual nodes

补丁正确性验证技术—发展历程

Overview **APCA:** Automated Patch Correctness Assessment

DiffTGen [ISSTA 2017] **Opad** [FSE 2017] **Patch-Sim** [ICSE 2018]

Daikon [SANER 2020]

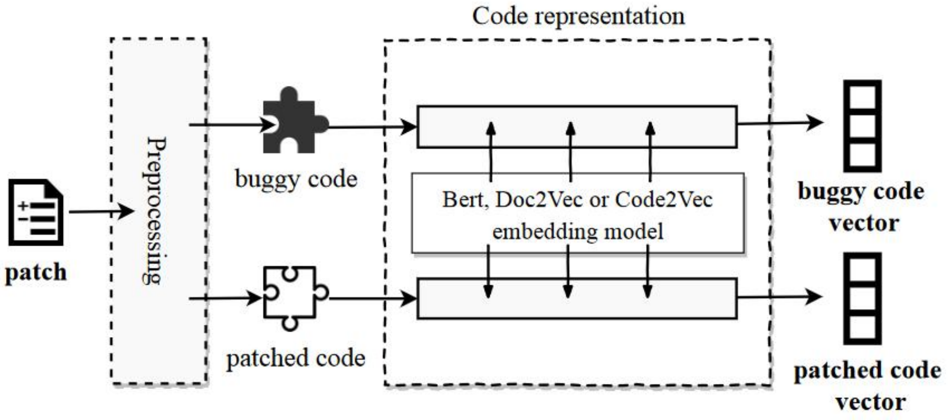
补丁正确性验证技术—发展历程

Overview **APCA:** Automated Patch Correctness Assessment

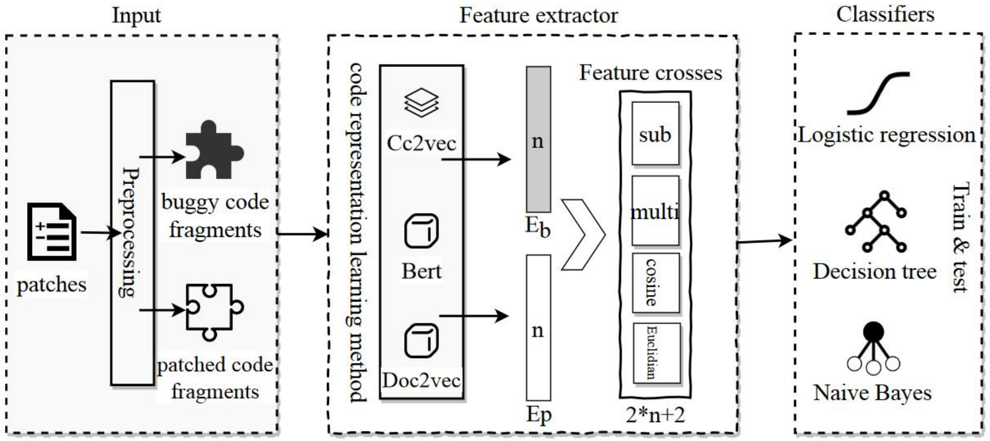
Static Feature Based

Dynamic Based

Embedding Based



Embedding Patches



Patch Classification [ASE 2020]

补丁正确性验证技术—发展历程

Overview **APCA:** Automated Patch Correctness Assessment

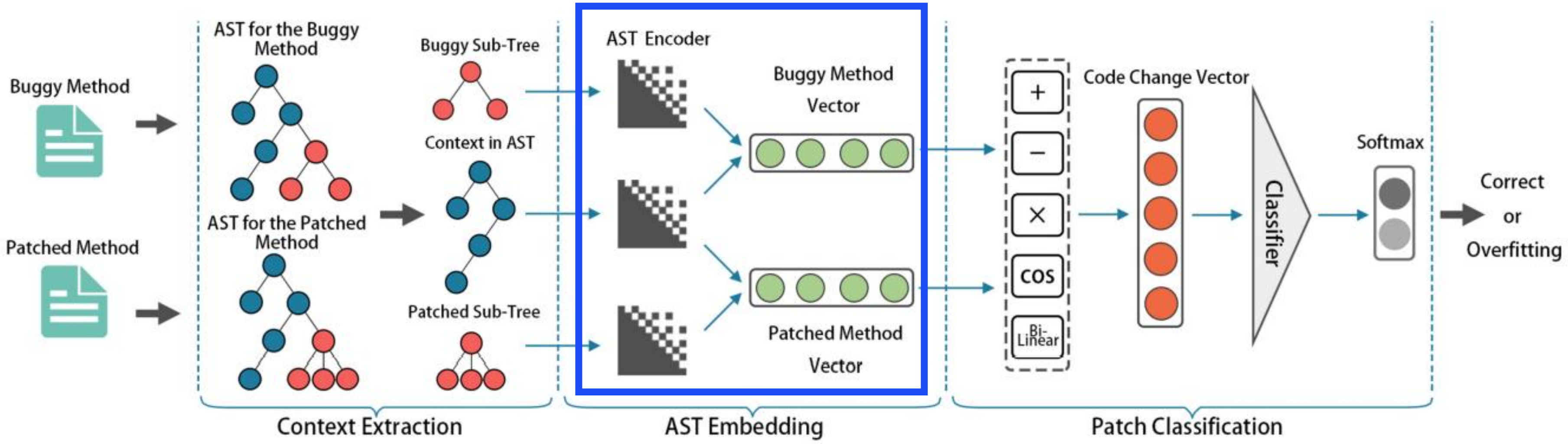
Static Feature Based

Dynamic Based

Embedding Based

Cache

[TOSEM 2022]



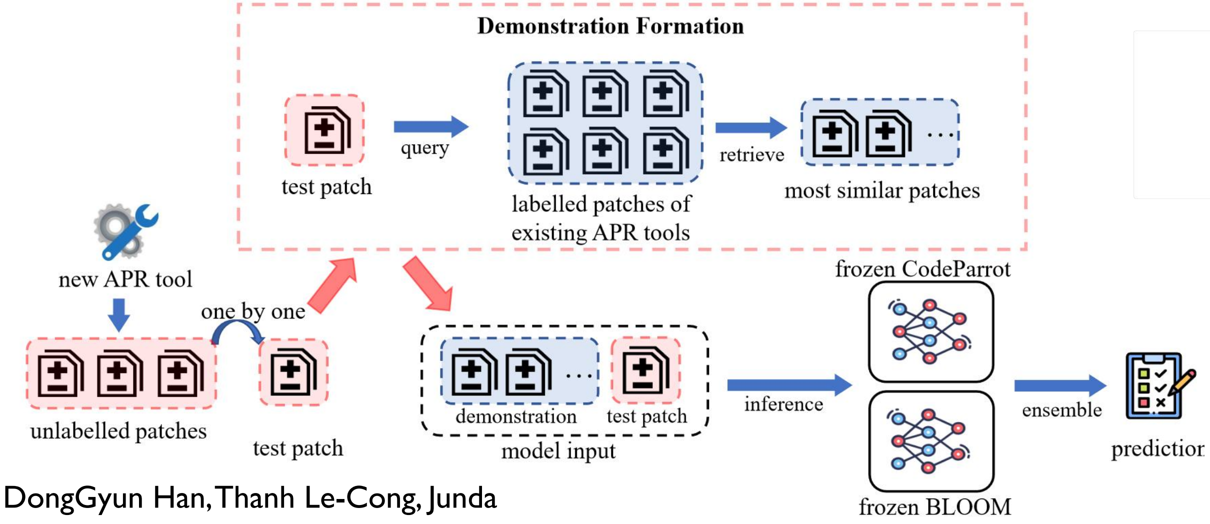
补丁正确性验证技术—发展历程

Overview **APCA:** Automated Patch Correctness Assessment

- Static Feature Based
- Dynamic Based
- Embedding Based
- Large Pre-Trained Model Based

PatchZero

[Arxiv 2023]



a BigScience initiative
BLOOM
176B params · 59 languages · Open-access

CodeParrot
& Friends

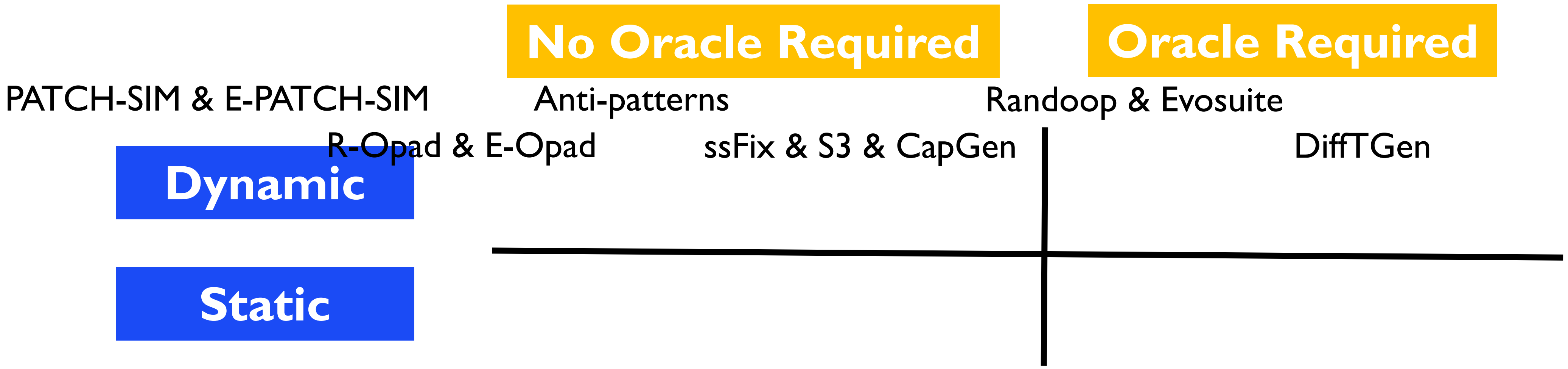
Prompt Template

$f_{prompt}(x) =$
$“[X]\ Q:\ It\ was\ wrong\ or\ correct?\ A:\ It\ was\ [Z].”$

$$V = \begin{cases} +: wrong, \\ -: correct, \end{cases}$$

Zhou, Xin, Bowen Xu, Kisub Kim, DongGyun Han, Thanh Le-Cong, Junda He, Bach Le, and David Lo. "Patchzero: Zero-shot automatic patch correctness assessment." arXiv preprint arXiv:2303.00202 (2023).

补丁正确性验证技术—实证研究



Which types of technique are more effective in identifying correct patches?
Are existing techniques complementary to each other?

Empirical Investigation

Technique	**9** different techniques and **3** heuristics based on **8** static code features
Patch	**902** patches automatically generated by **21** APR tools from **4** different categories

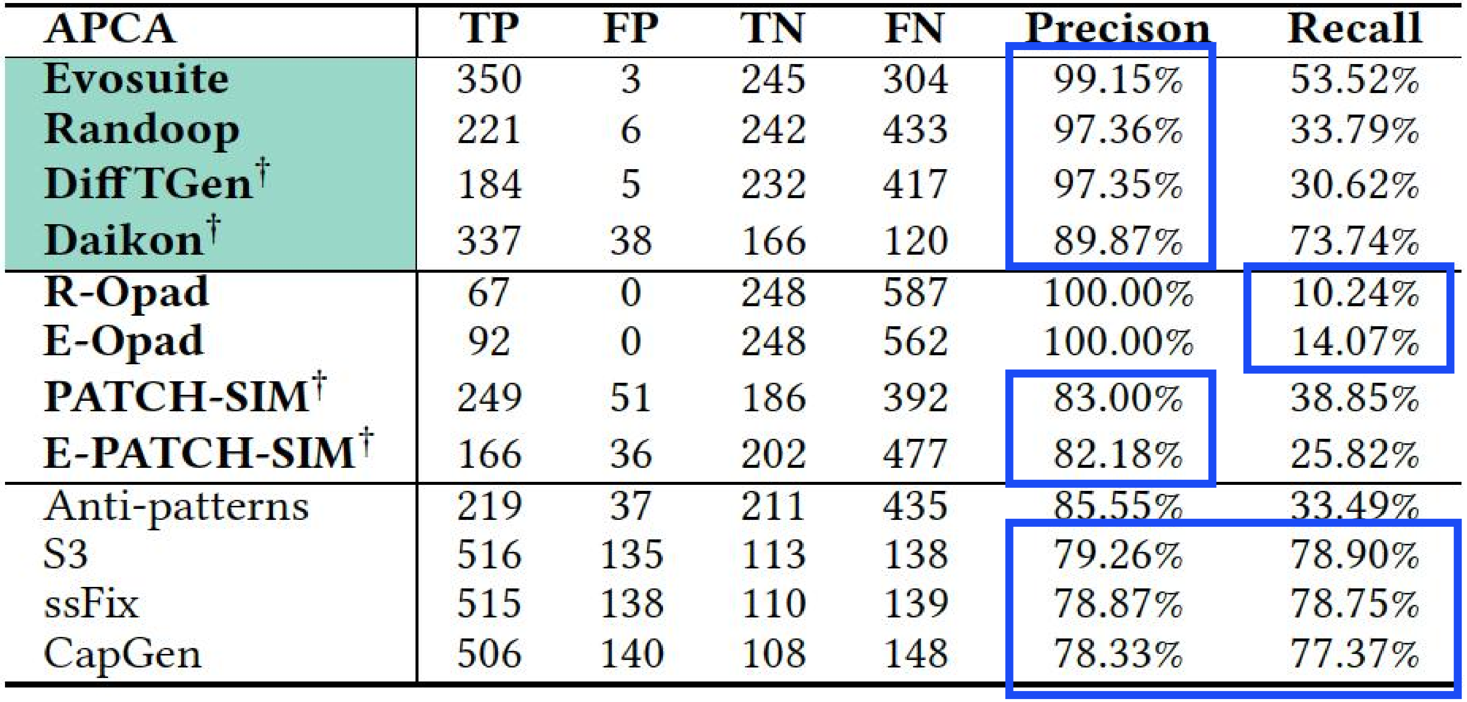
▶ 补丁正确性验证技术—实证研究

Oracle required

Dynamic

Static Heuristics

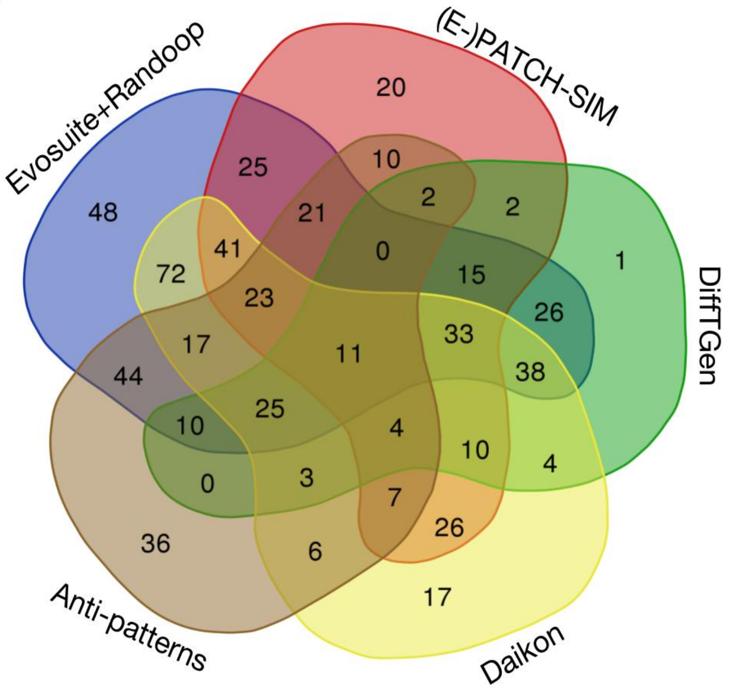
Effectiveness of each APCA Technique



APCA	TP	FP	TN	FN	Precison	Recall
Evosuite	350	3	245	304	99.15%	53.52%
Randoop	221	6	242	433	97.36%	33.79%
DiffTGen†	184	5	232	417	97.35%	30.62%
Daikon†	337	38	166	120	89.87%	73.74%
R-Opad	67	0	248	587	100.00%	10.24%
E-Opad	92	0	248	562	100.00%	14.07%
PATCH-SIM†	249	51	186	392	83.00%	38.85%
E-PATCH-SIM†	166	36	202	477	82.18%	25.82%
Anti-patterns	219	37	211	435	85.55%	33.49%
S3	516	135	113	138	79.26%	78.90%
ssFix	515	138	110	139	78.87%	78.75%
CapGen	506	140	108	148	78.33%	77.37%

- Dynamic APCA techniques **with oracles** can generate **a fewer number** of false positives than those without oracles.
- Opad can achieve **100% precision** while the recall is **rather low**.
- Heuristics based on **static features** can achieve higher recalls but are **less precise**.

补丁正确性验证技术—实证研究



Existing APCA techniques are highly **complementary** to each other;

Distribution of the overfitting patches identified by different APCA techniques.

- Only **11** overfitting patches are detected by all the displayed techniques.
- **Substantial overfitting patches** are detected exclusively by specific techniques.
- **610** unique overfitting patches **(93.3%)** can be detected by at least one technique.

补丁正确性验证技术—实证研究

Target Integrate static features and dynamic techniques for effectiveness enhancement.

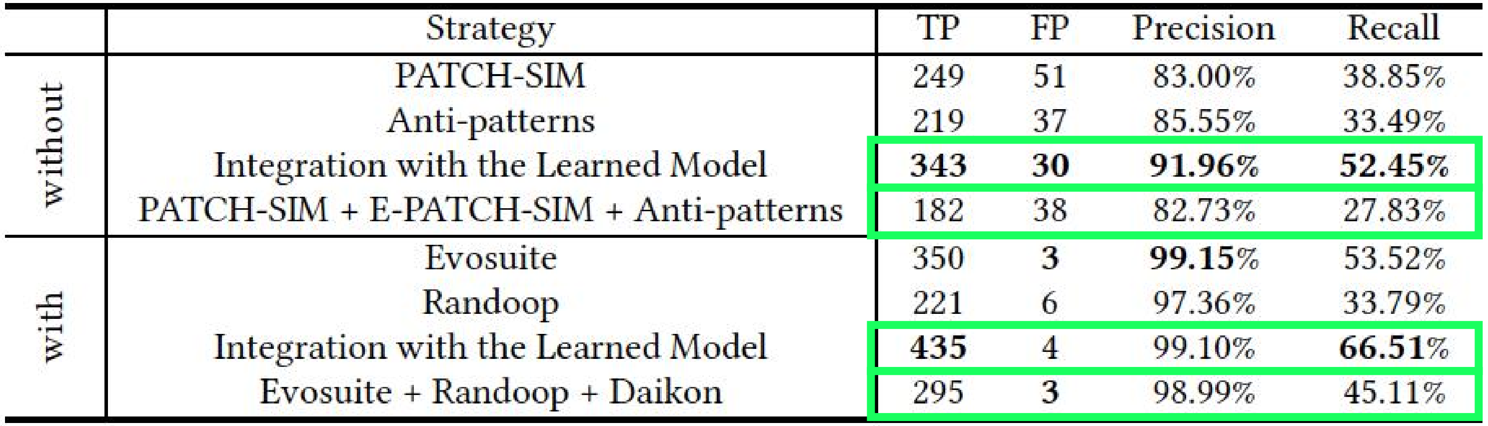
Step 1: integrate the eight static features via learning

- six classification model: *Random Forest, Decision Table, J48, Naive Bayes, Logistic Regression, SMO*.
- patch benchmark separation.
- 10-fold cross validation.

Step 2: combine trained model with existing techniques via majority voting

- without oracle: PATCH-SIM + Anti-patterns + Model① ☞ patch validation.
- with oracle: Evosuite + Randoop + Model② ☞ patch evaluation.

Integration Results with and without the Oracle



	Strategy	TP	FP	Precision	Recall
without	PATCH-SIM	249	51	83.00%	38.85%
	Anti-patterns	219	37	85.55%	33.49%
	Integration with the Learned Model	**343**	**30**	**91.96%**	**52.45%**
	PATCH-SIM + E-PATCH-SIM + Anti-patterns	182	38	82.73%	27.83%
with	Evosuite	350	**3**	**99.15%**	53.52%
	Randoop	221	6	97.36%	33.79%
	Integration with the Learned Model	**435**	4	99.10%	**66.51%**
	Evosuite + Randoop + Daikon	295	**3**	98.99%	45.11%

Under both scenarios, the integration results significantly outperform existing techniques.

The learned model makes contributions for performance enhancement.

PART 02

基于表示学习的补丁验证

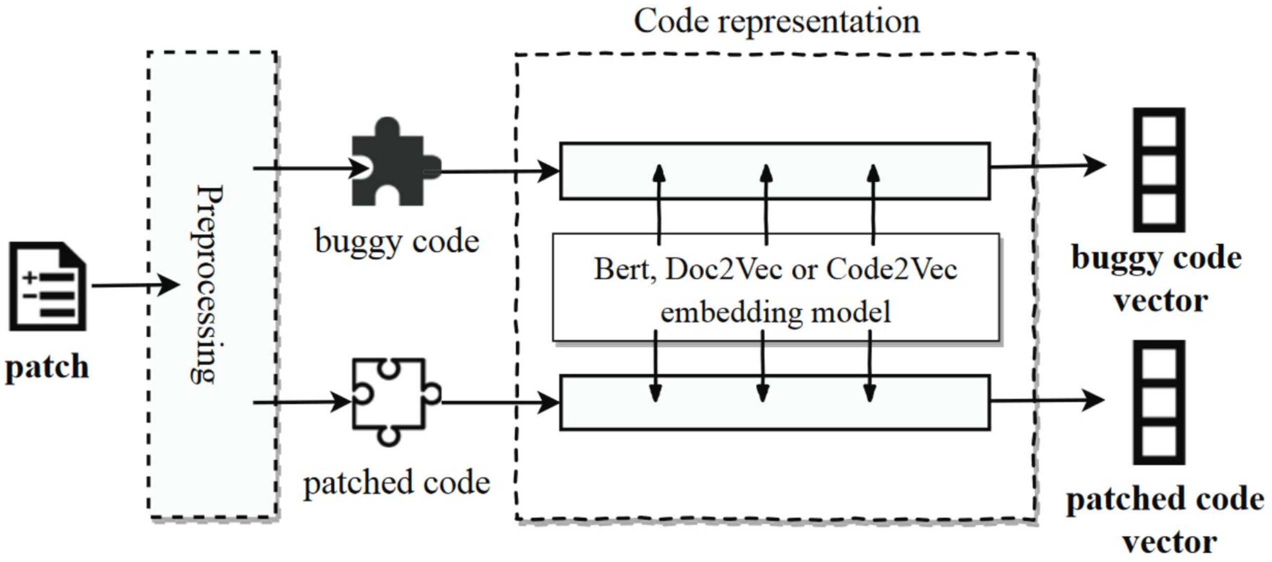
基于表示学习的补丁验证

Wang, Shangwen, Ming Wen, Bo Lin, Hongjun Wu, Yihao Qin, Deqing Zou, Xiaoguang Mao, and Hai Jin. "**Automated patch correctness assessment: How far are we?**." *In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 968-980. 2020.

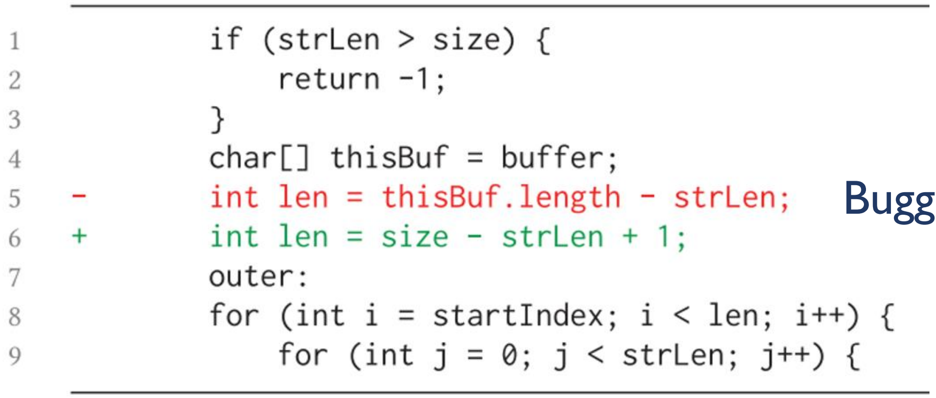
Dynamic Techniques *Time-consuming to generate and execute tests*

Static Techniques *More efficient while less precise. Relying on manually designed features.*

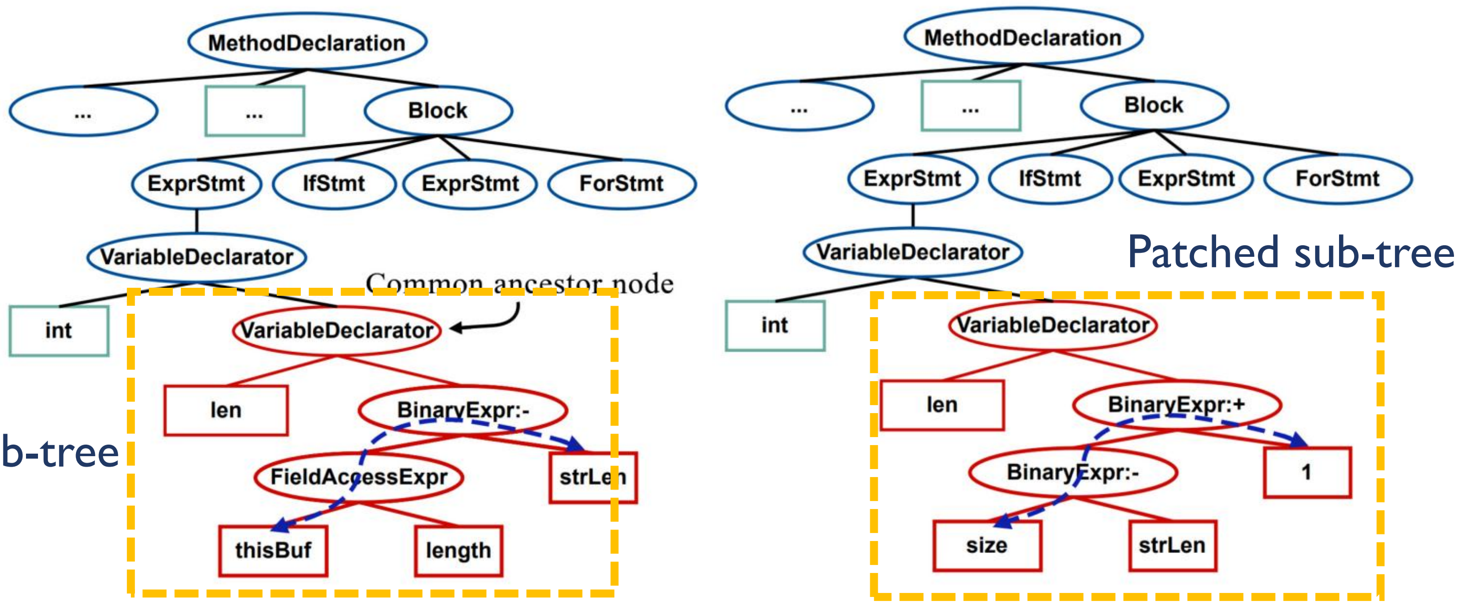
With the emerging of code embedding techniques, patches can be easily transferred to vectors, thus facilitating the utilization of deep learning techniques



基于表示学习的补丁验证



```
    if (strLen > size) {
        return -1;
    }
    char[] thisBuf = buffer;
-   int len = thisBuf.length - strLen;
+   int len = size - strLen + 1;
    outer:
    for (int i = startIndex; i < len; i++) {
        for (int j = 0; j < strLen; j++) {
```



Buggy sub-tree

(a) AST for buggy code snippet.

(b) AST for patched code snippet.

Missing Contexts

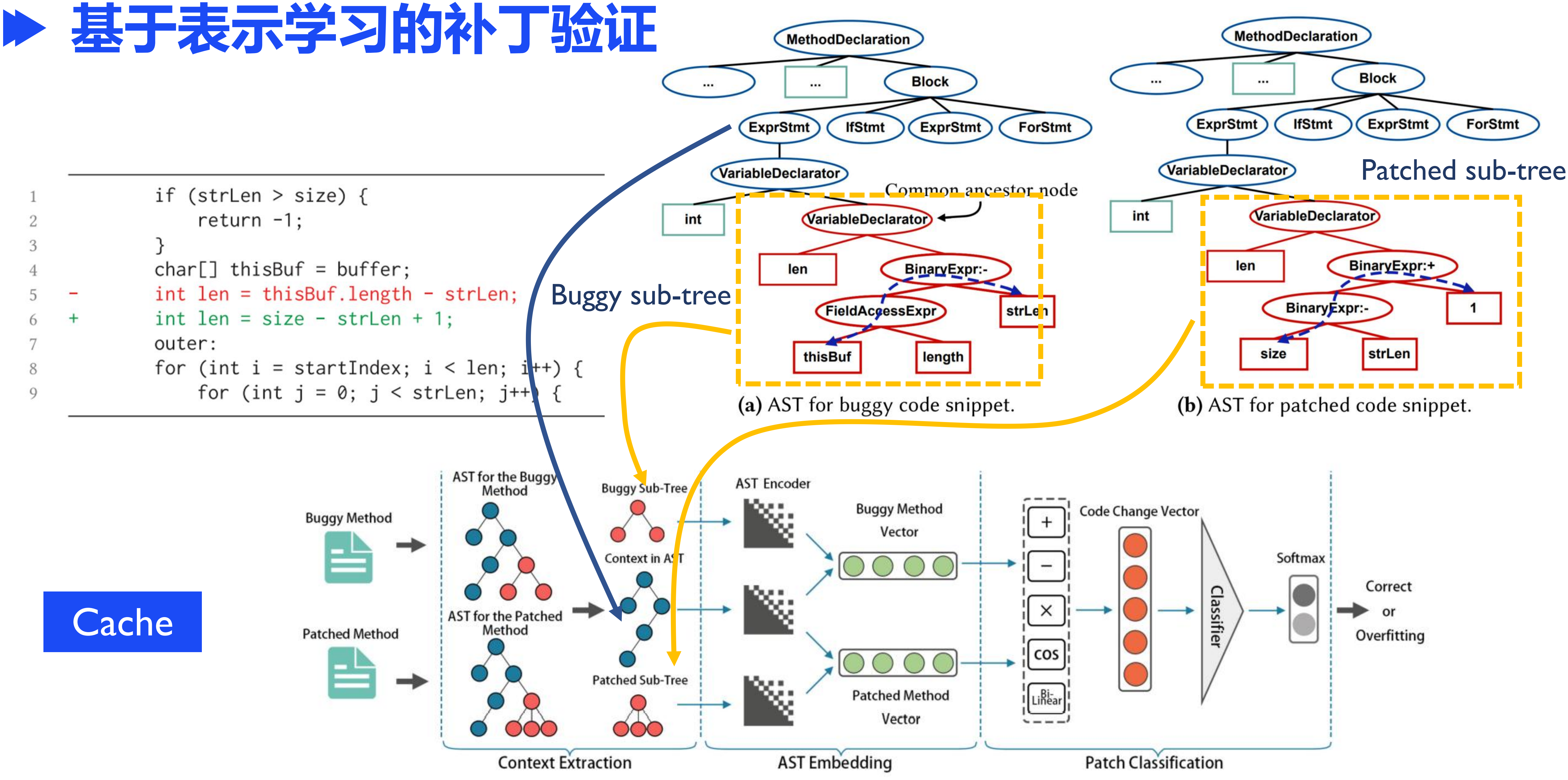
Treating the added and deleted lines separately by considering the whole line as a token sequence and embedding each single token.

`{ int, len, =, thisBuf, length, -, strLen }`

Missing Structures

Treating ***keywords*** in a program in the same way with other ***tokens*** (e.g., variable names), thus overlooking the program's inherent structures.

基于表示学习的补丁验证



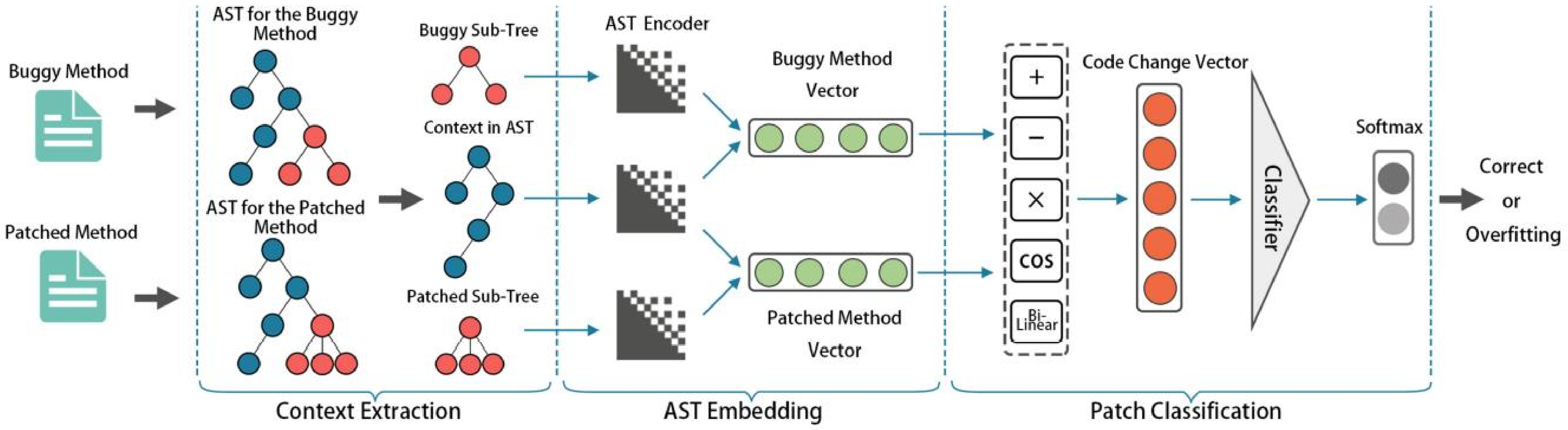
```
        if (strLen > size) {
            return -1;
        }
        char[] thisBuf = buffer;
-       int len = thisBuf.length - strLen;
+       int len = size - strLen + 1;
        outer:
        for (int i = startIndex; i < len; i++) {
            for (int j = 0; j < strLen; j++) {
```

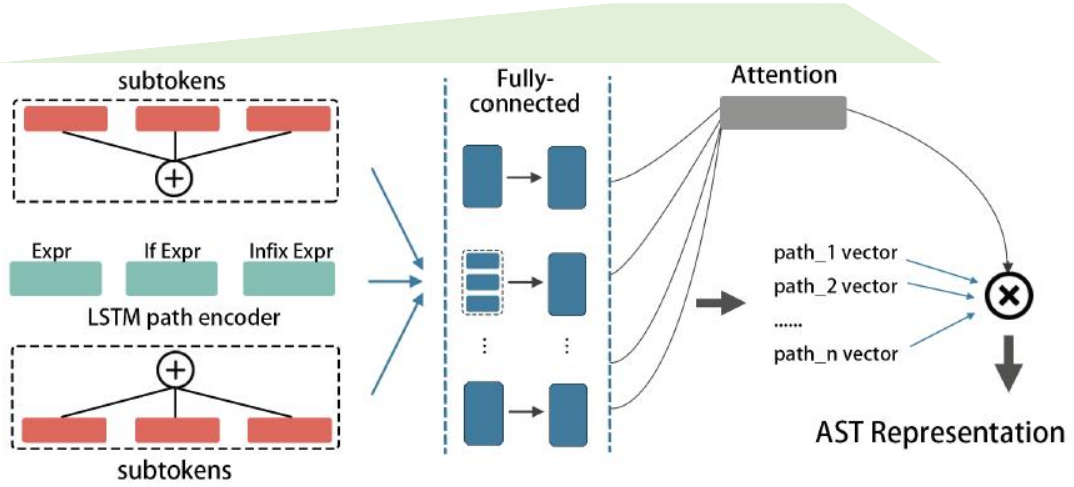
(a) AST for buggy code snippet.

(b) AST for patched code snippet.

Cache

基于表示学习的补丁验证—整体框架





AST Embedding

*We use **AST paths** to embed patches which, to a large extend, preserves the **structure** of the original program since any two consecutive nodes in the AST path are inherent with the **parent & child** relation in the parsed AST.*

基于表示学习的补丁验证—数据集

Dataset

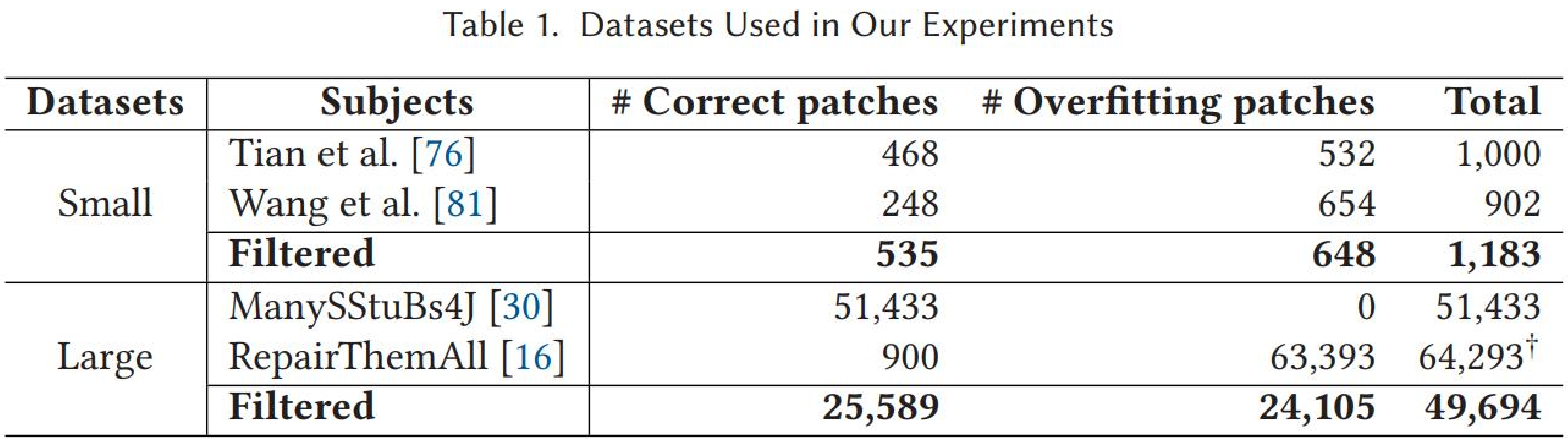
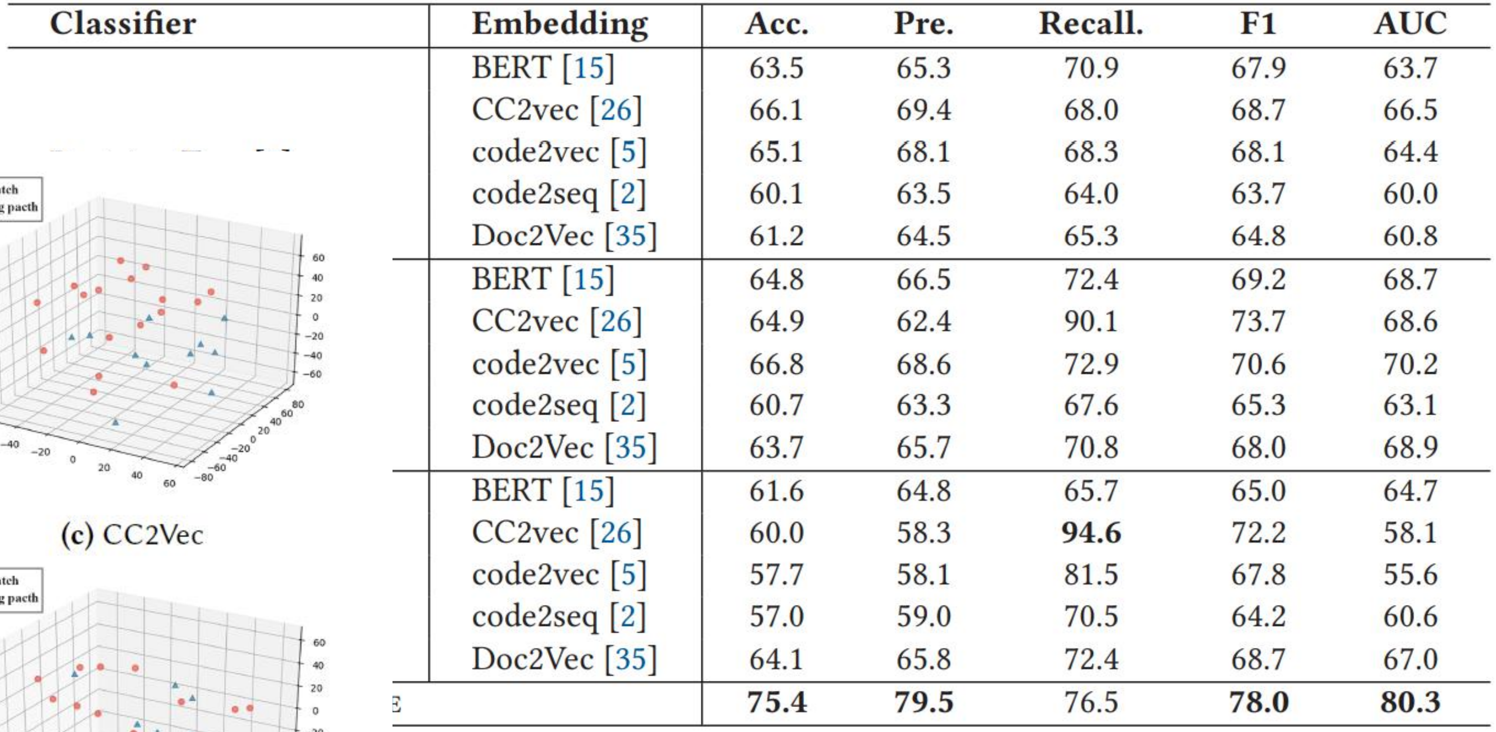


Table 1. Datasets Used in Our Experiments

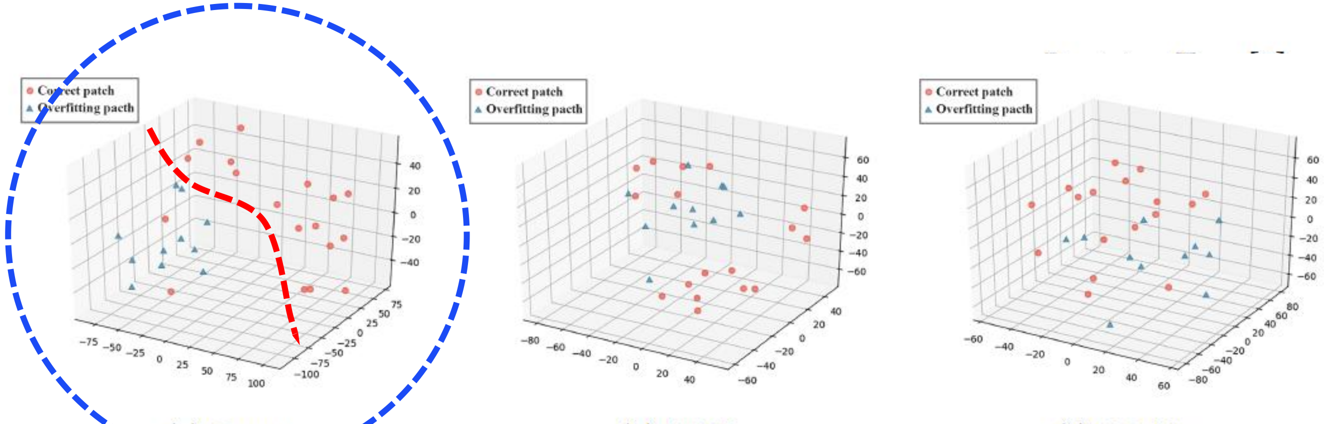
Datasets	Subjects	# Correct patches	# Overfitting patches	Total
Small	Tian et al. [76]	468	532	1,000
	Wang et al. [81]	248	654	902
	Filtered	**535**	**648**	**1,183**
Large	ManySStuBs4J [30]	51,433	0	51,433
	RepairThemAll [16]	900	63,393	64,293†
	Filtered	**25,589**	**24,105**	**49,694**

▶ 基于表示学习的补丁验证—实验结果

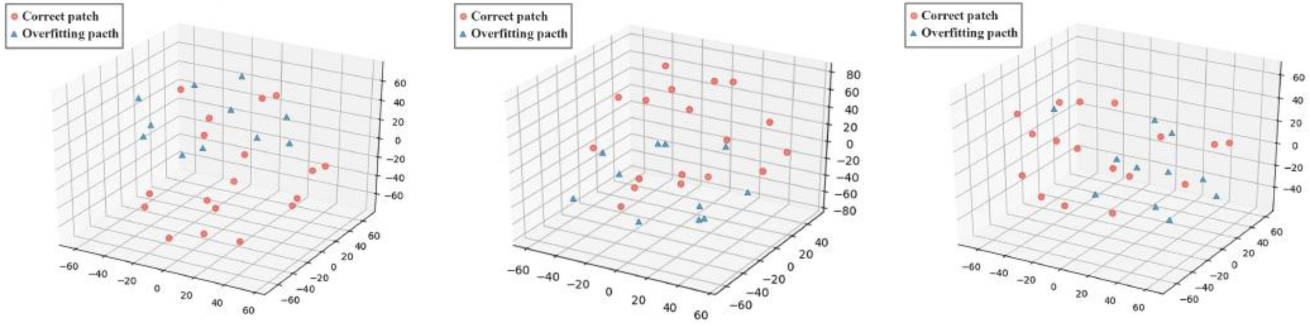
Effectiveness of Cache Compared with Other Representation Learning Techniques



Classifier	Embedding	Acc.	Pre.	Recall.	F1	AUC
	BERT [15]	63.5	65.3	70.9	67.9	63.7
	CC2vec [26]	66.1	69.4	68.0	68.7	66.5
	code2vec [5]	65.1	68.1	68.3	68.1	64.4
	code2seq [2]	60.1	63.5	64.0	63.7	60.0
	Doc2Vec [35]	61.2	64.5	65.3	64.8	60.8
	BERT [15]	64.8	66.5	72.4	69.2	68.7
	CC2vec [26]	64.9	62.4	90.1	73.7	68.6
	code2vec [5]	66.8	68.6	72.9	70.6	70.2
	code2seq [2]	60.7	63.3	67.6	65.3	63.1
	Doc2Vec [35]	63.7	65.7	70.8	68.0	68.9
	BERT [15]	61.6	64.8	65.7	65.0	64.7
	CC2vec [26]	60.0	58.3	**94.6**	72.2	58.1
	code2vec [5]	57.7	58.1	81.5	67.8	55.6
	code2seq [2]	57.0	59.0	70.5	64.2	60.6
	Doc2Vec [35]	64.1	65.8	72.4	68.7	67.0
E		**75.4**	**79.5**	76.5	**78.0**	**80.3**



(a) CACHE (b) BERT (c) CC2Vec



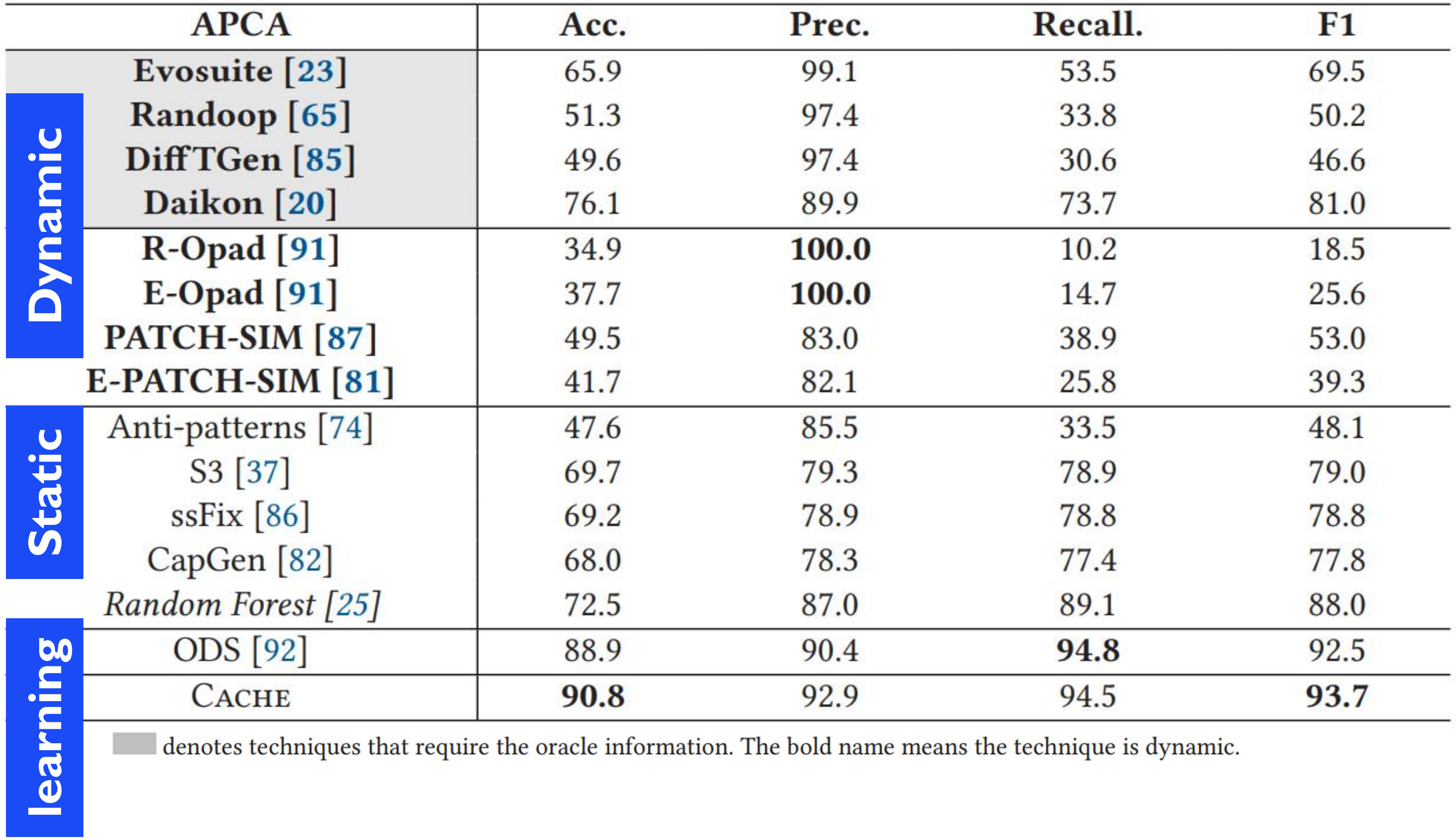
(d) code2seq (e) code2vec (f) Doc2vec

Distribution of embedding vectors generated by Cache and other Representation Learning Techniques

***Cache** can outperform other representation learning based techniques significantly under different experimental settings. Specifically, it achieves an average F1-score of **78.0%**.*

基于表示学习的补丁验证—实验结果

Effectiveness of Cache Compared with Other APCA Techniques



	APCA	Acc.	Prec.	Recall.	F1
Dynamic	**Evosuite [23]**	65.9	99.1	53.5	69.5
	Randoop [65]	51.3	97.4	33.8	50.2
	DiffTGen [85]	49.6	97.4	30.6	46.6
	Daikon [20]	76.1	89.9	73.7	81.0
	R-Opad [91]	34.9	**100.0**	10.2	18.5
	E-Opad [91]	37.7	**100.0**	14.7	25.6
	PATCH-SIM [87]	49.5	83.0	38.9	53.0
	E-PATCH-SIM [81]	41.7	82.1	25.8	39.3
Static	Anti-patterns [74]	47.6	85.5	33.5	48.1
	S3 [37]	69.7	79.3	78.9	79.0
	ssFix [86]	69.2	78.9	78.8	78.8
	CapGen [82]	68.0	78.3	77.4	77.8
	Random Forest [25]	72.5	87.0	89.1	88.0
learning	ODS [92]	88.9	90.4	**94.8**	92.5
	CACHE	**90.8**	92.9	94.5	**93.7**

▨ denotes techniques that require the oracle information. The bold name means the technique is dynamic.

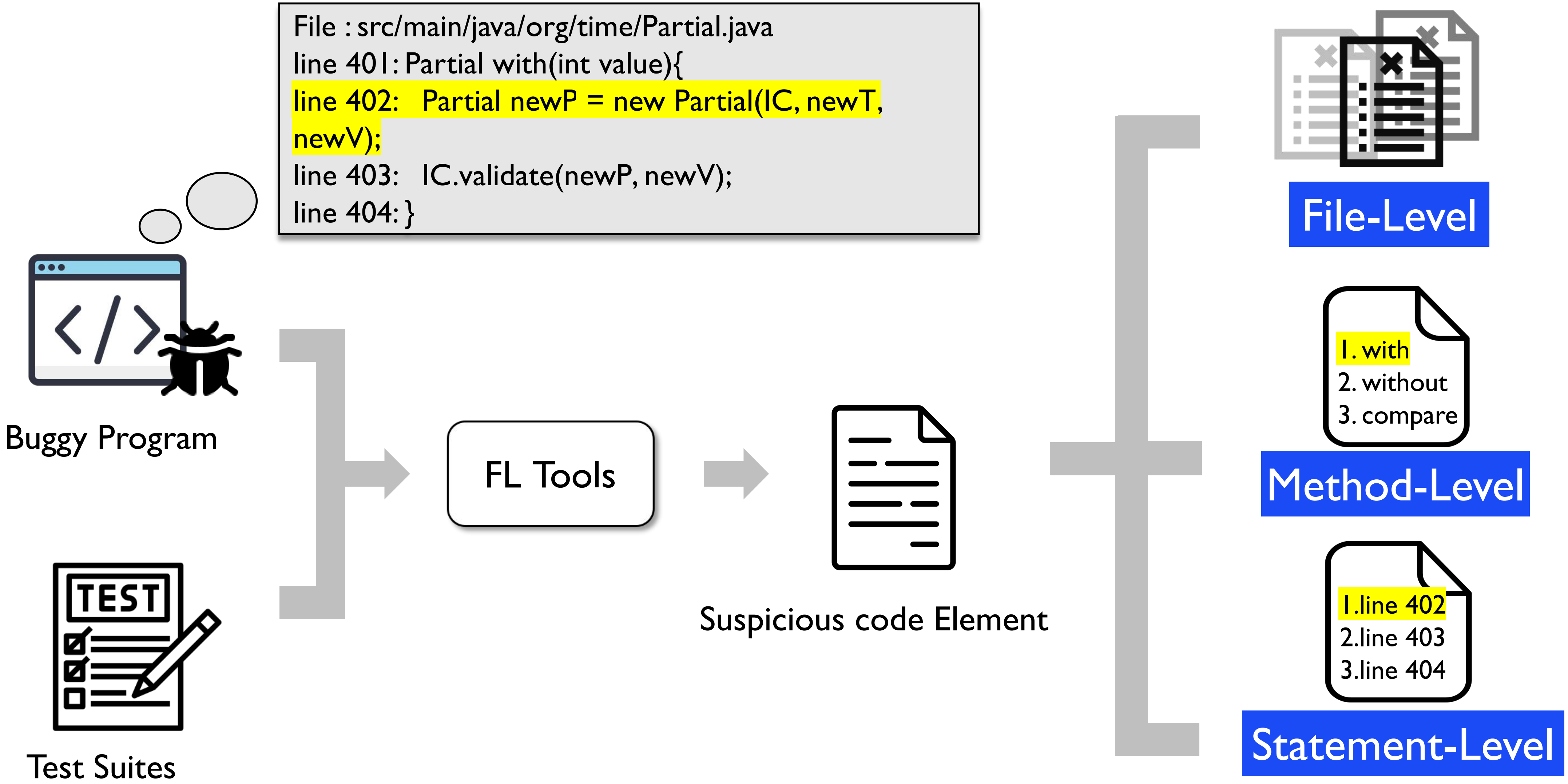
As a static technique, Cache achieves the optimum overall performance compared with existing APCA techniques with a ***high F1-score*** *reaching* ***93.7%****. Furthermore, it can even achieve a higher precision than certain* ***dynamic techniques****, e.g., PATCH-SIM.*

Lin, Bo, Shangwen Wang, Ming Wen, and Xiaoguang Mao. "**Context-aware code change embedding for better patch correctness assessment**." *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, no. 3 (2022): 1-29.

PART 03

基于缺陷定位的补丁排序

基于缺陷定位的补丁排序—背景



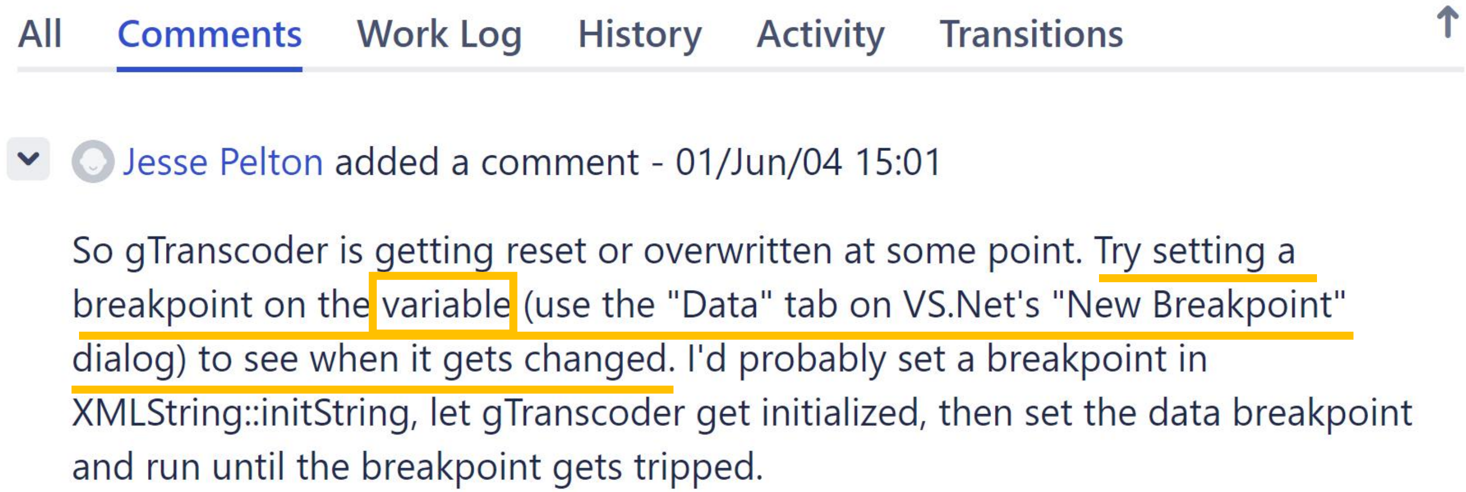
```
File : src/main/java/org/time/Partial.java
line 401: Partial with(int value){
line 402:   Partial newP = new Partial(IC, newT,
newV);
line 403:   IC.validate(newP, newV);
line 404: }
```

基于缺陷定位的补丁排序—动因

Developers look for **variable-level** FL approaches

- Monitoring variable are widely used **in practice**
- Variables values are useful to understand the **root cause** of the bug

Community comments



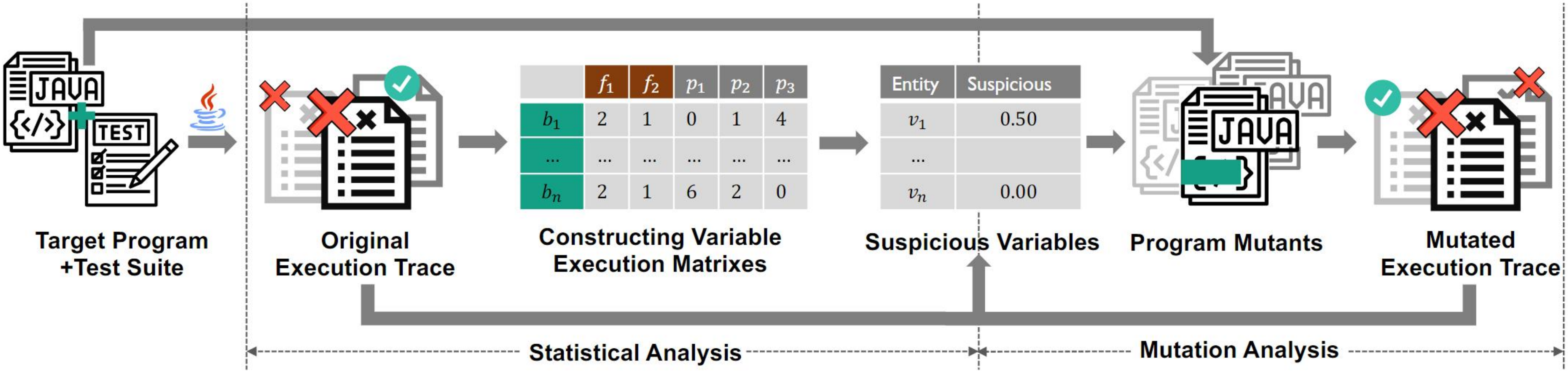
Jesse Pelton added a comment - 01/Jun/04 15:01

So gTranscoder is getting reset or overwritten at some point. Try setting a breakpoint on the variable (use the "Data" tab on VS.Net's "New Breakpoint" dialog) to see when it gets changed. I'd probably set a breakpoint in XMLString::initString, let gTranscoder get initialized, then set the data breakpoint and run until the breakpoint gets tripped.

Reference URL: https://issues.apache.org/jira/browse/XERCESC-1222

基于缺陷定位的补丁排序—框架

IsoVar **Iso**lating Fault-Correlated **Var**iables



Target Program +Test Suite

Original Execution Trace

	f_1	f_2	p_1	p_2	p_3
b_1	2	1	0	1	4
...	...	...	...	...	...
b_n	2	1	6	2	0

Constructing Variable Execution Matrixes

Entity	Suspicious
v_1	0.50
...	
v_n	0.00

Suspicious Variables

Program Mutants

Mutated Execution Trace

Statistical Analysis

Mutation Analysis

Spectrum-Based Fault Localization

- Fault-correlated elements should be more frequently covered by failing tests rather than passing tests

Mutation-Based Fault Localization

- A change of Fault-correlated elements is more likely to change the results in failing tests, and less likely in passing tests

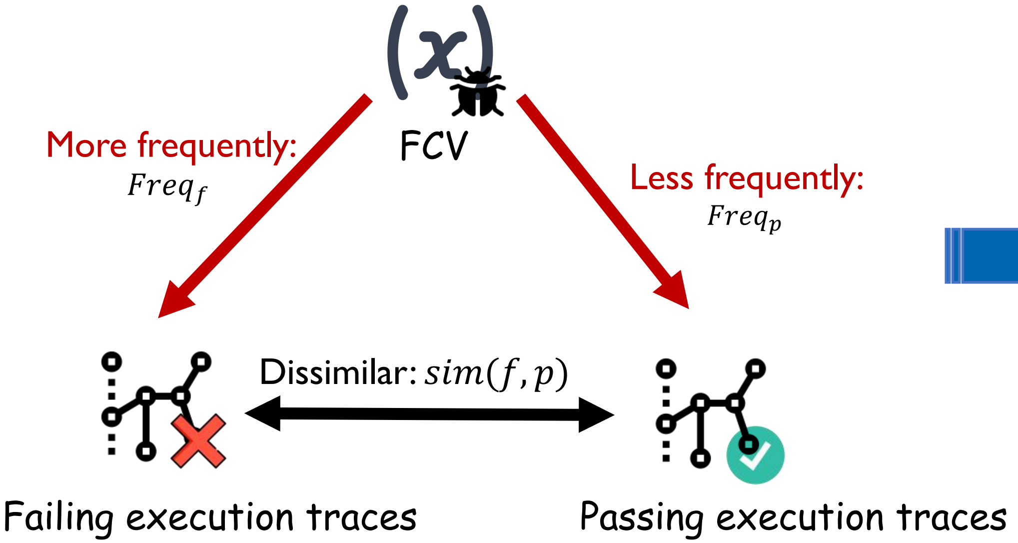
基于缺陷定位的补丁排序—方法

		Failing$_1$	Failing$_2$	Passing$_1$	Passing$_2$	Passing$_3$
Variable	BB$_1$	2	1	0	0	0
	BB$_2$	2	0	0	1	0
	BB$_3$	2	1	0	0	0
	BB$_4$	2	0	6	2	4

- Build variable execution matrix

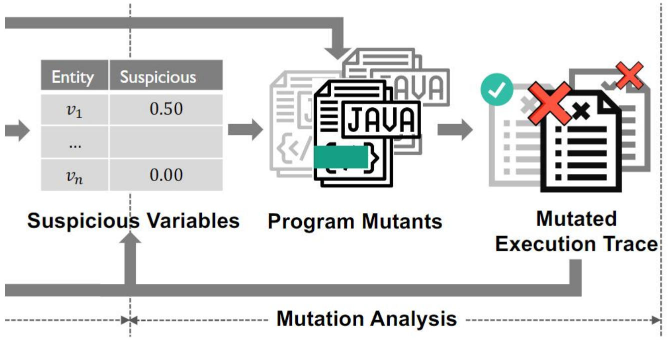
For each variable, IsoVar record the spectrum of the basic blocks containing that variable

- Fulfill the insights from spectrum-based fault localization



$$Susp(v) = \frac{Freq_f}{Freq_p + Freq_f} - \alpha * sim(f,p)$$

基于缺陷定位的补丁排序—方法

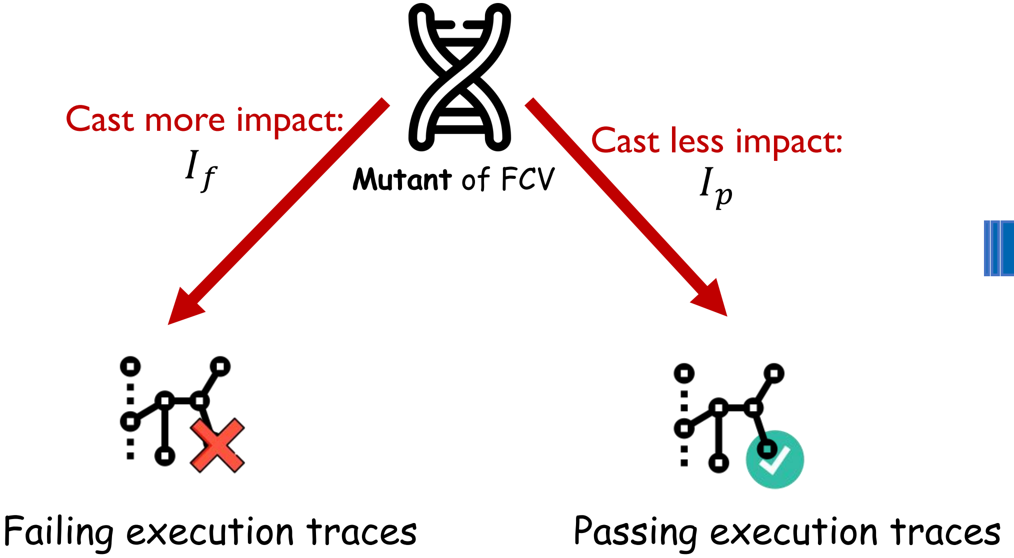


$int\ x\ =\ 1;$

$int\ x\ =\ 1 + 30;$

…

- Mutate the variable as mutants based on variable type
- Fulfill the insights from mutation-based fault localization



Cast more impact: I_f

Mutant of FCV

Cast less impact: I_p

Failing execution traces

Passing execution traces

$$fitness(v) = I_f - \beta * I_p$$

$$isolateion(v) = Susp(v) + r * fitness(v)$$

基于缺陷定位的补丁排序—实验结果

- $if\ (screen < 0\ ||\ screen >= numScreens)\ \{$

+ $if\ (screen >= numScreens)\ \{$

Refine **Patch Priority Score** by IsoVar outputs

Insight correct patches should involve more fault-correlated variables

TABLE 3: Patch Prioritization for Existing APR Techniques

APR	Cardumen	jMutRepair	NPEFix	Nopol	Arja	DynaMoth	GenProg	JGenProg	jKali	RSRepair	Kali	PraPR	TBar	SimFix	Summary
#CP	3	4	4	3	774	2	45	3	2	43	3	39	80	33	1,034
#CPBP	3	4	0	3	764	2	3	3	2	13	3	17	55	29	901
#CPBP^b	3	4	1	3	767	2	36	3	2	20	3	19	56	31	950

Enhance **14** automated program repair techniques to rank the correct patches. The precision improvement is **69.6%-79.9%**

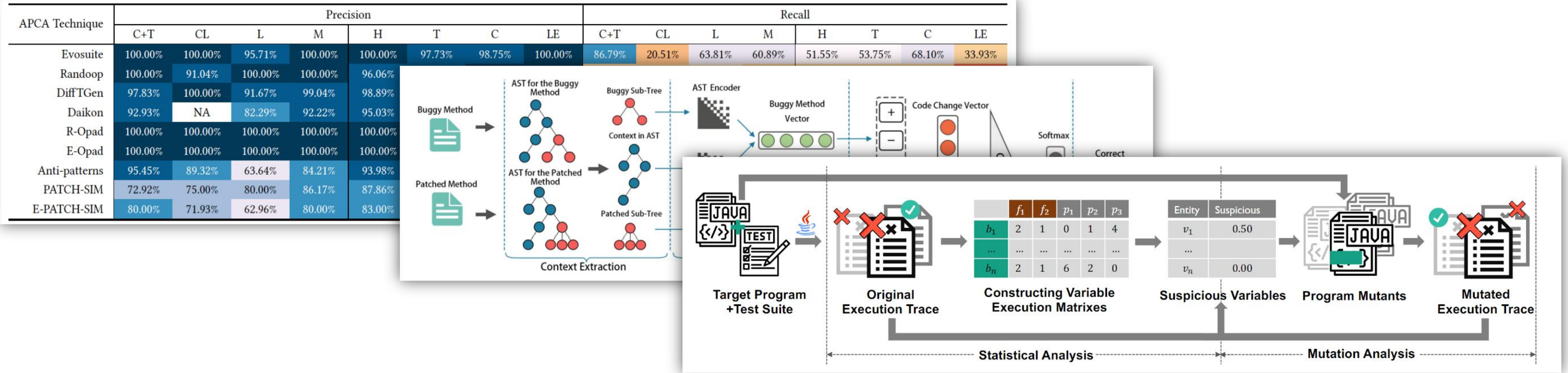
prioritize 49 more correct patches

PART 04

总结与展望

总结与展望

- Automated program repair is important, especially in the large language model era.
- Precise patch correctness assessment (PCA) is the key for practical automated program repair.
- Existing PCA efforts are yet insufficient. Our proposed static analysis based, and learning based methods have demonstrated promising performance.

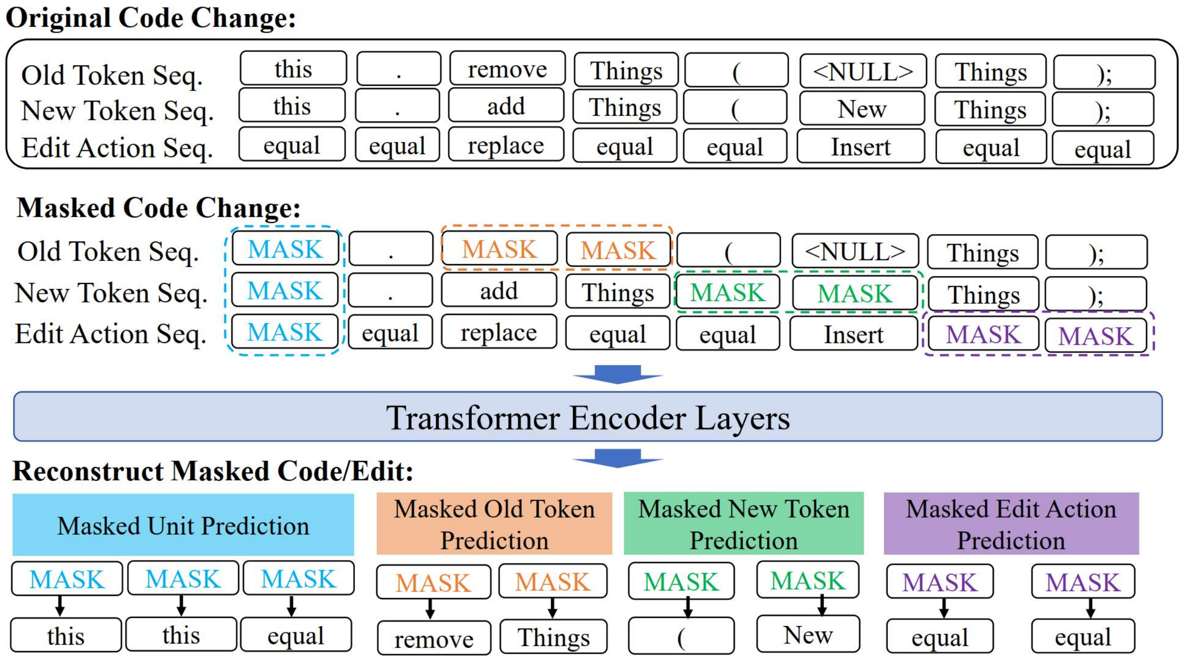


APCA Technique	Precision								Recall							
	C+T	CL	L	M	H	T	C	LE	C+T	CL	L	M	H	T	C	LE
Evosuite	100.00%	100.00%	95.71%	100.00%	100.00%	97.73%	98.75%	100.00%	86.79%	20.51%	63.81%	60.89%	51.55%	53.75%	68.10%	33.93%
Randoop	100.00%	91.04%	100.00%	100.00%	96.06%											
DiffTGen	97.83%	100.00%	91.67%	99.04%	98.89%											
Daikon	92.93%	NA	82.29%	92.22%	95.03%											
R-Opad	100.00%	100.00%	100.00%	100.00%	100.00%											
E-Opad	100.00%	100.00%	100.00%	100.00%	100.00%											
Anti-patterns	95.45%	89.32%	63.64%	84.21%	93.98%											
PATCH-SIM	72.92%	75.00%	80.00%	86.17%	87.86%											
E-PATCH-SIM	80.00%	71.93%	62.96%	80.00%	83.00%											

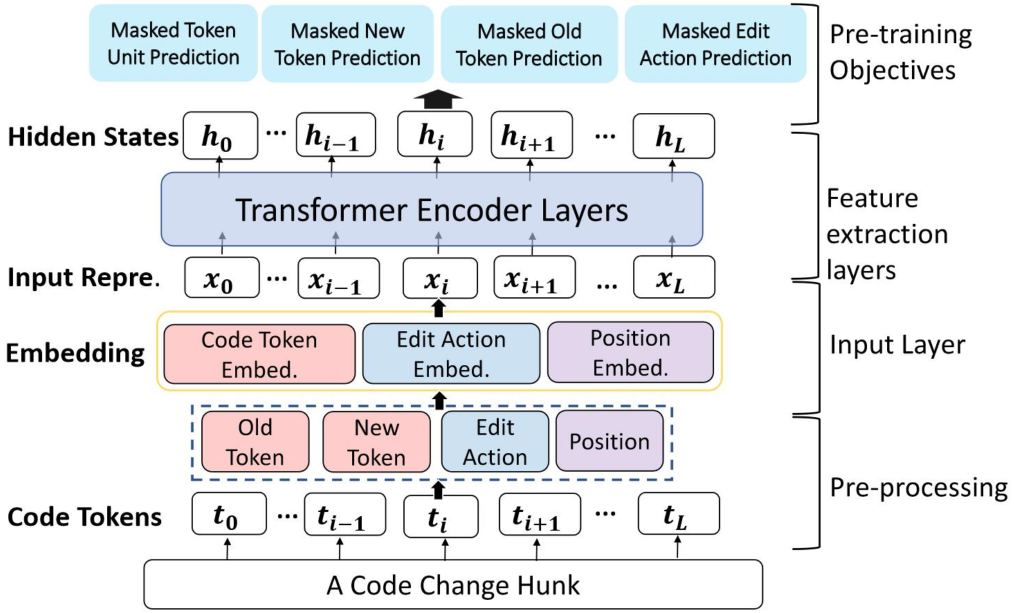
总结与展望—补丁表示学习

Direction How to better represent a patch?

Code-Change-Oriented Pre-Trained Model



Pre-training Objectives



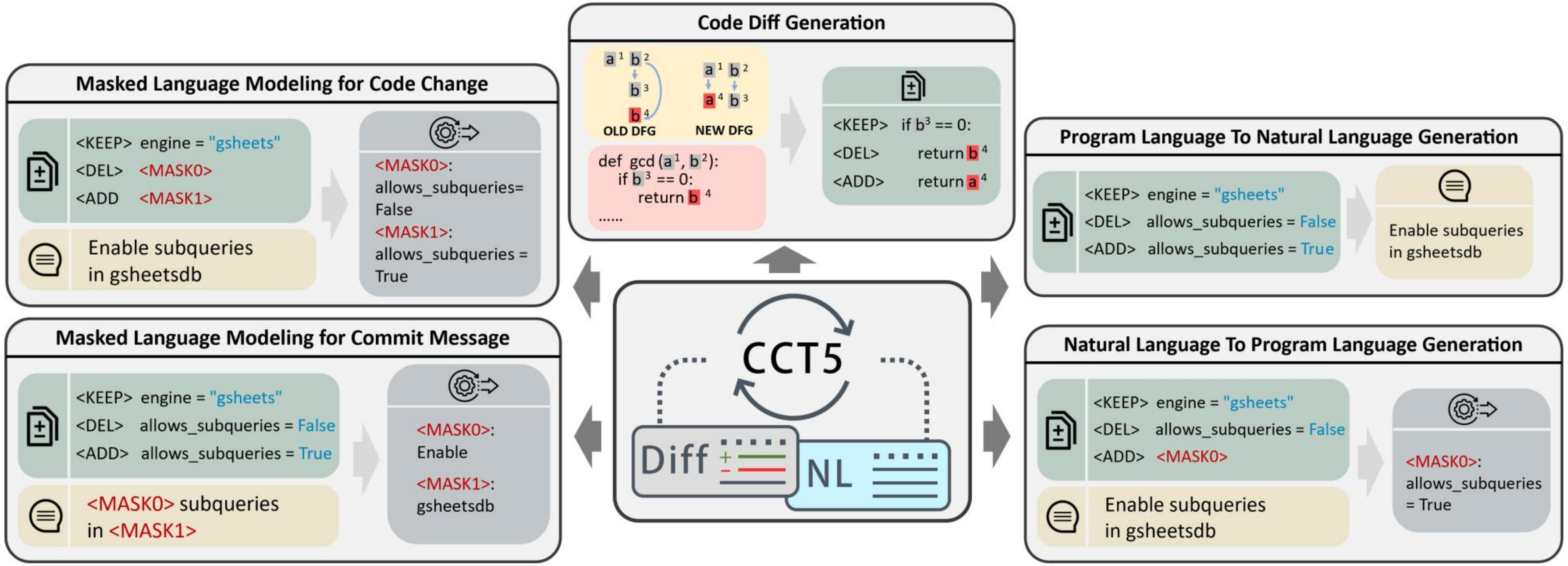
Overview of CCBERT

CCBERT: **Self-Supervised Code Change Representation Learning**, Arxiv 2023

总结与展望—补丁表示学习

Direction How to better represent a patch?

Code-Change-Oriented Pre-Trained Model

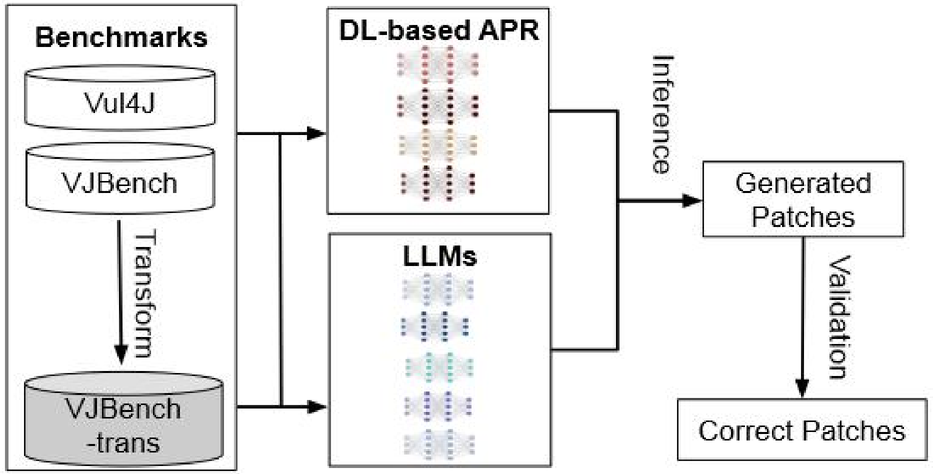


Overview of CCT5

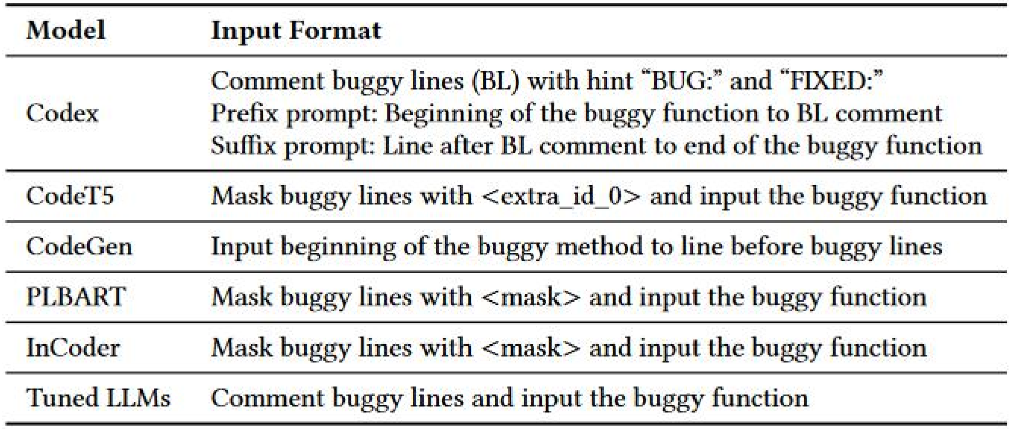
Lin, Bo, Shangwen Wang, Zhongxin Liu, Yepang Liu, Xin Xia, and Xiaoguang Mao. "**CCT5: A Code-Change-Oriented Pre-Trained Model**." ESEC/FSE (2023).

总结与展望—拥抱大模型

Direction How to design better prompts?



Prompt Design



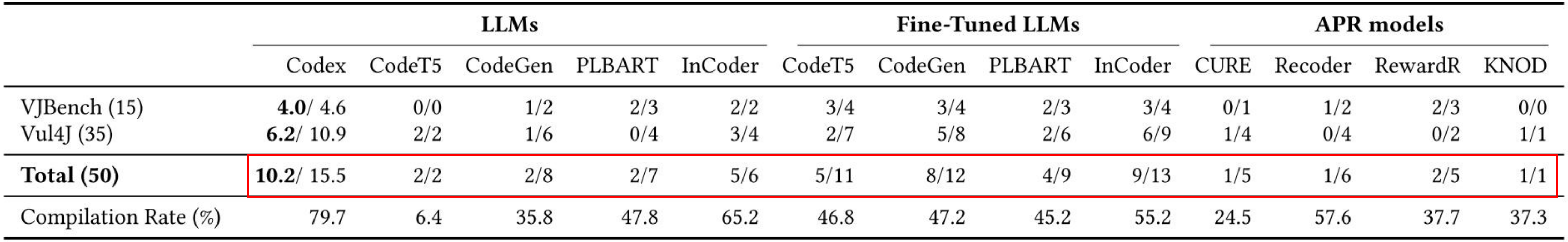
Model	Input Format
Codex	Comment buggy lines (BL) with hint "BUG:" and "FIXED:" Prefix prompt: Beginning of the buggy function to BL comment Suffix prompt: Line after BL comment to end of the buggy function
CodeT5	Mask buggy lines with <extra_id_0> and input the buggy function
CodeGen	Input beginning of the buggy method to line before buggy lines
PLBART	Mask buggy lines with <mask> and input the buggy function
InCoder	Mask buggy lines with <mask> and input the buggy function
Tuned LLMs	Comment buggy lines and input the buggy function

Wu, Yi, Nan Jiang, Hung Viet Pham, Thibaud Lutellier, Jordan Davis, Lin Tan, Petr Babkin, and Sameena Shah. "**How Effective Are Neural Networks for Fixing Security Vulnerabilities**." (ISSTA 2023).

总结与展望—拥抱大模型

Direction How to design better prompts?

Correctly Repaired Vulnerabilities. (**X/Y: X** denotes **correct patches** while **Y** denotes **plausible patches**)

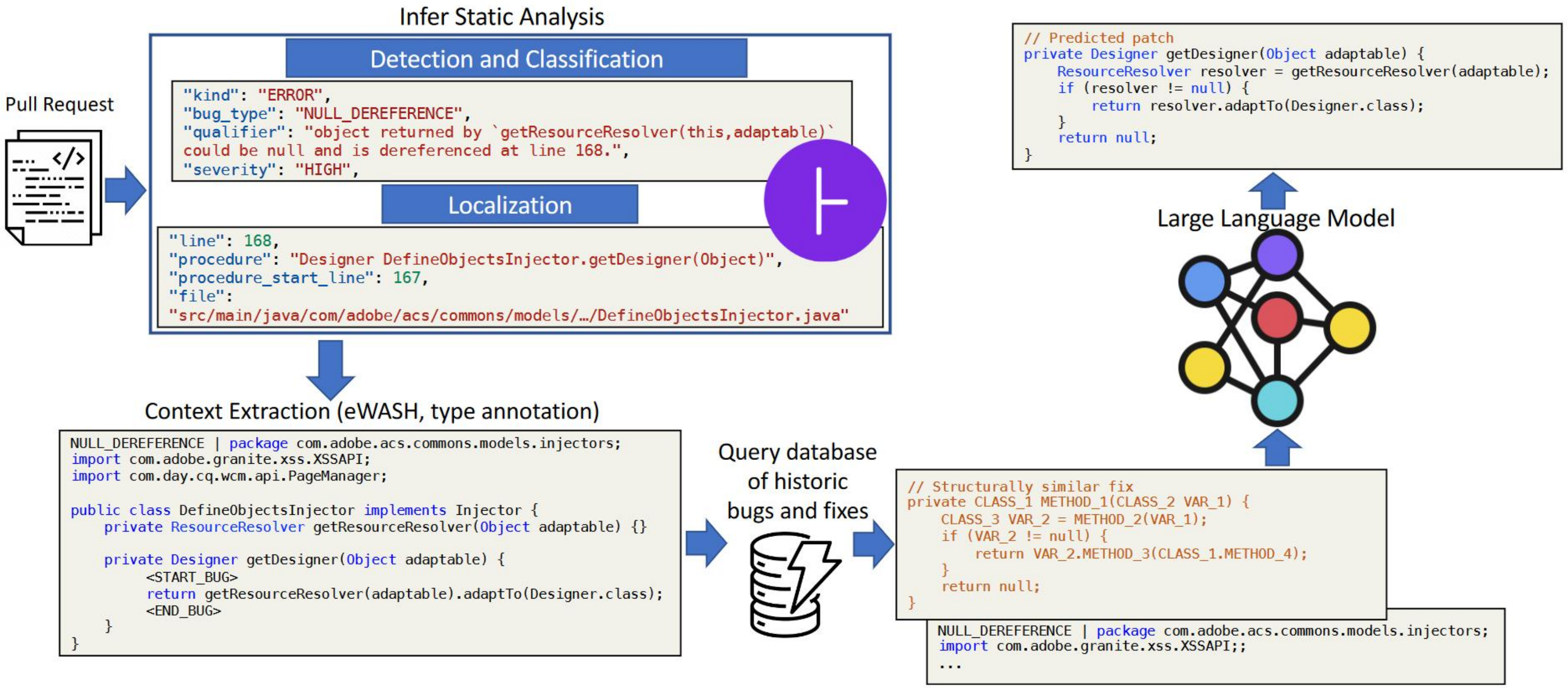


	LLMs					Fine-Tuned LLMs				APR models			
	Codex	CodeT5	CodeGen	PLBART	InCoder	CodeT5	CodeGen	PLBART	InCoder	CURE	Recoder	RewardR	KNOD
VJBench (15)	**4.0**/ 4.6	0/0	1/2	2/3	2/2	3/4	3/4	2/3	3/4	0/1	1/2	2/3	0/0
Vul4J (35)	**6.2**/ 10.9	2/2	1/6	0/4	3/4	2/7	5/8	2/6	6/9	1/4	0/4	0/2	1/1
Total (50)	**10.2**/ 15.5	2/2	2/8	2/7	5/6	5/11	8/12	4/9	9/13	1/5	1/6	2/5	1/1
Compilation Rate (%)	79.7	6.4	35.8	47.8	65.2	46.8	47.2	45.2	55.2	24.5	57.6	37.7	37.3

- Existing LLMs and APR models **fix very few Java vulnerabilities.** Codex fixes 10.2 (20.4%) vulnerabilities on average, exhibiting the best fixing capability.
- Manually examine whether a patch is correct, and **44.9%** of the patches are **plausible** but **incorrect**. It calls for action to design better APCA techniques.
- **Model fine-tuning can enhance the repair performance**. It calls for action to create larger vulnerability repair training datasets, and fine-tune LLMs with such data.

总结与展望—拥抱大模型

Direction How to design better prompts?



```
"kind": "ERROR",
"bug_type": "NULL_DEREFERENCE",
"qualifier": "object returned by `getResourceResolver(this,adaptable)`
could be null and is dereferenced at line 168.",
"severity": "HIGH",
```

```
"line": 168,
"procedure": "Designer DefineObjectsInjector.getDesigner(Object)",
"procedure_start_line": 167,
"file":
"src/main/java/com/adobe/acs/commons/models/…/DefineObjectsInjector.java"
```

```
NULL_DEREFERENCE | package com.adobe.acs.commons.models.injectors;
import com.adobe.granite.xss.XSSAPI;
import com.day.cq.wcm.api.PageManager;

public class DefineObjectsInjector implements Injector {
    private ResourceResolver getResourceResolver(Object adaptable) {}

    private Designer getDesigner(Object adaptable) {
          <START_BUG>
          return getResourceResolver(adaptable).adaptTo(Designer.class);
          <END_BUG>
    }
}
```

```
// Structurally similar fix
private CLASS_1 METHOD_1(CLASS_2 VAR_1) {
    CLASS_3 VAR_2 = METHOD_2(VAR_1);
    if (VAR_2 != null) {
        return VAR_2.METHOD_3(CLASS_1.METHOD_4);
    }
    return null;
}
```

```
NULL_DEREFERENCE | package com.adobe.acs.commons.models.injectors;
import com.adobe.granite.xss.XSSAPI;;
...
```

```
// Predicted patch
private Designer getDesigner(Object adaptable) {
    ResourceResolver resolver = getResourceResolver(adaptable);
    if (resolver != null) {
        return resolver.adaptTo(Designer.class);
    }
    return null;
}
```

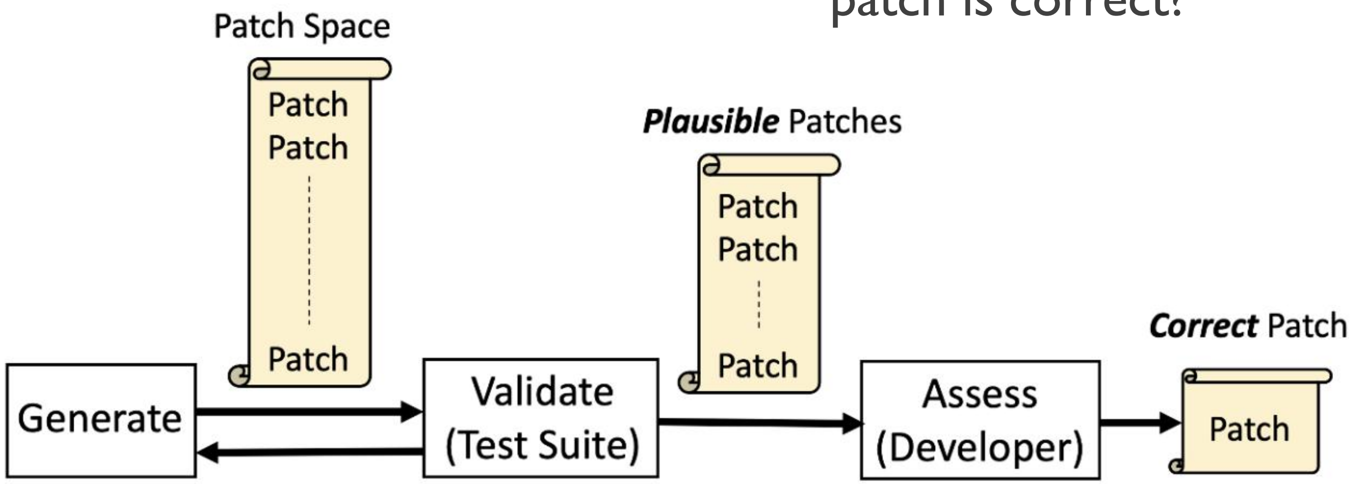
Prompt Design

- Bug Type
- Patch Contexts
- Historical Bug Fixes

Jin, Matthew, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. "**Inferfix: End-to-end program repair with LLMs**." arXiv preprint (2023).

总结与展望—拥抱大模型

Direction How to design better prompts?



PROMPT

- Whether this plausible patch is correct?

Enhanced PROMPT

- Whether this plausible patch is correct?

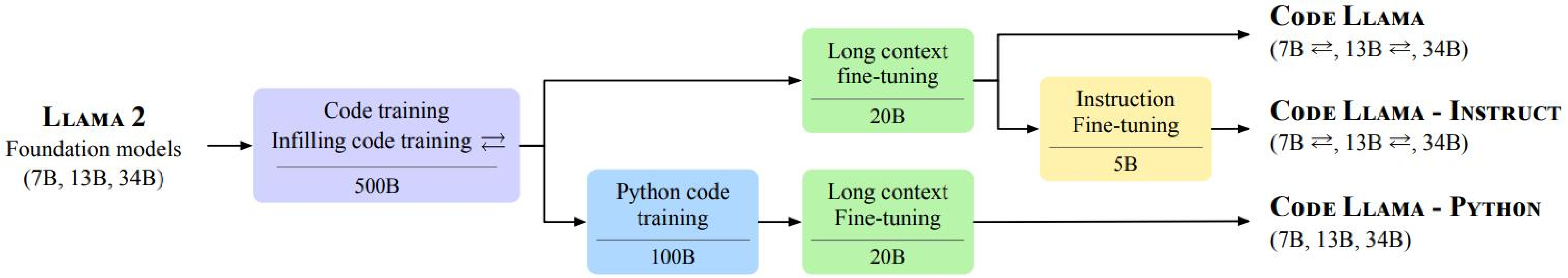
- Bug Type
- Patch Contexts
- Historical Bug Fixes
- ……

Static Analysis

Data Mining

总结与展望—微调大模型

Direction How to fine-tune an specified LLM model?



LLAMA 2
Foundation models
(7B, 13B, 34B)

Code training
Infilling code training ⇄
500B

Python code training
100B

Long context fine-tuning
20B

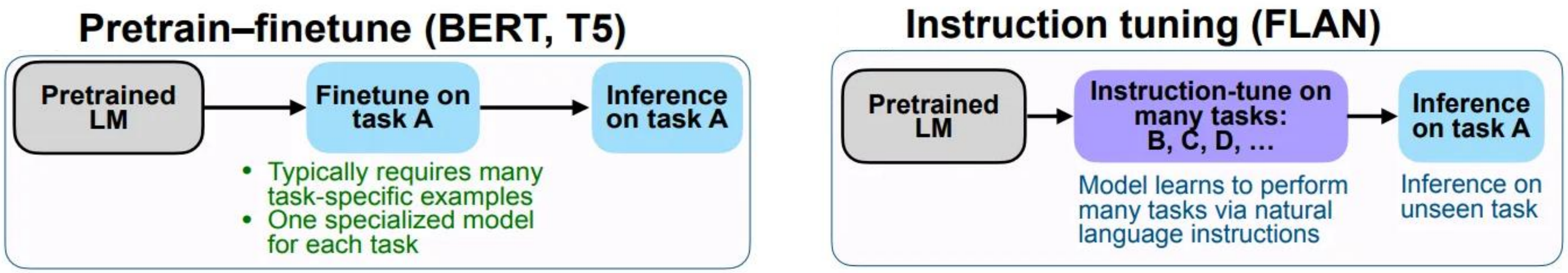
Long context Fine-tuning
20B

Instruction Fine-tuning
5B

CODE LLAMA
(7B ⇄, 13B ⇄, 34B)

CODE LLAMA - INSTRUCT
(7B ⇄, 13B ⇄, 34B)

CODE LLAMA - PYTHON
(7B, 13B, 34B)



Pretrain–finetune (BERT, T5)

Pretrained LM → Finetune on task A → Inference on task A

- Typically requires many task-specific examples
- One specialized model for each task

Instruction tuning (FLAN)

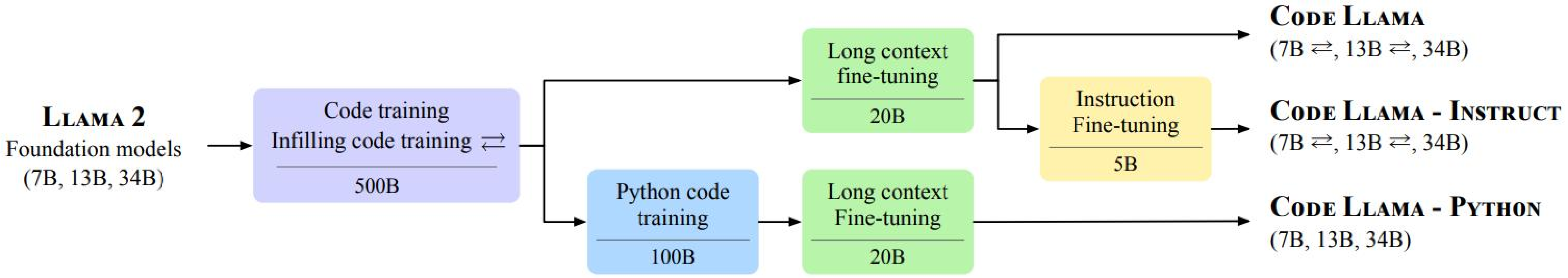
Pretrained LM → Instruction-tune on many tasks: B, C, D, … → Inference on task A

Model learns to perform many tasks via natural language instructions

Inference on unseen task

总结与展望—微调大模型

Direction How to fine-tune an specified LLM model?



LLAMA 2
Foundation models
(7B, 13B, 34B)

Code training
Infilling code training ⇄
500B

Python code training
100B

Long context fine-tuning
20B

Long context Fine-tuning
20B

Instruction Fine-tuning
5B

CODE LLAMA
(7B ⇄, 13B ⇄, 34B)

CODE LLAMA - INSTRUCT
(7B ⇄, 13B ⇄, 34B)

CODE LLAMA - PYTHON
(7B, 13B, 34B)

```
{
    "instruction": "Detect whether the following code contains
                                          vulnerabilities.",
    "input": "static struct pktcdvd_device
                   *pkt_find_dev_from_minor(int dev_minor)\n{\n\tif
                   (dev_minor >= MAX_WRITERS)\n\t\treturn
                   NULL;\n\treturn pkt_devs[dev_minor];\n}",
    "output": "1"
```

```
{
    “instruction”:“Detect whether the following code contains vulnerabilities.”,
    “input”:“static struct pktcdvd_device *pkt_find_dev_from_minor(int dev_minor)\n{\n\tif (dev_minor >=
MAX_WRITERS)\n\t\treturn NULL;\n\treturn pkt_devs[dev_minor];\n}”,
    “output”:“vulnerable. The sign of 'dev_minor' is not checked,  which could permit a negative integer to bypass the
'dev_minor >= MAX_WRITERS' check. This can allow an invalid memory access to occur when 'dev_minor' is used as an
index for 'pkt_devs', leading to sensitive information leaks or system crashes. "
}
```

THANKS