南京郵電大學 Nanjing University of Posts and Telecommunications

Code Difference Guided Adversarial Example Generation for Deep Code Models

《代码差异引导的深度代码模型对抗样本生成》

汇报人：张扬
指导老师：黄海平

Tian Z, Chen J, Jin Z. Code Difference Guided Adversarial Example Generation for Deep Code Models[C]//2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2023: 850-862.

目 录

01

背景介绍

背景介绍

模型的对抗鲁棒性(Adversarial Robustness)

深度学习模型的对抗鲁棒性是指通过对测试集中的样本添加微小的扰动，导致神经网络对其进行误分类，而人类依然能够正确分类该样本，这一行为可以使得经过良好训练的神经网络的检测准确率降为0。

```
void f1(int a[], int n){
  int i; int j; int k;
  for (i = 0; i < n; i++) {
    for (j = 0; j < ((n - i) - 1); j++) {
      if (a[j] > a[j + 1]){
        k = a[j];
        a[j] = a[j + 1];
        a[j + 1] = k;
      }
    }
  }
}
```

Ground-truth Label: sort
Prediction Result: sort (96.52%)

```
int f2(int t[], int len){
  int i; int j;
  i = 0; j = 0;
  while (len != 0) {
    t[i] = len % 10;
    len /= 10;
    i = i + 1;
  }
  while (j < i){
    if (t[j] != t[(i - j) - 1]) return 0;
    j = j + 1;
  }
  return 1;
}
```

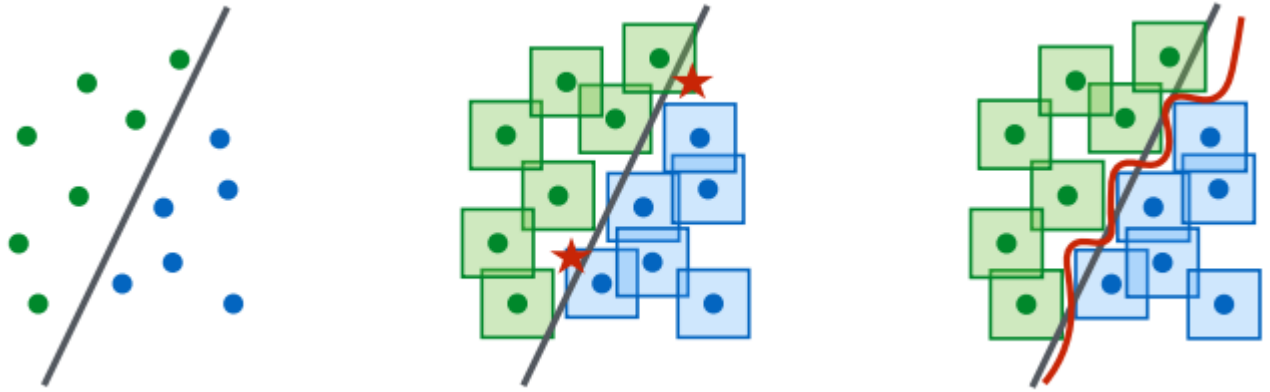
Ground-truth Label: palindrome
Prediction Result: palindrome (99.98%)

```
void f3(int t[], int len){
  int i; int j; int k;
  i = 0;
  while (i < len) {
    j = 0;
    while (j < ((len - i) - 1)) {
      if (t[j] > t[j + 1]){
        k = t[j];
        t[j] = t[j + 1];
        t[j + 1] = k;
      } j = j + 1;
    } i = i + 1;
  }
}
```

Ground-truth Label: sort
Prediction Result: palindrome (90.88%)

对抗样本(Adversarial Samples)及对抗训练(Adversarial Training)

- 上述能够在人眼难以察觉的条件下扰乱深度学习模型预测结果的样本，即为对抗样本。这种攻击方式的存在使得深度学习模型的鲁棒性受到挑战，研究者们不得不提出应对举措。
- 学术界的主流方法是对抗训练，最初由$Goodfellow$[1]等人提出。其思路非常简单直接，将生成的对抗样本加入到训练集中，做一个数据增强，让模型在训练时就学习对抗样本的特征，如下图所示[2]。



[1]Goodfellow I J, Shlens J, Szegedy C. Explaining and harnessing adversarial examples[J]. arXiv preprint arXiv:1412.6572, 2014.
[2]Madry A, Makelov A, Schmidt L, et al. Towards deep learning models resistant to adversarial attacks[J]. arXiv preprint arXiv:1706.06083, 2017.

背景介绍

编程语言模型中的对抗样本生成：

- Casalnuovo et al.在2020年的一篇论文[1]中提出了代码的“双通道”理念，实际上是对代码文件的不同理解形式，它们分别为(1)formal channel：用于编译与运行代码文件的信息，不包含与代码逻辑无关的噪音信息；(2)natural channel：以便于人类理解代码为目的，通常包含与代码运行无关的噪音信息。
- 由于编程语言中包含的自然语言属性，许多对抗样本生成方法均基于此进行相关工作，然而自然语言处理等操作**可能导致代码原始语义的改变**，从而影响代码的原始功能。
- 此外，一些深度学习模型可以识别代码的结构和调用链，从而实现漏洞检测等功能，这是其他领域中对抗样本生成方法没有涉及到的。

```
public override void WriteByte(
      byte b) {
  if (outerInstance.upto ==
        outerInstance.blockSize)
         {... }}
```

[1] Casalnuovo, C.; Barr, E. T.; Dash, S. K.; Devanbu, P.; and Morgan, E. 2020. A theory of dual channel constraints. In 2020 IEEE/ACM 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSENIER), 25–28. IEEE.

问题描述

- **巨大的搜索空间：**代码中通常包含数量庞大的自定义变量和方法，要通过变换这些变量和方法生成对抗样本，如果不施加有效的约束条件限制搜索空间，那么算法的时间复杂度将会很高。
- **频繁请求目标模型：**现有的对抗样本生成技术大都频繁请求目标模型的返回结果，以辅助对抗样本的生成，这会大大降低对抗样本的生成效率。
- **对代码语义的保护：**保护代码的自然语义有利于研究者理解模型的错误，因此在生成对抗样本时要注意维护其自然语义。现有的方法均未考虑到这一点，有些方法在生成样本时添加过多的不可达代码段(dead code)，导致代码自然语义受到损害。

方法建模

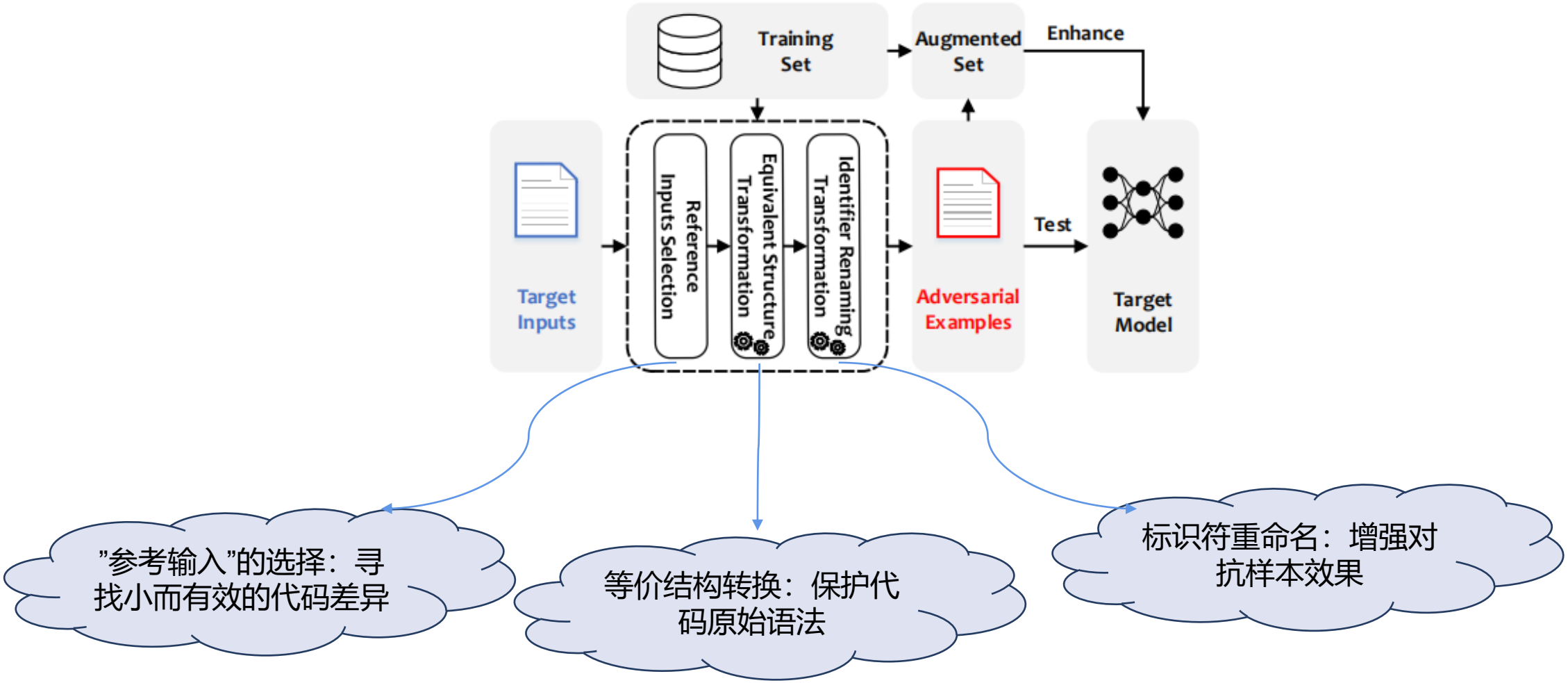
代码差异引导的深度代码模型对抗样本生成方法(COde Difference guided Adversarial examples generation，CODA)：

- 使用不同代码之间的代码差异，引导对抗样本生成，缩小搜索空间；
- 评估代码结构以及标识符的差异，设计保护代码语义的转换规则；
- 改善方案的整体实施流程，避免在过程中频繁请求目标模型。

02

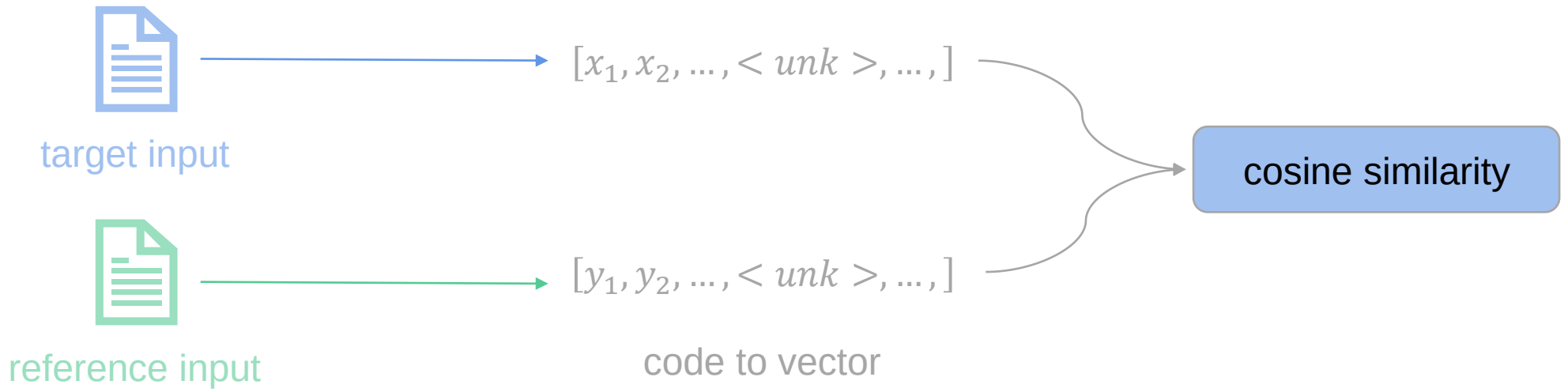
方法概述

总体架构


Training Set
Augmented Set
Enhance
Reference Inputs Selection
Equivalent Structure Transformation
Identifier Renaming Transformation
Target Inputs
Adversarial Examples
Test
Target Model
”参考输入”的选择：寻找小而有效的代码差异
等价结构转换：保护代码原始语法
标识符重命名：增强对抗样本效果

“参考输入”的选择

- **选择范围**：假定“目标输入”的真实预测标签为 c_i，且该标签有较大可能性转换为可能性第二的预测标签 c_j。则“参考输入”的选取应从预测标签为c_j的样本中选取。
- **筛选条件**：比较“目标输入”与“参考输入”的相似度，选用相似度高的“参考输入”。由于单一“参考输入”可能仅含有较少代码差异，因此需选用$Top-N$“参考输入”构成集合。
- **噪声处理**：不同代码段可能含有不同的标识符命名，导致代码段间相似度较低，因此引入$< unk >$占位符，在计算相似度时替换标识符所在位置，剔除代码结构外的影响。



等价结构转换

- **结构定义**：每种规则均对应前后两种代码结构，记为s_a和s_b，并将两种结构在“参考输入”集中的出现次数记为n_a和n_b，易得两者的分布分别为$\frac{n_a}{n_a+n_b}$、$\frac{n_b}{n_a+n_b}$。
- **实施方法**：等价结构转换的目标是通过改变“目标输入”中代码结构的分布，缩小与“参考输入”集的差异度。因此，本文采用概率的方法，将“目标输入”中的各结构转换为“参考输入”中分布较大的表示形式。

Transformation	Description
R_1-*loop*	equivalent transformation among `for` structure and `while` structure
R_2-*branch*	equivalent transformation between `if-else(-if)` structure and `if-if` structure
R_3-*calculation*	equivalent numerical calculation transformation, e.g., `++`, `--`, `+=`, `-=`, `*=`, `/=`, `%=`, `<<=`, `>>=`, `&=`, `|=` , `^=`
R_4-*constant*	equivalent transformation between a constant and a variable assigned by the same constant

```c
void f1(int a[], int n){
  int i; int j; int k;
  for (i = 0; i < n; i++) {
    for (j = 0; j < ((n - i) - 1); j++) {
      if (a[j] > a[j + 1]){
        k = a[j];
        a[j] = a[j + 1];
        a[j + 1] = k;
      }
    }
  }
}
```

Ground-truth Label: sort
Prediction Result: sort (96.52%)

```c
int f2(int t[], int len){
  int i; int j;
  i = 0; j = 0;
  while (len != 0) {
    t[i] = len % 10;
    len /= 10;
    i = i + 1;
  }
  while (j < i){
    if (t[j] != t[(i - j) - 1]) return 0;
    j = j + 1;
  }
  return 1;
}
```

Ground-truth Label: palindrome
Prediction Result: palindrome (99.98%)

```c
void f3(int t[], int len){
  int i; int j; int k;
  i = 0;
  while (i < len) {
    j = 0;
    while (j < ((len - i) - 1)) {
      if (t[j] > t[j + 1]){
        k = t[j];
        t[j] = t[j + 1];
        t[j + 1] = k;
      } j = j + 1;
    } i = i + 1;
  }
}
```

Ground-truth Label: sort
Prediction Result: palindrome (90.88%)

“目标输入”
for型循环较多

“参考输入”
while型循环较多

标识符重命名

- 基于自然语言的标识符重命名变换十分容易导致代码原始语义的改变，因此本文将该变换作为辅助模块，以加强对抗样本的强度。
- **实施方法**：使用$FastText$模型将“目标输入”和“参考输入”中的标识符转化为向量集，分别记为V_t和V_r。随后通过比较向量的余弦相似度，优先替换相似度高的向量组。
- **终止条件**：在样本已经能够转变“目标输入”的预测标签时，即终止重命名，此时得到的输入样本即为最终对抗样本。

```
void f1(int a[], int n){
  int i; int j; int k;
  for (i = 0; i < n; i++) {
    for (j = 0; j < ((n - i) - 1); j++) {
      if (a[j] > a[j + 1]){
        k = a[j];
        a[j] = a[j + 1];
        a[j + 1] = k;
      }
    }
  }
}
```

Ground-truth Label: sort
Prediction Result: sort (96.52%)

```
int f2(int t[], int len){
  int i; int j;
  i = 0; j = 0;
  while (len != 0) {
    t[i] = len % 10;
    len /= 10;
    i = i + 1;
  }
  while (j < i){
    if (t[j] != t[(i - j) - 1]) return 0;
    j = j + 1;
  }
  return 1;
}
```

Ground-truth Label: palindrome
Prediction Result: palindrome (99.98%)

```
void f3(int t[], int len){
  int i; int j; int k;
  i = 0;
  while (i < len) {
    j = 0;
    while (j < ((len - i) - 1)) {
      if (t[j] > t[j + 1]){
        k = t[j];
        t[j] = t[j + 1];
        t[j + 1] = k;
      } j = j + 1;
    } i = i + 1;
  }
}
```

Ground-truth Label: sort
Prediction Result: palindrome (90.88%)

“目标输入”
含a、n等变量

“参考输入”
含t、len等变量

03

实验结果分析

研究问题（RQ）

- RQ1：对比基线方法，CODA在有效性和执行效率方面表现如何?
- RQ2：CODA生成的对抗样本是否有利于提高模型鲁棒性?
- RQ3：不同组件是否均对整体方法做出了贡献?
- RQ4：人类是否能分辨CODA生成的对抗样本?

数据集选择

Task	Train/Val/Test	Class	Language	Model	Acc.
Vulnerability Prediction	21,854/2,732/2,732	2	C	CodeBERT GCBERT CodeT5	63.76% 63.65% 63.83%
Clone Detection	90,102/4,000/4,000	2	Java	CodeBERT GCBERT CodeT5	96.97% 97.36% 98.08%
Authorship Attribution	528/–/132	66	Python	CodeBERT GCBERT CodeT5	90.35% 89.48% 92.30%
Functionality Classification	41,581/–/10,395	104	C	CodeBERT GCBERT CodeT5	98.18% 98.66% 98.79%
Defect Prediction	27,058/–/6,764	4	C/C++	CodeBERT GCBERT CodeT5	84.37% 83.98% 81.54%

* GCBERT is short for GraphCodeBERT.

- 测试模型：$CodeBERT$、$GraphCodeBERT$、$CodeT5$
- 基线方法：$ALERT$[1]、$CARROT$[2]

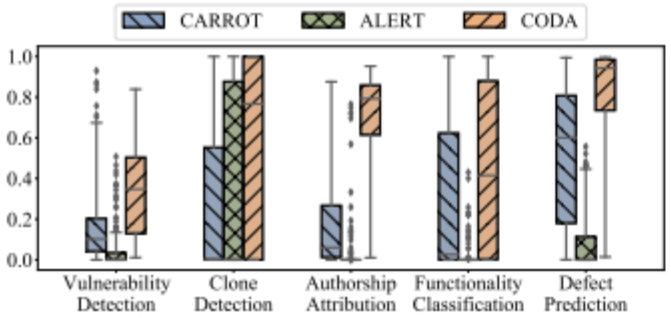
[1] Yang Z, Shi J, He J, et al. Natural attack for pre-trained models of code[C]//Proceedings of the 44th International Conference on Software Engineering. 2022: 1482-1493.
[2] Zhang H, Fu Z, Li G, et al. Towards robustness of deep program processing models—detection, estimation, and enhancement[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2022, 31(3): 1-40.

RQ1：有效性和执行效率

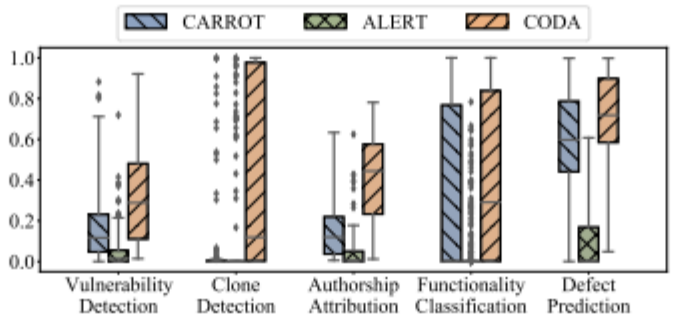
模型故障率RFR

Task	CodeBERT			GraphCodeBERT			CodeT5		
	CARROT	ALERT	CODA	CARROT	ALERT	CODA	CARROT	ALERT	CODA
Vulnerability Prediction	33.72%	53.62%	**89.58%**	37.40%	76.95%	**94.72%**	84.32%	82.69%	**98.87%**
Clone Detection	20.78%	27.79%	**44.65%**	3.50%	7.96%	**27.37%**	12.89%	14.29%	**42.07%**
Authorship Attribution	44.44%	35.78%	**79.05%**	31.68%	61.47%	**92.00%**	20.56%	66.41%	**97.17%**
Functionality Classification	44.15%	10.04%	**56.74%**	42.76%	11.22%	**57.44%**	38.26%	35.37%	**78.07%**
Defect Prediction	71.59%	65.15%	**95.18%**	79.08%	75.87%	**96.58%**	38.26%	35.37%	**78.07%**
Average	42.94%	38.48%	**73.04%**	38.88%	46.69%	**73.62%**	33.91%	40.99%	**70.96%**

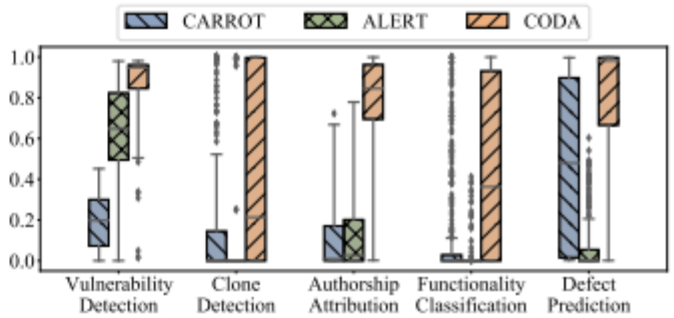
预测置信度PCD



(a) PCD on testing CodeBERT

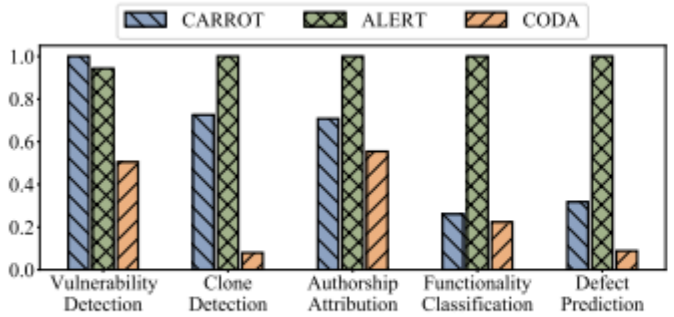


(b) PCD on testing GraphCodeBERT

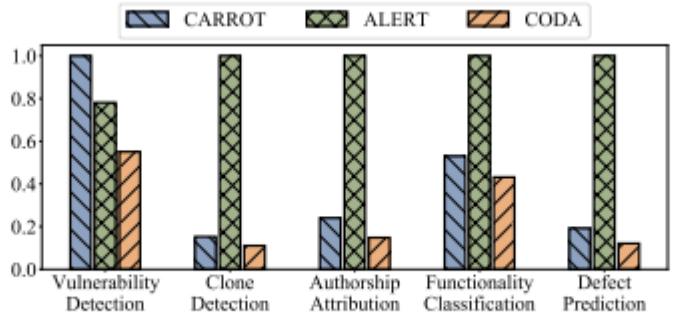


(c) PCD on testing CodeT5

模型询问次数



(a) Model invocations on testing CodeBERT



(b) Model invocations on testing GraphCodeBERT

(c) Model invocations on testing CodeT5

RQ2：对抗训练效果

Task	Model	Ori			CARROT			ALERT			CODA		
		CARROT	ALERT	CODA	CARROT	ALERT	CODA	CARROT	ALERT	CODA	CARROT	ALERT	CODA
Vulnerability Prediction	CodeBERT	62.96%	62.77%	**63.03%**	29.14%	21.11%	**29.69%**	23.43%	26.27%	**34.44%**	32.16%	31.73%	**38.82%**
	GraphCodeBERT	**62.99%**	62.88%	62.92%	12.37%	19.59%	**21.65%**	16.33%	17.35%	**23.71%**	25.77%	24.74%	**34.02%**
	CodeT5	63.69%	63.81%	**63.92%**	52.03%	39.76%	**82.03%**	42.26%	**49.11%**	44.26%	41.43%	45.52%	**52.54%**
Clone Detection	CodeBERT	97.39%	96.45%	**97.45%**	83.15%	42.31%	**94.44%**	52.65%	72.46%	**75.32%**	38.51%	71.45%	**89.78%**
	GraphCodeBERT	97.01%	97.22%	**97.43%**	75.00%	66.67%	**77.50%**	79.17%	84.29%	**92.31%**	35.71%	57.69%	**92.97%**
	CodeT5	97.73%	97.14%	**98.10%**	67.77%	57.63%	**75.85%**	69.94%	64.36%	**81.63%**	42.15%	51.74%	**79.88%**
Authorship Attribution	CodeBERT	90.55%	89.39%	**90.91%**	**45.06%**	40.67%	41.03%	51.25%	56.25%	**58.82%**	45.67%	43.33%	**76.47%**
	GraphCodeBERT	89.39%	88.72%	**90.35%**	**81.75%**	67.08%	72.40%	79.41%	78.67%	**100.00%**	45.59%	80.39%	**84.75%**
	CodeT5	92.43%	92.68%	**93.03%**	70.95%	65.91%	**73.48%**	55.73%	71.88%	**76.44%**	44.31%	52.56%	**72.37%**
Functionality Classification	CodeBERT	98.11%	98.52%	**98.56%**	**83.46%**	72.80%	81.51%	70.83%	71.75%	**79.41%**	78.92%	71.18%	**95.43%**
	GraphCodeBERT	98.48%	98.55%	**98.72%**	67.53%	75.19%	**77.27%**	32.04%	52.62%	**62.98%**	91.22%	90.81%	**93.08%**
	CodeT5	97.92%	98.46%	**98.63%**	25.31%	21.33%	**27.36%**	41.07%	57.14%	**57.42%**	24.87%	59.58%	**63.76%**
Defect Prediction	CodeBERT	83.50%	84.16%	**84.44%**	52.73%	25.81%	**66.03%**	74.88%	75.87%	**83.12%**	76.86%	68.66%	**85.36%**
	GraphCodeBERT	83.34%	84.00%	**84.53%**	68.20%	48.54%	**74.88%**	52.73%	**63.91%**	59.45%	67.08%	68.66%	**76.14%**
	CodeT5	80.92%	81.32%	**81.57%**	31.48%	34.08%	**37.73%**	31.75%	42.22%	**55.77%**	54.45%	54.18%	**73.83%**
Average		86.43%	86.40%	**86.91%**	56.40%	46.57%	**62.19%**	51.56%	58.94%	**65.67%**	49.65%	58.15%	**73.95%**

进行对抗训练后，$CODA$在原始预测准确率不变的情况下，有效地降低了对抗攻击成功率。

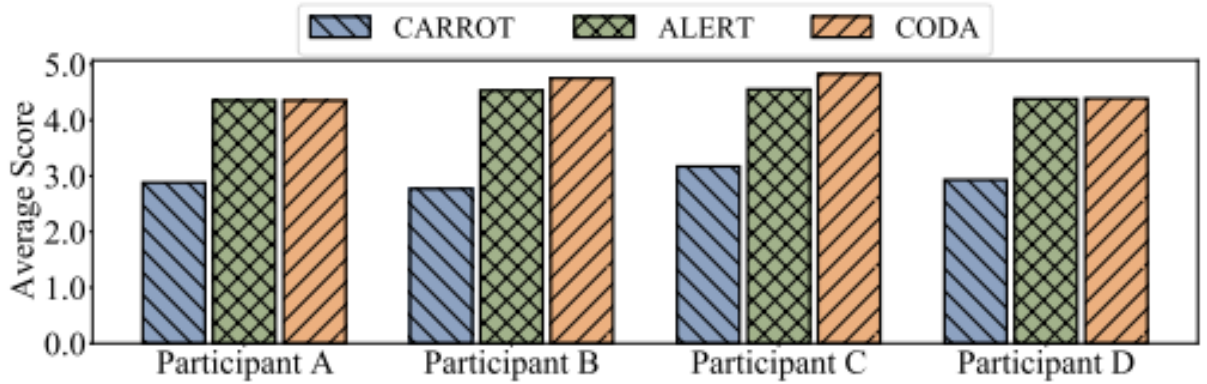
RQ3：组件评估

Model	w/o RIS	w/o EST	w/o CDG	w/o IRT	CODA
CodeBERT	30.83%	62.73%	63.08%	35.14%	**73.04%**
GraphCodeBERT	29.49%	62.41%	61.98%	26.24%	**73.62%**
CodeT5	26.75%	50.74%	57.98%	38.21%	**70.96%**

- RIS：“参考输入”的选择
- EST：等价结构转换
- CDG：等价结构转换中的代码差异引导机制
- IRT：标识符重命名

缺少任一组件均会导致攻击成功率的降低，证明各组件均对整体有效性做出了贡献

RQ4：对抗样本质量



上图所示为4位参与者对不同方法生成的对抗样本进行评估的结果，分数越高表示该样本更加符合代码语义。结果显示$CODA$生成的样本质量与$ALERT$几乎持平，而$ALERT$方法的主要贡献就是维持生成对抗样本的代码语义。

04

思考总结

总结&思考

- $CODA$创新性地使用“参考输入”来辅助对抗样本的生成，大大缩小了流程中的搜索空间，以及对目标模型的查询次数。
- **在组件评估实验中**，实验结果显示去除“标识符重命名”模块对攻击成功率的影响大于去除“等价结构替换”模块，然而本文中“等价结构替换”为主要模块，重命名模块为辅助模块，这一结果是否有些喧宾夺主？究其原因，测试中使用的各种预训练代码模型，均是基于预训练语言表征模型 $BERT$ 进行开发，固然对自然语言的改变较为敏感，因此在应对此类模型时，重命名操作贡献较大。而面对其他基于代码调用链进行相关检测的模型，也许更能体现代码结构替换带来的作用。

Model	w/o RIS	w/o EST	w/o CDG	w/o IRT	CODA
CodeBERT	30.83%	62.73%	63.08%	35.14%	**73.04%**
GraphCodeBERT	29.49%	62.41%	61.98%	26.24%	**73.62%**
CodeT5	26.75%	50.74%	57.98%	38.21%	**70.96%**

感谢观看