Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

面向深度学习软件库的API层漏洞挖掘方法

硕士研究生 赵智洋

2023年02月26日

内容提要

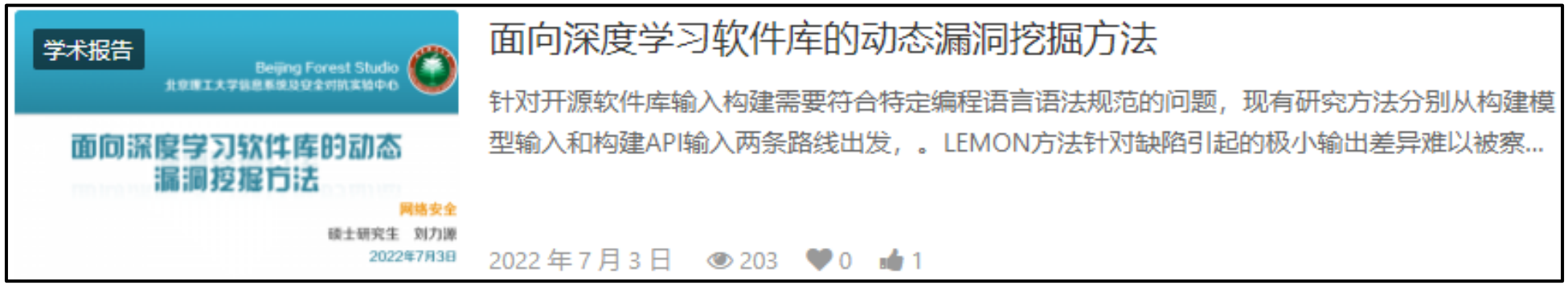
- 背景简介
- 基本概念
- 算法原理
- 总结

背景简介

- 预期收获
  - 了解深度学习软件库缺陷或漏洞现状
  - 了解深度学习软件库漏洞挖掘的分类
  - 理解最新的API层深度学习软件库漏洞挖掘方法

背景简介

- 前情提要
  - 2022年7月3日：《面向深度学习软件库的动态漏洞挖掘方法》

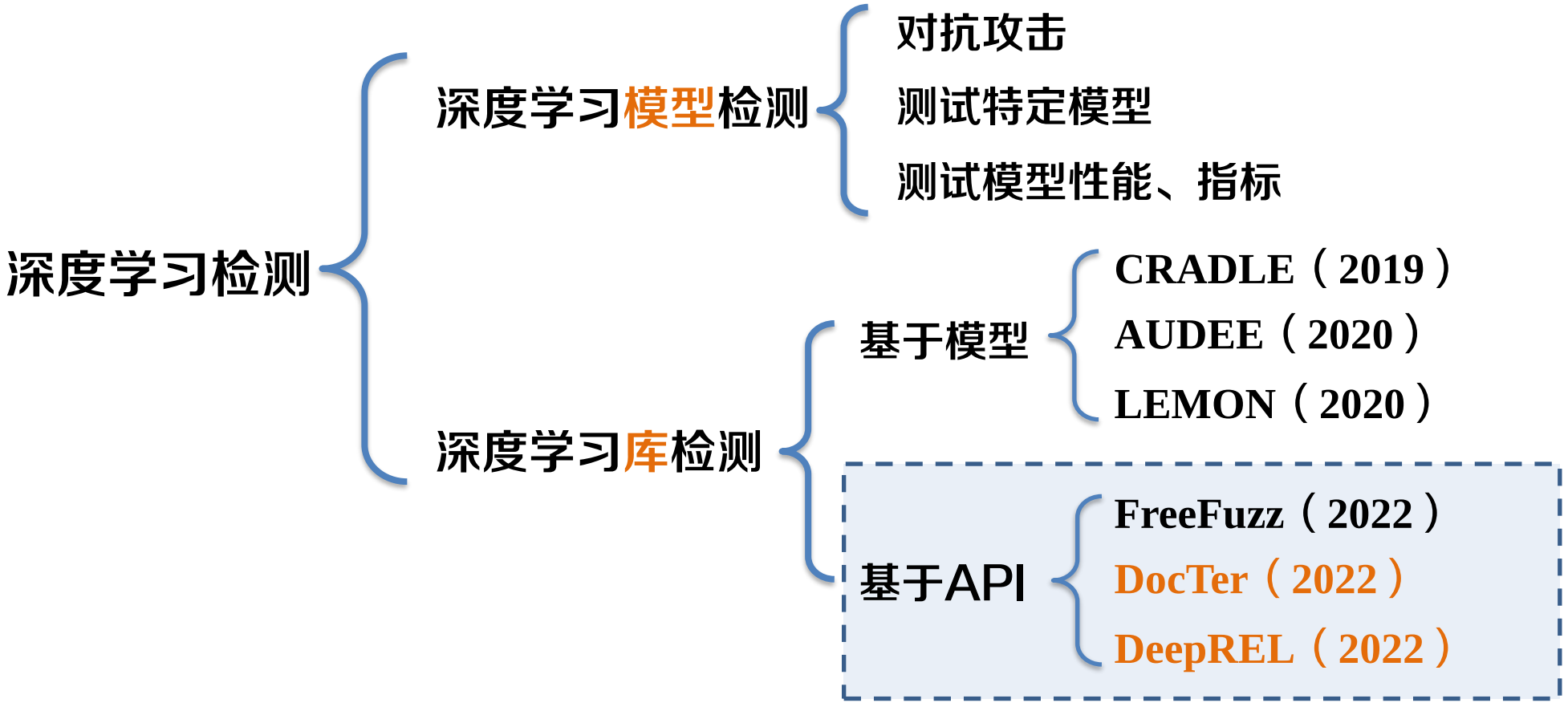


  - 2022年11月29日：《筑牢AI安全基石——BFS团队提交15个TensorFlow现存漏洞》

漏洞名称	`CHECK` fail in `CollectiveGather`
CVE编号	CVE-2022-35994
影响软件包	TensorFlow、TensorFlow-CPU、TensorFlow-GPU(pip)
影响版本	<2.10.0
补丁版本	2.7.4, 2.8.3, 2.9.2, 2.10.0

背景简介

- 深度学习检测分类



背景简介

- 进行深度学习库检测的重要性
  - 深度学习库是深度学习模型的基础
  - PyTorch、TensorFlow等深度学习库使用广泛
    - 坊间传闻：TensorFlow适合业界，PyTorch适合学界？
- 常见深度学习库

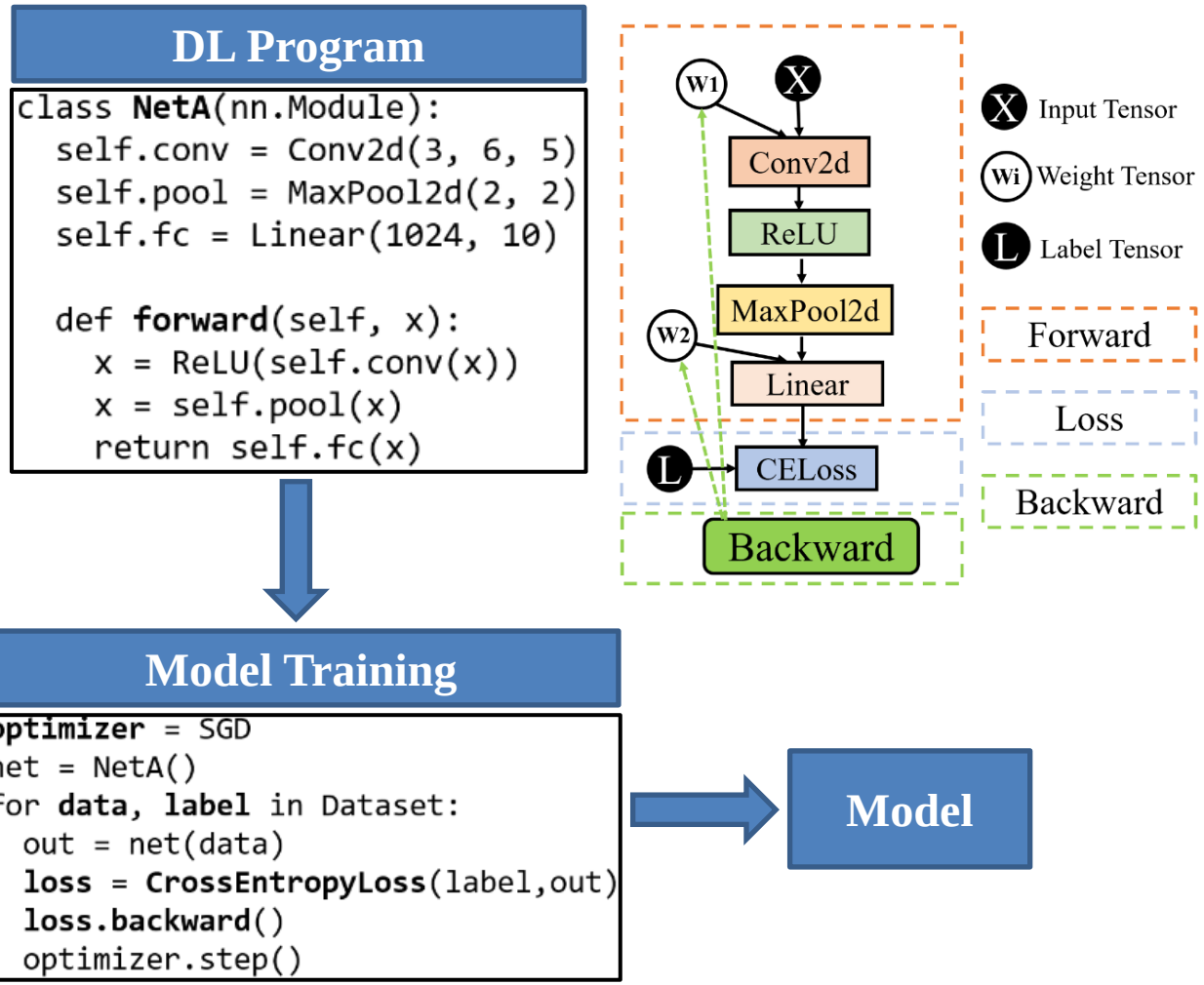
mxnet

PyTorch

TensorFlow

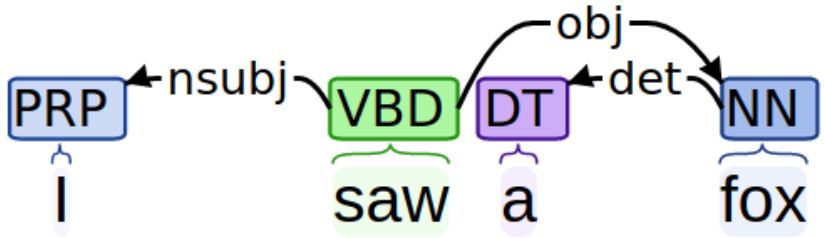
基本概念

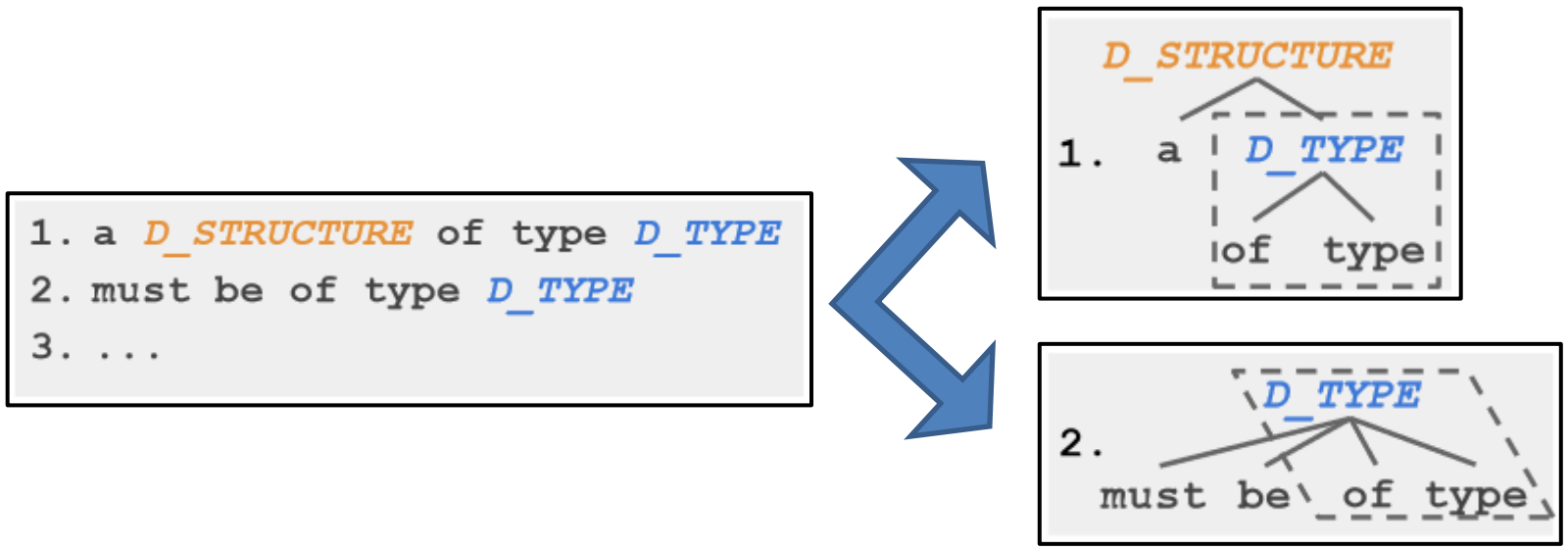
- 深度学习模型、深度学习库和API
  - 深度学习模型：
    - Deep Learning Model， DL Model
  - 深度学习库：
    - Deep Learnining Library，DL Library
  - 应用程序编程接口：
    - Application Programming Interface， API



基本概念

- 依存句法树（Dependency Parse Tree）
  - Parsing：“解析”，从给定的句子中创建解析树
    - 解析树：根据正式语法突出句子句法结构的树
  - 依存句法基本假设：
    - 句法结构本质上包含词和词之间的依存（修饰）关系
  - 依存关系构成：核心词（head）、依存词（dependent）



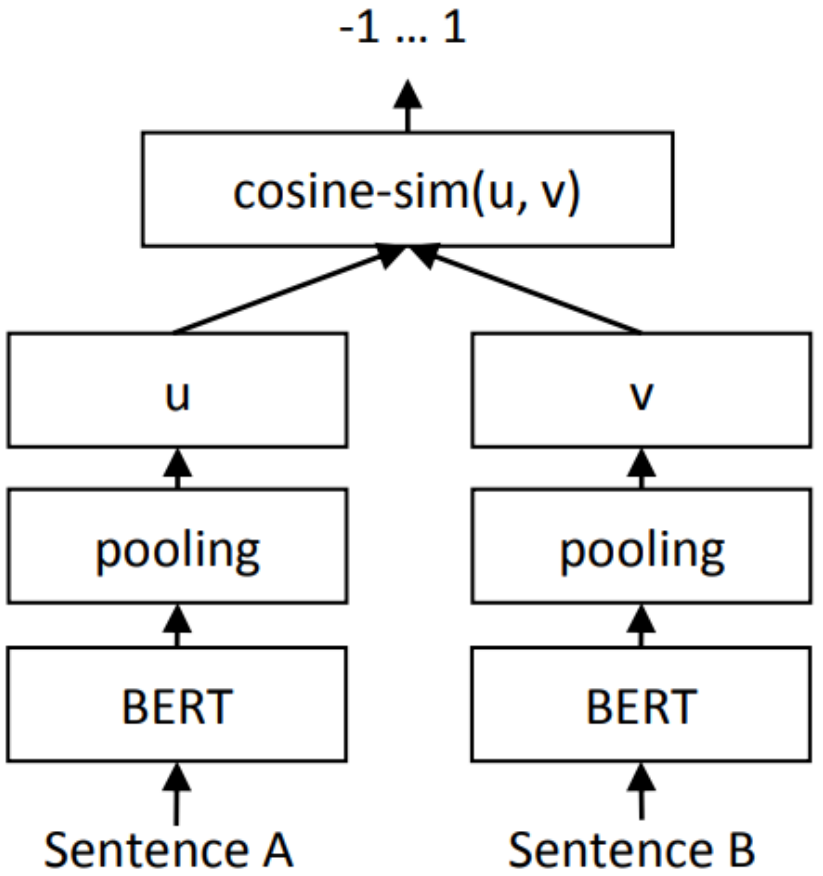


基本概念

- 词频-逆文本频率指数
  - TF-IDF： Term Frequency-Inverse Document Frequency
    - $TF-IDF = TF * IDF$
  - 词频（TF）
    - 某个词在某篇文中出现的频率
    - 词频 $= \frac{\text{某个词在某文章中出现的总次数}}{\text{某文章的总词数}}$
  - 逆文本频率（IDF）
    - 一个词普遍程度的度量
    - 逆文档频率$= log(\frac{\text{语料库的总文档数目}}{\text{包含该词的文档数}+1})$

基本概念

- Sentence-BERT（2019）
  - 目的
    - 获取2条输入语句的相似度
  - 结构
    - 基于BERT
    - 使用孪生网络的思想
      - 两个BERT参数共享
    - 计算余弦相似度
  - 优势
    - 减少BERT搜索最相似句子所需的时间



算法原理——DocTer

【DocTer】
Documentation-Guided Fuzzing for Testing Deep Learning API Functions

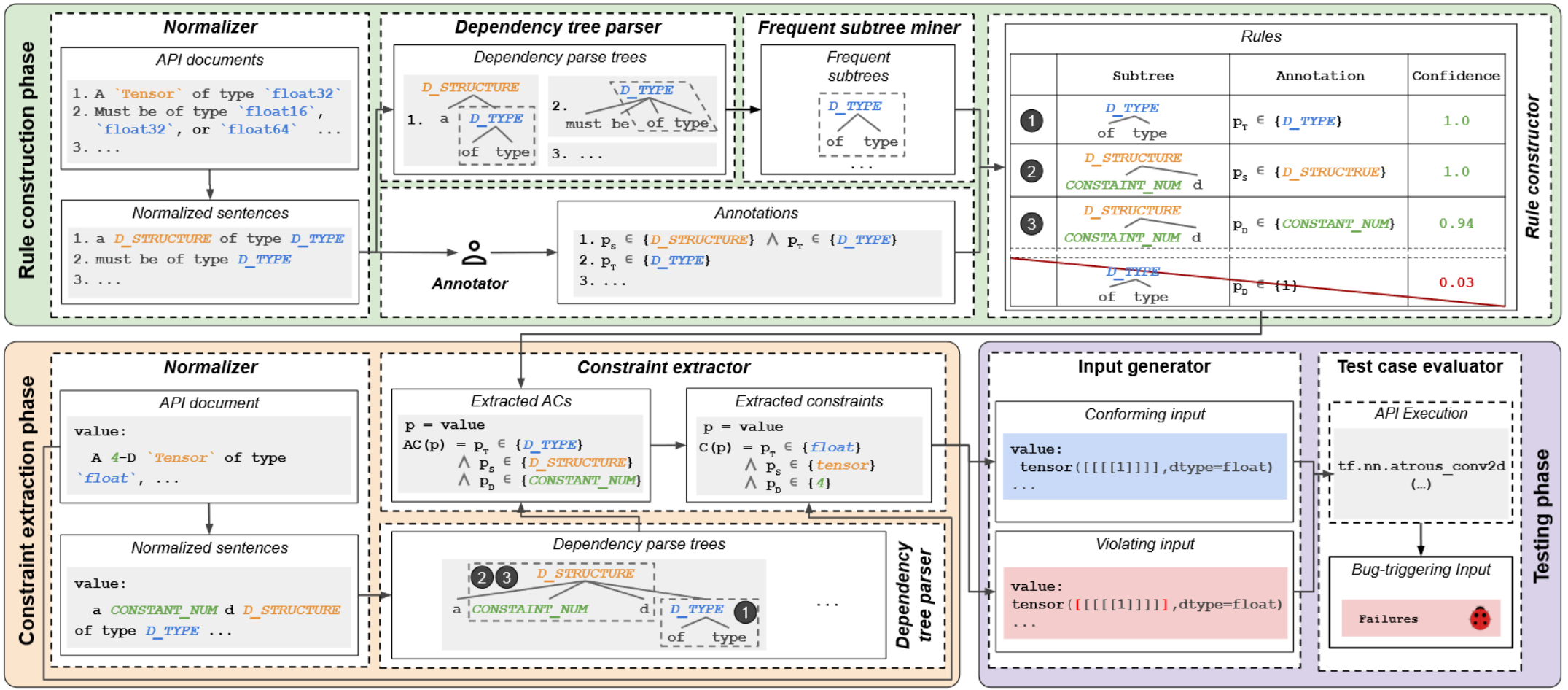
算法原理——DocTer

T	深度学习软件库漏洞挖掘
I	深度学习软件库；深度学习软件库的API文档
P	1. 构建规则：抽取API文档内容，建立文档描述与参数抽象约束之间的映射规则 2. 提取约束：根据规则，从文档的参数描述部分获取参数的具体约束 3. 测试：根据具体约束为API生成输入测试用例，检测运行结果
O	深度学习软件库及其API文档的漏洞

P	在为深度学习库中API生成输入时，缺乏考虑输入的约束
C	需要手工标注30%的参数
D	如何从自然语言编写的文档中提取输入约束
L	ISSTA 2022.07 (CCF A)

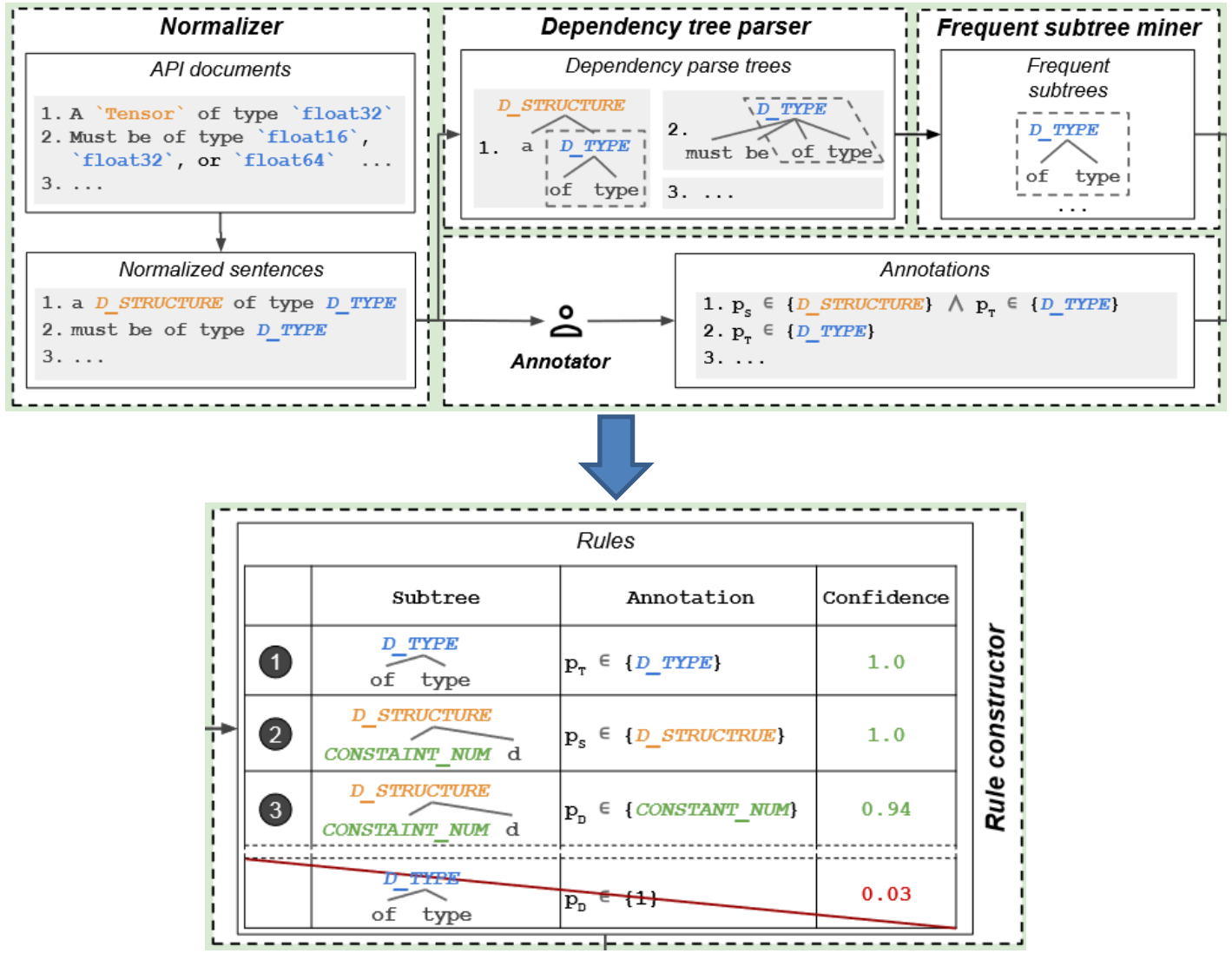
算法原理——DocTer

- 算法流程图
  - 构建规则、提取约束、测试



算法原理——DocTer

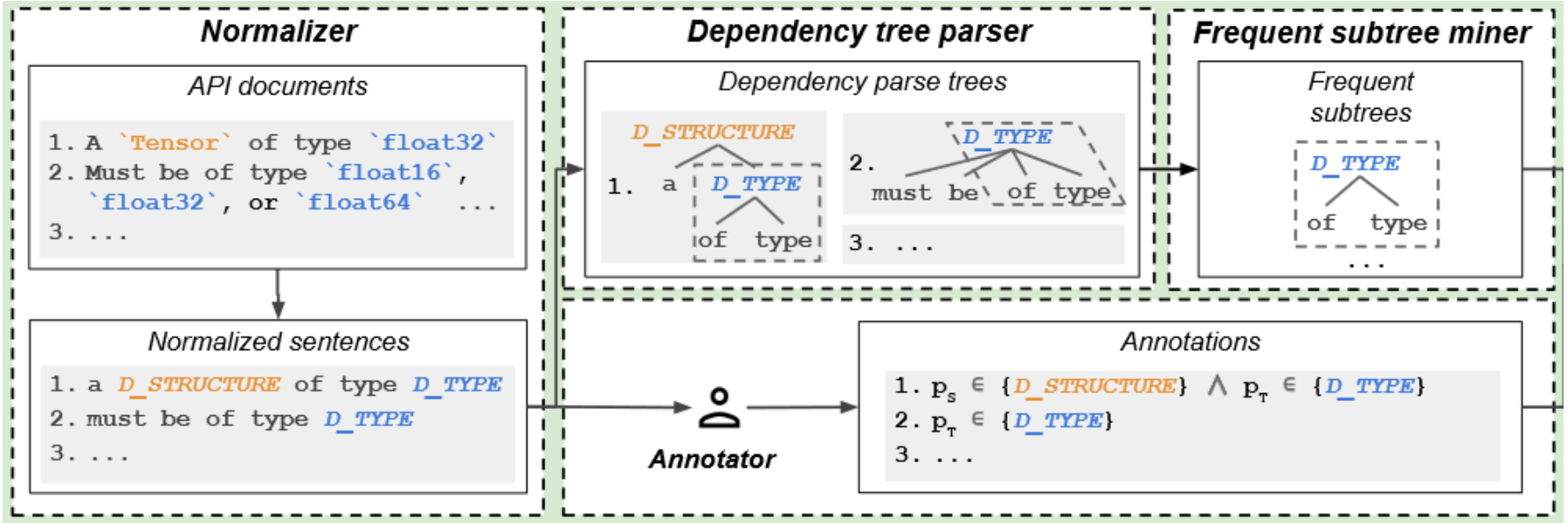
- 构建规则
  - 目的
    - 应对API文档中不同的实例化表述方式
    - 提取API文档中共有语法格式的部分
  - 步骤
    - 对API文档进行预处理
    - 建立映射
      - 建模为优化问题



算法原理——DocTer

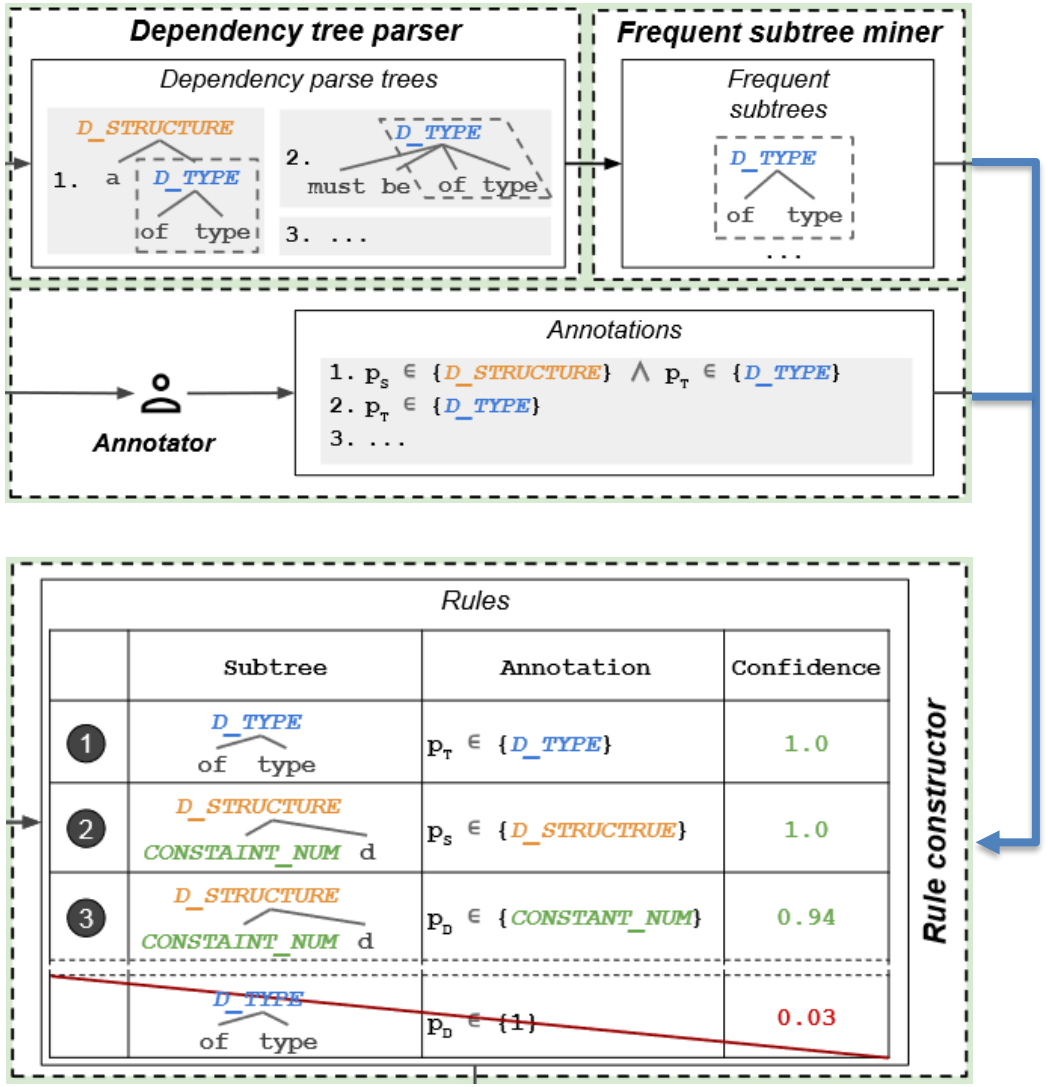
- 构建规则：对API文档进行预处理
  - 步骤
    - 抽取API文档进行规范化
    - 将规范化文档转换为依存树
      - 筛选出高频子树
    - 标注参数抽象约束

具象描述	具象描述实例	抽象表示
数据类型	Int32、float32	D_TYPE
数据结构	Tensor	D_STRUCTURE
整数	4	CONSTANT_NUM
参数名称	x	PARAM
…	…	…



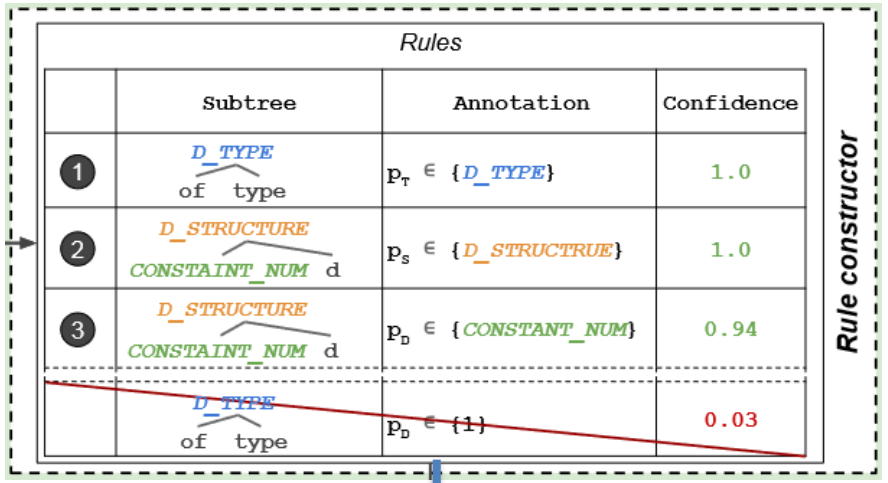
算法原理——DocTer

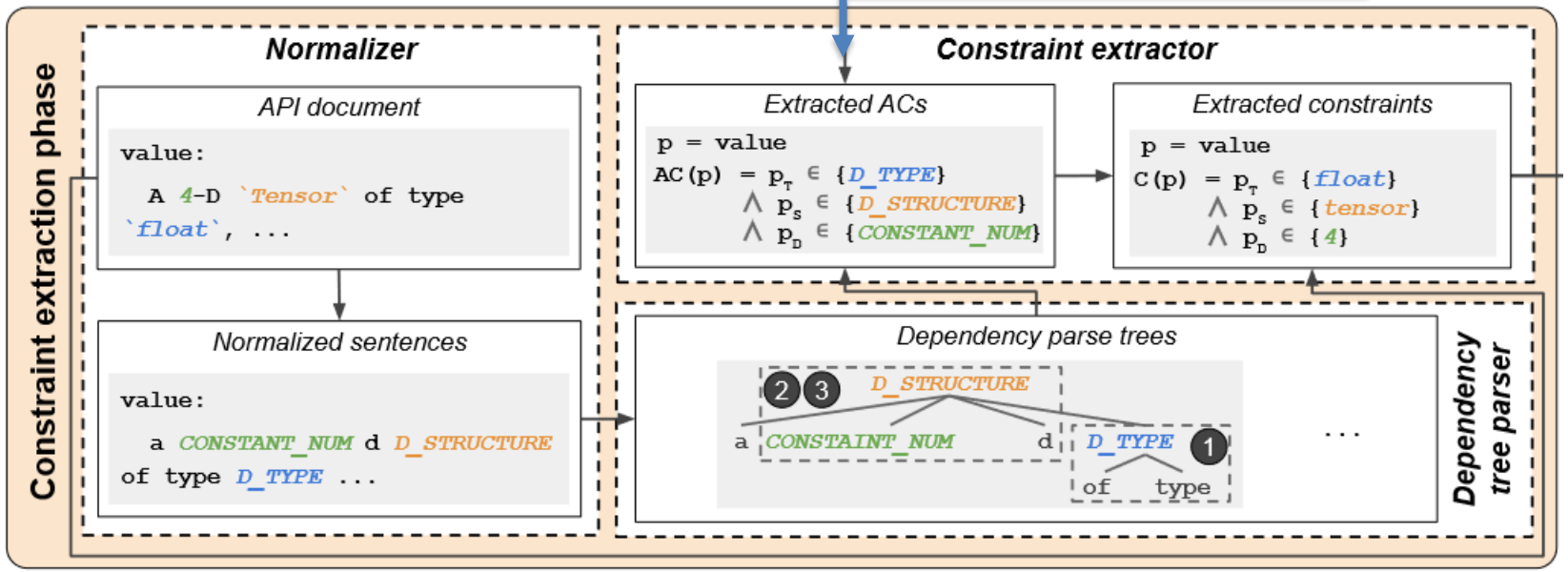
- 构建规则：建立映射
  - 步骤：建立高频子树与抽象约束间的映射
    - R: {高频子树 → 抽象约束}
  - 映射满足条件：
    - 全面
      - 抽象约束都与子树有映射关系
    - 数目少
    - 子树规模小
      - 提升泛化性
    - 子树能够精确预测抽象约束
      - 给定子树
      - 选择条件概率最高的抽象约束建立映射



算法原理——DocTer

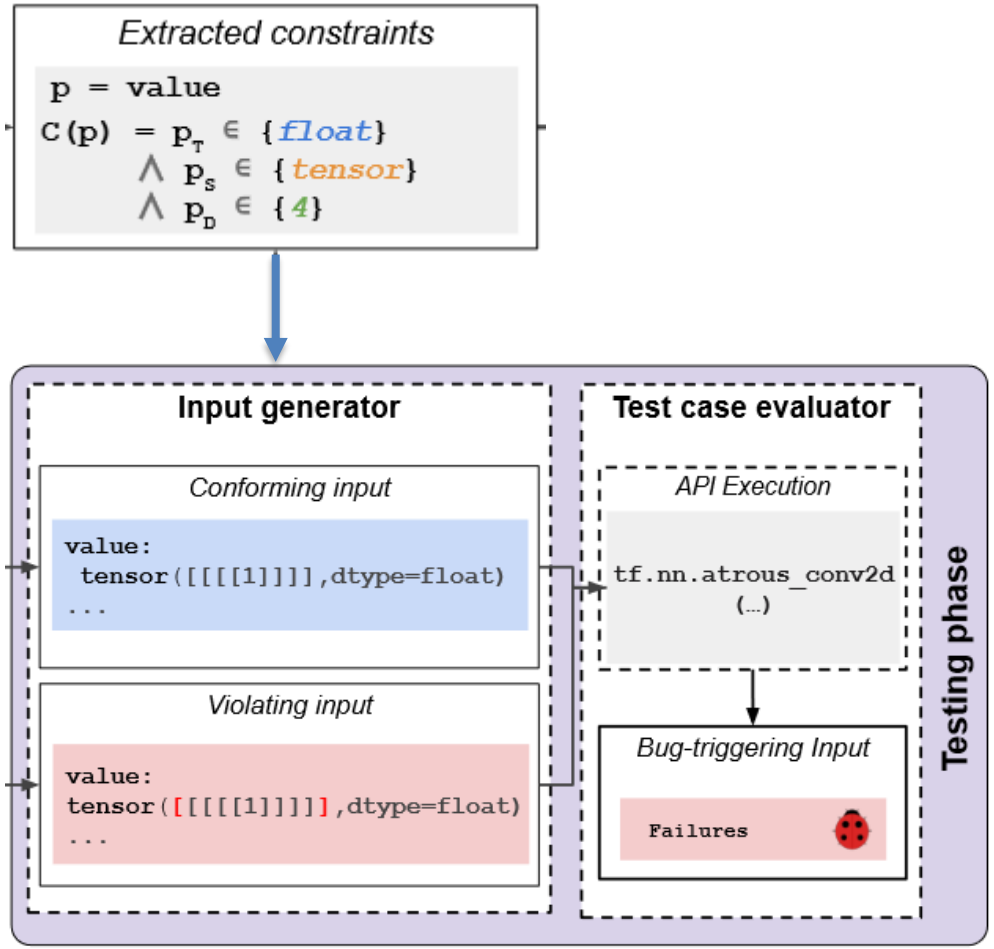
- 提取约束
  - 步骤：
    - 规范化API文档的参数描述
    - 将规范化文档转换为依存树
    - 根据树结构及规则给出抽象约束
    - 根据抽象约束和参数描述给出具体约束





算法原理——DocTer

- 测试
  - 根据参数具体约束，为API生成输入
    - 合法输入（conforming inputs, CIs）
      - 测试API函数的核心功能代码
    - 非法输入（violating inputs, VIs）
      - 测试API函数的输入有效性检测代码
  - 调用API，检测潜在bug
    - 引发非预期结果的输入及对应位点
      - 段错误（segmentation fault）
      - 浮点溢出
      - C++后端的总线错误



实验分析——DocTer

- 约束提取效果
  - 对比：手工标注
    - 抽取5%的输入参数
  - 实验结果
    - 从2415个API中抽取到了16035条参数约束
- 缺陷检测
  - 对比Baseline
    - 没有约束指导，生成输入
  - 实验结果
    - 检测到94个代码bug，43个此前未知的文档bug
      - 包括63个此前未知的代码bug
    - 假阳性：报告的66个未知bug中，仅3个无需修复

	TensorFlow	PyTorch	MXNet	Total/Avg
# APIs with constr. extracted	911	498	1,006	2,415
# constr. extracted	5,908	3,201	6,926	**16,035**
# constr. per API: *Avg (Min-Max)*	6.5 (1-51)	6.4 (1-33)	6.9 (1-111)	6.6 (1-65)
# evaluated param.	190	93	320	603
# evaluated param. with constr.	161	83	229	473
# evaluated constr.	350	170	363	883
Precision/Recall/F1 for All (%)	90.0/74.8/81.7	78.4/77.4/77.9	87.9/82.4/85.1	**85.4**/78.2/81.6
Precision/Recall/F1 for *dtype* (%)	93.0/82.3/87.3	78.1/79.4/78.7	92.9/81.9/87.0	88.0/81.2/84.3
Precision/Recall/F1 for *structure* (%)	78.9/88.2/83.3	85.7/90.0/87.8	91.7/90.2/92.4	85.4/89.5/87.8
Precision/Recall/F1 for *shape* (%)	89.1/74.5/81.2	80.0/76.9/78.4	76.1/79.8/77.9	81.7/77.1/79.2
Precision/Recall/F1 for *valid value* (%)	87.5/47.7/61.8	66.7/60.0/63.2	90.0/60.0/72.0	81.4/55.9/65.7

Approach		TensorFlow	PyTorch	MXNet	Total
Baseline		22 / 26 / 41 (79)	6 / 6 / 7 (8)	6 / 9 / 11 (21)	34 / 41 / 59 (108)
DocTer	All	31 / 38 / 61 (114)	13 / 13 / 18 (28)	10 / 12 / 15 (32)	54 / 63 / **94 (174)**
	CI	21 / 28 / 47 (93)	11 / 11 / 14 (23)	10 / 12 / 14 (27)	42 / 51 / 75 (143)
	VI	28 / 32 / 51 (83)	13 / 13 / 18 (25)	8 / 10 / 12 (27)	49 / 55 / 81 (135)

*：已被修复的bug/此前未知的bug/发现的总bug数目

实验分析——DocTer

- 漏洞示例
  - TensorFlow 2.4.0
    - API：$tf.quantization.quantize\_and\_dequantize$
  - 缺陷名称
    - 跨界内存读
  - CVE（Common Vulnerabilities & Exposures）

CVE编号	CVE-2020-15265
纰漏时间	2020/10/22
CVSS3评分	7.5/10
攻击复杂度	低
可用性	高

4 tensorflow/core/kernels/quantize_and_dequantize_op.cc

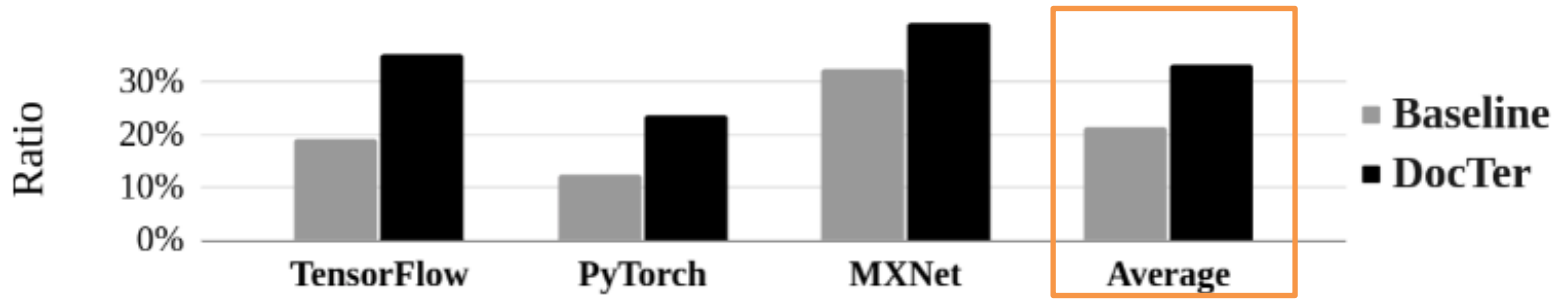
```
@@ -71,6 +71,10 @@ class QuantizeAndDequantizeV2Op : public OpKernel {
71  71
72  72    void Compute(OpKernelContext* ctx) override {
73  73      const Tensor& input = ctx->input(0);
    74  +   OP_REQUIRES(
    75  +       ctx, (axis_ == -1 || axis_ < input.shape().dims()),
    76  +       errors::InvalidArgument("Shape must be at least rank ", axis_ + 1,
    77  +                               " but is rank ", input.shape().dims()));
74  78      const int depth = (axis_ == -1) ? 1 : input.dim_size(axis_);
75  79      Tensor input_min_tensor;
76  80      Tensor input_max_tensor;
```

16 tensorflow/python/kernel_tests/array_ops_test.py

```
@@ -1628,6 +1628,22 @@ def testAxis(self):
1628 1628                    axis=(axis - 4)))
1629 1629          self.assertAllClose(fake_quantized, expected)
1630 1630
     1631 +  def testBadAxis(self):
     1632 +    input_tensor = [2.5, 2.5]
     1633 +    input_min = [0, 0]
     1634 +    input_max = [1, 1]
     1635 +    error_message_pattern = "Shape must be at least rank 11 but is rank 1"
     1636 +    # TODO(b/171260356): Eager mode and graph mode throw different error types
     1637 +    error = errors.InvalidArgumentError if context.executing_eagerly(
     1638 +    ) else ValueError
     1639 +    with self.assertRaisesRegex(error, error_message_pattern):
     1640 +      self.evaluate(
     1641 +          array_ops.quantize_and_dequantize_v2(
     1642 +              input=input_tensor,
     1643 +              input_min=input_min,
     1644 +              input_max=input_max,
     1645 +              axis=10))
     1646 +
1631 1647    def testQuantizeDequantizeGrad(self):
1632 1648      shape = (2, 2)
1633 1649      max_threshold = 0
```

实验分析——DocTer

- 输入有效性
  - 对比Baseline
    - 没有约束指导，生成输入
  - 评价
    - Baseline生成输入的通过率
    - DocTer生成合法输入的通过率
  - 实验结果
    - DocTer有33.4%的输入通过率，比Baseline（21.5%）提高了55.3%



算法原理——DeepREL

【DeepREL】
Fuzzing Deep-Learning Libraries via Automated Relational API Inference

算法原理——DeepREL

T	深度学习软件库漏洞挖掘
I	待测试的深度学习软件库；深度学习软件库的API文档；API调用用例数据集
P	1. 根据API文档计算API相似度，构建候选API对 2. 为候选API对生成调用代码 3. 根据调用代码从候选API对中，筛选出相似API对 4. 对相似API对进行模糊测试，以检测潜在漏洞
O	深度学习软件库及其API文档的漏洞

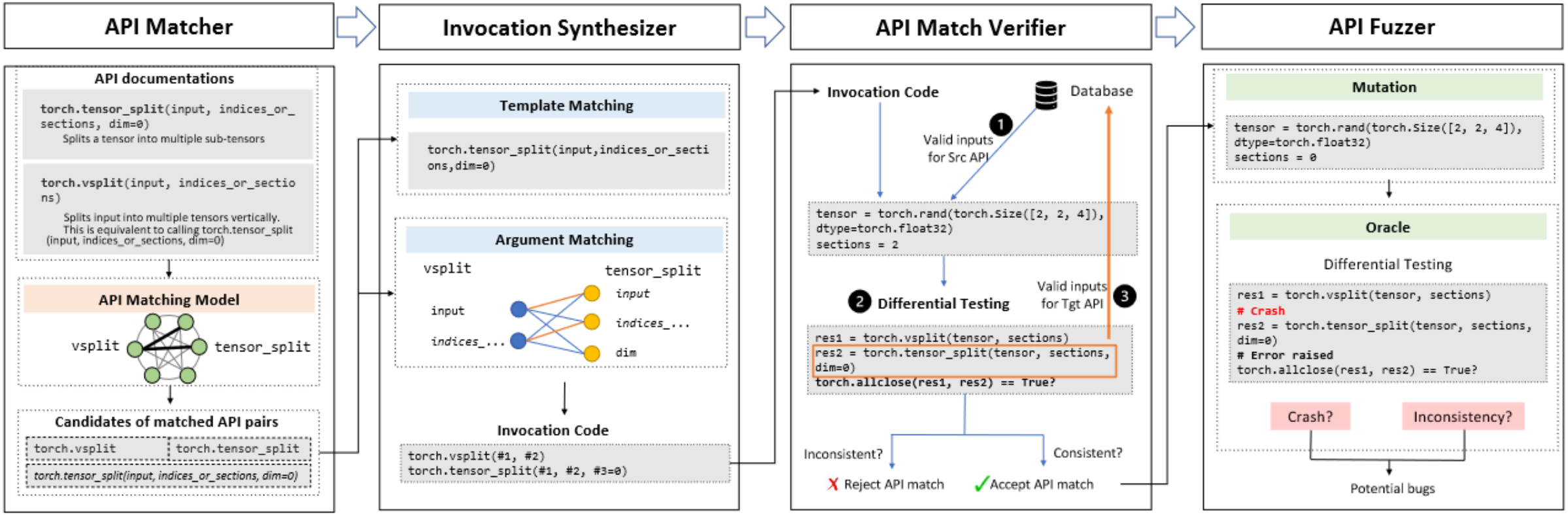
P	用于对深度学习软件库进行模糊测试的API需要手动标注，且API覆盖率低
C	深度学习库的API存在普遍的等价关系
D	如何自动地从API文档中获取准确的API关系
L	ESEC/FSE 2022.11 (CCF A)

算法原理——DeepREL

- 启发
  - 在传统软件测试中，发现多个等价API
  - 猜想：推断API之间的关系，或许有助于对深度学习库进行模糊测试
    - 值等效：相同输入->APIs->相同数值结果
    - 状态等效：相同输入->APIs->相同状态结果
- 例子：状态等效
  - $torch.nn.AdaptiveAvgPool3d$ 和 $torch.nn.AdaptiveMaxPool3d$
    - 参数：$(output\_size, return\_indices)$
- 解决局限
  - 借用已知API的测试输入信息，指导相似API生成输入
  - 相似API可直接用于差分测试

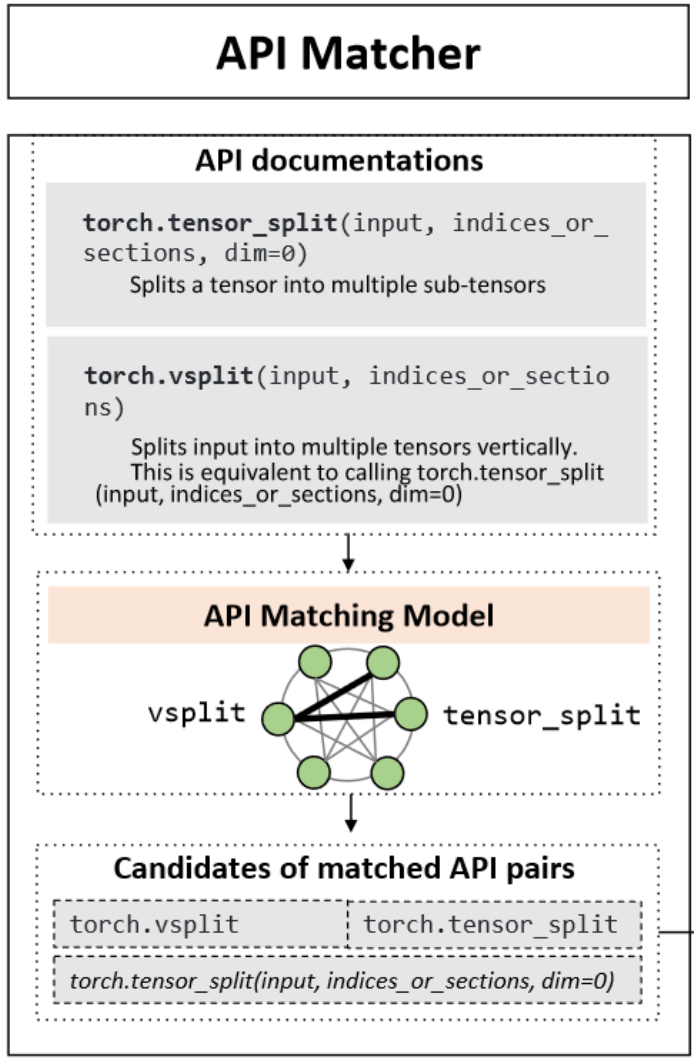
算法原理——DeepREL

- 算法流程图
  - API匹配、调用代码合成、API匹配验证、对API进行模糊测试
  - 迭代进行，直到覆盖API新增数目为0或达到迭代阈值（默认：10）



算法原理——DeepREL

- API匹配
  - 目的：
    - 根据API说明文档，构建候选API对
  - API对：（源API / S，目标API / T）
  - 步骤：
    - 从API文档收集签名和描述
      - 爬虫包：$Python\ package: bs4$
    - 计算不同API之间的相似度
      - 综合考虑签名相似度和描述相似度
    - 构建候选API对
      - 目标API：与源API距离最近的K个API
        - » 其中，K为超参数，设置为10



算法原理——DeepREL

- API匹配——计算不同API之间的相似度
  - 源和目标API之间相似度定义：

$$Sim_{API}(S,T) = Max(Sim_{sig}(S,T), Sim_{doc}(S,T))$$

  - 签名相似度：包含语法信息
    - 签名构成：API名称（参数名称）
      - API：$tf.math.maximum$、$tf.math.minimum$
      - 签名：$tf.math.maximum(x, y, name = None)$、$tf.math.\text{minimum}(x, y, name = None)$
    - 过程
      - 对签名进行分词
      - 以TF-IDF嵌入向量表示签名
      - 计算源和目的API之间的签名相似度

源API（S）的词嵌入

$$[c_1^S, c_2^S, \dots, c_n^S]$$

*：n为所有API签名分词后的总词数

$$Rep_{sig}(S) = [\frac{c_1^S}{\sum_{S' \in \mathcal{A}} c_1^{S'}}, \frac{c_2^S}{\sum_{S' \in \mathcal{A}} c_2^{S'}}, \dots, \frac{c_2^S}{\sum_{S' \in \mathcal{A}} c_2^{S'}}]$$

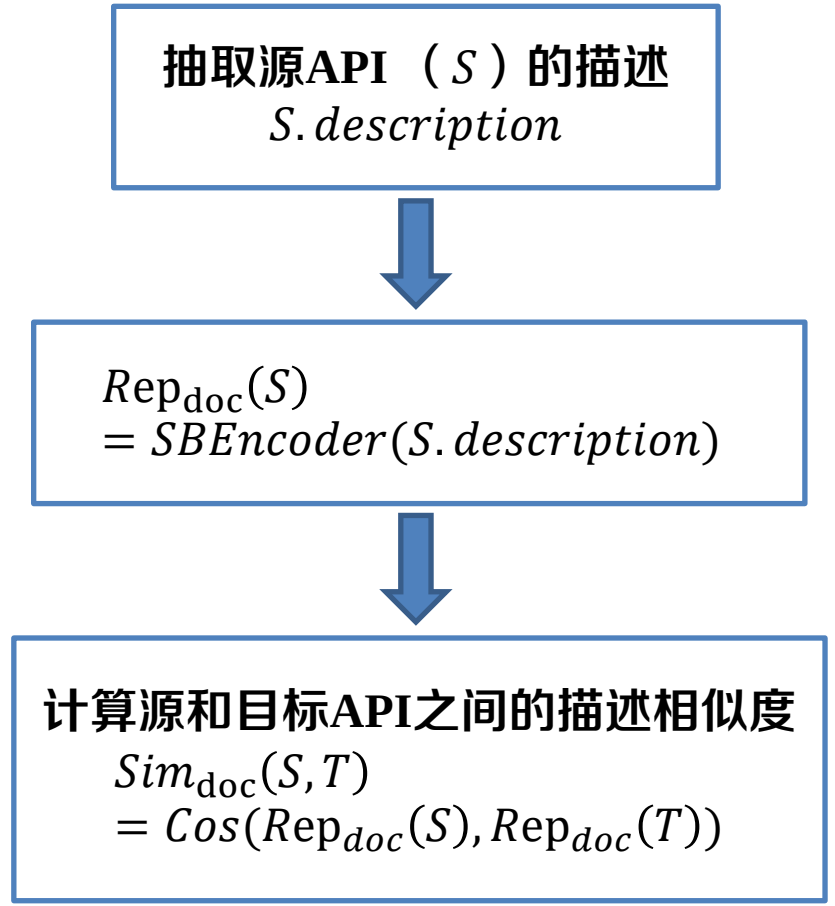
*：$\mathcal{A}$为所有API集合

计算源和目标API之间的签名相似度

$$Sim_{sig}(S,T) = Cos(Rep_{sig}(S), Rep_{sig}(T))$$

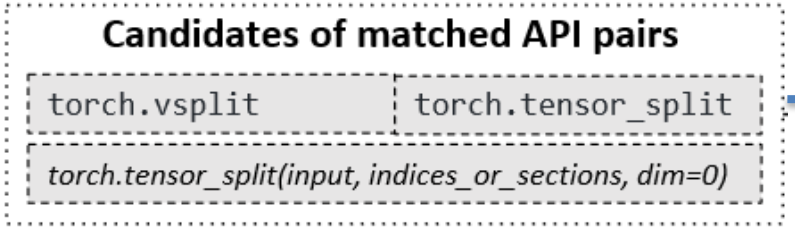
算法原理——DeepREL

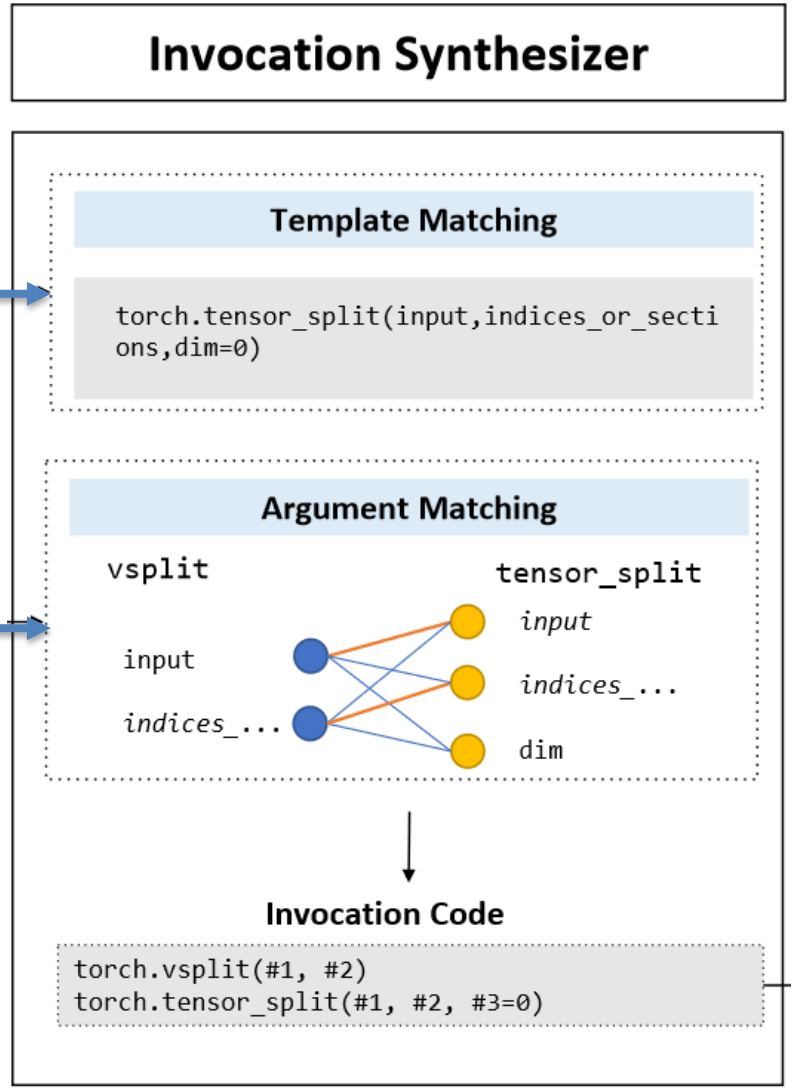
- API匹配——计算不同API之间的相似度
  - 描述相似度
    - 包含语义信息
    - 描述构成：描述API的第一句话
      - API：$\mathrm{torch}.vsplit$
      - 描述：
        - » $Splits\ input,$
        - » $a\ tensor\ with\ two\ or\ more\ dimensions, \ldots$
    - 过程
      - 抽取源API（S）的描述
      - 将描述编码为含有语义的句子嵌入
        - » $SBEncoder(Sentence - BERT\ Encoder)$
      - 计算源和目的API之间的描述相似度



算法原理——DeepREL

- 调用代码合成
  - 目的：
    - 为候选API对合成调用代码
    - 用于确定候选API对之间的关系
  - 步骤：
    - 输入候选API对
    - 进行参数匹配和模板匹配
    - 得到调用代码





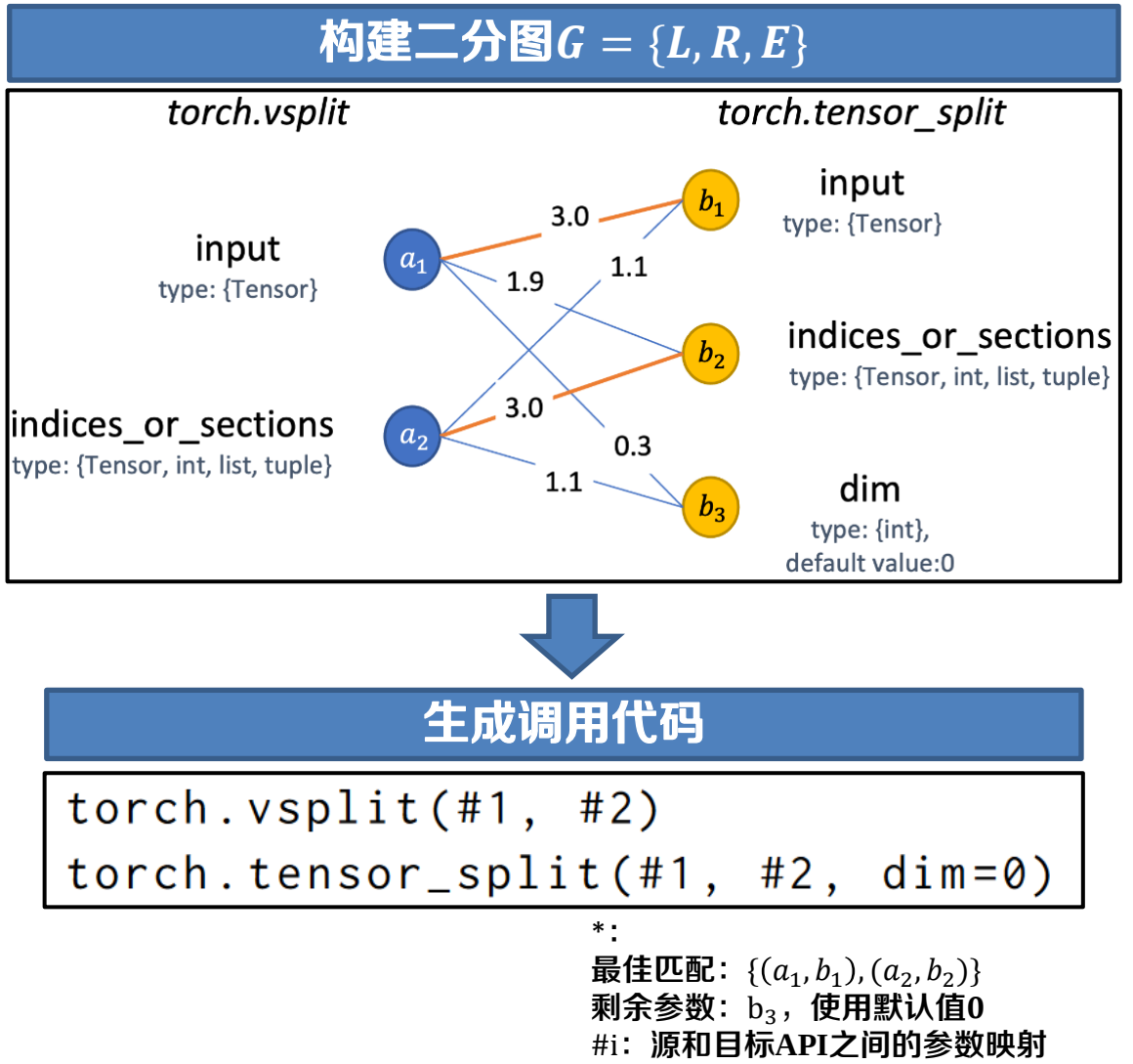
*：torch.vsplit：**源API**
torch.tensor_split：**目的API**
torch.$tensor\_split(input, indices_{or_{sections}}, dim = 0)$：**目的API的签名**

算法原理——DeepREL

- 调用代码合成：参数匹配
  - 步骤：
    - 给定源API（S）和目标API（T）
    - 及其对应参数列表
      - $S.args$、$T.args$
    - 构建二分图$G = \{L, R, E\}$
      - S的参数集合：$L = \{a | a \in S.args\}$
      - T的参数集合：$\mathrm{R} = \{b | b \in S.args\}$
      - 参数a、b的相似度：

$$\mathrm{E} = \{(a, b) | a \in L, b \in R\}$$

    - 得到最佳参数匹配
    - 生成调用代码

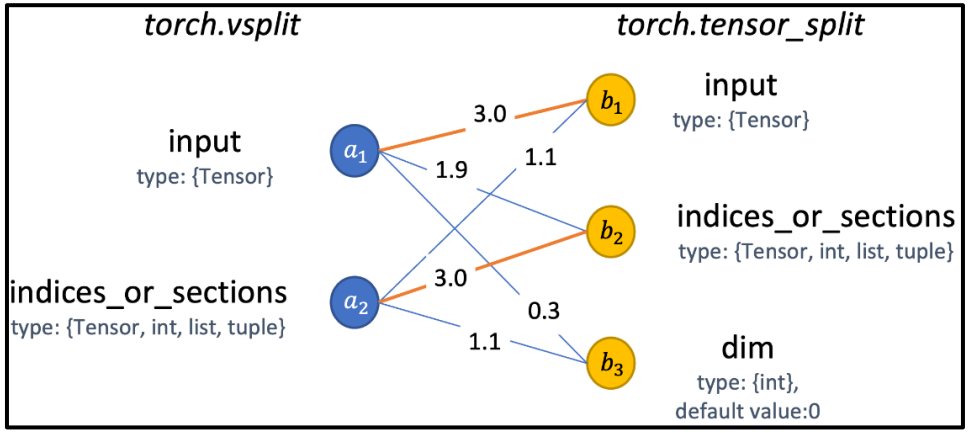


*：
最佳匹配：$\{(a_1, b_1), (a_2, b_2)\}$
剩余参数：b_3，使用默认值0
#i：源和目标API之间的参数映射

算法原理——DeepREL

- 调用代码合成：参数匹配
  - 计算参数a、b的相似度
    - $Sim_{arg}(a,b) = Sim_{name}(a,b) + Sim_{type}(a,b) + Sim_{pos}(a,b)$
    - 名称相似度
      - $Sim_{name}(a,b) = 1 - \frac{Levenshtein(a_{name},b_{name})}{Max(Len(a_{name}),Len(b_{name}))}$

*：$Lenvenshtein()$：计算两个字符串之间的编辑距离，即两个字符串之间，至少需要多少次操作，可以由一个字符串转换为另一个。



    - 类型相似度
      - $Sim_{type}(a,b) = \frac{|a_{type} \cap b_{type}|}{|a_{type}|}$
    - 位置相似度
      - $Sim_{pos}(a,b) = 1 - \frac{|a_{idx} - b_{idx}|}{Max(Len(S.args),Len(T.args))}$

算法原理——DeepREL

- 调用代码合成：模板匹配
  - 原因：
    - 对于一些复杂API对，无法使用参数匹配生成调用代码
  - 例子：

```
tf.scatter_nd(indices, updates, shape, name=None)
```

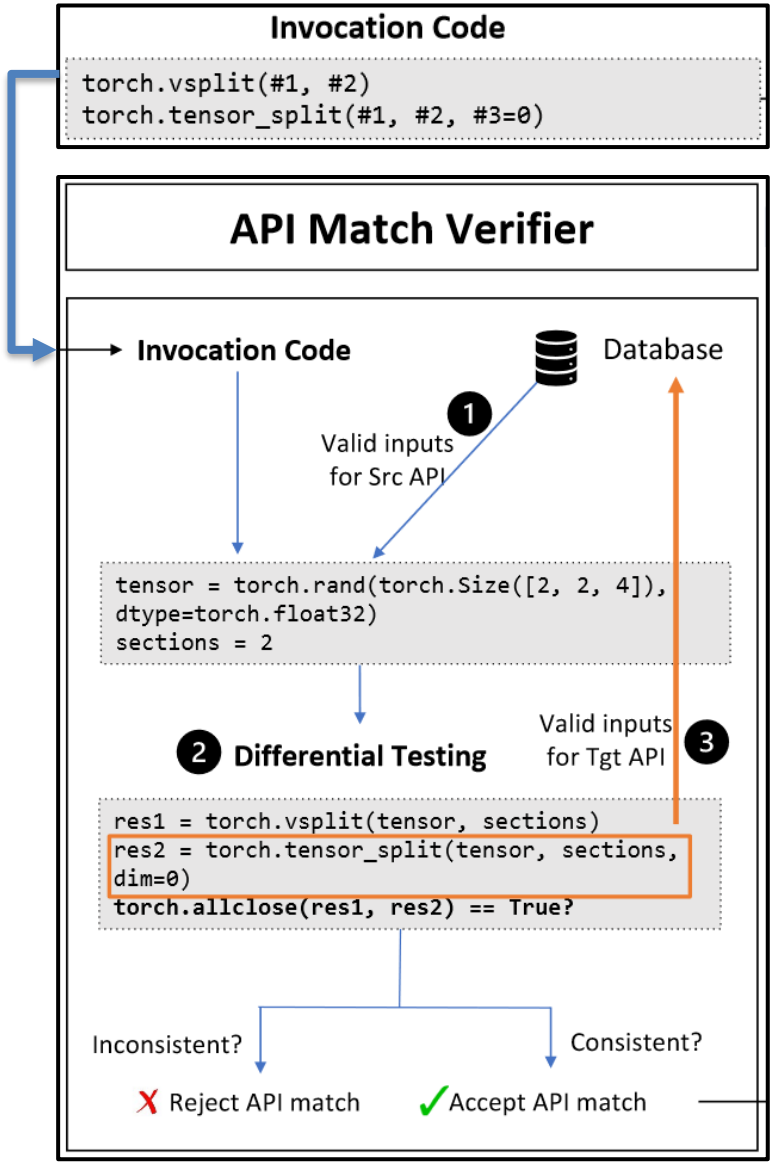
Calling tf.scatter_nd(indices, updates, shape) is identical to calling tf.tensor_scatter_nd_add(tf.zeros(shape, updates.dtype), indices, updates).

  - 做法：
    - 检查API文档中的代码块
    - 若某API描述中，包含对其他API的引用
    - 则定义为模板匹配，生成调用代码

```
tf.scatter_nd(#1, #2, #3)
tf.tensor_scatter_nd_add(tf.zeros(#3, #2.dtype), #1, #2)
```

算法原理——DeepREL

- API匹配验证
  - 目的：筛选出相似API
  - 步骤：
    - 根据调用代码和源API输入集合，为候选API对生成测试用例
      - 输入集合来源：文档、开发测试用例、深度学习模型
    - 差分测试
      - 值等价、状态等价
      - 不等价
    - 等价的候选API对记为相似API
      - 相似API的目标API在下一轮迭代中作为源API



算法原理——DeepREL

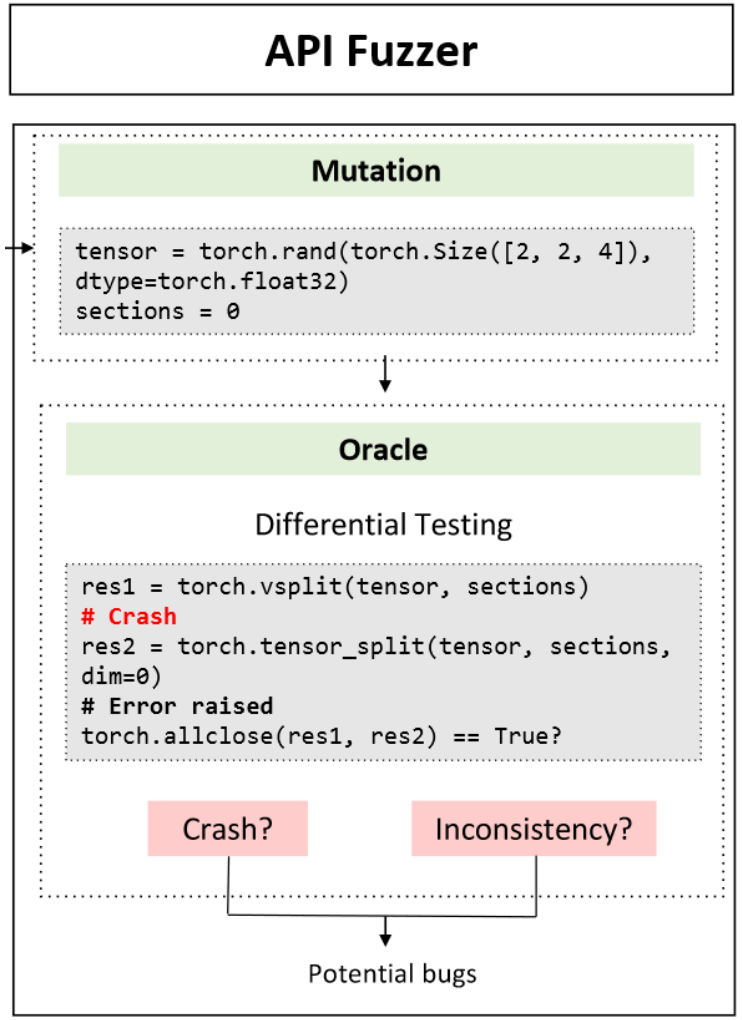
- **API匹配验证：值等价、状态等价**
  - 值等价定义
    - $S \equiv T(mod\ \mathcal{D}) \Leftrightarrow \forall x \in \mathcal{D}.S(x) = T(x)$
    - 输入集合：$\mathcal{D}$，源API：S，目标API：T，mod：取模
  - 状态等价
    - 定义
      - $S \sim T(mod\ \mathcal{D}) \Leftrightarrow \forall x \in \mathcal{D}.[\![S(x)]\!] = [\![T(x)]\!]$
      - $[\![\cdot]\!] \in \{Success, Exception, Crash\}$
    - 例子：都是进行池化操作

```
layer1 = torch.nn.AdaptiveAvgPool3d(output_size)
result1 = layer1(input)
layer2 = torch.nn.AdaptiveMaxPool3d(output_size)
result2 = layer2(input)
```

程序调用结果进行粗粒度划分：3类状态	
状态	注释
$Success$	程序正常运行
$Exception$	程序抛出已知异常
$Crash$	程序执行时遇到意料外的错误

算法原理——DeepREL

- 对API进行模糊测试
  - 方式：
    - 检测一致性缺陷
  - 步骤：
    - 使用FreeFuzz的变异模块，对源API的输入进行变异
    - 利用变异输入，对相似API进行差分测试
      - 值等效：希望执行后输出的值相等
      - 状态等效：希望执行后输出的状态相等
    - 在输出不一致处，检测bug



实验分析——DeepREL

- API覆盖率
  - 检测到的API
    - FreeFuzz覆盖较少原因
      - API分析的三个来源中缺乏使用频率低的API
  - 相似API对数目
    - 实验结论
      - API等效在深度学习库中普遍存在
      - PyTorch状态等价情况多
      - TensorFlow值等价情况多
        - » 包含大量用于兼容和低级访问操作的API

	#Total	#FreeFuzz	#DeepREL	Improvement(%)
PyTorch	1592	470	1071	601 (128%)
TensorFlow	6381	688	1902	1214 (176%)
Total	7973	1158	2973	1815 (157%)

*：API覆盖数目

	Equivalence$_{value}$	Equivalence$_{status}$	Total
PyTorch	1357	2933	4290
TensorFlow	5132	3676	8808
Total	6489	6609	13098

*：相似API对数目

实验分析——DeepREL

- 假阳率（False Positive Rate，FPR）
  - 指标：
    - ALL：检测到的不一致性总数
    - TP：True Positive
    - FP：False Positive
  - 实验结论
    - DeepREL假阳率为30.72%
    - 值等价假阳率小于状态等价假阳率
      - 值等价更严格
    - 假阳原因：API关系验证出错
      - 有效输入集未覆盖全部输入

	Oracle	# All	TP	FP	FPR
PyTorch	$\text{Equivalence}_{value}$	412	364	48	11.65%
	$\text{Equivalence}_{status}$	853	554	299	35.05%
	Overall	1265	918	347	27.43%
TensorFlow	$\text{Equivalence}_{value}$	97	68	29	29.90%
	$\text{Equivalence}_{status}$	363	209	154	42.42%
	Overall	460	277	183	39.78%
Total	$\text{Equivalence}_{value}$	509	432	77	15.13%
	$\text{Equivalence}_{status}$	1216	763	453	37.25%
	Overall	1725	1195	530	30.72%

实验分析——DeepREL

- 缺陷检测
  - 实验结论
    - 已确认此前未知的106个代码bug
      - FreeFuzz只能找到9个
      - CRADLE、LEMON、AUDEE都没有检测到
    - 检测出PyTorch中23个高优先级的bug
      - 2021.11.23-2022.3.1：23/170=13.5%
    - 文档bug：14个（PyTorch：10，TensorFlow：4）

	Total			Rejected			Confirmed			Fixed		
	Total	Value	Status	**Total**	Value	Status	**Total**	Value	Status	**Total**	Value	Status
PyTorch	**121**	76	45	**6**	3	3	**71**	33	38	**17**	9	8
TensorFlow	**41**	22	19	**1**	0	1	**35**	22	13	**12**	3	9
Total	**162**	98	64	**7**	3	4	**106**	55	51	**29**	12	17

Rejected：开发者拒绝的bug数目
Confirmed：确认为先前未知的bug数目
Fixed：已修复的先前未知的bug数目

总结

面向深度学习软件库的API层漏洞挖掘方法

- 深度学习软件库检测的流行趋势
  - API-Fuzz优于Model-Fuzz
    - 时间成本、运算成本、代码覆盖率、功能覆盖率…
  - 扩展获取API相关信息的渠道
    - 从API说明文档中提取信息
      - DocTer：输入的约束
      - DeepREL：API的关系

- 劣势
  - DocTer、DeepREL都只能检测含参的API是否存在缺陷
  - 是否有通用的方法，以适应深度学习软件库版本迭代的问题或适用于其它的库
  - API覆盖率仍有提升空间，但追求高API覆盖率，是否本末倒置
    - 大部分都是低危漏洞类型，较难发现高危漏洞

参考文献

[1] Xie D, Li Y, Kim M, et al. DocTer: documentation-guided fuzzing for testing deep learning API functions[C]. Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. 2022: 176-188.

[2] Deng Y, Yang C, Wei A, et al. Fuzzing deep-learning libraries via automated relational API inference[C]. Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2022: 44-56.

[3] https://avd.aliyun.com/detail?id=AVD-2020-15265

[4] https://github.com/tensorflow/tensorflow/commit/eccb7ec454e6617738554a255d77f08e60ee0808

[5] Reimers N, Gurevych I. Sentence-bert: Sentence embeddings using siamese bert-networks[J]. arXiv preprint arXiv:1908.10084, 2019.

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！