Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

源代码漏洞检测

硕士研究生 孔令迪

2023年5月14日

内容提要

- 背景简介
- 基础概念
  - 源代码安全
  - 漏洞检测基本概况
  - 理论及应用的发展方向
- 算法原理
  - FUNDED
  - LineVD
- 总结
- 参考文献

背景简介

- 预期收获
  - 1. 了解源代码安全的相关研究领域
  - 2. 理解源码漏洞检测的基本流程
  - 3. 理解源码漏洞检测的主流处理方法
  - 4. 了解源码漏洞检测的未来发展方向

背景简介 源码安全

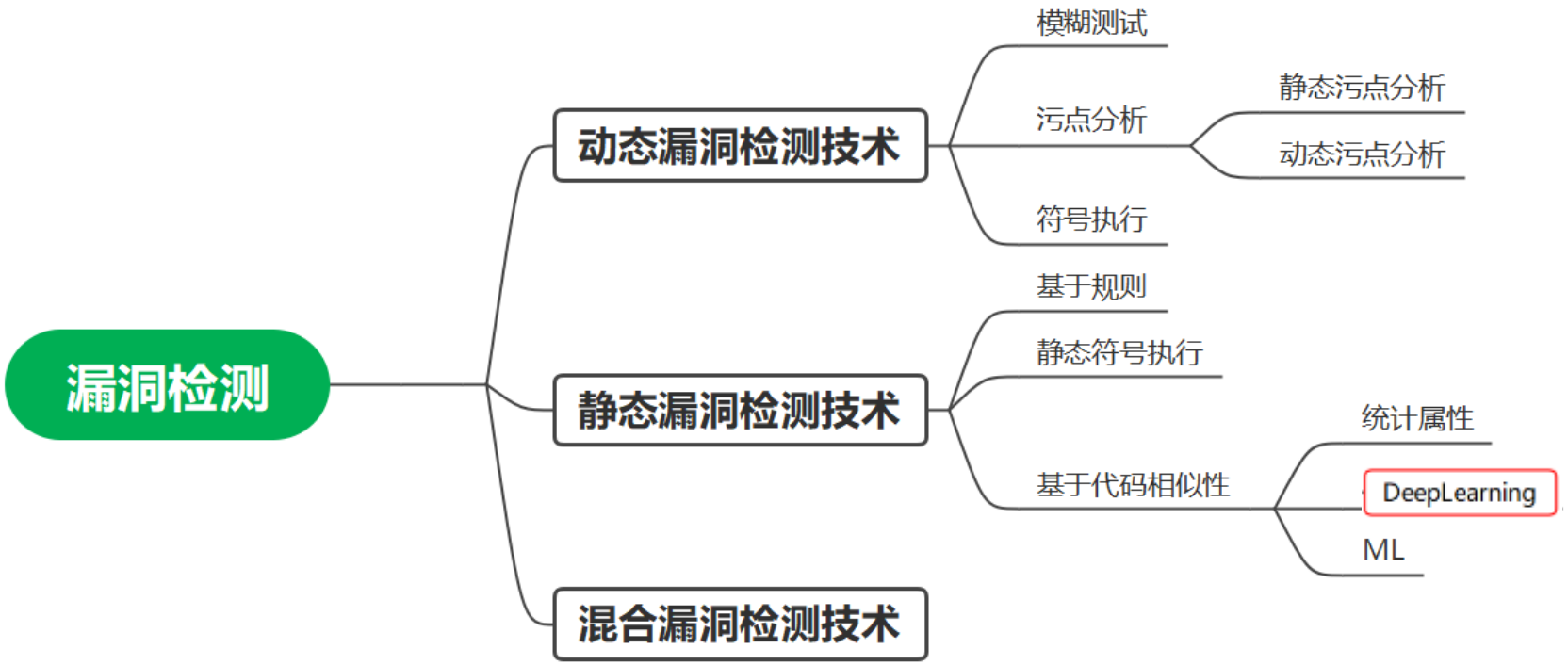
- 软件代码中安全漏洞和未声明功能的存在是信息安全事件频繁发生的根源
  - 对于软件安全保护很有必要
    - 在软件开发期就能及时识别漏洞，提示用户修改
  - 近年来相关研究非常热门
    - Dblp上检索vulnerability detection有1241篇文章，是源代码漏洞安全保护其他方向的3~4倍
    - 由于不需要运行程序，静态源码漏洞检测是近年来的研究热点
  - 有较大提升空间
    - 目前的研究与应用落差很大，实验中的F1指标在60%附近，实际应用更低，商业软件的F1在30%左右
    - 泛化性非常差，需要为每个项目单独训练模型
      - 源代码形式多样，漏洞结构复杂等原因

基本概念

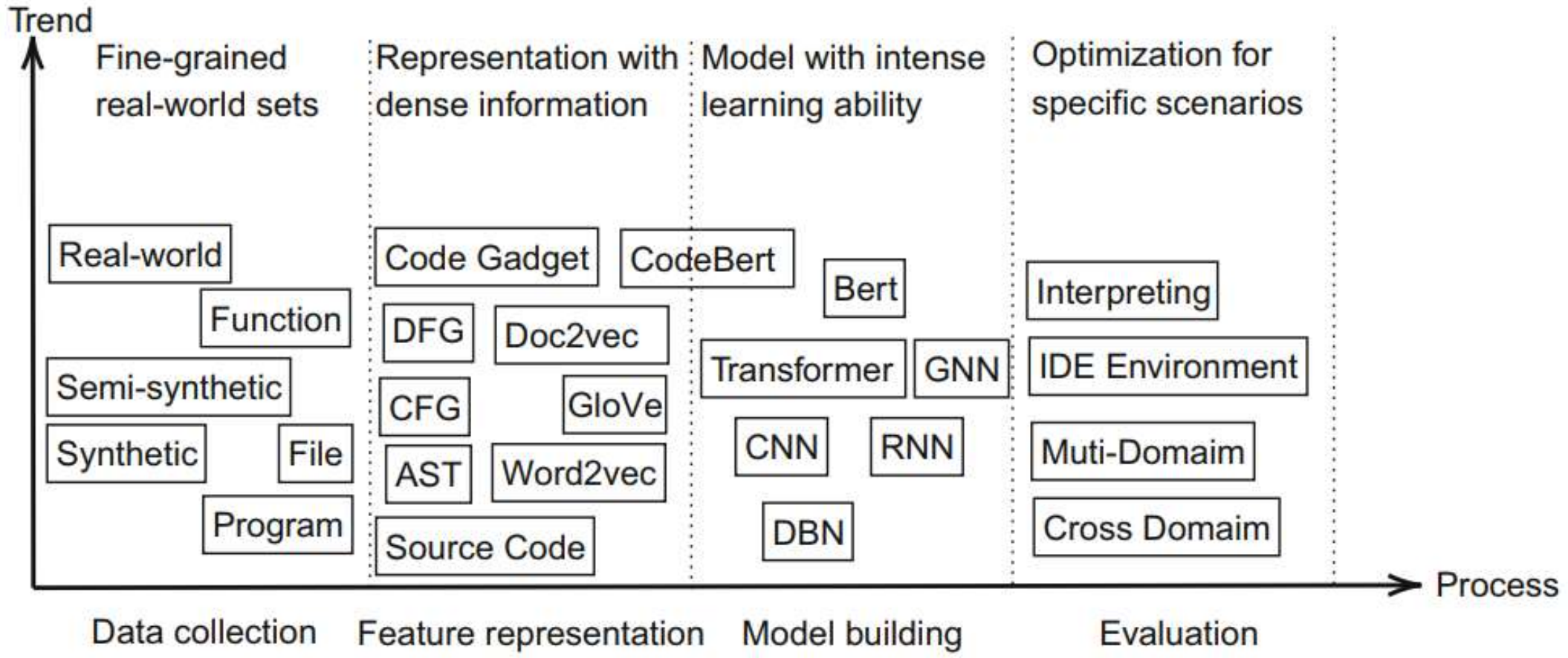
源代码安全 研究方向概述

- 研究方向

  - 漏洞检测
    - 静态漏洞检测
    - 动态漏洞检测
    - 混合漏洞检测
  - 漏洞评估
    - 可利用性评估
    - 危险性评分评级
  - 漏洞修复
  - 漏洞利用
    - 测试样例生成

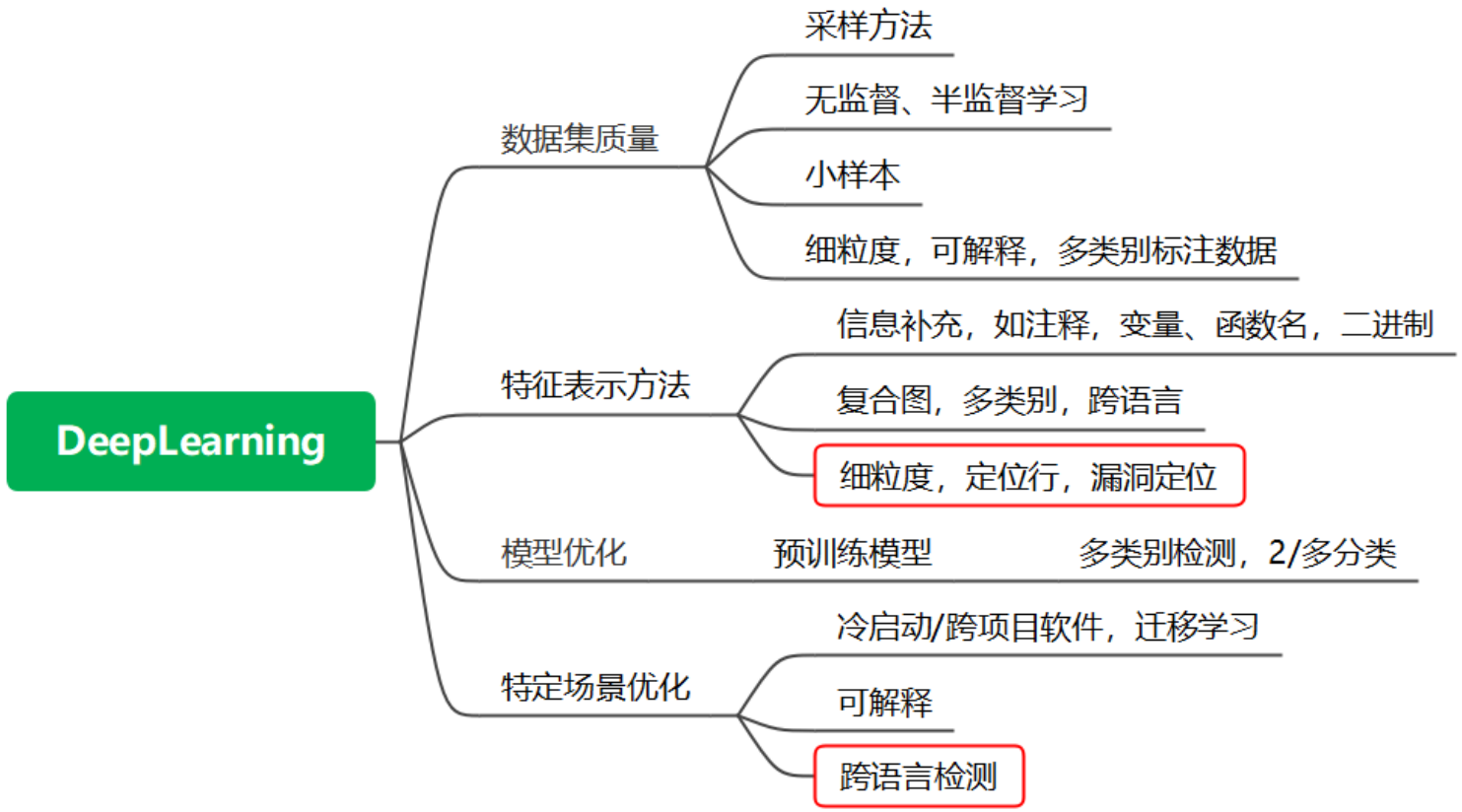


- 基本过程与处理方法


Trend
Fine-grained real-world sets
Representation with dense information
Model with intense learning ability
Optimization for specific scenarios
Real-world
Function
Semi-synthetic
Synthetic
File
Program
Code Gadget
CodeBert
DFG
Doc2vec
CFG
GloVe
AST
Word2vec
Source Code
Bert
Transformer
GNN
CNN
RNN
DBN
Interpreting
IDE Environment
Muti-Domaim
Cross Domaim
Process
Data collection
Feature representation
Model building
Evaluation

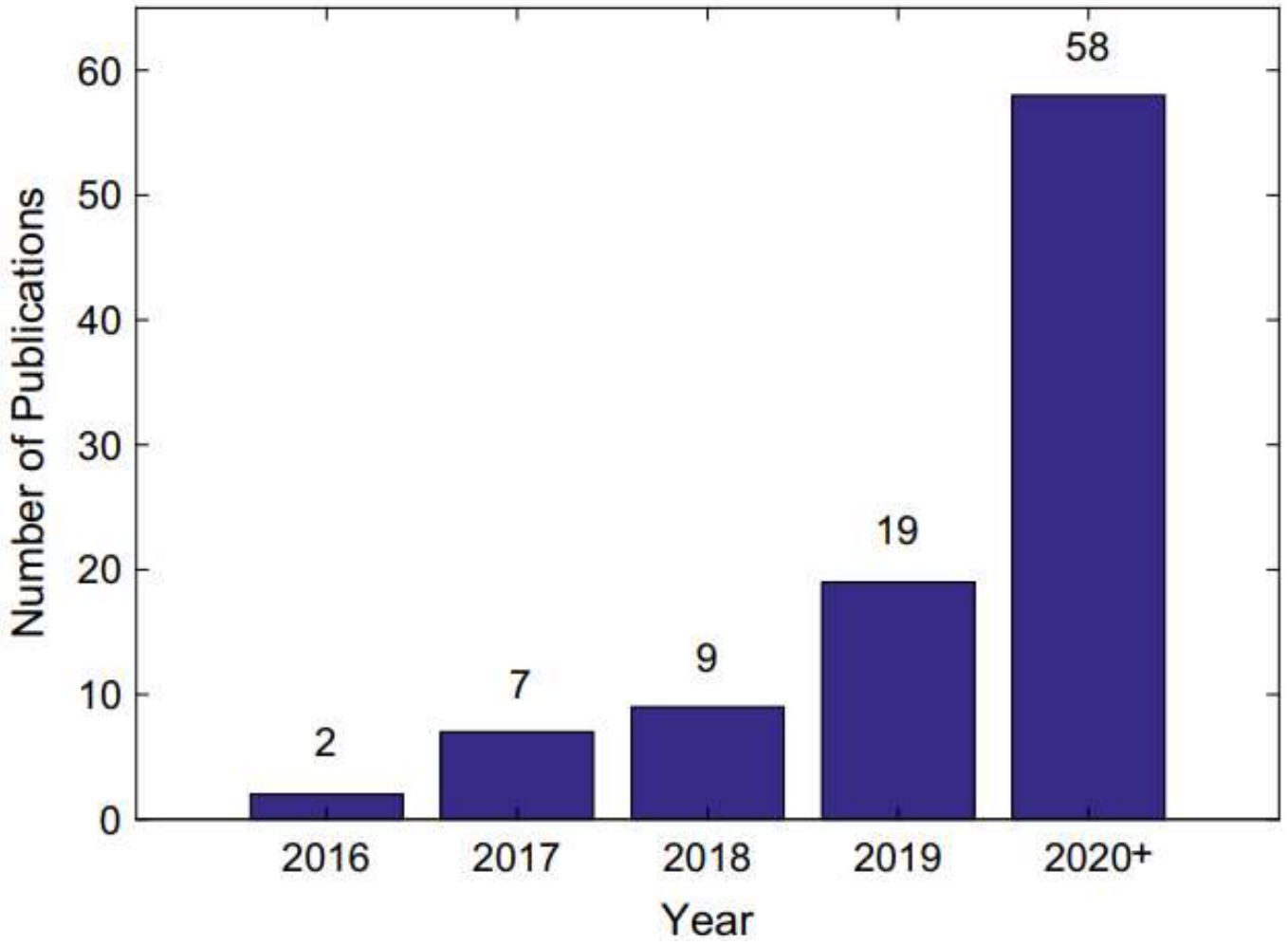
研究综述 技术与应用概况

- 从基本处理过程延伸出的研究方向
  - 数据集质量
    - 采样方法
    - 无监督，半监督学习
    - 小样本
    - 获取方法
  - 代码表征方法
    - 复合图
    - 信息补充，如注释等
  - 模型优化
    - 预训练模型
  - 特定场景优化
    - 跨项目、跨语言
    - 可解释



研究综述 技术与应用未来发展趋势

- 技术层面
  - 图网络模型
  - 预训练模型
- 应用层面
  - 检测范围
    - 多类型漏洞检测
    - 跨项目
    - 跨语言
  - 检测粒度
    - 切片级
    - 行级
  - 可解释



FUNDED
泛化性——跨语言漏洞检测

FUNDED TIPO

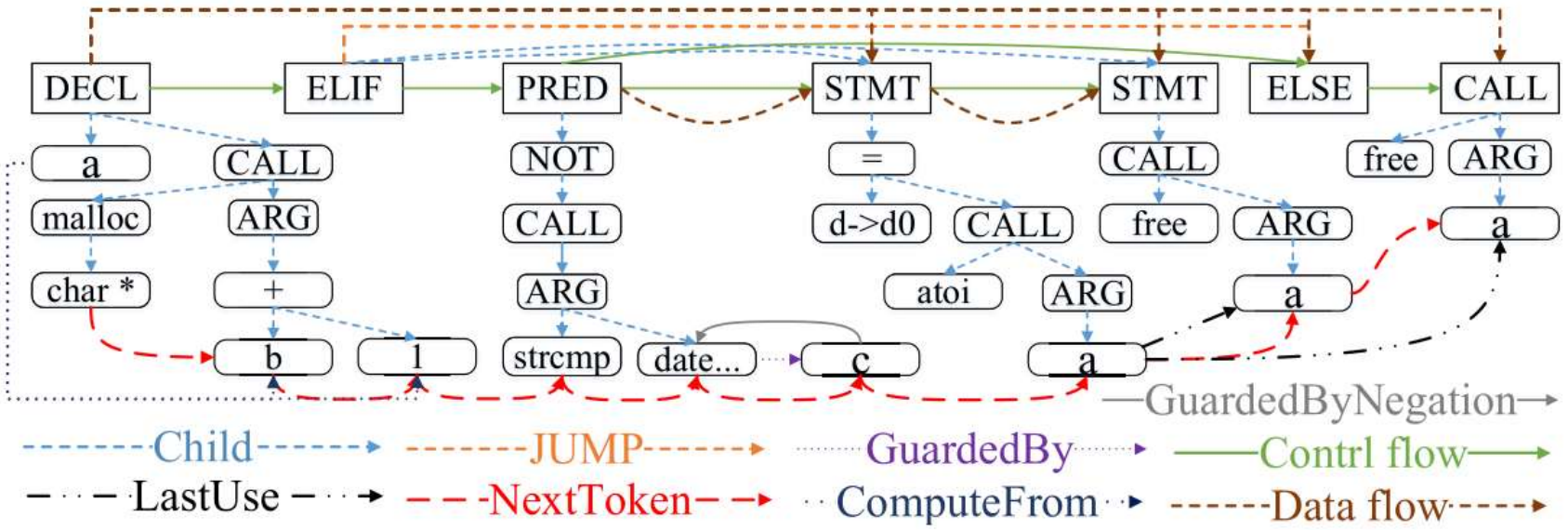
T	实现漏洞数据自动收集和函数级跨语言源码漏洞检测
I	多种语言程序源代码
P	1.组合AST和程序控制与依赖图PCDG构建程序图 2.多关系GGNN网络聚合传播信息 3.利用迁移学习适应不同语言漏洞检测 4.FC+Softmax多分类
O	是否包含漏洞、漏洞类型（多分类）

P	实现对多种代码语言漏洞的检测
C	函数级检测粒度
D	跨语言漏洞检测
L	2021 IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY

FUNDED 代码表示

- 在AST基础上扩展
  - 节点：语句、代码块或值
  - 边：8种关系，一对节点可能有多种关系
  - 原因
    - 标准的AST只有简单的父子从属关系，通过添加边可以得到额外的语法、数据和控制信息

```
1   a = (char*)malloc(b+1);
2   ...
3   else if(!strcmp(c,"dateadded")) {
4     d->d0 = atoi(a);
5     free(a);
6   } else {
7     free(a);
8   }
```

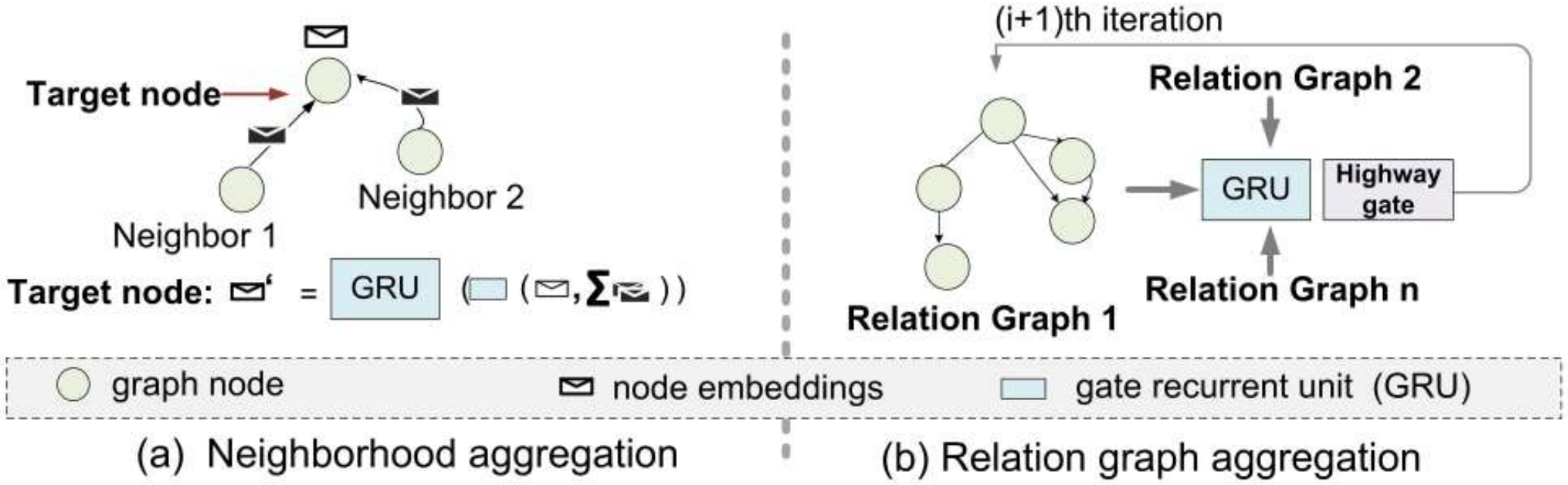


FUNDED 代码表示

- 生成程序图（以AST为基础添加7种边）
  - AST的边称为子边，表示父子关系
  - Data and control flows：PCDG中的数据流和控制流（PCDG=PDG+CDG）
  - GuardedBy：变量连接到所在域的封闭表达式 *确定错误操作数顺序*
  - Jump：连接具有控制依赖关系的变量 *资源锁定类漏洞*
  - ComputedFrom：变量连接到赋值语句 *变量或缓冲区的使用位置*
  - NextToken：代码序列
  - LastUse and LastLexicalUse： LastUse连接使用同一个变量的所有边，if语句中的变量只考虑最后一次关联边
- 节点类型表示
  - Word2vec分别对节点类型和节点内容做嵌入，两者拼接作为节点向量表示

FUNDED GGNN提取多关系图特征

- 特征提取模型GGNN
  - GGNN是基于GRU的GNN，可以获得图节点的更高阶邻域信息
  - 与标准GGNN不同，论文在多个关系图中传播和聚合信息



- 在不同关系图的对应节点间进行GRU聚合更新

FUNDED 代码切片code gadgets

- 整图信息聚合——图池化
  - 如何把有多个节点表示的图转换为单个向量的整图表示

$$h_G = \mathrm{CONCAT}\left(\sum_{i=1}^{m}\left(\left\{h_{v,i}^{(t)} | v \in G_i\right\}\right) | t = 0, 1, \ldots, n\right)$$

  - *t*表示节点聚合更新第*t*步；*m*表示关系图的个数；CONCAT表示向量拼接操作
  - 拼接“*m*个关系图的第*t*步节点嵌入表示求和”的结果，作为整图最终向量嵌入

- 模型输出
  - 全连接网络
  - Softmax多分类

FUNDED 跨语言漏洞检测

- 理论基础
  - 在具有相似输入的不同任务中训练的神经网络模型可以学习到共性特征
  - 神经网络的起始层抽象的输入属性大多与任务无关
  - 相比来说，网络的最后几层学到的信息更贴合特定任务

- 论文做法
  - 根据上述理论重复使用已训练的模型，加快新语言的学习
  - 图的结构也可以缓解语法之间的差异性（重用了图级知识）
  - 迁移学习
    - 使用与已训练好的基线模型相同的网络结构与初始权重
    - 对不同语言分别训练

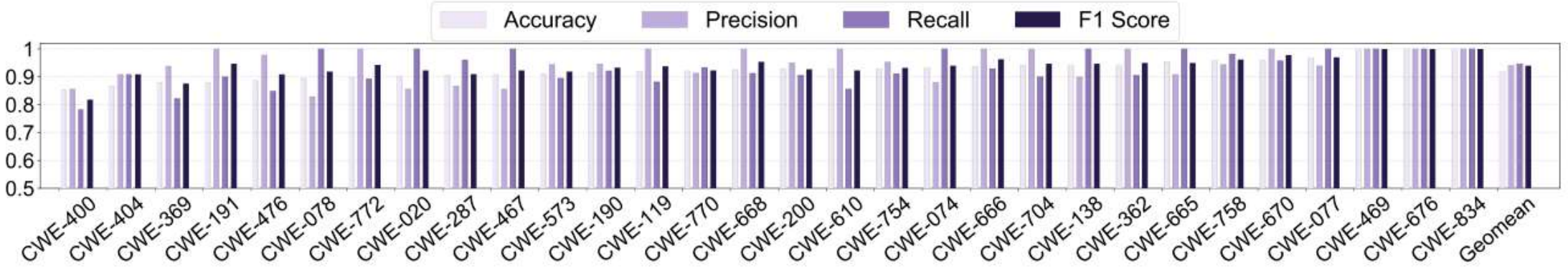
- 实验设计
  - 数据集

Source	Language	#vuln. types	#samples	#positive samples
SARD & NVD	C	30	90,954	45,477
	Java	14	29,512	14,756
	Php	5	17,578	8789
GitHub	C	10	10,400	5,200
	Swift	5	2,506	1253

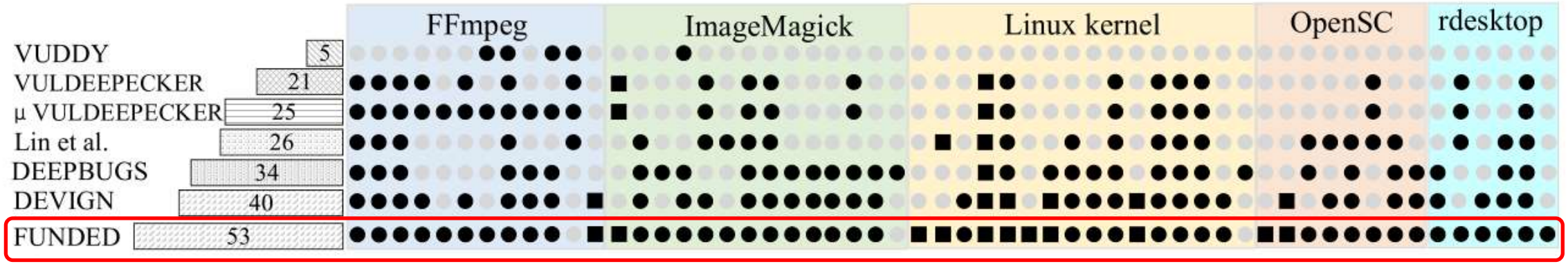
  - 对比方法
    - Vuldeepecker，μ Vuldeepecker，LIN，VUDDY，DEEPBUGS，devign
  - 评价指标
    - 五折交叉验证取几何平均值
    - Precision，Recall，Accuracy，F1 score

FUNDED 实验结果分析

- 多分类效果：C语言30个常见CWE的测试效果



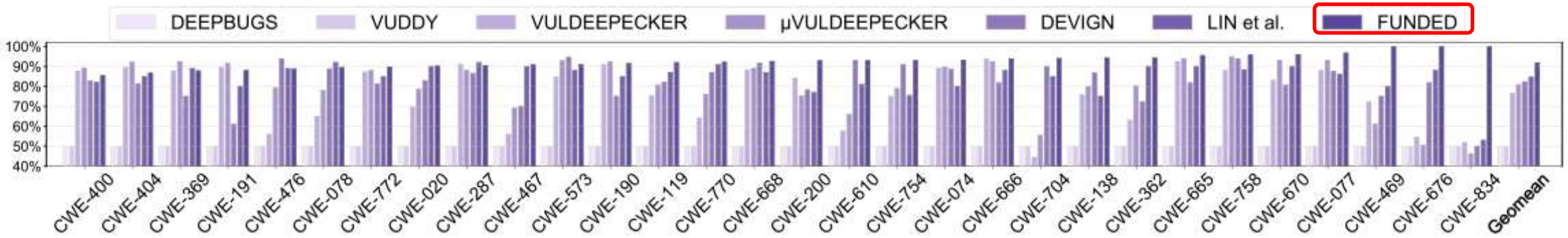
- 在真实数据中的检测效果（这些项目代码没有参与训练）



  - 注：仅含少数样本（5个项目总共56个漏洞）

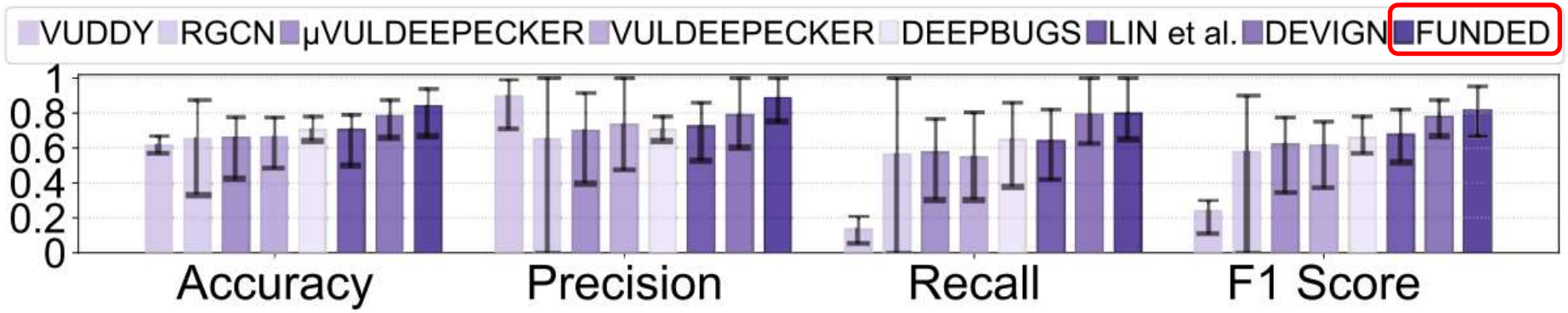
FUNDED 实验结果分析

- SARD和NVD的混合数据集



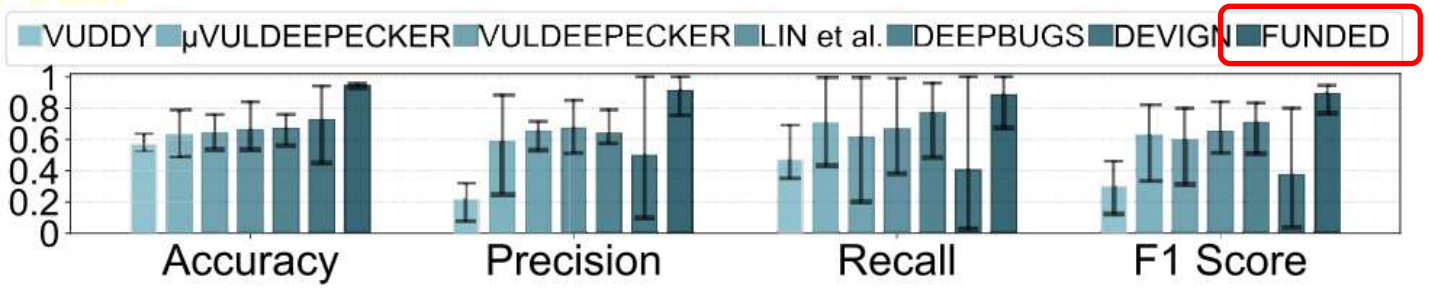
  - 平均准确率超过90%

- Github上收集的数据集

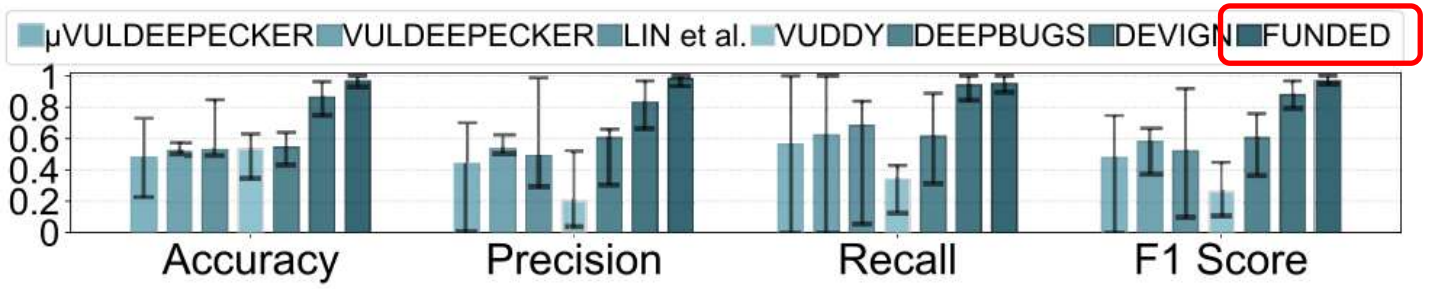


FUNDED 实验结果分析

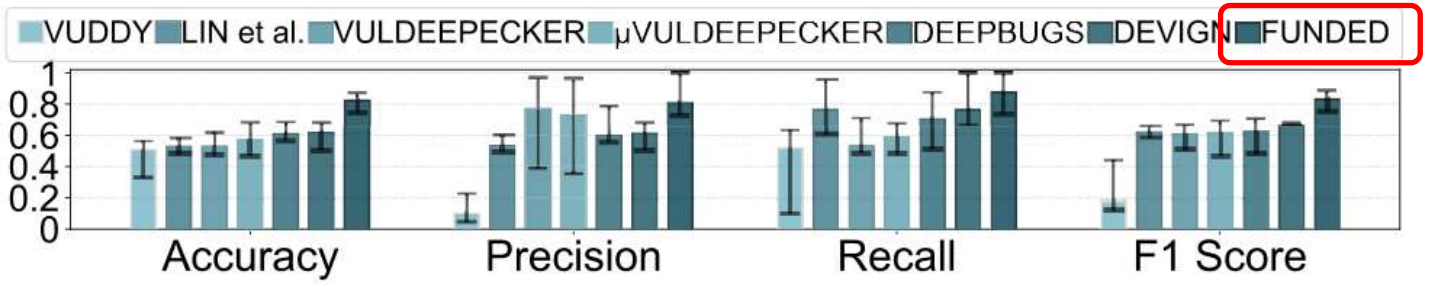
- 跨语言漏洞检测
  - C->Java
  - Java->Php
  - C->Swift
  - 注：文章里有不同语言的图生成工具
- 结论
  - FUNDED的迁移学习方法可以实现较好的跨语言漏洞检测



(a) C to Java



(b) Java to Php



(c) C to Swift

FUNDED 优劣分析

- 优势
  - 通过迁移学习和图结构削弱语言异构性实现跨语言漏洞检测
  - 多分类检测任务，可以识别漏洞类型
  - 提出了一种多关系图及聚合方法

- 劣势
  - 用于初始化图节点的word2vec有待改进
    - 难以区分重点语言关键词等重点token，降低跨语言检测性能
  - 迁移学习方法可以改进
  - 图池化方法只是简单的节点拼接，可以进一步完善
  - 尝试一下异质图嵌入？

LineVD
细粒度——定位漏洞语句

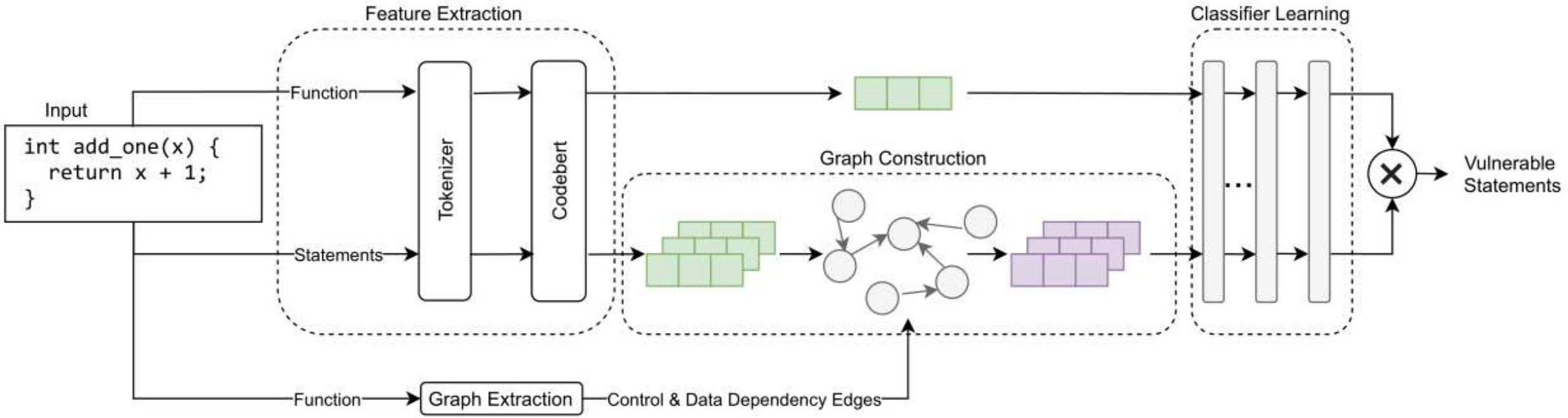
LineVD TIPO

T	实现代码行级的检测
I	函数代码
P	建模为图节点分类任务 1.特征提取：token嵌入、句子嵌入、函数嵌入 2.图构造 3.分类器学习
O	预测包含漏洞的代码行集合，按照置信度排序

P	函数级、切片级漏洞不易定位漏洞，可解释性差
C	C/C++代码
D	如何实现代码行级的漏洞定位
L	2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)

LineVD 算法原理

- 核心思想
  - 利用已识别的语句间数据、控制依赖作为上下文信息，预测代码行是否有漏洞
  - 视为节点分类任务

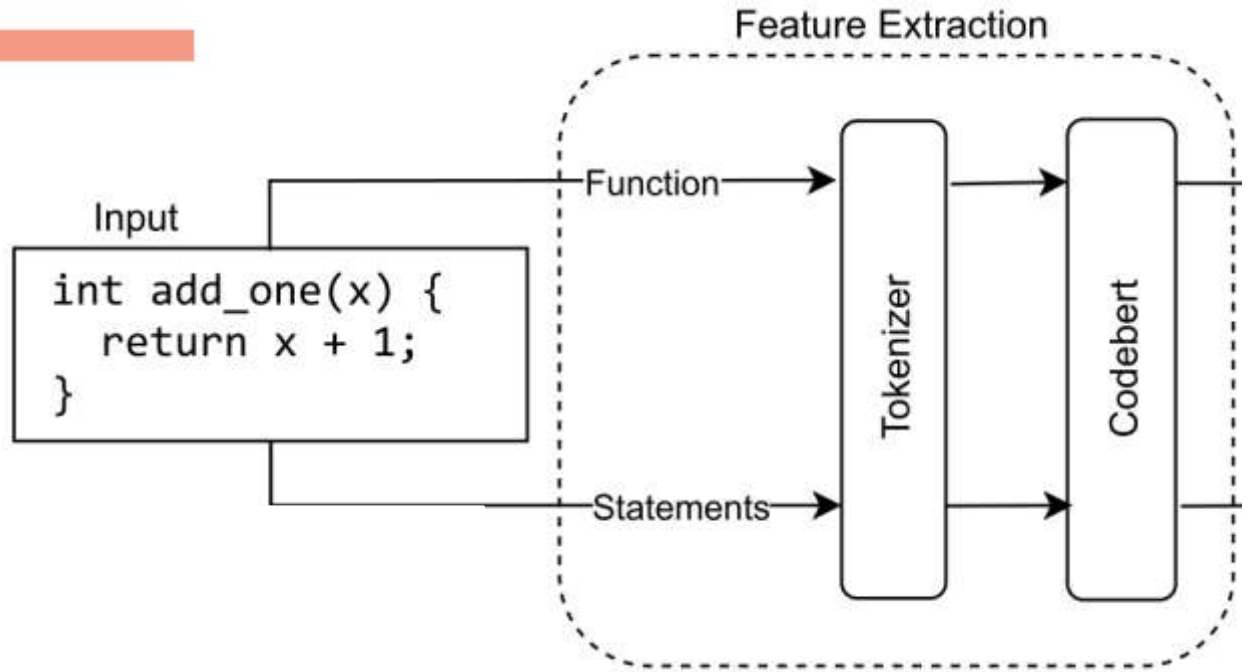


LineVD 模块分解

- 输出示例可视化

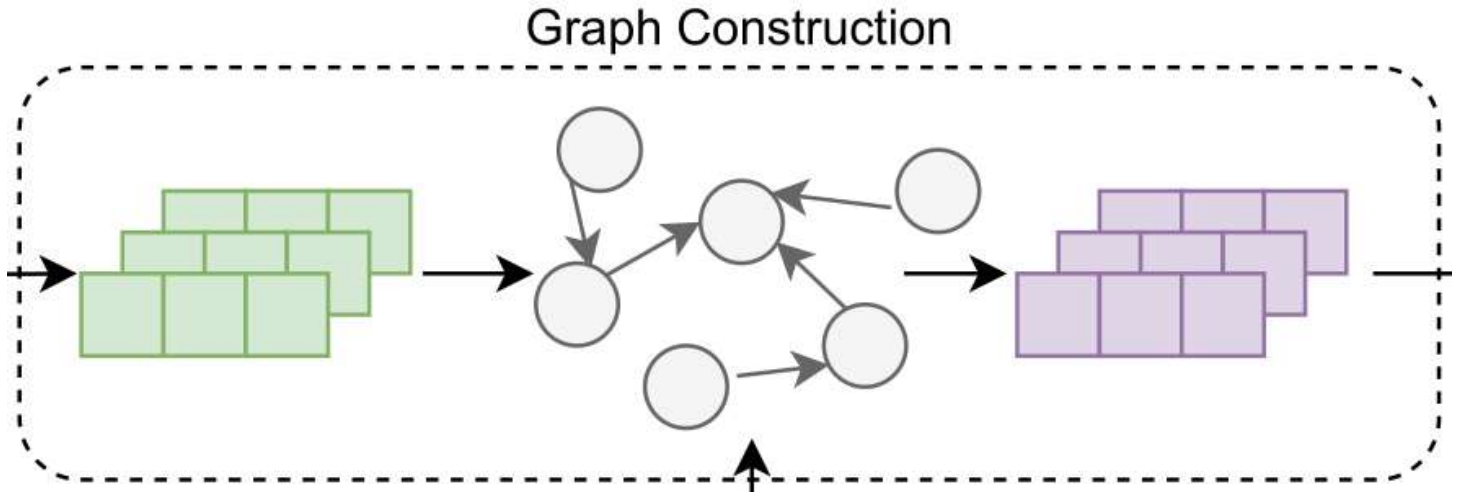
```
18    kc->clock_get(timr->it_clock, &ts64);
19    now = timespec64_to_ktime(ts64);
20
21    if (iv && (timr->it_requeue_pending & REQUEUE_PENDING || sig_none))
22      timr->it_overrun += (int)kc->timer_forward(timr, now);
23      // Added: timr->it_overrun += kc->timer_forward(timr, now);
24
25    remaining = kc->timer_remaining(timr, now);
```

- 特征提取（向量化的过程）
  - 拆分为单独的语句
  - 生成语句及函数的向量表示
    - 函数分词，生成token序列
    - 送进CodeBERT
    - 生成了1+n个嵌入
      - 函数整体嵌入
      - 每条语句的嵌入



LineVD 模块分解

- 图构造
  - 通过GAT学习图结构信息，包括数据依赖和控制依赖两个GAT（PDG+CDG）
  - 漏洞检测的GNN模型常用优化
    - 自循环、反向边
    - 使用Joern解析，将同行的节点分组到一起构成图节点



- 分类器学习
  - 从函数级和语句级代码中联合学习的模型
    - 近似认为两者的贡献相等
  - 结构：MLP+dropout+ReLU激活

- 实验设计
  - 数据集
    - 特点：真实项目非合成数据集、保持样本不平衡、数目够大、最好跨项目
    - 300多个不同的开源C/C++项目中提取代码漏洞
    - 含91种不同漏洞类型（含CVE编号），人工保证数据质量
    - 约10000个漏洞样本和17.7万个正常样本
  - 构造过程
    - Diff文件的删除行及其相关依赖语句视为漏洞语句
    - 人工清洗数据，保证数据质量
  - 评价指标
    - F1-score，Precision，Accuracy，Recall
    - ROCAUC，PRAUC（更适合不平衡数据集的模型效果评估）
    - 平均精度MAP，归一化累积增益nDCG，平均第一排名MFR
      - 第一排名：第一个正确预测的漏洞语句排名

LineVD 实验结果分析

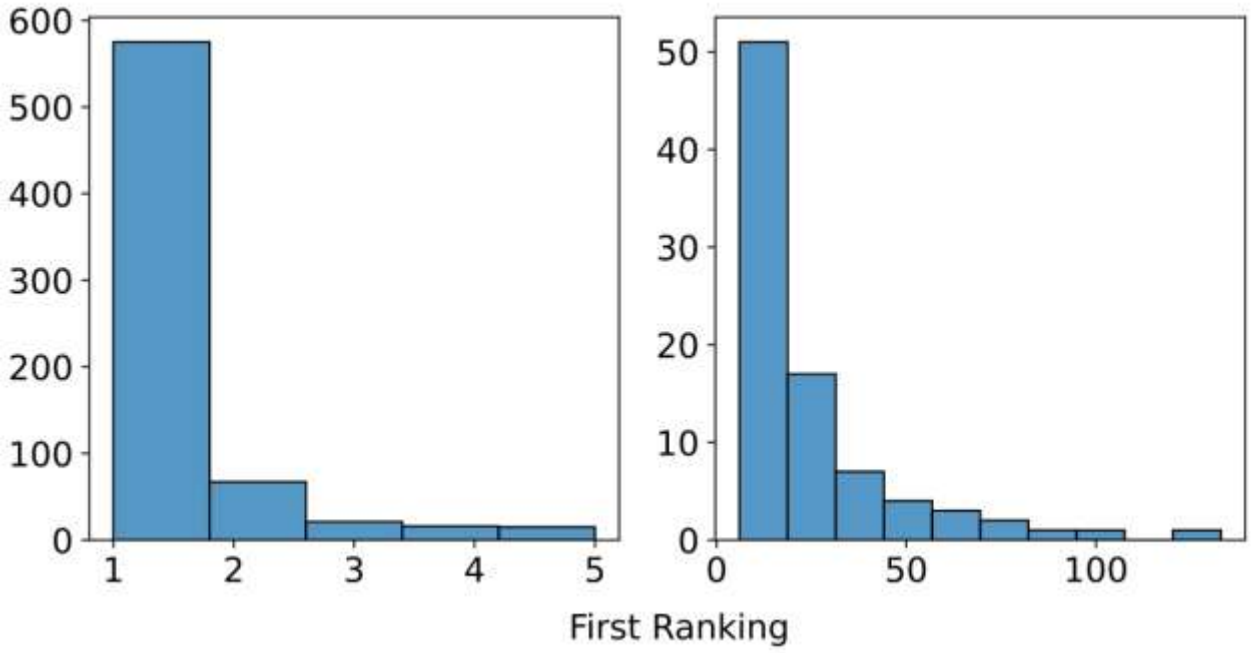
- 实验一：对比实验
  - 与可解释的SVD模型对比：IVDetect

Table 1: RQ1: Statement-level Performance (Ranked)

Methods	N5	MAP@5	NDCG@5	MFR
IVDetect	0.695	0.424	0.517	8.192
LineVD	**0.900**	**0.760**	**0.804**	**3.819**

Table 2: RQ1: Statement-level Performance (classification)

Methods	F1	Rec	Prec	ROCAUC	PRAUC
IVDetect	0.176	0.140	0.238	0.463	0.520
LineVD	**0.360**	**0.533**	**0.271**	**0.913**	**0.642**



  - 结论：各项指标显著优于对比模型；从排名分布上看，整体效果较好，存在一些较长序列的错误检测拉高了MFR

LineVD 实验结果分析

- 实验二：代码嵌入方法的影响

  - 对比方法
    - **CodeBERT**
    - **Doc2Vec**
    - **GloVe**

Embedding	F1	Rec	Prec	ROCAUC	PRAUC
Doc2Vec	0.064	0.167	0.040	0.580	0.508
GloVe	0.129	0.166	0.106	0.666	0.529
CodeBERT	**0.150**	**0.254**	**0.121**	**0.703**	**0.534**

  - 结论
    - **CodeBERT的效果显著**
    - **尽管CodeBERT在5种编程语言训练（没有C/C++），但是仍然能够表征C/C++代码。（普通BERT也可以，并且效果还可以）**

LineVD 实验结果分析

- 问题三：图神经网络和函数级信息的贡献

  - 对比消融模块
    - GCN与GAT
    - PDG和CDG
    - 是否包含函数级信息
  - 结论
    - GAT+PDG+Func是最佳组合
    - 不同的图结构类型、GNN类型影响相对不大
    - 函数代表的上下文依赖最重要

Model Type	F1	Rec	Prec	ROCAUC	PRAUC
GAT+CDG	0.115	0.120	0.112	0.657	0.528
GAT+CDG+Func	0.304	0.491	0.221	0.907	0.624
GAT+PDG	0.150	0.254	0.121	0.703	0.534
GAT+PDG+Func	**0.360**	0.533	**0.271**	0.913	**0.642**
GCN+CDG	0.084	0.143	0.060	0.632	0.514
GCN+CDG+Func	0.283	**0.558**	0.190	0.911	0.597
GCN+PDG	0.085	0.125	0.067	0.599	0.513
GCN+PDG+Func	0.310	0.460	0.235	0.905	0.616
No GNN	0.129	0.166	0.106	0.666	0.529
No GNN+Func	0.296	0.537	0.205	**0.921**	0.619

LineVD 实验结果分析

- 实验四：泛化性实验——跨项目分类场景中的表现
  - 实验设置
    - 测试数据所在项目不参与模型训练
  - 结论
    - 在不同项目上的检测性能差不多
    - 跨项目会降低检测性能
      - 不同项目之间存在差异，使用相同的检测模型效果必然有损失

Project	F1	Rec	Prec	ROCAUC	PRAUC	Vuln
Chromium	0.298	0.470	0.219	0.923	0.625	3103
Linux	0.301	0.502	0.216	0.925	0.630	1847
Android	0.290	0.494	0.208	0.922	0.625	962
ImageMagick	0.333	0.504	0.249	0.925	0.644	331
PHP	0.290	0.487	0.208	0.928	0.622	200
TCPDump	0.284	0.452	0.207	0.925	0.622	197
OpenSSL	0.298	0.508	0.211	0.926	0.628	157
Krb5	0.259	0.535	0.186	0.903	0.605	139
QEMU	0.250	0.478	0.177	0.910	0.601	120
FFmpeg	0.269	0.483	0.193	0.917	0.612	115

LineVD 实验结果分析

- 实验五：对于真实世界的数据，LineVD最能区分哪些语句类型？

  - 用于确定未来的优化方向
    - 哪种类型语句不好被检测出来
  - 使用Joern提供的节点类型
    - 将控制结构类型分成了if，while和for；
    - 函数调用分成内置和外部两种函数调用；
    - 运算符节点分成了赋值、算术、比较、访问和逻辑5种
  - 结论：函数声明、控制语句最容易被检测，因为其中包含的信息较多

Statement Type	TP	FP	TN	FN	F1
Function Declaration	530	364	17572	126	0.68
While Statement	48	72	1374	23	0.50
Builtin Function Call	142	246	4537	72	0.47
Logical Operation	90	159	4601	60	0.45
Switch Statement	18	31	1424	31	0.37
For Statement	75	195	3632	88	0.35
If Statement	749	1930	50969	855	0.35
Assignment Operation	1206	3792	81490	1051	0.33
Other Operation	62	200	5401	54	0.33
Jump Target	18	53	11119	19	0.33
Arithmetic Operation	27	106	1379	10	0.32
Return Statement	166	526	27332	186	0.32
External Function Call	644	2212	63378	582	0.32
Comparison Operation	19	64	1489	17	0.32
Access Operation	44	168	3488	48	0.29
Cast Operation	25	115	3146	28	0.26
Continue	2	13	1039	1	0.22
Break	5	42	7966	43	0.11
Goto Statement	3	15	5115	48	0.09

LineVD 优劣分析

- 优势
  - 使用节点分类的思路实现了代码行级漏洞检测
- 劣势
  - 构造的图相对简单直接，可以更丰富图结构
- 问题与改进
  - 考虑处理宏和函数间调用
    - 受到输入的限制：输入只有一个函数
  - C/C++预训练模型的开发

总结

总结

- 优势
  - 应用上：从泛化性和细粒度两个方面进行了研究
  - 技术上：在图结构构造和预训练模型两个主流方向上研究

- 劣势
  - 针对跨语言、跨项目等泛化性问题只是做了简单的迁移学习研究或实验验证
  - 构图过于复杂，缺乏有效的生成工具

- 发展前沿
  - 面向应用：
    - 大规模数据集构建困难：自监督、对比学习、小样本
    - 跨环境漏洞检测：域适应、迁移学习等

参考文献

[1] Wang, H., et al., Combining Graph-Based Learning With Automated Data Collection for Code Vulnerability Detection. IEEE Transactions on Information Forensics and Security, 2021. 16: p. 1943-1958.

[2] Hin D, Kan A, Chen H, et al. LineVD: statement-level vulnerability detection using graph neural networks[C]//Proceedings of the 19th International Conference on Mining Software Repositories. 2022: 596-607.

谢谢！

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