**软件安全**

* 论文动机：

深度学习对当前自动检测漏洞（DLVD）有很大帮助，但当面对现实中的数据不平衡问题，表现并不是很好。（深度学习模型训练有漏洞和无漏洞的代码）

数据不平衡-现实生活中有漏洞的代码和没有漏洞的代码的数量会相差很大，所以深度学习模型可能会过拟合。

RQ1：数据抽样是否提高了现有DLVD方法的有效性？

RQ2：数据采样是否提高了DLVD学习脆弱模式的能力？

* 论文方法：

选择使用数据采样：随机过采样、随机欠采样、SMOTE（升级版过采样）、OSS（升级版欠采样）

数据采样方法：

过采样：把数据集中少数通过复制或生成使跟多数数量相近。

欠采样：把多数删除部分跟少数数量相近。

SMOTE：对于选定的每个少数类样本，SMOTE随机选择其中一个最近邻样本。在原始样本和最近邻样本之间的特征空间内，SMOTE通过插值生成新的合成样本。具体来说，对于每个选定的特征，新样本的该特征值是原始样本和最近邻样本对应特征值的线性组合，组合系数是一个[0, 1]区间内的随机数。由于SMOTE生成的是合成样本而不是简单复制现有样本，它有助于减少过拟合的风险。

OSS：随机欠采样是一种简单直接的方法，但它可能会导致信息丢失，因为删除的样本可能包含对模型学习有用的信息。更高级的欠采样技术，如OSS，通过识别并删除对分类决策影响不大的样本来减少信息丢失的风险。

所用的四个DLVD：

Devign、Reveal、IVDetect、LineVul

**Devign：**

Devign是一种基于图神经网络(GNN)的漏洞检测方法，它利用代码属性图(Code Property Graph, CPG)来表示代码。

在CPG中，代码的每个部分（例如变量、函数调用等）都被视为图中的一个节点，而它们之间的关系（如数据流和控制流）则作为边。

Devign使用门控图神经网络(Gated Graph Neural Network, GGNN)来学习代码的表示，并通过这些表示进行漏洞检测。

**Reveal：**

Reveal同样是利用GNN进行漏洞检测的方法，它也采用CPG作为代码的输入表示。

Reveal使用GGNN来学习代码的表示，并且特别关注代码的时序特性，以更好地捕捉程序执行过程中的漏洞模式。

**IVDetect：**

IVDetect是一种专注于工业级Java代码的漏洞检测方法，它使用图卷积网络(Graph Convolutional Network, GCN)来处理代码的抽象语法树(AST)和程序依赖图(PDG)。

该方法通过FeatureAttention GCN模型(FA-GCN)来学习图上的节点表示，专注于函数级别的漏洞检测。

**LineVul：**

LineVul是一种基于Transformer模型的漏洞检测方法，它使用预训练的CodeBERT模型来嵌入整个函数的令牌序列。

LineVul可以进行函数级别和行级别的漏洞检测，它利用CodeBERT学习代码的上下文关系，并通过多层感知器(MLP)进行分类。

所用三个数据集：

**BigVul：**

BigVul是一个大型的漏洞数据集，包含来自多个开源C/C++项目的代码漏洞。

它覆盖了从2002年到2019年的常见弱点枚举(Common Weakness Enumeration, CWE)，包含超过10,000个有漏洞的方法和超过160,000个无漏洞的方法。

BigVul数据集的特点之一是它包含了漏洞修复信息，这使得研究人员能够分析漏洞修复前后的代码变化。

**Reveal：**

Reveal数据集包含大约12,000个方法，其中有9.16%是漏洞方法。

该数据集由先前的研究工作创建，并被用于评估深度学习在漏洞检测中的有效性。

Reveal数据集的漏洞与非漏洞方法的比例相对不平衡，这为研究数据抽样技术提供了一个具有挑战性的环境。

**Devign**：

Devign数据集包含来自FFmpeg和Qemu项目的方法，总共超过22,000个方法，其中大约45%是漏洞方法。

与Reveal数据集相比，Devign数据集在漏洞和非漏洞方法之间的比例更加平衡，这为评估数据抽样技术在不同数据分布上的效果提供了机会。

采取的两种策略：

在原始数据（原始代码）上进行采样

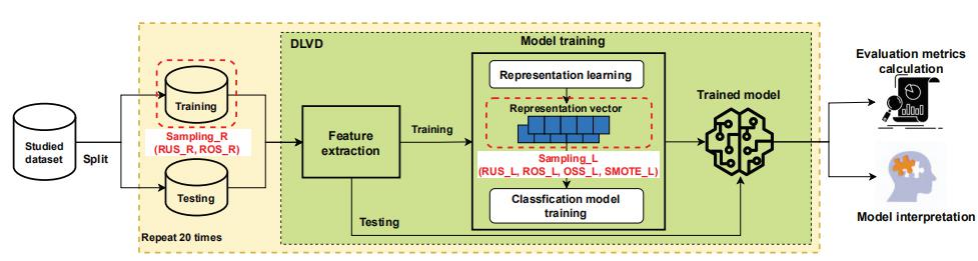
在投影空间（代码的表示向量）上进行采样

**原因：**一些先进的数据采样方法只能应用于已投影到潜在空间（特征空间）的数据点，因为它们需要在潜在空间中进行计算，比如SMOTE。而简单的数据，如随机欠/过采样，可以用于原始数据（没有任何投影）和潜在空间。一方面，潜在空间采样比对原始数据的采样更便宜，因为它节省了数据预处理的资源和时间(数据清理和特征提取)和训练模型。另一方面，投射到潜在空间会导致信息丢失。目前还不清楚哪个策略更好。

* DLVD方法

就是模型学习过程：

特征提取->模型训练->模型部署



两种分类：

基于标记的方法、基于图的方法

**基于标记(Token-based)的方法：**Devign、Reveal、IVDetect

代码表示：

在基于标记的方法中，源代码被视为一系列的标记(Token)，例如关键字、变量名、操作符等。

文本嵌入：

使用文本嵌入技术（如Word2Vec、GloVe或CodeBERT）将这些标记转换为数值向量，以捕获标记的语义信息。

序列建模：

将代码视为一个序列，并通过序列模型（如循环神经网络RNN、长短期记忆网络LSTM或Transformer）来处理这些嵌入的向量。

漏洞检测：

利用学习到的序列特征来训练一个分类器，以识别代码中是否存在漏洞。

优点：

易于实现，可以利用现有的自然语言处理技术。

能够较好地捕获代码的语义信息。

缺点：

可能无法充分捕获代码的结构信息，例如语法树的结构。

**基于图(Graph-based)的方法：**LineVul

代码结构：

基于图的方法考虑了代码的多种结构，如抽象语法树(AST)、程序依赖图(PDG)等。

图表示：

将代码的不同元素（如变量、函数调用、控制流语句等）表示为图中的节点，它们之间的关系表示为边。

图神经网络：

使用图神经网络(GNN)，特别是门控图神经网络(GGNN)，来学习图中节点和边的表示。

漏洞检测：

利用GNN学习到的图表示来训练一个分类器，以识别潜在的漏洞。

优点：

能够同时捕获代码的语义和结构信息。

适用于复杂的代码结构分析。

缺点：

实现相对复杂，需要对图论和GNN有深入的理解。

计算成本可能较高，尤其是在处理大型图时。

实验过程：

4种DLVD方法\*3种数据集\*三种采样方法（无采样，sample\_r（原始数据），sample\_l（嵌入空间））

在这篇论文中，用于评估深度学习漏洞检测（DLVD）模型性能的评价指标包括：

召回率 (Recall, R): 衡量模型识别所有漏洞代码的能力。召回率越高，表示模型漏掉的漏洞越少。

精确度 (Precision, P): 衡量模型预测为漏洞的代码中，实际为漏洞的比例。精确度越高，表示模型的假阳性（误报）越少。

F1 分数 (F1 Score, F1): 召回率和精确度的调和平均值，是一个综合考虑召回率和精确度的指标，用于评估模型的准确性和完整性的平衡。

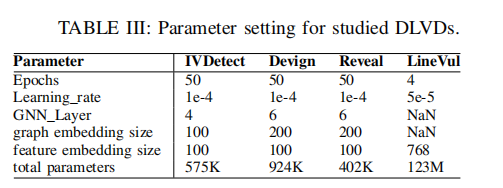
曲线下面积 (Area Under the Curve, AUC): 通常用于评估分类模型的二分类问题，表示模型对于随机选择的一个正样本和一个负样本，能够正确地将正样本排在负样本前面的概率。

Precision@k (P@k): 针对排名前k的样本，计算真正例的比例。这是一个考虑排名顺序的评价指标，常用于评估模型在返回较短列表时的性能。

RQ1 -有效性验证

RQ2-GNN解释器

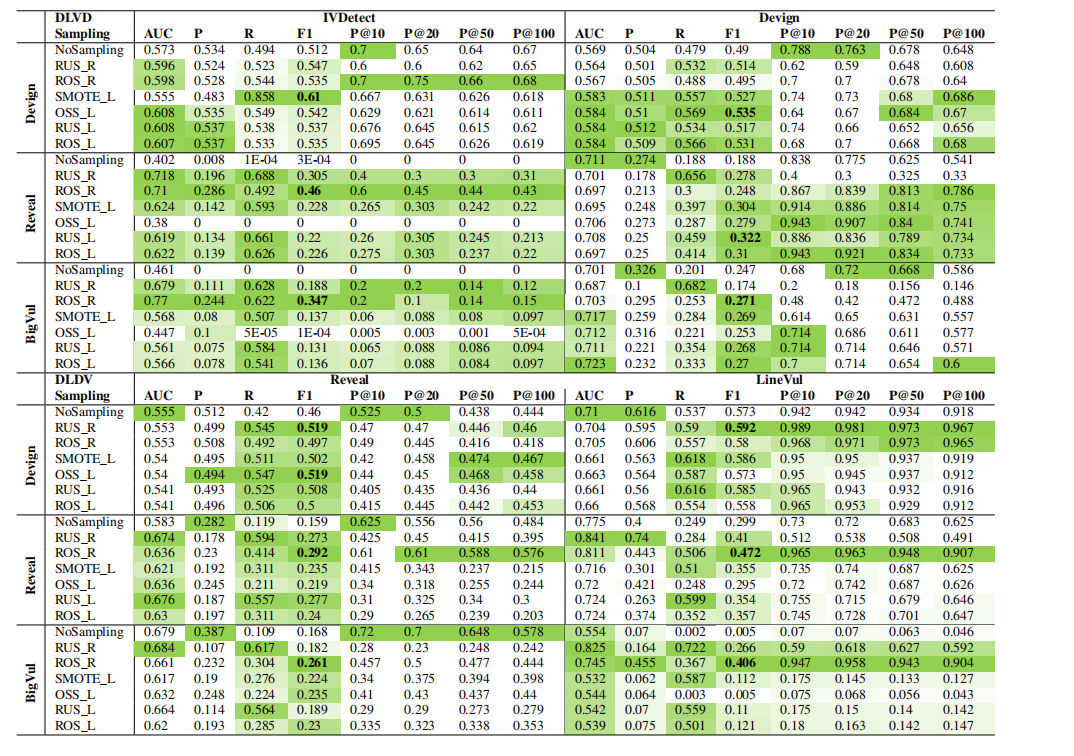
参数



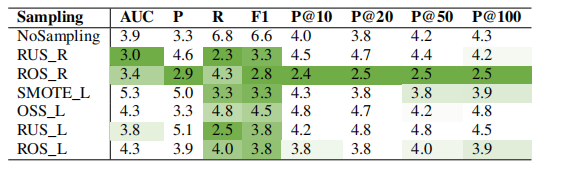
实验结果

RQ1结果

数据



这里面的数字是排名，并非数据



RQ2结果

