软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007433] [CSTR: 32375.14.jos.007433]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

VulFewShot: 利用对比学习改进少样本漏洞分类*

吴月明[1,2,4,7], 张笑睿[1,2,4,7], 李 志[1,2,4,5,7], 刘恺麟[3,4,7], 邹德清[1,2,4,5,7], 金 海[1,2,4,6,8]

[1](大数据技术与系统国家地方联合工程研究中心, 湖北 武汉 430074)
[2](服务计算技术与系统教育部重点实验室, 湖北 武汉 430074)
[3](分布式系统安全湖北省重点实验室, 湖北 武汉 430074)
[4](大数据安全湖北省工程研究中心, 湖北 武汉 430074)
[5](金银湖实验室, 湖北 武汉 430074)
[6](集群与网格计算湖北省重点实验室, 湖北 武汉 430074)
[7](华中科技大学 网络空间安全学院, 湖北 武汉 430074)
[8](华中科技大学 计算机科学与技术学院, 湖北 武汉 430074)
通信作者: 李志, E-mail: lizhi16@hust.edu.cn

摘 要: 为了对漏洞进行细粒度检测, 理想的模型必须确定软件是否包含漏洞, 并确定漏洞的类型 (即进行漏洞分类). 一系列深度学习模型在漏洞分类任务中取得了良好的整体性能. 然而, 观察到不同漏洞类型之间存在严重的数据不平衡. 许多漏洞类型只有少量的漏洞样本 (称为少样本类型), 这导致了对少样本类型的分类性能和泛化能力较差. 为了提高少样本漏洞类型的分类性能, 实现 VulFewShot. 这种基于对比学习的漏洞分类框架通过使相同类型的漏洞样本“接近”, 同时使不同类型的漏洞样品彼此“远离”, 从而为仅有少数漏洞样本类型赋予了更多的权重. 实验结果表明, VulFewShot 可以提高对所有类型漏洞的分类性能. 类型包含的漏洞样本数量越少, 改进就越显著. 因此, VulFewShot 可以提高样本不足的漏洞的分类性能, 并减少样本量对学习过程的影响.

关键词: 漏洞分类; 少样本; 对比学习
中图法分类号: TP311

中文引用格式: 吴月明, 张笑睿, 李志, 刘恺麟, 邹德清, 金海. VulFewShot: 利用对比学习改进少样本漏洞分类. 软件学报. http://www.jos.org.cn/1000-9825/7433.htm
英文引用格式: Wu YM, Zhang XR, Li Z, Liu KL, Zou DQ, Jin H. VulFewShot: Improving Few-shot Vulnerability Classification by Contrastive Learning. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7433.htm

VulFewShot: Improving Few-shot Vulnerability Classification by Contrastive Learning

WU Yue-Ming[1,2,4,7], ZHANG Xiao-Rui[1,2,4,7], LI Zhi[1,2,4,5,7], LIU Kai-Lin[3,4,7], ZOU De-Qing[1,2,4,5,7], JIN Hai[1,2,4,6,8]

[1](National Engineering Research Center for Big Data Technology and System, Wuhan 430074, China)
[2](Services Computing Technology and System Lab, Wuhan 430074, China)
[3](Hubei Key Laboratory of Distributed System Security, Wuhan 430074, China)
[4](Hubei Engineering Research Center on Big Data Security, Wuhan 430074, China)
[5](Jinyinhu Laboratory, Wuhan 430074, China)
[6](Cluster and Grid Computing Lab, Wuhan 430074, China)
[7](School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China)
[8](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

* 基金项目: 国家自然科学基金 (62202191)
收稿时间: 2024-05-30; 修改时间: 2024-10-10, 2025-01-21; 采用时间: 2025-03-25; jos 在线出版时间: 2025-07-23

Abstract: To perform fine-grained vulnerability detection, an ideal model must determine whether software contains vulnerabilities and identify the type of vulnerability (i.e., perform vulnerability classification). A series of deep learning models have demonstrated strong overall performance in vulnerability classification tasks. However, a severe data imbalance exists across different vulnerability types. Many vulnerability types are represented by only a small number of samples (referred to as few-shot types in this study), resulting in poor classification performance and generalization for these few-shot types. To enhance classification performance for these types, VulFewShot is proposed. This contrastive learning-based vulnerability classification framework assigns more weight to few-shot types by bringing samples of the same type closer together while keeping samples from different types further apart. Experimental results show that VulFewShot improves classification performance across all vulnerability types. The smaller the number of samples for a given type, the more significant the improvement. Therefore, VulFewShot improves classification performance for vulnerabilities with limited samples and mitigates the impact of sample size on the learning process.

Key words: vulnerability classification; few shot; contrastive learning

如今各种网络攻击层出不穷, 一系列事件表明漏洞是大多数的网络攻击主要原因[1,2]. 在实践中, 软件的设计缺陷使得漏洞在软件变得更加复杂的同时也更难以预防[3]. 因此, 以自动化、高效且准确的方式检测出软件漏洞是至关重要的. 目前已有多种方法被用于检测软件漏洞. 传统的检测方法基于人工定义的规则或特征对漏洞的描述[4−8], 这些方法对许多类型的漏洞都很有效但高度依赖人类专家的经验, 并不可避免地可能遇到大量的误报和漏报[9]. 为了减少专家工作量, 基于机器学习的检测方法被提出来自动学习和识别漏洞的特征[10,11]. 深度学习是相对于机器学习的突破性技术, 它在漏洞检测中得到了广泛应用, 这些应用包括利用卷积神经网络 (CNN)[12−15]、长短期记忆网络 (LSTM)[16−18]等模型来提取漏洞的特征.

目前基于深度学习的漏洞检测方法更多的是采用二分类的方法检测一段代码是否包含漏洞[14,16,17,19–22]. 但在实际操作中, 识别漏洞类型也是调查漏洞产生原因并修复漏洞的必要手段. 在这种情况下漏洞检测自然会利用多分类方法对漏洞进行分类. μVulDeePecker 是一种最先进的多分类系统, 它使用由 3 个 BiLSTM 网络块组成的神经网络对漏洞进行分类[23]. 根据他们论文的结果, 它在漏洞类型分类方面可以取得理想的性能.

但是根据我们的统计显示, 这些数据集中不同类型漏洞的数量是严重不平衡的. 在重新评估μVulDeePecker 之后我们发现随着漏洞样本数量的减少, 不同漏洞类型的分类性能会随之下降使得预防工作无效[24,25]. 在此结果上继续评估了 CNN、BiGRU、BiLSTM 和 Atten-BiLSTM 等多个其他深度学习模型在漏洞分类任务上的效果, 发现漏洞样本数量不平衡问题使得这些模型的分类性能依旧较差. 为了进一步研究性能较差的背后是否有其他的因素存在, 我们在 MVD 数据集的基础上构建了一个更平衡且更符合少样本特征的数据集 MVD-part. 实验表明在少样本数据集中分类性能较差的问题依然存在.

为了缓解分类性能较差的问题, 少样本漏洞类型中该问题尤为明显, 我们将对比学习引入到漏洞分类任务中. 具体来说, 利用有监督的对比学习设计了一个新颖的训练过程, 该过程使得同一类型的漏洞样本之间相互“接近”, 同时使得不同类型的漏洞样本彼此“远离”, 从而更好地从少样本中提取出漏洞特征. 基于此训练过程, 我们实现了基于对比学习的漏洞分类框架 VulFewShot, 并在 MVD[23]和 MVD-part 数据上对该框架进行评估. 实验结果表明, 使用对比学习可以提升不同样本规模的各类漏洞的分类性能, 该提升在少样本的漏洞分类上提升尤为明显. 因此 VulFewShot 在分类性能方面弥合了大规模和小规模样本数量漏洞类型之间的差距.

综上所述, 本文做出了以下贡献.

(1) 全面分析了漏洞数据集中样本不平衡问题以及该问题对漏洞分类性能, 特别是少样本学习场景下的影响.

(2) 用对比学习提高了不同样本规模的漏洞分类性能, 尤其是少样本漏洞类型的分类性能.

(3) 设计并实现了 VulFewShot (代码已发布在 https://github.com/VulFewShot/VulFewShot), 这是一个基于对比学习的漏洞分类框架. 同时对 4 种深度学习模型 (CNN、BiGRU、BiLSTM 和 Atten-BiLSTM) 进行了广泛实验, 证明了对比学习确实带来了改进.

本文第 1 节介绍相关工作. 第 2 节描述漏洞分类的过程. 第 3 节介绍传统深度学习模型的分类性能. 第 4 节介绍使用对比学习框架后模型的分类性能. 第 5 节讨论关键的潜在威胁. 第 6 节对本文总结并展望未来的工作.

1 相关工作

目前基于深度学习的漏洞分类研究主要有两类, 分别是: 二分类[14,16,17,19−22]和多分类[23,26−28]. 我们的研究集中在后者上.

深度学习模型以向量作为输入, 因此在使用这些模型时首先需要提取漏洞程序的特征向量表示. 提取特征向量的过程主要包括两个步骤: 第 1 步是对漏洞程序的数据进行静态处理 (例如删除注释、提取片段、归一化等), 第 2 步是生成代码嵌入 (即将第 1 步获得的数据转换为向量表示). 研究人员提出了不同的方法来处理第 1 步中的数据, 一种简单的方法是直接删除注释并规范化[22,29,30]. 也有方法专注于使用抽象语法树[31]或函数调用[32]来提取特定特征. 一些其他方法可能在代码分析工具如 Joern[33]的帮助下构建程序依赖图 (PDG)[34]或系统依赖图 (SDG)[35]. 在第 2 步中, 除了有使用包括 Word2Vec、GloVe 和 FastText[16,20,23]在内的一些主流嵌入方法, 也出现了一些新的具有不同特长方法. 例如基于图像的嵌入方法可以辅助对漏洞的视觉识别[22], 而基于 CodeBERT 的嵌入方法更适合下游任务[29]以及基于图的嵌入可以更好地描述代码连接[30,36]. μVulDeePecker[23]是一个使用最先进技术的多分类漏洞分类系统, 它为数据处理和开源数据集提供了一种经过验证的方法论. 因此我们遵循了 μVulDeePecker 的数据处理和向量表示方法, 并在其构建的数据集上进行了实验.

受不同漏洞类型之间严重的数据不平衡的启发, 我们引入对比学习来增强当前在少样本类型的漏洞上的分类性能. 自 Mikolov 等人[37]将对比学习引入自然语言处理 (NLP) 以来, 对比学习已逐步应用于图像、语音和视频等其他领域[38−50]. 例如, SimCLR[50]是将对比学习应用于图像分类任务的典型工作. 在之前的大多数研究中, 对比学习主要用于自监督学习. Khosla 等人[51]将对比学习方法扩展到监督学习, 他们通过对比训练数据的标签信息来提高分类性能. Wu 等人[52]将对比学习引入到 Android 恶意软件分类中, 这有助于减少混淆性差异, 同时增大了恶意软件和良性应用程序之间的区别. Gunel 等人[53]将监督对比学习应用于文本分类任务, 在交叉熵损失的基础上增加了一个额外的监督对比学习损失函数 (SCL). 新设计的损失函数使同一类样本更靠近, 同时使得不同类别的样本之间距离更远. 他们还通过实验证实了对比学习可以带来少样本学习场景的巨大改进, 这给了我们一些启示. 我们的实验结果表明, 对比学习方法可以提高整体性能和在不同规模的漏洞类型上的表现, 尤其是在少样本类型上.

2 漏洞分类

图 1 展示了漏洞分类的一般过程, 该过程一般包含 3 个步骤: 数据预处理、向量提取和模型训练/预测. 在数据预处理中, 漏洞样本会被提取为代码片段并根据预先定义的代码标签进一步提取为标准化函数片段. 在向量提取中, 标准化的函数片段会被转化为向量表示. 最后在模型训练/预测阶段, 一部分漏洞样本对应的特征向量将被用作输入训练用于分类任务的深度学习模型, 深度学习模型使用训练阶段未使用的漏洞样本进行测试, 并根据第 2.3 节中的评估指标进行评估.

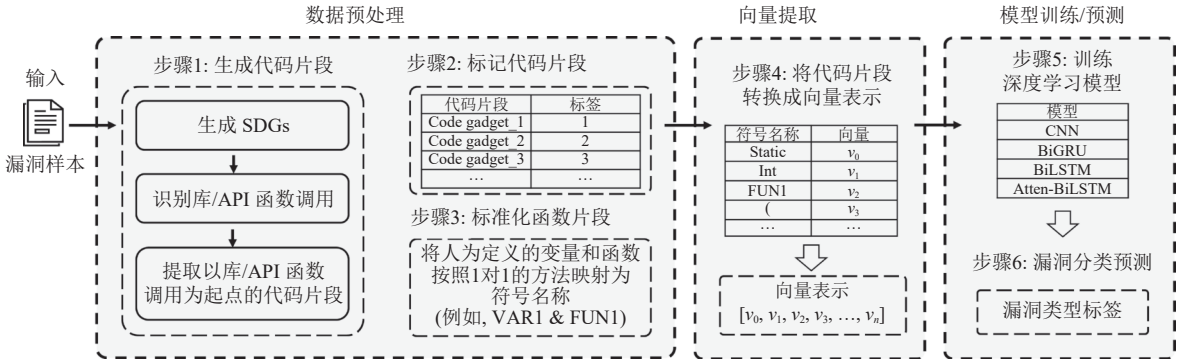

图 1 漏洞分类的一般过程

2.1 深度学习的模型选择

为了完成漏洞检测或分类任务, 各种深度学习架构被用于在漏洞特征的不同表示中进行学习. 我们主要考虑使用卷积神经网络 (CNN)[12−15,22]和循环神经网络 (RNN), 包括长短期记忆 (LSTM) 及其一些变体[16,20,23,54,55]. 现有相关工作绝大多数是二分类工具, 多分类工具很少. 在二分类工具当中, 绝大多数模型都采用了 CNN (如 VulCNN[22])、BiLSTM (如 VulDeePecker[16]) 或 BiGRU (如 SySeVR[20]); 而多分类工具同样如此, 如经典的多分类工具μVulDeePecker 就采用了 BiLSTM 模型[23]. 因此, 我们选择 CNN、双向长短期记忆 (BiLSTM)[16,23]、双向门循环单元 (BiGRU[20]) 和基于注意力的双向长短期记忆 (Atten-BiLSTM[56]) 作为实验模型.

CNN 是一种具有人工神经元的前馈神经网络, 包括卷积层、全连接层、关联权值层和池化层, 这使得 CNN 可以利用输入数据的二维结构. CNN 在计算机视觉领域取得了很好的效果, 也可以应用于文本分类. 在这个实验中, CNN 将 Word2Vec 标记的代码片段作为输入, 然后将标记向量在一个层中进行卷积. 在另一层进行最大池化后, 最终输出会连接到 Softmax 函数来完成分类任务. VulCNN 采用这种模型用于二分类漏洞检测[22].

RNN 模型由于其自身的结构特点, 可以有效地学习历史和位置信息, 解决了远程依赖问题. LSTM 是 RNN 的一种改进模型, 通过引入遗忘门、更新门和输出门来控制信息流, 有效地缓解了梯度消失问题. 它使用单元状态来表示每个时间节点的信息. 作为文本分类器使用时, LSTM 可以有效地捕捉文本上下文词之间的关联属性, 并利用遗忘门结构过滤无效信息. 然而, LSTM 是单向的[57]. 由于在漏洞分类任务中需要同时考虑上文和下文的影响, 所以采用 BiLSTM 作为实验模型之一, VulDeePecker 和μVulDeePecker 也采用了这种模型[16,23].

RNN 的另一个变体是 GRU, 它比 LSTM 具有更少的内部门控和更少的参数, 但可以实现类似的功能. GRU 也被安排在未来 VulDeePecker 的实验比较工作中. BiGRU 是 SySeVR 所采用的模型[20], 同时基于前面提到的同样原因, 我们将 BiGRU 作为实验模型之一.

此外, 传统的 RNN 模型倾向于截取或扩展向量表示的输入序列到固定长度, 这限制了解码过程, 特别是对于相对较长的输入序列[58]. 注意力机制通过保留 BiLSTM 编码器的中间输出, 在模型训练过程中选择性地从输入中学习, 并将它们与输出相关联. 使用注意力机制打破了传统模型中编码器和解码器的固定长度限制, 从而提高了实验性能. 因此, 我们也选择了基于注意力的双向长-短时记忆 (Atten-BiLSTM) 作为实验模型之一.

传统的深度学习模型以一批向量表示作为输入, 输出漏洞样本的类型标签. 在本文中, 使用两种方法训练的模型进行实验: 通过交叉熵微调的传统深度学习模型 (参见第 3.1 节) 与使用对比学习框架训练的模型 (参见第 4.1 节). 从整体性能和不同规模的漏洞类型上比较了两个数据集的性能.

2.2 数据集

我们在μVulDeePecker 的开源数据集 MVD 上进行实验, 该数据集共收集了 43 119 个漏洞样本, 涵盖了国家漏洞数据库 (NVD)[59]和软件保障参考数据集 (SARD)[60]中的 40 种漏洞类型. MVD 数据集包含了处理过的代码片段及其对应的标签. 虽然数据预处理已经在图 1 的步骤 2 中进行, 我们仍然需要对代码进行规范化. 规范化过程主要包括 3 个步骤: 步骤 1 是从代码片段文本中删除非 ASCII 字符和注释, 因为它们与漏洞无关. 第 2 步是在符号名称 (例如“VAR1”“VAR2”) 和用户定义变量之间应用一对一的映射. 第 3 步是在符号名称 (如“FUN1”“FUN2”) 和用户定义函数之间应用一对一的映射. 之后, 对数据集进行合并和去重, 得到一个包含 41 660 个漏洞样本的 MVD 数据集, 涵盖 40 种漏洞类型.

我们对 MVD 中的漏洞类型标签进行了统计分析. 不同类型漏洞样本数量的分布情况, 从大到小排序, 如图 2 所示. 图中横轴表示不同的漏洞类型 (未标记), 纵轴表示对应类型的漏洞样本数量. 可以看到, 不同漏洞类型的大小符合长尾幂律分布, 这表明数据集中漏洞类型的样本数量之间存在严重的数据不平衡. 具体来说, 缓冲区溢出是数量最大的漏洞类型 (包含了 CWE122, CWE121, CWE124, CWE127, CWE680, CWE126, CWE416 和 CWE123), 它有 17 872 个漏洞样本, 而最小的是由于调用方不正确的遵循规范导致的共享资源同步错误 (包含了 CWE662 和 CWE573), 它只有 6 个漏洞样本.

为了更全面地研究少样本漏洞类型所造成的影响, 我们在 MVD 的基础上构建了一个 MVD-part 数据集, 该数

据集更加平衡, 更符合传统的少样本特征. 在构建该数据集时不能简单地将所有漏洞类型样本数量降采样到和原始数据集中样本最少的漏洞类型数量一样, 因为这样新数据集将会丢失原始样本数量大小关系. 相反, 对于不同的类型大小, 我们采用分层的降采样策略[61,62]. 具体来说, 由于漏洞类型的第 1 个 1/3 区域的样本数量均在 400 以上, 第 2 个 1/3 区域的均在 80 以上, 因此将样本数量在 400 以上的漏洞类型视为大规模类型, 样本数量在 80–400 之间的漏洞类型视为中等规模类型, 样本数量小于 80 的漏洞类型视为小规模类型. 基于这些定义, 我们将 40 种漏洞类型相对平均地划分为大、中、小规模类型, 不同的规模类型分别包括 13、13、14 种不同的漏洞类型.

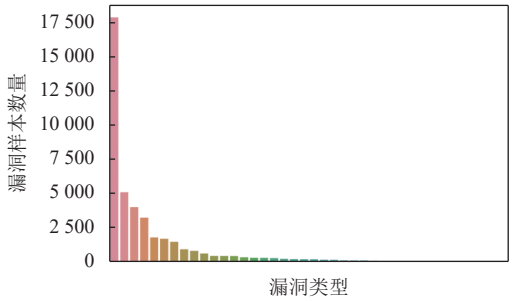

图 2　不同类型漏洞的数量分布

在 MVD 数据集中, 不同漏洞类型样本数量的不平衡会影响传统深度学习模型中的学习过程. 因此在 MVD-part 中采用了上述划分方式通过控制变量法减少了样本数量对学习过程的影响, 使我们更容易探索不同模型对不同漏洞类型的分类性能. 然后将每个大规模的漏洞类型样本数量降采样到 400 个, 每个中等规模的漏洞类型样本数量降采样到 80 个而小规模漏洞类型的样本在被完整保留的同时, 若样本总数不足 10 个, 则会通过随机复制样本的方法将样本数量补足为 10 个. 最终获得的 MVD-part 数据集中样本总数占 MVD 样本总数的 16.16%, 其中大规模样本占 77.00%, 中等规模样本占 15.40% 而小规模样本占 7.59%. MVD-part 相较于小规模样本仅占据 1.23% 的 MVD 数据集平衡得多.

我们没有按漏洞样本数量的固定百分比进行降采样, 因为这样会使得一些大规模类型转换为中等规模类型, 一些中等规模类型转换为小规模类型. 这会导致我们虽然能比较整体性能, 但是由于这种降采样会使得 MVD-part 与 MVD 数据集的大/中/小规模漏洞类型集合不同而无法在同一漏洞类型的尺度上比较 MVD 和 MVD-part 的结果.

2.3　评价指标

本文参考了 μVulDeePecker[23]中使用的评价指标以及针对不同规模类型的几个指标来评估模型的分类性能. 请注意, 所有的指标均为值的百分比. 本文用 N 表示样本总数, FPR_i、W_i、FNR_i、$F1_i$ 和 STD_i 分别表示第 i 个漏洞类型的 M_F1 的假阳性率、权重、假阴性率、$F1$ 分数和标准差值. 其中 W_i 范围为 0–1, 由样本数量决定, 其总和为 1. 度量标准如下.

• *M_FPR* (根据 10 路交叉验证的漏洞类型总数得到的假阳性率的多分类对应值的平均值). *M_FPR* 的计算公式如公式 (1) 所示:

$$M\_FPR = (1/N) \times \sum_{i=1}^{N} FPR_i \tag{1}$$

• *W_FPR* (根据 10 路交叉验证的每个漏洞类型数量计算的权重得到的假阳性率的多分类对应值的平均值). *W_FPR* 的计算公式如公式 (2) 所示:

$$W\_FPR = \sum_{i=1}^{N} (W_i \times FPR_i) \tag{2}$$

• *M_FNR* (根据 10 路交叉验证的漏洞类型总数得到的假阴性率的多分类对应值的平均值). *M_FNR* 的计算公式如公式 (3) 所示:

$$M\_FNR = (1/N) \times \sum_{i=1}^{N} FNR_i \tag{3}$$

• *W_FNR* (根据 10 路交叉验证的每个漏洞类型数量计算的权重得到的假阴性率的多分类对应值的平均值). *W_FNR* 的计算公式如公式 (4) 所示:

$$W\_FNR = \sum_{i=1}^{N} (W_i \times FNR_i) \tag{4}$$

• *M_F*1 (根据 10 路交叉验证的漏洞类型总数得到的 *F*1 度量的多分类对应值的平均值). *M_F*1 的计算公式如公式 (5) 所示:

$$M\_F1 = (1/N) \times \sum_{i=1}^{N} F1_i \tag{5}$$

• *W_F*1 (根据 10 路交叉验证的每个漏洞类型数量计算的权重得到的 *F*1 度量的多分类对应值的平均值). *W_F*1 的计算公式如公式 (6) 所示:

$$W\_F1 = \sum_{i=1}^{N} (W_i \times F1_i) \tag{6}$$

• *Mean_L* (*M_F*1 值在 10 路交叉验证里大规模漏洞类型上的平均值).

• *Mean_M* (*M_F*1 值在 10 路交叉验证里中规模漏洞类型上的平均值).

• *Mean_S* (*M_F*1 值在 10 路交叉验证里小规模漏洞类型上的平均值). 我们用 *Mean* 来表示 *Mean_L*、*Mean_M* 和 *Mean_S*, *Mean* 的计算公式如公式 (7) 所示:

$$Mean = (1/N) \times \sum_{i=1}^{N} M\_F1_i \tag{7}$$

• *STD_L* (*M_F*1 值在 10 路交叉验证里大规模漏洞类型上的标准偏差值的平均值).

• *STD_M* (*M_F*1 值在 10 路交叉验证里中规模漏洞类型上的标准偏差值的平均值).

• *STD_S* (*M_F*1 值在 10 路交叉验证里小规模漏洞类型上的标准偏差值的平均值). 我们用 *STD* 来表示 *STD_L*, *STD_M* 和 *STD_S*, *STD* 的计算公式如公式 (8) 所示:

$$STD = (1/N) \times \sum_{i=1}^{N} \sqrt{\sum_{j=1}^{10} (F1_{ij} - Mean)^2 / 9} \tag{8}$$

3 传统深度学习模型在少样本漏洞类型上的表现

在本节中, 分别使用 4 种传统的深度学习模型 (CNN、BiGRU、BiLSTM 和 Atten-BiLSTM) 在 MVD 和 MVD-part 数据集上进行了实验. 研究了它们的整体性能以及在大、中、小规模的漏洞类型上的性能. 结果表明, 尽管模型在 MVD 和 MVD-part 数据集上的总体性能令人满意, 但其性能从大到小依次下降.

3.1 训练方法

实验中使用了 10 路交叉验证方法. *N* 个样本 (*N*≥10) 的漏洞类型被随机分成 10 个大小相同的子集. 而切分剩余的样本则随机分散到 10 个子集中 (每个子集 1 个). 对于具有 *N* 个漏洞样本 (*N*<10) 的小规模类型, 首先利用留一法获得只有 *N* 路的交叉验证数据, 然后从这 *N* 路中随机选择 10–*N* 次并每次进行复制. 通过这样的方式得到了小规模漏洞类型的 10 路交叉验证数据. 这样可以保证测试数据中的每个漏洞类型至少包含一个样本. 在 10 路交叉验证数据中, 9 个子集将用于训练, 1 个子集将用于整个实验的测试.

3.2 超参数优化

在 10 路交叉验证的实验中, 我们使用了网格探索策略[63]对模型超参数进行优化. 之后根据这些不同超参数

组合在前 3 组实验中的平均表现选择了最佳的组合, 并将该超参数组合应用到了后续实验中. 表 1 展示了本文在 CNN 模型中所使用的参数.

表 2 则展示了本文在包括 BiGRU、BiLSTM 和 Atten-BiLSTM 等在内的 RNN 模型中所使用的参数. 使用 Adam 优化算法以及 0.000 02 的学习率训练所有的 4 种传统深度学习模型[64]. 在模型训练好后, 不再修改这些模型的参数, 并使用这些模型进行漏洞分类任务.

表 1 CNN 使用的参数

参数名	配置
损失函数	交叉熵损失函数
滤波器尺寸	(2, 3, 4)
滤波器数量	256
激活函数	ReLU
池化层策略	最大池化
优化策略	Adam
批量大小	32
学习率	0.000 02

表 2 BiGRU, BiLSTM 和 Atten-BiLSTM 使用的参数

参数名	配置
损失函数	交叉熵损失函数
激活函数	tanh
层数	2
优化策略	Adam
批量大小	32
学习率	0.000 02

3.3 结 果

为了评估传统深度学习模型在少样本漏洞类型上的分类性能, 本文研究了以下两个问题.

• RQ1: 传统深度学习模型在 MVD 和 MVD-part 数据集上的整体性能如何?

• RQ2: 4 种传统模型在 MVD 和 MVD-part 数据集上的大、中、小规模漏洞类型上的表现如何?

3.3.1 RQ1

4 种传统深度学习模型在 MVD 和 MVD-part 数据集上的实验结果总结如表 3 所示.

表 3 传统深度学习模型在 MVD 和 MVD-part 数据集上各性能指标表现

数据集	模型	*M_FPR*	*M_FNR*	*M_F*1	*W_FPR*	*W_FNR*	*W_F*1	*Mean_L*	*Mean_M*	*Mean_S*	*STD_L*	*STD_M*	*STD_S*
MVD	CNN	0.015	3.020	97.582	0.113	**0.494**	**99.493**	**99.182**	**98.982**	94.797	1.466	2.330	16.824
	BiGRU	0.018	4.046	96.694	0.090	0.638	99.348	99.006	99.463	92.904	1.476	2.138	17.153
	BiLSTM	0.020	4.775	95.957	0.115	0.662	99.317	99.026	98.528	90.721	1.520	2.233	19.086
	Atten-BiLSTM	**0.014**	**2.824**	**97.621**	**0.066**	0.503	99.487	99.169	98.797	**95.092**	**1.338**	**2.137**	**16.281**
MVD-part	CNN	**0.054**	**2.959**	**96.818**	**0.104**	**2.058**	**97.910**	**97.920**	**98.594**	**94.145**	**2.291**	**4.161**	**17.402**
	BiGRU	0.119	6.751	92.579	0.191	4.557	95.410	96.090	94.412	87.616	3.619	7.120	23.217
	BiLSTM	0.141	8.555	91.493	0.242	5.390	94.572	95.380	93.739	85.797	4.622	7.903	23.602
	Atten-BiLSTM	0.117	5.786	93.960	0.201	4.460	95.537	95.925	96.329	89.936	4.186	6.656	17.895

数据显示, CNN 的综合性能最好, 其次是 Atten-BiLSTM, 再次是 BiGRU, BiLSTM 表现最差. 为了更好地展示模型在 MVD 和 MVD-part 上的性能, 我们以 *M_F*1 作为主要评价标准, 并在图 3 中绘制了一个箱型图以便更直观地表示.

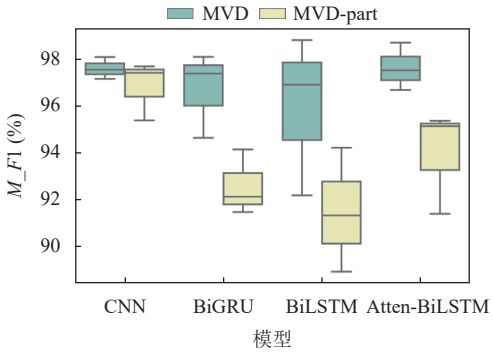

图 3 传统深度学习模型在 MVD 和 MVD-part 上的总体性能

如图 3 所示, 4 种模型在 MVD 上的表现都优于 MVD-part, 这证明传统的深度学习模型依赖于具有足够样本的训练集, 因此在少样本漏洞类型的分类上表现不佳. CNN 的 diff 最小, 而 BiLSTM 的 diff 最大, 这进一步证实了我们对模型的评估. BiLSTM 的最差表现符合我们的预期, 因为 BiGRU 和 Atten-BiLSTM 都是基于 BiLSTM 的改进模型.

综上所述, 对于漏洞分类任务, 4 种传统深度学习模型在 MVD 和 MVD-part 数据集上都有较好的性能表现都取得了较好的性能, 但是他们在 MVD-part 的表现较差.

3.3.2 RQ2

图 4 分别展示了 4 种深度学习模型在 MVD 和 MVD-part 数据集上对大、中、小规模漏洞类型的性能. 我们以 *M_F*1 分数作为评价标准, 代表模型在所有类型上的性能的平均值. 图 4 显示每个模型的 *M_F*1 分数从大型到小型漏洞类型均在稳步下降. 由表 3 可知, 4 个模型的平均 *M_F*1 分数在 MVD 上由大到中型下降 0.40%, 由中到小型下降 5.31%. MVD-part 下降幅度更大, 大到中型下降 0.56%, 中到小型下降 6.40%. 可以观察到, 对于大规模漏洞类型, 在 10 个交叉验证集中, *M_F*1 值的下降幅度较小. 同时, 中小规模的漏洞类型下降幅度更大. 这一现象表明, 模型在大规模类型上具有较好的泛化能力, 但在少样本漏洞类型进行分类时表现不佳.

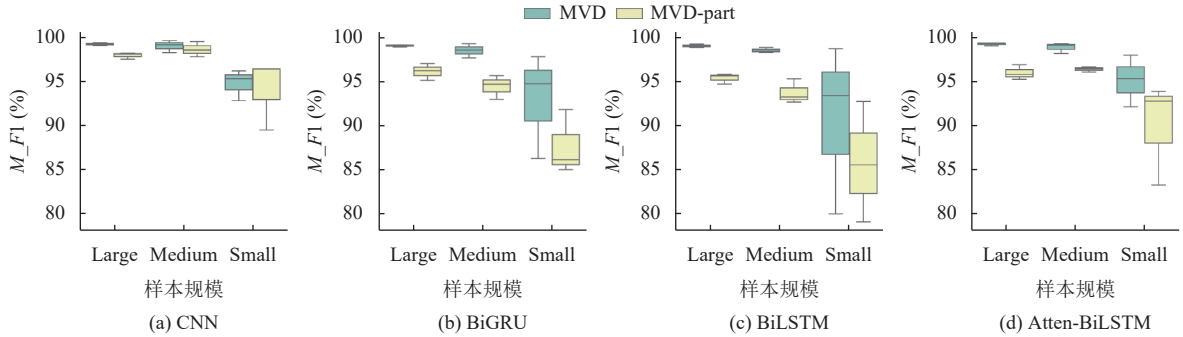

图 4 MVD 和 MVD-part 数据集上传统深度学习模型在不同漏洞类型规模上的性能

通过进一步的比较, 我们发现这 4 个模型在两个数据集上的大规模漏洞类型上具有非常相似的 *M_F*1 值, 这意味着 MVD-part 数据集的降采样对大规模漏洞类型的检测性能几乎没有影响. 但对于中等规模的漏洞类型, 4 个模型中的 3 个在 MVD-part 数据集上的表现更差. 而对于小规模的漏洞类型, 4 个模型中的两个在 MVD 部分数据集上表现较差, 而其他两个则有所改善.

理论上, 在 MVD-part 数据集中, 小规模类型的样本受到大规模和中等规模类型的影响更小, 这使我们更容易捕捉它们的特征, 从而提高性能. 然而通过图 4 可以观察到在大规模漏洞类型的 10 组交叉验证中, *M_F*1 值的变化较小, 而在小规模和中等规模的漏洞类型中, 变化较大. 这一现象表明传统模型只在较大的类型尺度上具有较好的泛化能力. 通过查看具体数据, 可以发现 MVD 和 MVD-part 的标准偏差 (*STD*) 值从大型到中型分别增长 0.76% 和 2.78%, 从中型到小型分别增长 15.13% 和 14.07%. 且中小规模类型上许多 *M_F*1 值的 STD 大于 0.3, 最低的 *M_F*1 值为 0, 最高的为 1, 这进一步表明在更平衡的数据集上, 传统模型在少样本类型上的较差性能并没有改善.

综上所述, 在漏洞分类任务上, 4 种传统深度学习模型在不同规模的漏洞类型上表现不一致, 且在 MVD 和 MVD-part 数据集上其性能表现从大规模到小规模类型上均依次下降.

4 通过对比学习进行优化

为了解决第 3 节中暴露的对少样本漏洞类型分类性能差的挑战, 在本节中提出了基于对比学习的漏洞分类框架 VulFewShot. 比较了传统模型和使用 VulFewShot 框架在 MVD 和 MVD-part 数据集上训练的模型的整体性能. 还分别比较了它们在大、中、小规模漏洞类型上的性能. 实验结果表明, 使用 VulFewShot 框架训练的模型在少样本漏洞类型上优于传统深度学习模型, 从而进一步提高了整体分类性能. 通过比较 MVD-part 的性能, 本文展示了对比学习在漏洞分类任务中对少样本学习的巨大改进.

4.1 基于对比学习的方法

在本节中, 将介绍基于对比学习的漏洞分类框架 VulFewShot.

4.1.1 概 述

如图 5 所示, VulFewShot 由两个主要阶段组成: 训练和分类阶段. 训练阶段的目的是训练出一个性能良好的分类模型. 这个阶段由以下 3 个步骤组成. (1) 代码静态分析: 该步骤旨在从漏洞样本中提取代码片段, 对代码片段进行标注和归一化. (2) 向量提取: 该步骤旨在将规范化的代码片段转化为向量表示. (3) 基于对比学习的模型训练: 这一步使用上一步的向量表示, 通过对比学习的方式对传统深度学习模型进行微调, 训练出准确的分类模型. 其中, 对比学习旨在最大化相似数据之间的相似度并最小化不相似数据的相似度. 这里我们并未使用数据增强来生成新样本与原始样本对比, 而是将训练中不同轮次中的相同漏洞样本类型作为对比, 具体的方法将在第 4.1.3 节进行介绍.

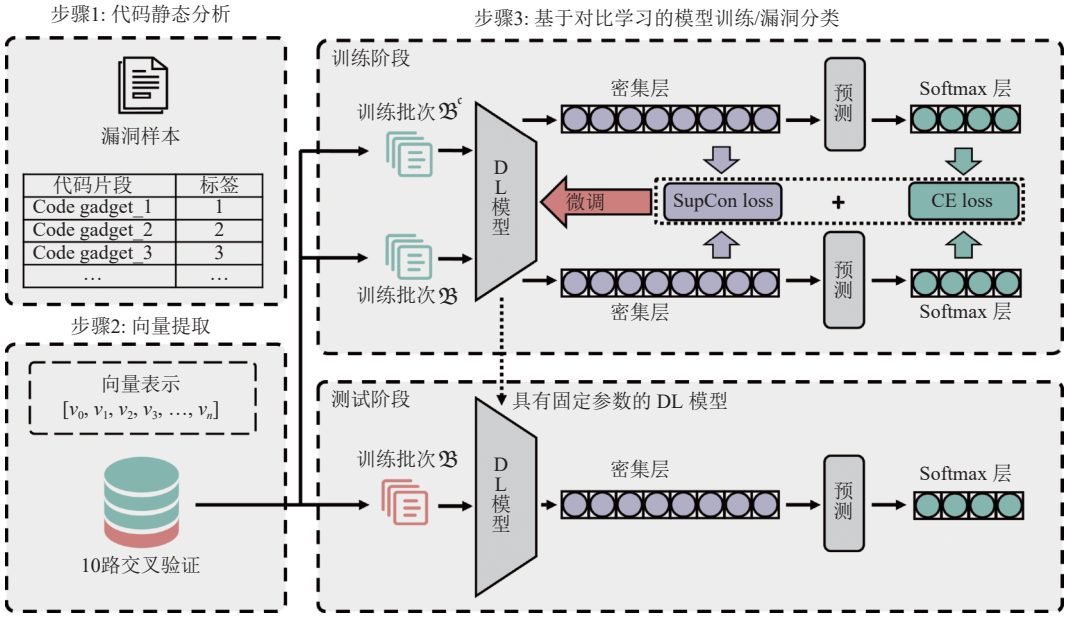

图 5 VulFewShot 的系统概述

分类阶段的目标是将未标记的漏洞分类为相应的类型. 该阶段包括以下 3 个步骤: (1) 代码静态分析, (2) 向量提取, (3) 漏洞分类. 前两步的执行方式与训练阶段完全相同. 第 3 步应用从对比学习框架中训练出来已固定参数的模型对漏洞进行分类, 然后将一批未标记漏洞样本的向量表示作为输入, 输出相应的类型标签.

4.1.2 代码静态分析&向量提取

正如第 2.2 节和图 1 所示, 在μVulDeePecker 中按照代码静态分析和向量提取的步骤处理数据. 首先, 从代码片段文本中删除无关的非 ASCII 字符和注释. 其次, 在符号名称和代码之间应用一对一的映射, 包括“VAR”到变量, “FUN”到函数, “STR”或“NUM”到常量等. 最后, 我们生成一个序列化文件, 该文件包含 3 类信息: 代码片段、代码标签和规范化后的代码片段. 其中代码标签来自μVulDeePecker.

在向量提取的步骤中, 将规范化的代码片段转换为固定长度的向量. 转换是通过在μVulDeePecker 中使用的一个简单的词嵌入实现的. 这些向量是训练模型的输入.

4.1.3 基于对比学习的模型训练

在日常生活中, 孩子们只需要几张动物图片就能识别出动物之间的差异[65]. 人类在只提供少量样本的情况下, 往往会在比较不同类别样本的同时寻求同一类别样本之间的相似性, 这使我们能够捕捉到物体的高级特征, 从而将其与其他物体区分开来.

在这种学习策略的启发下, 我们假设基于对比学习的损失可以驱动模型提取样本的更高级特征. 这样, 模型可

以通过更少的样本实现更好的性能, 这可能有助于改善在少样本的漏洞类型上较差的分类性能以及在保证良好的性能的同时最小化时间和计算资源的投入. 因此, 本文提出了一种基于监督对比学习的模型训练框架, 使相同类型的漏洞样本更紧密地聚集在一起, 同时使不同类型的漏洞样本彼此远离.

对比学习已经在许多方面得到了应用, 包括图像、语音和视频[38−50]. 它们倾向于用不同的增强方法扩展样本, 并将增强样本作为与原始样本的比较项. 然而, 与这些方法不同的是, 我们并不进行数据增强. 相反, 如图 6 所示, 我们将漏洞类型相同但位于不同训练轮次中的样本作为比较对象.

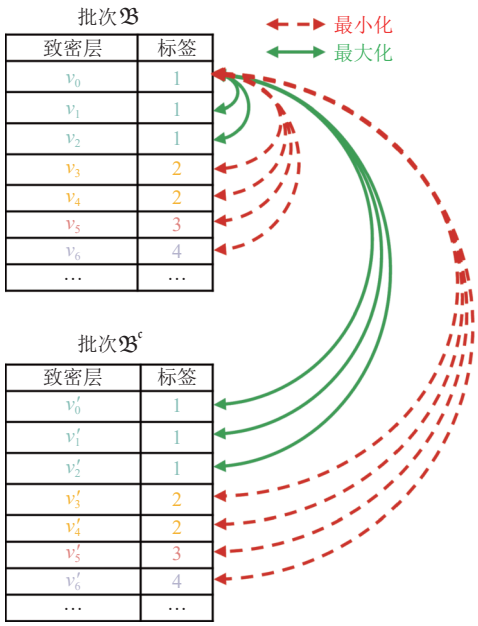

图 6 VulFewShot 中的监督对比学习

具体来说, 对于具有 T 类对象和批量大小为 N 的漏洞分类任务, 将一个批量大小的训练示例 $\{v_i, l_i\}_{i=1,\ldots,N}$ 称为 $\mathfrak{B}$, 其中 v_i 表示 $\mathfrak{B}$ 中第 i 个漏洞样本的输入向量表示, l_i 表示对应的标签. $l_{i,t}$ 表示标签类型, $\hat{l}_{i,t}$ 表示第 i 个漏洞样本的模型输出属于类型 t 的概率. 交叉熵 (CE) 损失 $\mathcal{L}_{\mathrm{CE}}$ 可以定义为公式 (9):

$$\mathcal{L}_{\mathrm{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} l_{i,t}\cdot\log\hat{l}_{i,t} \tag{9}$$

然后本文生成了一个比较数据集 $\{v'_i, l_i\}_{i=1,\ldots,N}$ 称为 $\mathfrak{B}^c$, 其中 v'_i 表示训练数据集中与 v'_i 标签相同但在 $\mathfrak{B}$ 中不相同的漏洞样本的向量表示. 将两个数据集合并在一起, 得到 $\mathfrak{B}^a = \mathfrak{B}\cup\mathfrak{B}^c$ 作为训练批次.

如图 6 所示, 本文的对比学习框架的目标是最大化具有相同类型标签的漏洞样本之间的相似性, 最小化不同类型样本之间的相似性. 对于 $\mathfrak{B}^a$ 中的 v'_i, 我们借鉴了 SimCLR 的思路[50], 尝试通过最小化公式 (10) 所表示的标准化温度范围交叉熵损失 $\mathcal{L}_i$ 来使其与相同标签项的距离更近, 与不同标签项的距离更远. 然后对 $\mathfrak{B}^a$ 中所有样本的监督对比学习 (SupCon) 损失 $\mathcal{L}_{\mathrm{SupCon}}$ 进行加权平均, 其定义为公式 (11), N_{l_i} 表示批次中与 l_i 具有相同标签的样本总数:

$$\mathcal{L}_i = -\sum_{j=1}^{2N}\mathbf{1}_{i\neq j}\mathbf{1}_{l_i=l_j}\log\frac{\exp(\Phi(v_i)\cdot\Phi(v_j)/\tau)}{\sum_{k=1}^{2N}\mathbf{1}_{i\neq k}\exp(\Phi(v_i)\cdot\Phi(v_k)/\tau)} \tag{10}$$

其中, $\Phi(\cdot)$ 表示由深度学习构建的编码器, 输出在最终的 Softmax 操作之前已经过 L2 范数归一化的最后一个密集层, τ 表示控制类分离的温度参数.

$$\mathcal{L}_{\text{SupCon}}=\sum_{i=1}^{2N}\frac{1}{N_{l_i}-1}\mathcal{L}_i \tag{11}$$

综上所述, 总体客观损失 $\mathcal{L}_{\text{All}}$ 的定义为公式 (12):

$$\mathcal{L}_{\text{All}}=\eta\mathcal{L}_{\text{CE}}+(1-\eta)\mathcal{L}_{\text{SupCon}} \tag{12}$$

$\mathcal{L}_{\text{CE}}$ 在公式 (9) 中定义, $\mathcal{L}_{\text{SupCon}}$ 在公式 (11) 中定义, 我们使用 η 作为标量加权超参数, 以保持交叉熵损失和监督对比损失的平衡.

4.1.4 漏洞类型分类

在训练阶段结束后, 可以获得经过训练的带有固定参数的深度学习分类模型, 对新的未标记的漏洞样本进行分类. 具体来说, 给定一个新的漏洞样本, 首先进行静态分析, 以提取代码片段. 然后片段中的每一个单词都会被转换成一个 768 维的向量, 并组合成漏洞样本的向量表示. 最后, 模型将向量表示作为输入, 输出相应的类型标签, 完成漏洞分类任务.

4.2 模型超参数优化

由于本文使用与第 3 节相同的深度学习模型, 因此大多数超参数与第 3.2 节相同. 然后在对比学习中引入了两个新的超参数 τ 和 η. 对于使用对比学习框架的每个实验, 使用 $\tau\in\{0.1,0.3,0.5,0.7\}$ 和 $\eta\in\{0.1,0.3,0.5,0.7,0.9\}$ 进行基于网格的超参数扫描. 最后, 根据当 $\tau=0.1$ 和 $\eta=0.3$ 的情况在前 3 组实验中的平均表现, 选择 $\tau=0.1$ 和 $\eta=0.3$ 作为最佳的超参数组合.

4.3 结 果

为了评估 VulFewShot 是否能有效解决对少样本类型分类差的问题, 我们研究了以下两个问题.

• RQ3: 与传统深度学习模型相比, 使用 VulFewShot 框架训练的模型在整体性能方面表现如何?

• RQ4: 在 MVD 和 MVD-part 数据集上, 与传统深度学习模型相比, 使用 VulFewShot 框架训练的模型在大、中、小规模漏洞类型上的表现如何?

4.3.1 RQ3

图 7 比较了传统深度学习模型和使用 VulFewShot 框架训练的模型在 MVD 和 MVD-part 上的性能. 其中, CE 表示采用交叉熵损失而 CE+SupCon 表示结合交叉熵损失和对比学习损失. 可以看到, 使用 VulFewShot 框架训练的模型在 MVD 和 MVD-part 上都比传统模型获得了更高的分数. 由于页面限制, 只在表 4 中展示了整体性能评估指标的 *M_F*1 和 *W_F*1 分数. 本文的框架与 CNN、BiGRU、BiLSTM 和 Atten-BiLSTM 这 4 种模型相比在 MVD 上分别提高了 0.205%、0.964%、1.627% 和 0.535% 的 *M_F*1 分数, 在 MVD-part 上分别提高了 0.478%、2.769%、2.853% 和 3.336% 的 *M_F*1 分数.

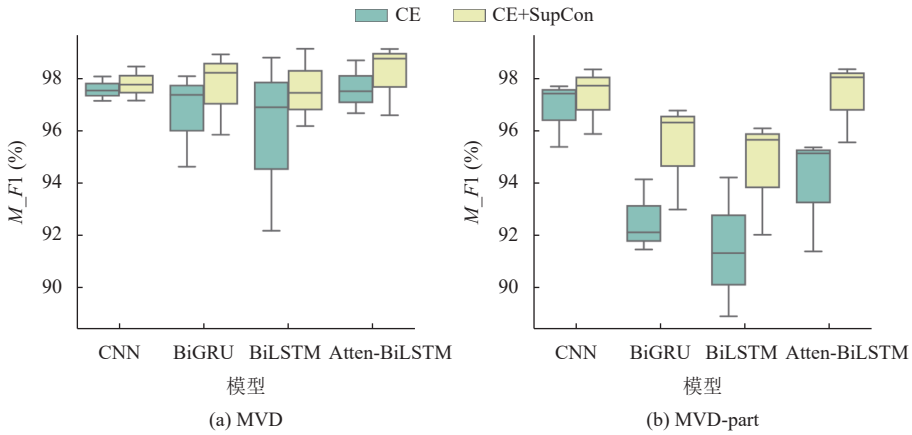

图 7 MVD 和 MVD-part 数据集上使用传统深度学习模型和使用 VulFewShot 框架训练的模型的总体性能

表 4 传统深度学习模型和使用 VulFewShot 框架训练的模型在 MVD 和 MVD-part 数据集上各项性能指标表现 (%)

模型框架	模型名称	MVD					MVD-part				
		*M_F*1	*W_F*1	*Mean_L*	*Mean_M*	*Mean_S*	*M_F*1	*W_F*1	*Mean_L*	*Mean_M*	*Mean_S*
传统模型	CNN	97.582	99.493	**99.182**	98.982	94.797	96.818	97.910	**97.920**	98.594	94.145
	BiGRU	96.694	**99.348**	**99.006**	98.463	92.904	92.579	95.410	96.090	94.412	87.616
	BiLSTM	95.957	99.317	**99.026**	98.528	90.721	91.493	94.572	**95.380**	93.739	85.797
	Atten-BiLSTM	97.621	99.487	99.169	98.797	95.092	93.960	95.537	95.925	96.329	89.936
VulFewShot	CNN	**97.788**	**99.512**	99.121	**99.262**	**95.180**	**97.296**	**97.920**	97.760	**99.336**	**94.970**
	BiGRU	**97.658**	99.335	98.955	**98.589**	**95.590**	**95.348**	**96.411**	**96.745**	**95.561**	**93.853**
	BiLSTM	**97.585**	**99.395**	98.977	**98.856**	**95.111**	**94.346**	**95.142**	95.360	**95.344**	**92.478**
	Atten-BiLSTM	**98.156**	**99.520**	**99.175**	**99.315**	**96.133**	**97.296**	**97.815**	**97.763**	**98.372**	**95.863**

一方面, 虽然与 CNN 模型相比整体性能提升并不明显, 但与其他 3 种模型相比性能提升的表现则较为突出. 通过表 4 中的性能指标, 我们可以发现 CNN 在所有 4 种传统模型中的平均得分是最好的, 并且得分非常高, 所以即使是轻微的改进也可以证明 VulFewShot 的优秀.

另一方面, 传统深度学习模型在 MVD-part 上的平均性能比 MVD 差 3.25%, VulFewShot 将这一差距缩小到 1.73%. 从第 2.2 节可知, MVD-part 在少样本漏洞类型的评价标准上优于 MVD, 因为 MVD 中不同漏洞类型样本数量的不平衡会影响传统深度学习模型的学习过程, 与控制变量的原则相悖. 因此, 我们可以得出结论, VulFewShot 在少样本漏洞类型的分类上比传统的深度学习模型表现得更好.

从交叉实验结果中的数据分布也可以看出, 使用 VulFewShot 框架训练的模型具有更好的泛化能力, 这意味着它们在多个实验中具有更稳定的性能.

综上所述, 根据评估指标的结果, 使用 VulFewShot 框架训练的模型优于传统深度学习模型, 在 MVD-part 数据集上的优势更为明显.

4.3.2 RQ4

为了更好地展示传统深度学习模型和使用 VulFewShot 框架训练的模型在不同漏洞类型尺度上的性能比较, 用箱形图展示了每个模型在 MVD 和 MVD-part 上的性能, 如图 8 所示. 其中, CE 表示采用交叉熵损失而 CE+SupCon 表示结合交叉熵损失和对比学习损失. 我们在图 8 中可以看到的是本文的框架带来的不同漏洞类型规模上的性能提升, 尤其是在少样本类型的漏洞上的提升是不可否认的. 同时, 本文的框架减少了从大规模漏洞类型到小规模漏洞类型的性能下降程度.

具体而言, 对于小规模类型的平均 *M_F*1 分数, VulFewShot 相比于 CNN、BiGRU、BiLSTM 和 Atten-BiLSTM 在 MVD 上提高分别为 0.383%、2.686%、4.390% 和 1.041%, 在 MVD-part 上分别提高 0.825%、6.237%、6.681% 和 5.927%. 对于中等规模类型的平均 *M_F*1 分数, VulFewShot 相比于 CNN、BiGRU、BiLSTM 和 Atten-BiLSTM 在 MVD 上提高了 0.126%–0.518%, 在 MVD-part 上提高了 0.742%–2.043%. 对于大规模类型的平均 *M_F*1 分数, VulFewShot 与 4 种传统模型在两个数据集上的比较实验中, 8 例中有 5 例性能略有下降 (0.020%–0.160%). 在其他 3 个案例中, 性能略有提升 (0.006%–1.838%). 在 CNN 上的改善效果比在其他模型上的改善效果少得多的原因是 CNN 的分数很高, 一点点的增加都是有价值的. 可以看到在 MVD 和 MVD-part 上 CNN 和 VulFewShot 的结果差异在大规模到小规模漏洞类型上是稳定增加的, 符合结果预期, 因此可以认为在排除错误的影响下, VulFewShot 在对少样本漏洞类型的分类上确实优于 CNN. 此外, 部分案例在大规模类型上略有下降的原因可能有两个: 一是模型在大规模类型上已经取得了很好的结果 (MVD 为 99.006%–99.182%, MVD-part 为 95.380%–97.920%), 改进空间有限. 另一个原因是实验中的随机性和偶然性.

图 9 显示了 MVD 和 MVD-part 上模型的平均标准偏差 (*STD*) 值. 从图 9 中可以看出, 本文的框架提高了模型的泛化能力, 并且在 MVD-part 数据集中的改进更为明显. 在比较不同规模的漏洞类型时, 从大规模到小规模类型, 改进程度依次增加.

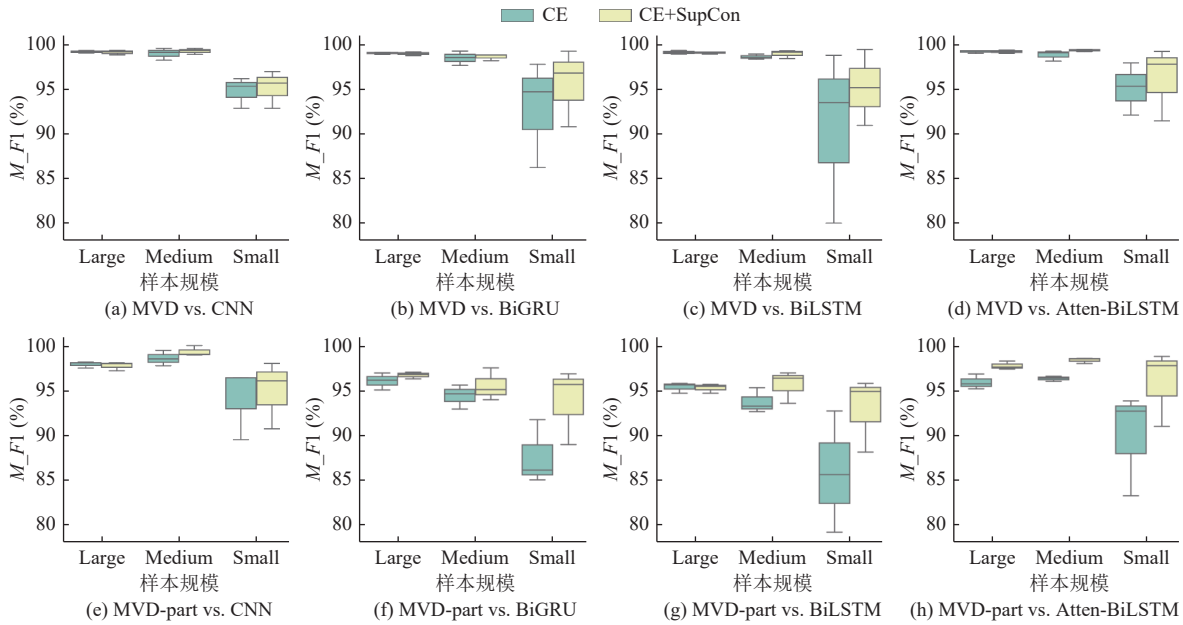

图 8　MVD 和 MVD-part 数据集中, 传统深度学习模型和使用 VulFewShot 框架训练的模型在不同漏洞类型规模上的性能

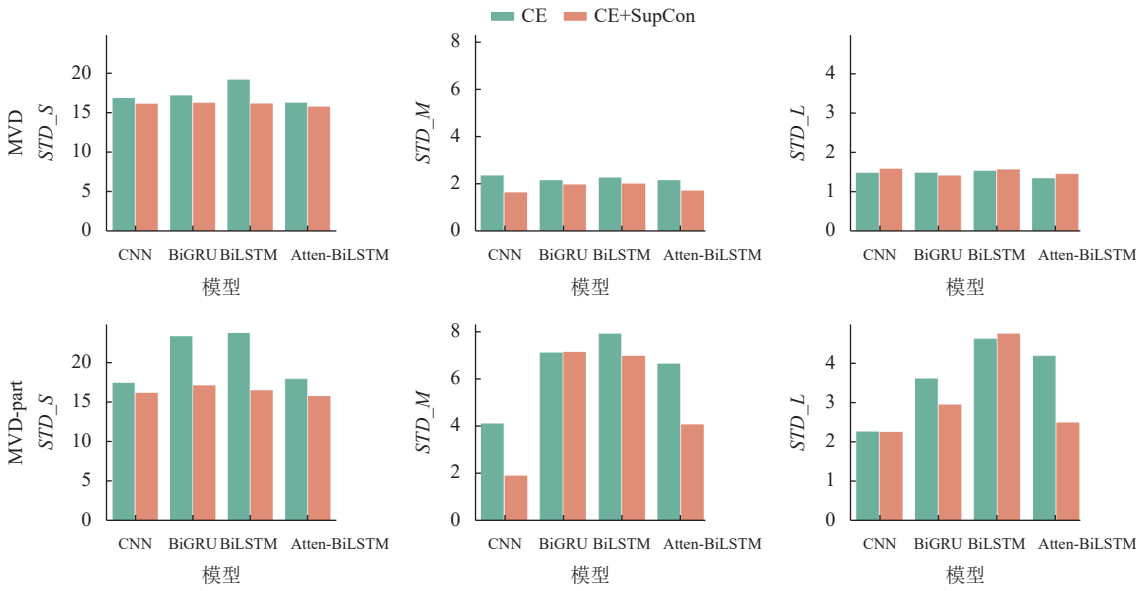

图 9　MVD 和 MVD-part 数据集上传统深度学习模型和使用 VulFewShot 框架训练的模型的 STD 性能

综上所述, VulFewShot 可以显著提高传统深度学习模型在中小规模漏洞类型上的性能, 从而缓解了在少数漏洞类型上的性能下降. 这种改进从大尺度到小尺度依次增加, 在 MVD-part 数据集上尤为明显.

5　对有效性的威胁

本文的实验方法和结论可能在内部和外部有效性方面受到以下描述的某些方面的潜在威胁.

一种可能的内部威胁是以特定库/API 函数调用为起点的代码片段漏洞. 我们没有使用整个程序代码, 而是使用这些代码片段来提取特征, 这些特征在多项研究中被证明对漏洞的检测和分类是有效的[16,20,23]. 并且提取的代码片段不仅是原始漏洞代码的一部分而且是可解释的. 相关研究正试图基于代码片段检测并以图片的形式直观地表示漏洞.

另一个潜在的内部威胁可能是我们对类型尺度的定义. 由于不同漏洞类型之间的严重数据不平衡 (最大的有 17 872 个漏洞样本, 最小的只有 6 个), 不能将所有漏洞类型降采样到最小的大小, 因为大多数样本将会丢失. 因此, 本文采用了分层的下采样策略, 以保持不同规模的漏洞数量相对均匀. 明确地说, 将样本量在 400 以上的类型设置为大规模类型, 样本量在 80–400 之间的为中等规模类型, 样本量小于 80 的为小规模类型. 这样一来, 数据相对比较均衡, 从而便于考察模型在各种规模的漏洞类型上的表现.

最后, 数据集的质量和代表性可能是我们研究的外部威胁. 我们相信 MVD 的质量, 因为它是由专门从事安全工作的研究人员构建的并且已经在各种现有的研究中使用. 与此同时, 数据集中漏洞片段的提取是基于相应的 CWE ID, 我们已经观察到具有相同类型标签的代码片段的数据特征有相似之处. 未来, 可以在更多的漏洞分类数据集上进一步验证我们的发现.

6 结 论

在本文工作中, 我们发现漏洞类型之间存在严重的数据不平衡. 然后本文评估了多个传统深度学习模型, 包括 CNN、BiGRU、BiLSTM 和 Atten-BiLSTM, 来对漏洞进行分类. 结果表明, 不同漏洞类型的分类性能随着漏洞样本数量的减少而下降, 导致在少样本类型上的分类性能较差. 为了应对这一挑战, 本文实现了 VulFewShot, 这是一个基于对比学习的漏洞分类框架, 以提高当前的分类性能. VulFewShot 显著提高了整体水平和不同样本规模下所有类型漏洞的分类性能, 并且在少样本类型上的提升是不可否认的, 这进一步提升了整体性能.

MVD 和 MVD-part 数据集的评估结果表明, VulFewShot 不仅在总体水平上, 而且在具有不同样本规模的所有类型的漏洞上, 都显著提高了分类性能. 这种改进少样本类型上尤其明显, 这进一步提高了整体性能.未来, 我们有兴趣应用更多的特征提取方法来探索其他编程语言 (如 Python、Java 等) 中的漏洞分类任务, 并在 3 种不同规模的漏洞大小 (大、中、小) 上进一步实验 4 种传统的深度学习模型.

References:

[1] Sun N, Zhang J, Rimba P, Gao S, Zhang LY, Xiang Y. Data-driven cybersecurity incident prediction: A survey. IEEE Communications Surveys & Tutorials, 2019, 21(2): 1744–1772. [doi: 10.1109/COMST.2018.2885561]

[2] Liu L, De Vel O, Han QL, Zhang J, Xiang Y. Detecting and preventing cyber insider threats: A survey. IEEE Communications Surveys & Tutorials, 2018, 20(2): 1397–1417. [doi: 10.1109/COMST.2018.2800740]

[3] Rajeh W, Jin H, Zou DQ. Saudi cloud infrastructure: A security analysis. Science China Information Sciences, 2017, 60(12): 122102. [doi: 10.1007/s11432-016-0322-7]

[4] Cherem S, Princehouse L, Rugina R. Practical memory leak detection using guarded value-flow analysis. In: Proc. of the 28th ACM SIGPLAN Conf. on Programming Language Design and Implementation. San Diego: ACM, 2007. 480–491. [doi: 10.1145/1250734.1250789]

[5] Fan G, Wu RX, Shi QK, Xiao X, Zhou JG, Zhang C. SMOKE: Scalable path-sensitive memory leak detection for millions of lines of code. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 72–82. [doi: 10.1109/ICSE.2019.00025]

[6] Li W, Cai HP, Sui YL, Manz D. PCA: Memory leak detection using partial call-path analysis. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 1621–1625. [doi: 10.1145/3368089.3417923]

[7] Shi QK, Xiao X, Wu RX, Zhou JG, Fan G, Zhang C. Pinpoint: Fast and precise sparse value flow analysis for million lines of code. In: Proc. of the 39th ACM SIGPLAN Conf. on Programming Language Design and Implementation. Philadelphia: ACM, 2018. 693–706. [doi: 10.1145/3192366.3192418]

[8] Smith J, Johnson B, Murphy-Hill E, Chu B, Lipford HR. How developers diagnose potential security vulnerabilities with a static analysis

tool. IEEE Trans. on Software Engineering, 2019, 45(9): 877–897. [doi: 10.1109/TSE.2018.2810116]

[9] Aloraini B, Nagappan M, German DM, Hayashi S, Higo Y. An empirical study of security warnings from static application security testing tools. Journal of Systems and Software, 2019, 158: 110427. [doi: 10.1016/j.jss.2019.110427]

[10] Yamaguchi F, Lindner F, Rieck K. Vulnerability extrapolation: Assisted discovery of vulnerabilities using machine learning. In: Proc. of the 5th USENIX Workshop on Offensive Technologies. San Francisco: USENIX Association, 2011.

[11] Walden J, Stuckman J, Scandariato R. Predicting vulnerable components: Software metrics vs. text mining. In: Proc. of the 25th IEEE Int'l Symp. on Software Reliability Engineering. IEEE, 2014. 23–33.

[12] Harer JA, Kim LY, Russell RL, Ozdemir O, Kosta LR, Rangamani A, Hamilton LH, Centeno GI, Key JR, Ellingwood PM, Antelman E, Mackay A, McConley MW, Opper JM, Chin P, Lazovich T. Automated software vulnerability detection with machine learning. arXiv: 1803.04497, 2018.

[13] Lee YJ, Choi SH, Kim C, Lim SH, Park KW. Learning binary code with deep learning to detect software weakness. 2017. http://syscore.sejong.ac.kr/~woongbak/publications/C37.pdf

[14] Russell R, Kim L, Hamilton L, Lazovich T, Harer J, Ozdemir O, Ellingwood P, McConley M. Automated vulnerability detection in source code using deep representation learning. In: Proc. of the 17th IEEE Int'l Conf. on Machine Learning and Applications (ICMLA). Orlando: IEEE, 2018. 757–762. [doi: 10.1109/ICMLA.2018.00120]

[15] Shar LK, Tan HBK. Predicting common Web application vulnerabilities from input validation and sanitization code patterns. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. Essen: IEEE, 2012. 310–313. [doi: 10.1145/2351676.2351733]

[16] Li Z, Zou DQ, Xu SH, Ou XY, Jin H, Wang SJ, Deng ZJ, Zhong YY. VulDeePecker: A deep learning-based system for vulnerability detection. arXiv:1801.01681, 2018.

[17] Lin GJ, Zhang J, Luo W, Pan L, Xiang Y. POSTER: Vulnerability discovery with function representation learning from unlabeled projects. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security. Dallas: ACM, 2017. 2539–2541. [doi: 10.1145/3133956.3138840]

[18] Lin GJ, Zhang J, Luo W, Pan L, Xiang Y, De Vel O, Montague P. Cross-project transfer representation learning for vulnerable function discovery. IEEE Trans. on Industrial Informatics, 2018, 14(7): 3289–3297. [doi: 10.1109/TII.2018.2821768]

[19] Duan X, Wu JZ, Ji SL, Rui ZQ, Luo TY, Yang MT, Wu YJ. VulSniper: Focus your attention to shoot fine-grained vulnerabilities. In: Proc. of the 28th Int'l Joint Conf. on Artificial Intelligence. Macao: AAAI, 2019. 4665–4671.

[20] Li Z, Zou DQ, Xu SH, Jin H, Zhu YW, Chen ZX. SySeVR: A framework for using deep learning to detect software vulnerabilities. IEEE Trans. on Dependable and Secure Computing, 2022, 19(4): 2244–2258. [doi: 10.1109/TDSC.2021.3051525]

[21] Zhou YQ, Liu SQ, Siow J, Du XN, Liu Y. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 10197–10207.

[22] Wu YM, Zou DQ, Dou SH, Yang W, Xu D, Jin H. VulCNN: An image-inspired scalable vulnerability detection system. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Pittsburgh: IEEE, 2022. 2365–2376. [doi: 10.1145/3510003.3510229]

[23] Zou DQ, Wang SJ, Xu SH, Li Z, Jin H. μVulDeePecker: A deep learning-based system for multiclass vulnerability detection. IEEE Trans. on Dependable and Secure Computing, 2021, 18(5): 2224–2236. [doi: 10.1109/TDSC.2019.2942930]

[24] Fan LL, Su T, Chen S, Meng GZ, Liu Y, Xu LH, Pu GG, Su ZD. Large-scale analysis of framework-specific exceptions in Android APPs. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Gothenburg: IEEE, 2018. 408–419. [doi: 10.1145/3180155.3180222]

[25] Tang CB, Chen S, Fan LL, Xu LH, Liu Y, Tang ZS, Dou L. A large-scale empirical study on industrial fake APPs. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Montreal: IEEE, 2019. 183–192. [doi: 10.1109/ICSE-SEIP.2019.00028]

[26] Evangelista JF. Cybersecurity vulnerability classification utilizing natural language processing methods [Ph.D. Thesis]. Washington: The George Washington University, 2021.

[27] Siewruk G, Mazurczyk W. Context-aware software vulnerability classification using machine learning. IEEE Access, 2021, 9: 88852–88867. [doi: 10.1109/ACCESS.2021.3075385]

[28] Wang Q, Li YZ, Wang Y, Ren JD. An automatic algorithm for software vulnerability classification based on CNN and GRU. Multimedia Tools and Applications, 2022, 81(5): 7103–7124. [doi: 10.1007/s11042-022-12049-1]

[29] Yuan X, Lin GJ, Tai YH, Zhang J. Deep neural embedding for software vulnerability discovery: Comparison and optimization. Security and Communication Networks, 2022, 2022: 5203217. [doi: 10.1155/2022/5203217]

[30] Hin D, Kan A, Chen HM, Babar MA. LineVD: Statement-level vulnerability detection using graph neural networks. arXiv:2203.05181,

2022.

[31] Yamaguchi F, Lottmann M, Rieck K. Generalized vulnerability extrapolation using abstract syntax trees. In: Proc. of the 28th Annual Computer Security Applications Conf. Orlando: ACM, 2012. 359–368. [doi: 10.1145/2420950.2421003]

[32] Neuhaus S, Zimmermann T, Holler C, Zeller A. Predicting vulnerable software components. In: Proc. of the 14th ACM Conf. on Computer and Communications Security. Alexandria: ACM, 2007. 529–540. [doi: 10.1145/1315245.1315311]

[33] Joern. 2019. https://github.com/ShiftLeftSecurity/

[34] Ferrante J, Ottenstein KJ, Warren JD. The program dependence graph and its use in optimization. ACM Trans. on Programming Languages and Systems, 1987, 9(3): 319–349. [doi: 10.1145/24039.24041]

[35] Sinha S, Harrold MJ, Rothermel G. System-dependence-graph-based slicing of programs with arbitrary interprocedural control flow. In: Proc. of the 21st Int'l Conf. on Software Engineering. Los Angeles: ACM, 1999. 432–441. [doi: 10.1145/302405.302675]

[36] Wang HT, Ye GX, Tang ZY, Tan SH, Huang SF, Fang DY, Feng YS, Bian LZ, Wang Z. Combining graph-based learning with automated data collection for code vulnerability detection. IEEE Trans. on Information Forensics and Security, 2020, 16: 1943–1958. [doi: 10.1109/TIFS.2020.3044773]

[37] Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. In: Proc. of the 27th Int'l Conf. on Neural Information Processing Systems. Lake Tahoe: Curran Associates Inc., 2013. 3111–3119.

[38] Sohn K. Improved deep metric learning with multi-class N-pair loss objective. In: Proc. of the 30th Int'l Conf. on Neural Information Processing Systems. Barcelona: Curran Associates Inc., 2016. 1857–1865.

[39] van den Oord A, Li YZ, Vinyals O. Representation learning with contrastive predictive coding. arXiv:1807.03748, 2019.

[40] Wu ZR, Xiong YJ, Yu SX, Lin DH. Unsupervised feature learning via non-parametric instance discrimination. In: Proc. of the 2018 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Salt Lake City: IEEE, 2018. 3733–3742. [doi: 10.1109/CVPR.2018.00393]

[41] Bachman P, Hjelm RD, Buchwalter W. Learning representations by maximizing mutual information across views. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 15535–15545.

[42] Hénaff OJ, Srinivas A, De Fauw J, Razavi A, Doersch C, Ali Eslami SM, van den Oord A. Data-efficient image recognition with contrastive predictive coding. In: Proc. of the 37th Int'l Conf. on Machine Learning. JMLR.org, 2020. 4182–4192.

[43] Baevski A, Zhou H, Mohamed A, Auli M. Wav2vec 2.0: A framework for self-supervised learning of speech representations. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2020. 12449–12460.

[44] Conneau A, Baevski A, Collobert R, Mohamed A, Auli M. Unsupervised cross-lingual representation learning for speech recognition. arXiv:2006.13979, 2020.

[45] Tian YL, Krishnan D, Isola P. Contrastive multiview coding. In: Proc. of the 16th European Conf. on Computer Vision. Glasgow: Springer, 2020. 776–794. [doi: 10.1007/978-3-030-58621-8_45]

[46] Hjelm RD, Fedorov A, Lavoie-Marchildon S, Grewal K, Bachman P, Trischler A, Bengio Y. Learning deep representations by mutual information estimation and maximization. arXiv:1808.06670, 2019.

[47] Han TD, Xie WD, Zisserman A. Video representation learning by dense predictive coding. In: Proc. of the 2019 IEEE/CVF Int'l Conf. on Computer Vision Workshop. Seoul: IEEE, 2019. 1483–1492. [doi: 10.1109/ICCVW.2019.00186]

[48] He KM, Fan HQ, Wu YX, Xie SN, Girshick R. Momentum contrast for unsupervised visual representation learning. In: Proc. of the 2020 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Seattle: IEEE, 2020. 9726–9735. [doi: 10.1109/CVPR42600.2020.00975]

[49] Misra I, van der Maaten L. Self-supervised learning of pretext-invariant representations. In: Proc. of the 2020 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Seattle: IEEE, 2020. 6706–6716. [doi: 10.1109/CVPR42600.2020.00674]

[50] Chen T, Kornblith S, Norouzi M, Hinton G. A simple framework for contrastive learning of visual representations. In: Proc. of the 37th Int'l Conf. on Machine Learning. JMLR.org, 2020. 1597–1607.

[51] Khosla P, Teterwak P, Wang C, Sarna A, Tian YL, Isola P, Maschinot A, Liu C, Krishnan D. Supervised contrastive learning. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2020. 18661–18673.

[52] Wu YM, Dou SH, Zou DQ, Yang W, Qiang WZ, Jin H. Obfuscation-resilient Android malware analysis based on contrastive learning. arXiv:2107.03799, 2022.

[53] Gunel B, Du JF, Conneau A, Stoyanov V. Supervised contrastive learning for pre-trained language model fine-tuning. arXiv:2011.01403, 2021.

[54] Lin GJ, Zhang J, Luo W, Pan L, De Vel O, Montague P, Xiang Y. Software vulnerability discovery via learning multi-domain knowledge bases. IEEE Trans. on Dependable and Secure Computing, 2021, 18(5): 2469–2485. [doi: 10.1109/TDSC.2019.2954088]

[55] Lin GJ, Wen S, Han QL, Zhang J, Xiang Y. Software vulnerability detection using deep neural networks: A survey. Proc. of the IEEE,

2020, 108(10): 1825–1848. [doi: 10.1109/JPROC.2020.2993293]

[56] Zhou P, Shi W, Tian J, Qi ZY, Li BC, Hao HW, Xu B. Attention-based bidirectional long short-term memory networks for relation classification. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (Vol. 2: Short Papers). Berlin: Association for Computational Linguistics, 2016. 207–212. [doi: 10.18653/v1/P16-2034]

[57] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735–1780. [doi: 10.1162/neco.1997.9.8.1735]

[58] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473, 2016.

[59] nvd. 2019. https://nvd.nist.gov/

[60] sard. 2019. https://samate.nist.gov/SRD/

[61] ElRafey A, Wojtusiak J. Recent advances in scaling-down sampling methods in machine learning. WIREs Computational Statistics, 2017, 9(6): e1414. [doi: 10.1002/wics.1414]

[62] Sun AX, Lim EP, Liu Y. On strategies for imbalanced text classification using SVM: A comparative study. Decision Support Systems, 2009, 48(1): 191–201. [doi: 10.1016/j.dss.2009.07.011]

[63] Ndiaye E, Le T, Fercoq O, Salmon J, Takeuchi I. Safe grid search with optimal complexity. In: Proc. of the 36th Int'l Conf. on Machine Learning. Long Beach: PMLR, 2019. 4771–4780.

[64] Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv:1412.6980, 2017.

[65] Meta-learning: Learning to learn fast. 2018. https://lilianweng.github.io/posts/2018-11-30-meta-learning/