一种基于大语言模型的多来源漏洞影响库识别方法

徐近伟[1),2)] 周 鑫[1),2)] 杨焱景[1),2)] 李晓康[1),2)] 余灏沣[1),2)]
杨岚心[1),2)] 张 贺[1),2)] 吴永航[3)]

[1)](南京大学软件学院 南京 210093)
[2)](计算机软件新技术国家重点实验室(南京大学) 南京 210093)
[3)](北京兴云数科技术有限公司 北京 100176)

摘 要 现代软件以软件供应链的模式进行开发,需要使用大量的第三方组件和软件库。第三方软件库中可能存在安全漏洞,危害下游依赖该软件库的用户,因此确保依赖软件库的安全对于确保企业的软件安全至关重要。NVD(National Vulnerability Database)等漏洞信息网站会定期发布软件漏洞报告,帮助安全从业者和用户及时了解软件的安全状况。但这些漏洞报告中并不会清晰准确地提供漏洞影响的软件库,需要安全专家手动对其进行分析,识别出受影响的软件库,这一过程费时费力且容易出错。因而自动从漏洞报告准确识别漏洞影响库可以更加快速高效地为软件安全提供保障。目前的漏洞影响库识别方法只聚焦于英文的NVD漏洞报告,而忽视了其他漏洞信息来源(如中文漏洞报告),导致准确率不高。同时它们也没有关注自动化识别方法对于不同包管理器中漏洞软件库(如Maven, PyPI等)的识别效果是否存在较大差别。因此本文提出了一个基于大语言模型的多来源漏洞影响库识别方法,该方法首先通过中文/英文漏洞报告相互补充进行输入增强,然后根据输入信息特征设计相应的Alpaca训练模板,并使用指令微调技术对大语言模型进行任务调优,使其可以自动地从中文/英文漏洞报告中识别漏洞的影响库,最后使用基于文本相似度的方法来消除大语言模型的幻觉问题(输出的漏洞软件库并不存在),并在不同的包管理器上对方法的效果进行评估。实验使用了9260份中文/英文漏洞报告作为数据集对本文的方法进行评估,结果表明本文提出的方法在中文和英文漏洞报告上都有更好的识别效果,相较于基线方法在中文和英文报告的平均$F1$分数上分别提升了4%和8%,且在中/英文漏洞报告相互补充的情况下效果最优,平均$F1$分数达到0.85,虽然在不同编程语言软件库上的识别效果存在差异,但本文方法在PyPI、Composer、NPM、Golang等多数包管理器上的识别效果优异,平均$F1$分数均达到0.85,在Maven上也取得了最优表现,平均$F1$分数达到0.71。基于实验结果,本文总结了在漏洞影响库识别任务上使用大语言模型的经验,并给出未来可能的改进方向。最后,本文公开了一个包含9260份中文/英文漏洞报告及其漏洞影响库标注的数据集,以支持后续研究更好地开展工作。

关键词 漏洞影响库;漏洞报告;大语言模型;软件供应链
中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2026.00261

收稿日期:2025-02-07;在线发布日期:2025-07-14。本课题得到江苏省自然科学基金(BK20241195)、国家自然科学基金(62202219,62302210)、工业和信息化部软件融合应用与测试验证重点实验室2025年度开放课题(RFT20250301)、新疆两区科技发展计划项目(2024LQ03004)、北京经济开发区"白菜心工程"资助。**徐近伟**,博士研究生,中国计算机学会(CCF)会员,主要研究领域为软件供应链安全、代码评审和软件过程安全。E-mail: jinwei_xu@smail.nju.edu.cn。**周 鑫**(通信作者),博士,助理研究员,中国计算机学会(CCF)会员,主要研究领域为软件供应链安全、漏洞检测、DevSecOps、经验软件工程和自然语言处理。E-mail: zhouxin@nju.edu.cn。**杨焱景**,博士研究生,中国计算机学会(CCF)会员,主要研究领域为AI鲁棒性、源码安全和软件供应链安全。**李晓康**,硕士,中国计算机学会(CCF)会员,主要研究领域为软件供应链安全和经验软件工程。**余灏沣**,本科,工程师,主要研究领域为软件供应链安全和大语言模型。**杨岚心**,博士,助理研究员,中国计算机学会(CCF)会员,主要研究领域为代码评审、经验软件工程和软件供应链安全。**张贺**,博士,教授,中国计算机学会(CCF)会员,主要研究领域为软件工程、开发运维一体化、软件供应链安全、经验软件工程、软件安全和区块链安全。**吴永航**,硕士,工程师,主要研究领域为智能软件工程和大语言模型研发。

An Approach for Identifying Affected Libraries from Multiple Sources Based on Large Language Models

XU Jin-Wei[1),2)] ZHOU Xin[1),2)] YANG Yan-Jing[1),2)] LI Xiao-Kang[1),2)] YU Hao-Feng[1),2)]
YANG Lan-Xin[1),2)] ZHANG He[1),2)] WU Yong-Hang[3)]

[1)](Software Institute, Nanjing University, Nanjing 210093)

[2)](State Key Laboratory of Novel Software Technology (Nanjing University), Nanjing 210093)

[3)](Beijing Xingyun Digital Technology Co., Ltd., Beijing 100176)

Abstract Modern software is developed in a software supply chain paradigm, which requires the use of a large number of third-party components and software libraries. There could be security vulnerabilities in third-party software libraries, which endanger downstream users who rely on these software libraries. Therefore, ensuring the security of the dependent software libraries is crucial to ensuring the software security of the enterprise. NVD(National Vulnerability Database) and other vulnerability information websites regularly release some software vulnerability reports to help the security practitioners and users understand the security status of the software in a timely manner. However, these vulnerability reports do not provide clear and accurate information about the software libraries affected by vulnerabilities. Security experts need to manually analyze them and identify the affected software libraries. This process is time-consuming, labor-intensive and error-prone. Therefore, automatically and accurately identifying the affected libraries from vulnerability reports provides faster and more efficient protection for software security. Currently, the approaches for identifying affected libraries only focus on English NVD vulnerability reports, while ignoring other sources of vulnerability information (such as Chinese vulnerability reports), resulting in low accuracy. They also ignore whether there is a significant difference in the performance of identifying affected libraries of different packaging managers (e. g., Maven and PyPI). Therefore, this article proposes an approach for identifying affected libraries from multiple sources based on large language models. This approach first enhances the input by mutual complementation of Chinese and English vulnerability reports, then designs the corresponding Alpaca training template according to the characteristics of input data, and uses instruction tuning techniques to fine-tune the large language model on the specific task, enabling it to automatically identify the affected libraries from Chinese/English vulnerability reports. Finally, a text similarity-based method is used to eliminate the hallucinations in the output of the large language model (the output affected library does not exist), and the effectiveness of the approach is evaluated across different packaging managers. The experiment uses 9260 Chinese and English vulnerability reports as a dataset to evaluate the proposed approach. The results show that the proposed approach has better performance in both Chinese and English vulnerability reports. Compared with the baseline method, the average $F1$ scores of Chinese and English reports are improved by 4% and 8%, respectively. The approach achieves the best performance when Chinese/English vulnerability reports complement each other, with an average $F1$ score of 0.85. Although there are differences in performance of identifying affected libraries of different packaging managers, the performance of the proposed approach is favorable on most packaging managers such as PyPI, Composer, NPM, Golang, etc., with the average $F1$ score all reaching 0.85. It also achieves the best performance on Maven, with an average $F1$ score of 0.71. Based on the experimental results, this article summarizes the experience of using large language models on the task of affected library identification and provides possible future improvement directions. Finally, this article discloses a dataset containing 9260 Chinese and

English vulnerability reports annotated with their affected libraries, aiming to facilitate future research efforts.

Keywords vulnerability affected library; vulnerability report; large language model; software supply chain

1 引 言

现代软件开发使用大量的第三方软件库来避免重复劳动、降低开发成本、提高开发灵活性[1-3]，进而提升自身的市场竞争力。作为重要的可复用资源[4]，第三方软件库可以为开发者提供常用功能[5]，避免重复造轮子[6]，使开发者聚焦业务场景，大大提升开发效率。已成为现代软件开发不可或缺的重要部分[1]，这种严重依赖于外部组件的开发模式被称为软件供应链模式。超过66%的企业在软件开发中使用开源第三方组件[7]，一个单独的软件便会依赖几十个不同的第三方库[8]，由于第三方库也会依赖其他的第三方库，这使得软件由于隐式的依赖关系间接依赖了更多的第三方库[9-12]。尽管带来诸多优势，第三方库也扩大了软件的攻击面，增加了外部风险[13-15]。近年来，对于开源软件库的攻击不断增加[16]，例如影响范围巨大的Log4j攻击[17]。若软件依赖的第三方库存在漏洞，则其也将面临威胁[18]，因此确保软件依赖的第三方库的安全对于维护企业的软件安全至关重要[19-21]。

NVD等安全披露平台会定期发布漏洞报告[22]，报告已知的公开漏洞（如CVE漏洞），漏洞报告中常含有漏洞的概要描述、漏洞的相关参考链接和漏洞影响的平台等信息，但却并不会清晰准确地提供漏洞影响的软件库[23]。虽然漏洞影响的平台指出了漏洞可能影响的系统或软件，但该信息并不等同于漏洞影响的软件库[24]。例如CVE-2015-7318的漏洞报告中提供的漏洞影响平台是Plone，但该漏洞影响的软件库是Plone管理系统中的一个名为Zope的Python库。此外，NVD报告常存在信息缺失或不准确[24-25]。例如CVE-2023-576的漏洞报告中提到漏洞影响软件是glassfish，但更具体地该漏洞影响的软件库是org. glassfish. main. orb分组里的orb-connector库。安全专家需要仔细地检查漏洞报告，并结合自身知识分析出漏洞影响的软件库，这一过程耗时费力[26]，导致安全防护延迟[5,27]（从漏洞被发布到开发者发现并更新依赖的漏洞库）。因此，研究者尝试自动识别漏洞影响的软件库，以提高相应效率。

目前从漏洞报告识别漏洞影响库的自动化方法基于XML（eXtreme Multi-label Learning）[23-24,26]。基于XML的方法先收集好漏洞报告和软件库列表，然后通过对XML模型进行监督训练，使模型学习从漏洞报告到对应软件库的映射关系。但目前漏洞影响库识别的信息来源只聚焦于NVD英文漏洞报告，忽略了其他安全信息平台，这种信息来源的不充分会导致识别准确率不高，安全防护不够全面立体。CNNVD（China National Vulnerability Database of Information Security）是由中国政府运营的权威漏洞搜集与发布平台[28]，每年会发布数万份CNNVD中文漏洞报告（包含很多高危漏洞），将CNNVD中文报告作为信息输入来源与NVD英文报告相互补充，可以提高识别准确率，增大安全防护范围，例如对于漏洞CVE-2022-4147，对应的NVD报告只提到它影响了Quarkus，但更具体地它影响的是Quarkus的quarkus-vertx-http组件，而CNNVD报告则补充了这点。此外，不同包管理器中软件库的命名方式、层次结构都存在差异，例如Maven库的命名遵循groupID：artifactID：version，而PyPI库则通常只有包名。但现有方法未对其加以区分，难以确定在不同包管理器上的识别效果。同时，他们使用的漏洞报告数据集截止到2019年，由于软件库和漏洞报告不断更新，难以评估其在最新漏洞报告上的识别效果。

针对上述问题，本文提出了一个基于大语言模型的多来源漏洞影响库识别方法。大语言模型对于自然语言文本有着很强的理解分析能力，尤其在处理漏洞报告这种以自然语言描述为主的任务中具有独特的优势。“通义千问”大模型在多个评估中英文语言理解能力的数据集上全面领先其他同规模的开源模型[29]，且在信息抽取任务上表现出良好的性能[30-31]。基于此，本文使用“通义千问”大模型从漏洞报告中识别漏洞的影响库。但大语言模型是在通用语料上进行预训练，直接应用于具体任务效果欠佳。因此，本文首先构建了一个含有中文/英文漏洞报告及其影响软件库的高质量数据集，接着根据具

体数据特征并参照Alpaca模板格式分别构建了中文和英文的训练模板，然后使用LoRA（Low-Rank Adaption）技术对大语言模型在漏洞报告数据集上进行局部参数微调。微调可以使大模型从大量预训练知识中排除干扰内容，更准确地利用与任务相关的知识，提升大模型在漏洞影响库识别任务上的效果，还可以减少输出结果中的无关内容，方便后续的自动化处理。由于大语言模型存在幻觉问题，其输出的影响库可能在现实场景并不存在，本文使用了多种基于文本相似度的算法（基于编辑距离、基于token、基于序列）进行幻觉消除，将大模型的输出结果与真实的软件库进行比较，并选择相似度最高的软件库作为最终的漏洞影响库识别结果。本文分别使用了9260份中文/英文漏洞报告进行试验，并评估了方法对于不同包管理器中漏洞影响库的识别效果。实验结果表明，本文方法在中文/英文漏洞报告上表现出色，相较于基线方法平均$F1$分数分别提升了4%和8%，在中/英文报告相互补充的情况下效果最优，平均$F1$分数达到0.85，虽然在不同包管理器上的识别效果存在一定差异，但在PyPI、Composer、NPM、Golang等多数包管理器上的识别效果优异，当考虑排名前3的结果时，本文方法在上述4个包管理器上的$F1$分数均超过0.85。

本文的主要贡献包括以下内容。

（1）相较于现有工作关注英文NVD漏洞报告，本文首次关注从更多漏洞报告来源，即中文漏洞报告对漏洞影响库进行识别，拓展了该领域的输入来源，提供了从更多数据源进行识别的新思路，多数据源相互补充提供了更全面的软件供应链安全防护。

（2）提出了一种基于大语言模型的多来源漏洞影响库识别方法，并使用基于文本相似度的方法消除大语言模型的幻觉问题，该方法在处理中文/英文漏洞报告以及5个主流包管理器上的识别效果均优于现有方法。

（3）提供了一个包含9260份中/英文漏洞报告及其影响库的数据集①，涵盖多种流行包管理的漏洞影响库，可以帮助后续相关研究更好地开展工作。

本文第2节介绍背景和相关工作；第3节介绍研究动机和问题定义；第4节介绍本文提出的自动化识别方法；第5节报告实验设计与结果分析；第6节讨论大语言模型的使用经验和未来改进方向；第7节讨论实验的效度威胁；最后第8节总结全文并提出未来工作方向。

2 背景和相关工作

2.1 背 景

NVD等安全披露平台会定期发布漏洞报告，披露软件漏洞，但漏洞报告通常未能清晰精准地提供漏洞影响范围。确定漏洞的影响范围（漏洞影响的软件库）对于保障软件供应链安全至关重要，许多软件静态分析工具（例如Veracode[32]）和软件成分分析工具（例如Scantisit[33]）都依赖于及时更新的漏洞库列表，它们首先会对待测软件进行扫描，分析出软件依赖的第三方库列表，然后将其与自身构建的漏洞库列表进行比对，判断出软件依赖的第三方库中哪些受到漏洞的威胁。从漏洞报告确定漏洞的影响库可以帮助安全工具不断更新它们的漏洞库列表，然而由于漏洞报告并不总是明确提供漏洞影响库的相关信息，而且提供的漏洞影响库信息也不准确完整，甚至可能出现错误，导致人工检查漏洞报告以识别漏洞影响库既费时费力，又容易出错。因此很多研究者致力于提供自动化的漏洞影响库识别方法，来降低人工成本，加快识别速度，提供更加及时的风险警报。目前从漏洞报告识别漏洞影响库的任务被定义为XML任务[23-24,26]，即从众多的软件库列表中找出该漏洞影响的软件库，实现的自动化方法也都基于各种XML模型。研究者首先从NVD收集漏洞报告，从中提取漏洞描述、参考链接等相关信息，然后对该信息进行一系列预处理操作（例如分词、去除停用词等）后将其作为XML模型的输入。接着对一批漏洞报告的影响库（标签）进行人工标注，然后使用这批标注好的数据训练XML模型，使模型学习漏洞报告和漏洞影响库间的关联关系。最后使用训练好的XML模型处理新的漏洞报告，并从软件库列表中返回得分最高的库作为该漏洞报告的影响库。

2.2 大语言模型

语言模型（Language Models）[34-36]是能够理解和生成人类语言的计算模型，而大语言模型（Large Language Models）[37-39]是具有大量参数和卓越学习能力的高级语言模型。其背后的核心模块是Transformer[40]中的自注意力模块，它是语言建模任务的基本构建块。Transformers以其高效处理顺序数据的能力推动了自然语言处理领域的变革，可以

① https://figshare.com/s/cbd35613e0e524614e1e

并行捕获文本中的长距离依赖关系。大语言模型对于处理自然语言形式的输入尤为擅长，而且可以进行上下文学习[41]，根据给定的上下文或提示生成文本。大语言模型已经在学术界和工业界引起了广泛关注[42-43]，并被证明在很多下游任务上有出色的表现[44-46]，例如问答系统、信息提取、文本总结等等。

尽管大语言模型在很多任务上展现出巨大的潜力，但仍面临诸多挑战。其中一个主要问题就是模型的训练目标与用户目标之间不匹配：大语言模型在大型语料库上进行预训练，它的训练目标是最小化语料库中上下文单词预测的误差，但用户则更希望模型可以有效且安全地遵循他们的指示，完成要求的任务[47-50]。指令调优(Instruction tuning)是解决这一问题的关键方法，它可以增强大语言模型对于指示遵循的能力和可控性[51]。指令调优使用＜指令，输出＞对来训练大语言模型，其中指令是人根据任务构建而得的模型需要遵循的指示，输出则是模型接受前面的指令输入后期望得到的输出结果。指令调优可以弥合大语言模型预测下一个词的目标与遵循用户指令目标之间的差距，还可以提升模型行为的可控性和可预测性。直接将原生大模型应用于具体任务不仅输出格式不统一，不便于后续提取，而且模型缺乏任务相关的领域知识，对任务要求不敏感，导致在具体任务上表现不佳。在领域数据集上对大模型进行指令调优一方面可以规范化模型输出，另一方面可以为模型补充任务相关的领域知识，使大模型专注于具体任务，激活大模型完成具体任务的能力，提升大语言模型在具体任务上的表现。

大语言模型还存在幻觉问题，可能输出一些看似合理但其实错误的答案[52]。大语言模型的幻觉问题大致可以分为3类：(1)与输入矛盾[53]：大语言模型生成的内容与用户的输入相矛盾；(2)与上下文矛盾[54]：大语言模型生成的内容与之前自身生成的内容相矛盾；(3)与事实矛盾[55-57]：大语言模型生成的内容与已知的事实常识不相符。大语言模型的幻觉问题大大降低了其在真实场景下的可靠性[58]。

2.3　幻觉消除方法

由于大语言模型的输出存在幻觉问题，需要对大语言模型的输出结果进行后处理，来消除其可能出现的幻觉。常见的幻觉消除方法包括在提示工程中引入思维链[59]和对模型生成的内容进行检索增强[60]。在漏洞影响库识别的任务中，大语言模型主要的幻觉问题是输出的软件库可能看似合理，但却与真实的漏洞影响库存在一定偏差，输出的库可能在现实中并不存在。为了消除该幻觉问题，需要对模型的生成内容进行检索增强。具体地，本文使用文本相似度算法将大模型输出的结果与真实软件库进行比较，使用相似度最高的软件库作为最终结果。

常见的文本相似度算法大致可以分为3类：(1)基于编辑的相似度算法；(2)基于token的相似度算法；(3)基于序列的相似度算法。

基于编辑：基于编辑的算法也被称为基于编辑距离的算法，通过度量将一个文本转换成另一个文本所需的最少插入、删除、替换操作次数来衡量二者相似度，代表性的算法有Levenshtein相似度[61]。

基于token：基于token的算法侧重于文本的组成，先将文本拆分为单词或token，然后基于token比较文本间相似度，代表性的算法有cosine相似度[62]。

基于序列：基于序列的算法侧重于比较整个文本序列，而不关注文本的具体组成，代表性的算法有最长公共子字符串相似度[63]。

2.4　漏洞影响库识别

目前对从漏洞报告识别漏洞影响库的问题定义主要分为两类，一类是将其定义为基本的XML任务[23-24]，另一类则是将其定义为零样本XML(Zero-shot XML[64])任务[26]。基本的XML任务首先需要构造一个固定的标签集合，然后将输入的样本映射到该标签集合上，一个样本对应一个或多个标签。对于漏洞影响库识别而言，首先需要构造一个由软件库组成的标签集合，然后训练模型将漏洞报告映射到标签集合中的一个或多个软件库，作为其对应的漏洞影响库。但目前的一些研究没有考虑漏洞报告的时间顺序，将未来漏洞报告放入训练集，将未来漏洞报告对应的漏洞影响库放入训练时的标签集合，这与现实情况不相符。正确的方法是应该使用历史上的漏洞报告及其影响库作为训练集合训练模型，然后对未来的漏洞报告进行识别。零样本XML任务旨在解决将输入样本映射到训练时未见过的标签(零样本标签)上的问题，在训练时构造的标签集合并不包含所有的标签，因而在预测的时候既需要考虑将样本映射到训练中已见过的标签上，又需要考虑将样本映射到未见过的标签上。对于漏洞影响库识别而言，在构造标签集合时需要考虑漏洞报告和漏洞影响库的时间顺序，使用一个时间点对漏洞报告数据集进行划分，使用该时间点之前的漏洞报告对模型进行训练，训练时的标签集合只包含训练集中漏洞报告对应的影响库，使用该时间点之后的漏洞报告对模型进行测试，测试

时的标签集合包含了训练集和测试集中漏洞报告对应的影响库，因而会含有训练时未见过的标签。零样本XML任务中模型在训练时并不会见到未来的漏洞报告与其影响库之间的映射关系，与现实场景更加贴近，能够更好地评估模型在真实场景下的漏洞影响库识别效果。

由于目前从漏洞报告识别漏洞影响库的问题都被定义为XML问题，即基本XML问题和零样本XML问题，因此现有的自动化方法都是基于XML模型进行实现。Chen等人[24]提出了一个基于FastXML的自动化漏洞影响库识别方法，首次将XML模型应用于该任务，先将NVD漏洞报告中的漏洞描述、参考链接和漏洞影响平台信息经过分词、过滤等预处理操作后作为模型的输入，然后训练FastXML模型，使其自动将漏洞报告映射至标签集合中的一个或多个软件库，以实现漏洞影响库的自动识别。该方法的平均$F1$分数在考虑排名前三的识别结果上可以达到0.51，这对于极端多分类任务已较为可观。Haryono等人[23]遵循Chen等人的数据提取流程，采用相同的输入评估了多类XML模型，包括一对多（One-vs-all）类的DiSMEC，基于树（Tree-based）类的FastXML，ExtremeText，Parabel，Bonsai，以及基于深度学习（Deep Learning）类的XML-CNN和LightXML。实验结果表明，基于Parabel模型的方法相较于之前基于FastXML的方法效率提高超过10倍，且$F1$分数也提高了7%，而基于LightXML模型的方法在识别准确率上提升最多，$F1$分数提高了10%。Lyu等人则认为应该考虑漏洞报告的时间顺序，即XML模型在训练时不应见到未来的漏洞报告和漏洞影响库（标签），而之前的研究并没有考虑这一点，选择的FastXML和LightXML模型都只关注训练中见过的标签，对于新标签（训练中未见过）的识别能力很弱。因此Lyu等人[26]选择使用ZestXML模型构建自动化漏洞影响库识别方法Chronos。ZestXML模型适用于解决Zero-short learning问题，该模型不仅关注训练数据中的已知标签，还能够处理新的标签，当样本输入与新标签存在较强关联性时，模型能够将其映射至该新标签，适用于解决从漏洞报告识别新漏洞影响库的问题。实验结果表明，Chronos在识别新漏洞影响库上的$F1$分数可以达到0.72，而基于LightXML模型的方法则无法识别出任何新库。

目前从漏洞报告识别漏洞影响库的研究均基于同一数据集，该数据集由安全专家手工检查得到，包含7,696份NVD漏洞报告及其对应的漏洞影响库。但是该数据集只关注英文NVD漏洞报告，而忽略掉其他有价值的信息来源，如中文CNNVD漏洞报告，导致安全防护范围不够全面。同时该数据集收集的漏洞报告只截止到2019年，由于新的软件库持续涌现，漏洞报告也不断更新，亟需评估在最新漏洞报告上的识别效果。此外，该数据集未标注出漏洞影响库对应的包管理器，由于不同编程语言包管理器在命名方式和层次结构上存在差别，因而需要评估对于不同包管理器中漏洞影响库的识别效果。

除了从漏洞报告识别漏洞影响库的通用方法外，还有一些针对特定生态系统的漏洞影响库识别方法。Chen等人[65]提出自动化识别方法VulLibMiner，从特定生态系统中收集软件库的描述信息，然后通过比较漏洞报告与软件库描述之间的相关性来识别漏洞影响库。Chen等人[66]提出自动化识别方法VulLibGen，使用软件库描述和VulLibMiner的top-k个预测结果来提示大模型识别漏洞的影响库，在PyPI等3个生态系统取得了最优的识别效果。Wu等人[67]提出自动化识别方法HOLMES，从GitHub仓库和漏洞报告中收集了产品名、文件名和代码类名等多类生态信息，然后通过逐一比较该生态中的所有库与收集信息之间的相关性来识别漏洞影响库，在Maven等4个生态系统中取得了最优的识别效果。这些针对特定生态系统的漏洞影响库识别方法利用了更多从生态系统中收集的辅助信息，因而在相应的生态系统中识别效果更优。

3 动机和问题定义

3.1 动 机

图1是CVE-2019-16543的中文漏洞报告，漏洞描述中提及了CloudBees Jenkins软件，并指出其使用的Spira Importer插件存在安全漏洞，然而该漏洞影响的第三方库却是Jenkins里的inflectra-spira-integration。事实上，Spira Importer插件并不是某一个具体的软件库，而是泛指一系列用于将数据导入Spira产品的软件/插件集合。inflectra-spira-integration则是Jenkins里的一个具体插件，用于将Jenkins持续集成构建服务器与Spira产品进行集成，它的功能就是将Jenkins数据导入Spira产品（如SpiraTest、SpiraPlan或SpiraTeam），因而也属于Spira Importer插件，这一点可以通过其README文件的标题为‘Spira Importer’加以验证。这个案例

说明虽然漏洞报告会提供一些漏洞和漏洞影响库的相关信息，但提供的信息并不一定准确详细。图2是CVE-2020-26290的英文漏洞包报告，在漏洞描述中提到了该漏洞影响的软件库Dex，但并没有说明该软件库的拥有者(dexidp)，References里则是提到了漏洞影响库“github. com/dexidp/dex”，但在CPE配置信息里又指出漏洞的影响平台是“linuxfoundation:dex”，前后存在不一致，而事实上该漏洞影响的软件库不仅有“github. com/dexidp/dex”，还有“github. com/russellhaering/goxmldsig”，虽然漏洞描述里提到了底层Go库的XML编码存在问题，但并没有明确指出其影响的软件库是“goxmldsig”，需要进一步分析才能确认。这个案例一方面说明漏洞报告中提供的信息可能前后不一致，容易造成干扰，另一方面说明漏洞报告中有些信息并不直接指出漏洞的影响库，只是间接地提供了一些证据。由于漏洞报告提供的影响库信息并不直接准确、全面详细，导致识别难度较高，因而单一使用中文或英文漏洞报告都难以保证漏洞影响库的精准识别，而将中文报告和英文报告相结合，通过多源数据的相互验证与补充，可以更好地提升漏洞影响库识别的准确性和全面性。

CNVD漏洞报告
漏洞ID：CNNVD-201911-1240 名称： CloudBees Jenkins Spira Importer Plugin 安全漏洞 发布时间：2019-11-21 漏洞描述：CloudBees Jenkins(Hudson Labs)是美国CloudBees公司的一套基于Java开发的持续集成工具。该产品主要用于监控持续的软件版本发布/测试项目和一些定时执行的任务。Spira Importer Plugin是使用在其中的一个SpiraPlan项目管理集成插件。CloudBees Jenkins Spira Importer Plugin 3.2.2及之前版本中存在安全漏洞，该漏洞源于程序将未加密的凭证存储在全局配置文件中。攻击者可利用该漏洞查看这些凭证。 CVE-ID：CVE-2019-16543 严重性：中危 漏洞类型：其他
漏洞影响库 inflectra-spira-integration

图1　CNNVD中文漏洞报告

NVD Vulnerability Report
CVE-ID: CVE-2020-26290 Description: Dex is a federated OpenID Connect provider written in Go. In Dex before version 2.27.0 there is a critical set of vulnerabilities which impacts users leveraging the SAML connector. The vulnerabilities enables potential signature bypass due to issues with XML encoding in the underlying Go library. The vulnerabilities have been addressed in version 2.27.0 by using the xml-roundtrip-validator from Mattermost (see related references). References: https: // github.com/dexidp/dex/commit/324b1c886b407594196113a3dbddebe38eecd4e8;https: // github.com/dexidp/dex/releases/tag/v2.27.0 https: // github.com/dexidp/dex/security/advisories/GHSA-m9hp-7r99-94h5 https: // github.com/mattermost/xml-roundtrip-validator/blob/master/advisories/unstable-attributes.md https: // github.com/mattermost/xml-roundtrip-validator/blob/master/advisories/unstable-directives.md https: // github.com/mattermost/xml-roundtrip-validator/blob/master/advisories/unstable-elements.md https: // github.com/russellhaering/goxmldsig/security/advisories/GHSA-q547-gmf8-8jr7 https: // mattermost.com/blog/coordinated-disclosure-go-xml-vulnerabilities/ CPE Configurations: cpe:2.3:a:linuxfoundation:dex:*:*:*:*:*:*:*:* Weakness Enumeration: Improper Verification of Cryptographic Signature
Affected Library ['github.com/russellhaering/goxmldsig','github.com/dexidp/dex']

图2　NVD英文漏洞报告

3.2　问题定义

从漏洞报告识别漏洞影响库的问题形式化定义如式(1)所示。

$$M(v)=l \quad v\in V,\ l\subseteq L \tag{1}$$

其中，V表示漏洞报告的集合，L表示软件库的集合，v是V中的元素，表示一份漏洞报告，l是L中元素构成的集合，表示一个或多个软件库，M是一种映射关系，将输入的漏洞报告映射一个或多个软件库上，表示从漏洞报告识别漏洞影响的软件库。模型在标签数据集上训练学习的是映射关系M，训练完成后，模型可以使用M接受新的漏洞报告v并输出它对应的影响库集合l。

4 漏洞影响库识别方法

本文提出的基于大语言模型的多来源漏洞影响库识识别方法主要包含3个部分：从中/英文漏洞报告进行信息抽取，构建中/英文训练模板然后对大语言模型进行指令调优，基于文本相似度算法消除大语言模型产生的幻觉问题。方法概览如图3所示。

4.1 漏洞报告数据抽取

本文从CNNVD收集中文漏洞报告，从NVD收集英文漏洞报告，然后分别从漏洞报告抽取与漏洞影响库相关的信息项作为后续模型的输入。同时从学术文献和安全网站收集经安全研究者或安全专家评审过的CVE漏洞影响的软件库。

本文从中文漏洞报告抽取漏洞标题、漏洞描述和漏洞类型三项信息。

（1）漏洞标题：简要描述漏洞事件的名称和大致的影响范围/组件。

（2）漏洞描述：描述了漏洞产生的原因、被利用的方式、可能造成的危害、影响的组件、组件的背景知识、如何升级修复等信息，每份漏洞报告中描述的信息并不固定。

（3）漏洞类型：描述了漏洞原因的大致类型。

漏洞标题中会包含有漏洞影响的厂商或组件信息，漏洞描述则会通过大量描述直接或间接地提供与漏洞影响库相关的信息，而漏洞类型则可能会暗示漏洞影响库的功能，例如“日志信息泄露”类型的漏洞更可能影响与日志记录和数据库驱动相关的库，因此，本文从中文漏洞报告中提取上述三项信息。漏洞的发布时间和严重性与漏洞影响库并不相关，为了减少噪声干扰，本文在大模型的输入内容中剔除了这两项信息。

本文从英文漏洞报告抽取漏洞描述、参考链接和CPE配置三项信息：

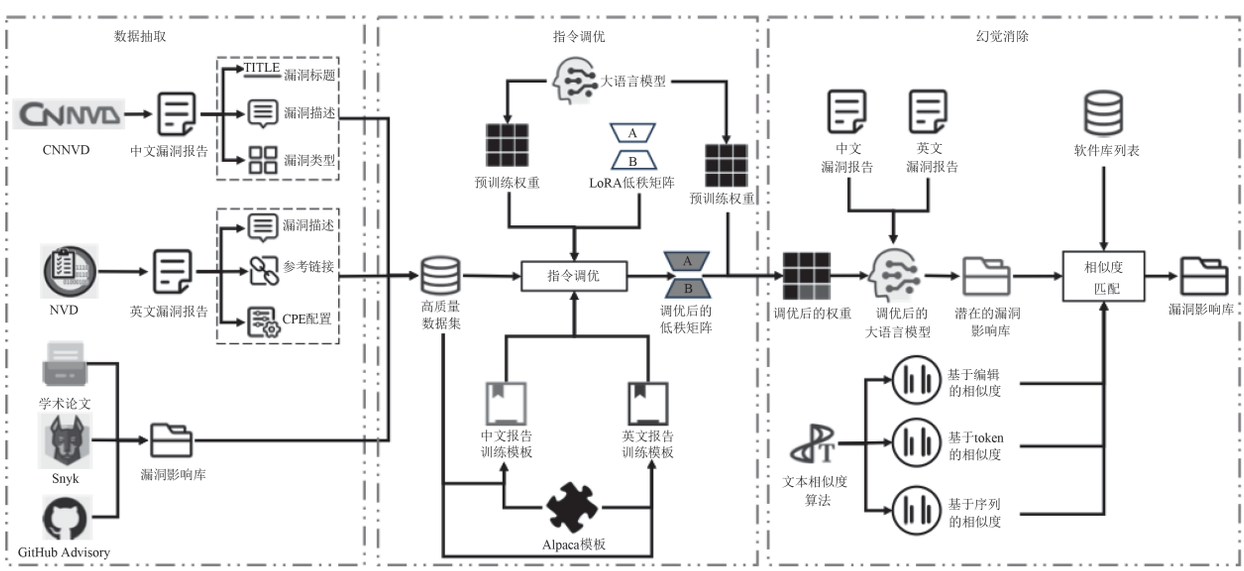

图3 提出方法的概览

（1）漏洞描述：描述了漏洞产生的原因、被利用的方式、可能造成的危害、影响的组件、如何升级修复等信息，每份漏洞报告中描述的信息并不固定。

（2）参考链接：描述了与漏洞相关的新闻、博客、修复提交等的URL链接。

（3）CPE配置信息：描述了漏洞可能影响的平台、系统、软件等。

漏洞描述、参考链接和CPE配置三项信息在之前的研究[23-24,26]中已被证明对于识别漏洞的影响库很有价值，因此本文从英文漏洞报告中抽取这三项信息。

本文从学术文献和软件安全信息网站收集CVE漏洞影响的软件库，该漏洞影响库均经过研究者或安全专家的评审。由于中文报告和英文报告都对应一个具体的CVE漏洞，可以通过CVE漏洞将漏洞报告和漏洞影响库关联起来，构建一个由＜漏洞报告，漏洞影响库＞对构成的高质量数据集用于接下来对大语言模型的调优。

4.2 大语言模型指令调优

本文选择Qwen1.5-14B（即“通义千问”大模型）作为基础大模型。实验结果表明Qwen1.5-14B在MML[68]、C-Eval[69]和CMMLU[70]等多个中英文

语言理解的测评集上全面领先同规模的开源模型[29]，且在信息提取任务（如实体命名识别[30]和关系抽取[31]）上展现出良好的性能，非常适合用于对中文和英文漏洞报告进行理解并从中识别漏洞的影响库。因此本文选择使用Qwen1.5-14B来接受指令输入，对复杂的漏洞报告进行分析，并输出漏洞影响的软件库。指令调优已经被证明对于提升大语言模型在专有任务上的表现有很大帮助，因此本文将Qwen1.5-14B在之前构建好的高质量漏洞报告数据集上进行指令调优。对大语言模型进行全量参数调优在设备和时间上开销巨大，一种更为高效的方式是对其进行局部参数微调，本文使用LoRA微调技术对模型进行局部微调，提升大语言模型对于漏洞报告的理解能力和对漏洞影响库的识别能力。对Qwen1.5-14B的LoRA局部微调主要包含以下四步：模板构建、数据标准化、LoRA矩阵训练、结果解析。

4.2.1　模板构建

虽然大语言模型可以接受非结构化的数据输入，但结构化的输入可以帮助模型更清晰地理解输入各部分的含义，因而对大语言模型的微调通常采用结构化的数据输入。Alpaca模板是对大语言模型进行指令微调的通用模板，由Instruction、Input和Output三部分组成，对于不同的任务需要定制化构建训练模板的Instruction。本文分别为中文报告和英文报告定制化了两个训练模板。模板的Instruction主要包括自我认知、任务描述和输出要求三部分内容。自我认知部分，通过提示大模型“你是一名拥有软件漏洞和软件库相关知识的软件供应链安全专家”来让模型进入相应的角色设定，使大语言模型从海量的预训练知识中聚焦到软件漏洞和供应链安全相关的知识域，减少无关知识的干扰，提升在特定任务上的准确性。任务描述部分，告诉大模型需要执行的任务是检查漏洞的相关信息然后分析出漏洞的影响库，并对漏洞信息的具体条目做出解释，帮助模型更好地理解这些信息。输出要求部分，对大语言模型的输出做了约束，要求模型直接输出漏洞影响的软件库，而不需要做过多的解释。

4.2.2　数据标准化

本文将中文漏洞报告、英文漏洞报告按照构建好的模板进行数据标准化，生成模板实例。图4、图5分别展示了中文训练模板和英文训练模板的两个实例，模板的Instruction部分按照先前构建好的内容进行填充，模板的Input部分则需要按照不同报告对应的具体数据项进行填充。中文训练模板的Input部分包含了从中文漏洞报告抽取的漏洞标题、漏洞描述和漏洞类型，英文漏洞报告的Input部分包含了从英文漏洞报告抽取的漏洞描述、参考链接CPE配置信息。模板的Output部分是漏洞报告对应的影响库（标签）。数据标准化构建出的模板实例将成为大语言模型的输入，在调优阶段，大语言模型遵循Instruction，依据Input生成内容，并让生成的内容更贴近output，在评估阶段，则直接根据Instruction和Input生成最终的结果。

Instruction:	您是一名拥有软件漏洞和软件库相关知识的软件供应链安全专家。 您需要仔细检查并分析漏洞的一些相关信息（漏洞名称、漏洞描述和漏洞类型），然后直接提供该漏洞影响的软件库。
Input:	漏洞标题： CloudBees Jenkins Job Import Plugin 代码问题漏洞 漏洞描述：CloudBees Jenkins (Hudson Labs)是美国CloudBees公司的一套基于Java开发的持续集成工具。该产品主要用于监控持续的软件版本发布/测试项目和一些定时执行的任务。Job Import Plugin是使用在其中的一个Jenkins任务实例导入插件。 Jenkins Job Import Plugin 2.1及之前版本中的src/main/java/org/jenkins/ci/plugins/jobimport/client/RestApiClient.java文件存在XML外部实体注入漏洞。攻击者可利用该漏洞读取任意文件，造成拒绝服务或其他危害。 漏洞类型：代码问题
Output:	['https:// github.com/jenkinsci/job-import-plugin']

图4　中文训练模板的实例

4.2.3　LoRA矩阵训练

本文首先根据小规模试验确定了大语言模型微调的一些超参数，例如模型的输入长度被设置为1024个token，这可以覆盖绝大多数漏洞报告的长度。然后本文设定了两个低秩矩阵$\boldsymbol{A}$和$\boldsymbol{B}$，低秩矩阵$\boldsymbol{A}$的输入维度和Qwen1.5-14B注意力层的输入维度相同，低秩矩阵B的输出维度和Qwen1.5-14B注意力层的输出维度相同，矩阵$\boldsymbol{A}$的输出维度和矩阵$\boldsymbol{B}$的输入维度相同，均设为8，这也是矩阵$\boldsymbol{A}$和$\boldsymbol{B}$的秩。随后使用随机高斯分布初始化矩阵$\boldsymbol{A}$，并将

Instruction:	You are a software supply chain security expert with knowledge of software vulnerabilities and software libraries. You need to carefully review and analyze some relevant information (description, references, and CPE) about a vulnerability, and directly provide the software libraries it affects.
Input:	vulnerability descriptions: In the Automattic WooCommerce plugin before 3.2.4 for WordPress, an attack is possible after gaining access to the target site with a user account that has at least Shop manager privileges. The attacker then constructs a specifically crafted string that will turn into a PHP object injection involving the includes/shortcodes/class-wc-shortcode-products.php WC_Shortcode_Products:get_products() use of cached queries within shortcodes. reference links: https:// blog.ripstech.com/2018/woocommerce-php-object-injection/ https://woocommerce.wordpress.com/2017/11/16/woocommerce-3-2-4-security-fix-release-notes/ CPE information: ['woocommerce']
Output:	['https:// github.com/woocommerce/woocommerce']

图5 英文训练模板的实例

矩阵$\boldsymbol{B}$初始化为零矩阵，确保在训练开始阶段，矩阵$\boldsymbol{A}$和$\boldsymbol{B}$的乘积为0，避免对大语言模型的输出结果产生较大的干扰，使训练更稳定。然后冻结Qwen1.5-14B的所有参数开始训练。如式2所示，x为模型注意力层的输入，W_0为注意力层的原始参数，$\boldsymbol{A}$和$\boldsymbol{B}$为之前定义好的低秩矩阵。模型的输入传播到注意力层后，会由模型的原始参数和低秩矩阵共同计算得到输出结果h。具体地，h由两部分独立计算后相加而得，即由大语言模型原始的注意力层输出(W_0x)和低秩矩阵与输入的矩阵乘积($\boldsymbol{BA}x$)相加而得。接着使用交叉熵损失函数计算模型的损失，但只计算低秩矩阵$\boldsymbol{A}$和$\boldsymbol{B}$的梯度并更新其参数。LoRA微调将大语言模型调优的成本从对大语言模型全量参数的计算与更新缩减到对两个低秩矩阵的更新，参数的训练量减少至原来的0.05%。此外，LoRA局部微调十分灵活，并不会影响大语言模型原本的计算和输出，只需在注意力层添加训练后的低秩矩阵的计算结果。

$$h = W_0x + \Delta Wx = W_0x + \boldsymbol{BA}x \tag{2}$$

4.2.4 结果解析

LoRA矩阵训练完成后，Qwen1.5-14B基础大模型和LoRA矩阵将共同组合成一个指令遵循的大语言模型，该模型将作为一个安全专家，结合其具备的与软件供应链安全相关的预训练知识，遵循检查漏洞报告信息并识别漏洞影响库的指令，给出输入漏洞报告对应的漏洞影响库，输出结果将是一段只包含漏洞影响库的文本。本文对输出长度做出截断，去除超过256个token的部分，然后对文本进行序列化处理，转换成一个包含所有漏洞影响库的列表，方便后续的自动化操作。

4.3 大语言模型幻觉消除

大语言模型的输出可能存在幻觉问题，即Qwen1.5-14B输出的漏洞影响库可能看似合理，但实际上并不存在。例如模型输出的漏洞影响库是Jenkins的thycotic-secrets-vault库，但真实的漏洞影响库是Jenkins的thycotic-devops-secrets-vault插件。针对这一问题，本文先收集真实的软件库构建软件库列表，然后利用多种文本相似度算法将模型的输出结果映射到软件库列表上，以此消除大语言模型的幻觉问题。本文使用了三类相似度算法：(1)基于编辑的相似度算法；(2)基于token的相似度算法；(3)基于序列的相似度算法。

4.3.1 基于编辑：Levenshtein相似度

Levenshtein距离是最常用的编辑距离，用于测量将一个字符串转换成另一个字符串所需的最少单字符允许操作(插入、删除或替换)的数量，而Levenshtein相似度就是1减去转换所需的最少单字符操作次数除以两个字符串长度的最大值。Levenshtein相似度范围为0到1。例如将“bellow”转换成“bow”最少需要删除3个字符，因此Levenshtein距离为3，字符串“bellow”长度为6，字符串“bow”长度为3，因此Levenshtein相似度为1-(3/Max(3,6))=0.5。

4.3.2 基于token：cosine相似度

cosine相似度(余弦相似度)是一种被广泛使用的基于token的相似度函数，cosine相似度计算量化多维空间中两个非零向量间角度的余弦值，常用于比较文本、文档等其他高维数据点之间的相似度。cosine相似度捕捉向量的方向，而不是大小。

$$cosine(\theta) = \frac{\boldsymbol{A}\cdot\boldsymbol{B}}{|\boldsymbol{A}|\cdot|\boldsymbol{B}|} = \frac{\sum_{i=1}^{n}\boldsymbol{A}_i\boldsymbol{B}_i}{\sqrt{\sum_{i=1}^{n}\boldsymbol{A}_i^2}\sqrt{\sum_{i=1}^{n}\boldsymbol{B}_i^2}} \tag{3}$$

cosine相似度的计算如式3所示，其中$\boldsymbol{A}\cdot\boldsymbol{B}$表示向量$\boldsymbol{A}$和$\boldsymbol{B}$之间的点积，$|\boldsymbol{A}|$表示向量$\boldsymbol{A}$的欧几里得范数，$|\boldsymbol{B}|$表示向量$\boldsymbol{B}$的欧几里得范数，cosine相似度

的取值范围通常为[-1,1]，其中-1表示完全相反，1表示完全相同，0表示正交或去相关。在文本匹配任务中，向量$\boldsymbol{A}$和向量$\boldsymbol{B}$通常表示文本的术语频率向量。cosine相似度在比较时标准化文本长度。由于术语频率不能为负，因此在文本相似度的场景中，两个文本之间的cosine相似度范围为0到1。

4.3.3　基于序列：最长公共子字符串相似度

最长公共子字符串指两个字符串之间最长的连续相同的子字符串，最长公共子字符串相似度则是将最长公共子字符串的长度除以两个字符串长度的最大值。例如字符串“pineapples”长度为10，字符串“apple”长度为5，它们的最长公共子字符串“apple”长度为5，因此它们的最长公共子字符串相似度为5/Max(5,10)=0.5。

本文的方法首先分别单独使用三种文本相似度算法将大语言模型的输出结果映射到软件库列表，以消除大语言模型的幻觉问题。同时由于不同种类相似度算法的侧重点不同，例如Levenshtein相似度更关注字符串整体间是否相似，cosine相似度更关注token的重合度，包含语义信息，而最长公共子字符串相似度则更关注字符串的相似主体，希望连续相同的部分更长，它们关注的目标不同，因而可以相互纠正。因此除单独使用Levenshtein相似度、cosine相似度和最长公共子字符串相似度以外，本文也将三种相似度算法进行两两组合以及三者结合，综合考虑模型结果与列表中每个软件库的相似度，提升方法识别的准确性。

本文的方法首次提出将中英文漏洞报告相互补充作为输入，扩展了该领域的数据来源，同时微调大语言模型对漏洞报告进行理解分析，识别出漏洞的影响库，并使用文本相似度方法消除大语言模型出现的幻觉问题。

5　实　验

实验旨在验证本文提出的方法是否能有效地从漏洞报告识别漏洞影响的软件库。

5.1　研究问题

本节提出以下三个研究问题(Research Questions)来驱动实验设计和结果分析。

RQ1：本文提出的方法能否有效地从中/英文漏洞报告中识别漏洞的影响库？

RQ2：本文提出的方法在中/英文报告相互补充的情况下效果提升如何？

RQ3：本文提出的方法对于不同包管理器中的漏洞影响库识别效果如何？

5.2　实验设计

5.2.1　数据准备

实验使用的数据由漏洞报告和漏洞影响库两部分组成，本文首先从CNNVD和NVD收集截至到2023年底的中文漏洞报告和英文漏洞报告。漏洞影响库的来源包括三个渠道：学术文献[23-24,26]、Snyk[71]和GitHub Advisory[72]。由于CNNVD漏洞报告和NVD漏洞报告未提供漏洞影响的软件库，所以我们只保留上面三个数据源中经过研究者或安全专家评审且提供明确漏洞影响库的漏洞报告，这些漏洞报告有明确的漏洞影响库，可以用于后续模型的训练与评估。最终本文的数据集含有9260份中文漏洞报告和9260份英文漏洞报告以及它们对应的漏洞影响库。每份中文漏洞报告和英文漏洞报告都对应一个CVE漏洞，所以数据集中共含有9260个CVE漏洞，即一个CVE漏洞对应一份中文报告和一份英文报告，这两份报告的漏洞影响库相同。图6展示了数据集中漏洞影响库所属的包管理器的数据分布，其中Maven(Java库)占比最高，PyPI(Python库)、Composer(PHP库)、NPM(JavaScript库)、Golang(Go库)次之，这五个包管理器的漏洞影响库占比达到数据集总体的92%。实验按时间顺序排列漏洞报告，取前80%用于训练，后20%用于测试。

5.2.2　基线方法

实验选择了目前最先进的两个漏洞影响库识别方法LightXML[23]和Chronos[26]作为基线方法。由

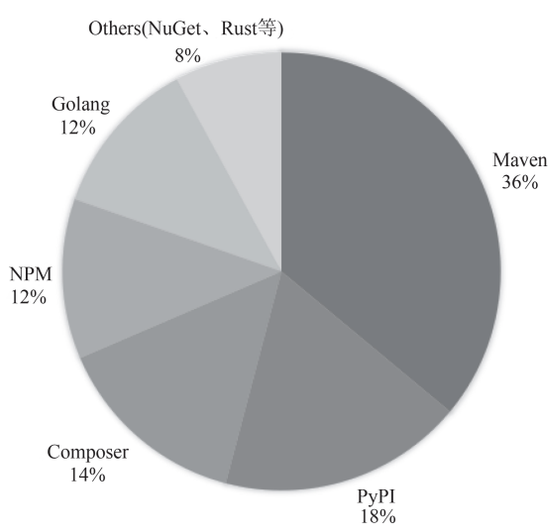

图6　漏洞影响库所属包管理器分布

于LightXML方法基于BERT，RoBERTa和XLNet三个模型，因此实验分别基于这三个模型实现了三个基线：LightXML-BERT，LightXML-RoBERTa和LightXML-XLNet。对于英文报告，Chronos会爬取漏洞报告中提供的参考链接的网页内容进行增强，因此针对英文报告本实验实现了基线Chronos-w/o-DE（Chronos without data enhancement）和Chronos，而由于中文报告并不提供参考链接，因此针对中文报告本实验实现了基线Chronos。LightXML和Chronos的数据预处理步骤与之前的研究保持一致。

5.2.3 模型选择和超参数设置

本文的任务要求模型对自然语言形式的中/英文漏洞报告进行理解，Vicuna-13B在中/英文任务上的表现都位居开源模型前列[73]，而Qwen1.5-14B在多个评估中英文理解能力的数据集上领先其他同规模开源模型[29]，且在信息提取任务（如实体命名识别[30]和关系抽取[31]）上展现出良好的性能，因此本文选择使用Vicuna-13B和Qwen1.5-14B两个模型作为基座模型进行实验。

超参数设置如表1所示。lora_r用于调整信息提取的丰富度，lora_alpha用于调整学习的速度，实验设置lora_r为8，足够模型高效地学习有价值的信息，lora_alpha与lora_r在通常情况下比值为2，所以实验设置lora_alpha为16。实验设置temperature为1，让模型的生成内容更加多样。top_k用于调整模型生成每个token时采样的候选列表的大小，top_p用于调整模型生成每个token时候选列表概率之和，num_beams用于调整模型在束搜索时候选序列的大小，实验设置top_k为40，设置top_p为0.75，设置num_beams为4来平衡模型输出的质量和多样性。max_new_tokens用于调整模型生成结果的最大长度，由于绝大部分漏洞报告对应的漏洞影响库数量不会太多，实验设置max_new_tokens为256，足以输出所有的漏洞影响库。max_length用于调整模型输入的最大长度，实验设置max_length为1024，可以覆盖绝大部分漏洞报告的长度。实验设置batch_size为128，可以增加梯度计算的稳定性，加快模型收敛的速度。实验设置epoch为30，足以确保模型完成收敛。实验使用1张NVIDIA A100-PCIE-40GB显卡进行训练，训练时间为两天。

表1 本文方法的超参数设置

超参数	设定值
lora_r	8
lora_alpha	16
lora_dropout	0.01
temperature	1.0
top_k	40
top_p	0.75
num_beams	4
max_new_tokens	256
max_length	1024
batch_size	128
epoch	30

5.2.4 评价指标

实验和之前的研究[23-24,26]保持一致，使用查准率（Precision P）、召回率（Recall R）和$F1$分数（$F1$ score）来评估本文提出的方法和基线方法。每个评价指标的计算都只考虑方法的前k个预测结果，实验和之前的研究保持一致，考虑k=1、2、3三种情况。式(4)和式(5)分别计算对于单样本的前k个预测结果的查准率和召回率，其中v表示漏洞报告，$lib_k(v)$表示方法对于该报告漏洞影响库的前k个预测结果，$\widehat{lib}_k(v)$表示该报告对应的漏洞影响库（标签）的前k个结果。$P@k$和$R@k$的分子相同，都是方法预测的前k个结果与标签前k个结果间重合的数量。而$P@k$的分母是k与标签数量二者间的最小值，$R@k$的分母是标签数量。

$$P@k(v)=\frac{\left|lib_{k(v)}\cap\widehat{lib}_{k(v)}\right|}{\min\left(k,\left|\widehat{lib}_{k(v)}\right|\right)} \tag{4}$$

$$R@k(v)=\frac{\left|lib_{k(v)}\cap\widehat{lib}_{k(v)}\right|}{\left|\widehat{lib}_{k(v)}\right|} \tag{5}$$

如式(6)、式(7)、式(8)所示，$P@k$表示对所有样本前k个预测结果的查准率的平均值，$R@k$表示对所有样本前k个预测结果的召回率的平均值，$F1@k$则是对$P@k$和$R@k$的综合评估，是二者的调和平均数。

$$P@k=\frac{\sum_{v=1}^{n}P@k(v)}{n} \tag{6}$$

$$R@k=\frac{\sum_{v=1}^{n}R@k(v)}{n} \tag{7}$$

$$F1@k=2\times\frac{P@k\times R@k}{P@k+R@k} \tag{8}$$

5.3 结果与分析

5.3.1 从中/英文漏洞报告分别识别漏洞影响库

表2和表3分别展示了本文方法和基线方法从中文报告和英文报告识别漏洞影响库的效果。

表2　中文报告漏洞影响库识别效果

方法	k=1			k=2			k=3			Avg		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
LightXML-BERT	37.2	34.8	36.0	41.0	40.3	40.6	42.7	42.5	42.6	40.3	39.2	39.7
LightXML-RoBERTa	34.3	32.0	33.1	37.4	36.7	37.1	39.5	39.3	39.4	37.1	36.0	36.5
LightXML-XLNet	36.7	34.3	35.5	40.2	39.6	39.9	42.0	41.9	42.0	39.6	38.6	39.1
Chronos-w/o-DE	67.9	64.4	66.1	76.1	75.1	75.6	79.3	79.0	79.2	74.4	72.8	73.6
Vicuna-13B+lev	65.8	62.6	64.2	67.1	66.5	66.8	68.8	68.7	68.8	67.2	65.9	66.6
Vicuna-13B+cos	72.6	69.0	70.8	73.6	72.9	73.3	74.3	74.2	74.3	73.5	72.0	72.8
Vicuna-13B+lcs	65.6	62.5	64.0	68.3	67.7	68.0	70.0	69.9	69.9	68.0	66.7	67.3
Vicuna-13B+lev+lcs	66.8	63.6	65.2	69.2	68.6	68.9	71.5	71.4	71.5	69.2	67.9	68.5
Vicuna-13B+cos+lcs	74.2	70.8	72.5	76.7	76.1	76.4	78.6	78.5	78.5	76.5	75.1	75.8
Vicuna-13B+lev+cos	74.8	71.4	73.0	76.5	77.9	76.2	78.1	77.9	78.0	76.5	75.7	75.7
Vicuna-13B+lev+cos+lcs	74.2	70.8	72.5	75.9	75.2	75.6	77.9	77.8	77.8	76.0	74.6	75.3
Qwen-14B+lev	68.9	65.3	67.1	70.7	69.9	70.3	72.5	72.2	72.3	70.7	69.1	69.9
Qwen-14B+cos	73.4	69.6	71.4	74.9	74.0	74.5	76.1	75.8	76.0	74.8	73.1	74.0
Qwen-14B+lcs	69.2	65.6	67.4	71.8	70.9	71.4	73.2	72.9	73.0	71.4	69.8	70.6
Qwen-14B+lev+lcs	70.6	66.9	68.7	72.0	71.2	71.6	74.4	74.1	74.3	72.3	70.7	71.5
Qwen-14B+cos+lcs	75.9	72.1	73.9	78.2	77.3	77.7	79.7	79.4	79.6	77.9	76.3	77.1
Qwen-14B+lev+cos	76.2	72.5	74.3	**78.6**	**77.8**	**78.2**	**80.4**	**80.0**	**80.2**	**78.4**	**76.8**	**77.6**
Qwen-14B+lev+cos+lcs	**76.3**	**72.6**	**74.4**	78.1	77.2	77.6	80.2	**80.0**	80.1	78.2	76.6	77.4

表3　英文报告漏洞影响库识别效果

方法	k=1			k=2			k=3			Avg		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
LightXML-BERT	46.8	44.7	45.7	49.2	48.7	48.9	50.4	50.1	50.3	48.8	47.8	48.3
LightXML-RoBERTa	41.3	39.2	40.2	43.8	43.3	43.5	45.4	45.1	45.3	43.5	42.5	43.0
LightXML-XLNet	47.4	45.2	46.3	49.9	49.3	49.6	50.8	50.5	50.6	49.4	48.3	48.8
Chronos-w/o-DE	67.1	63.4	65.2	75.7	74.6	75.2	79.3	78.9	79.1	74.0	72.3	73.2
Chronos	66.8	63.0	64.8	75.2	74.0	74.6	78.9	78.5	78.7	73.6	71.8	72.7
Vicuna-13B+lev	68.6	64.7	66.6	71.0	70.2	70.5	72.7	72.4	72.5	70.8	69.1	69.9
Vicuna-13B+cos	77.7	73.4	75.5	79.6	78.6	79.1	80.4	80.0	80.2	79.2	77.3	78.3
Vicuna-13B+lcs	70.9	67.1	68.9	72.8	72.0	72.4	74.9	74.7	74.8	72.9	71.3	72.0
Vicuna-13B+lev+lcs	71.0	67.0	68.9	72.4	71.6	72.0	74.9	74.6	74.7	72.8	71.1	71.9
Vicuna-13B+cos+lcs	79.5	75.1	77.2	81.5	80.4	80.9	83.1	82.8	83.0	81.4	79.4	80.4
Vicuna-13B+lev+cos	79.5	75.2	77.3	81.4	80.5	81.0	82.6	82.3	82.5	81.2	79.3	80.3
Vicuna-13B+lev+cos+lcs	78.5	74.4	76.4	80.9	80.0	80.4	81.9	81.6	81.7	80.4	78.7	79.5
Qwen-14B+lev	73.3	69.4	71.3	74.0	73.3	73.6	74.8	74.6	74.7	74.0	72.4	73.2
Qwen-14B+cos	79.6	75.6	77.5	80.1	79.3	79.7	80.8	80.6	80.7	80.2	78.5	79.3
Qwen-14B+lcs	75.1	70.8	72.8	75.8	75.0	75.4	77.6	77.4	77.5	76.2	74.4	75.2
Qwen-14B+lev+lcs	74.6	70.6	72.5	75.8	75.1	75.5	77.5	77.2	77.4	76.0	74.3	75.1
Qwen-14B+cos+lcs	**81.5**	**77.3**	**79.4**	**82.6**	**81.8**	**82.2**	**83.9**	**83.6**	**83.8**	**82.7**	**80.9**	**81.8**
Qwen-14B+lev+cos	81.3	77.2	79.2	82.0	81.3	81.7	83.5	83.3	83.4	82.3	80.6	81.4
Qwen-14B+lev+cos+lcs	80.8	76.7	78.7	81.7	80.9	81.3	83.2	83.0	83.1	81.9	80.2	81.0

如表2和表3所示，相较于基线方法，基于大语言模型加文本相似度的方法效果更优。对于中文漏洞报告，基于大语言模型加文本相似度方法的最优平均$F1$分数为0.78，相较于基线方法提升了4%。对于英文漏洞报告，基于大语言模型加文本相似度方法的最优平均$F1$分数为0.82，相较于基线方法提升了9%。大模型对漏洞报告有更好的理解分析能力，同时在预训练阶段见过很多软件库，因此基于大语言模型加文本相似度的方法效果更优。对于基座大模型而言，基于Qwen-14B的识别方法效果优

于基于Vicuna-13B的识别方法。对于中文漏洞报告，基于Qwen-14B方法的最优平均$F1$分数为0.78，而基于Vicuna-13B方法的最优平均$F1$分数为0.76。对于英文漏洞报告，基于Qwen-14B方法的最优平均$F1$分数为0.82，而基于Vicuna-13B方法的最优平均$F1$分数为0.80。Qwen-14B在高质量的中英文语料上进行了针对训练，对漏洞报告有更强的语言理解和信息抽取整合能力，因而识别效果更优。

对于大语言模型加文本相似度的方法而言，使用cosine相似度的方法效果最好，单独使用cosine相似度的平均$F1$分数为在中文报告上为0.74，在英文报告上为0.79，而将cosine相似度与其他相似度相结合的方法在中文报告上平均$F1$分数均高于0.77，在英文报告上均高于0.81。cosine相似度更关注字符串分词后的token而非字符串整体或连续相同的部分，例如大语言模型输出的预测结果为“ipfs/go_unix_fsnode”，而真实的漏洞影响库“github.com/ipfs/go-unixfsnode”，将二者进行分词后都含有“go”、“unix”和“fsnode”，因而cosine相似度较高，而二者的Levenshtein相似度较低，因为需要插入前面较长的“github.com/”，需要较多编辑操作，最长公共子字符串相似度也较低，因为连续相同的部分长度较短，最长公共子串为“fsnode”。因此相较于其他两类相似度算法，cosine相似度可以考虑更多的语义信息，能更准确地将模型的结果映射到正确的软件库。

总结来说，中文报告上Qwen大模型加cosine相似度和Levenshtein相似度以及Qwen大模型加cosine相似度和Levenshtein相似度和最长公共子字符串相似度两个方法效果最好，而英文报告上Qwen大模型加cosine相似度和最长公共子字符串相似度方法效果最好。

5.3.2 从相互补充的中/英文漏洞报告识别漏洞影库

表4展示了本文方法在中文报告和英文报告相互补充下的漏洞影响库识别效果。

RQ1中得到(1)Qwen大语言模型加cosine相似度和最长公共子字符串相似度，(2)Qwen大语言模型加cosine相似度和Levenshtein相似度和(3)Qwen大语言模型加cosine相似度和Levenshtein相似度和最长公共子字符串相似度这三个方法在中文报告或英文报告上识别效果最好，因此RQ2探究这三种方法在中文报告和英文报告相互补充下的识别效果。如表4所示，三种方法上中英文报告相互补充的效果均为最优，优于单一使用中文报告或英文报告，具体而言，中英文报告相互补充使大语言模型加cosine相似度和最长公共子字符串相似度方法的平均$F1$分数提升3.5%，使大语言模型加cosine相似度和Levenshtein相似度方法的平均$F1$分数提升3.6%，使大语言模型加cosine相似度和Levenshtein相似度和最长公共子字符串相似度方法的平均$F1$分数提升3.7%。

表4 中/英文报告相互补充的漏洞影响库识别效果

方法	k=1			k=2			k=3			Avg		
	P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$
Qwen-14B+cos+lcs-中	75.9	72.1	73.9	78.2	77.3	77.7	79.7	79.4	79.6	77.9	76.3	77.1
Qwen-14B+cos+lcs-英	81.5	77.3	79.4	82.6	81.8	82.2	83.9	83.6	83.8	82.7	80.9	81.8
Qwen-14B+cos+lcs-中/英	**84.0**	**79.8**	**81.8**	**86.7**	**85.6**	**86.1**	**88.1**	**87.7**	**87.9**	**86.3**	**84.4**	**85.3**
Qwen-14B+lev+cos-中	76.2	72.5	74.3	78.6	77.8	78.2	80.4	80.0	80.2	78.4	76.8	77.6
Qwen-14B+lev+cos-英	81.3	77.2	79.2	82.0	81.3	81.7	83.5	83.3	83.4	82.3	80.6	81.4
Qwen-14B+lev+cos-中/英	84.0	79.7	81.7	86.3	85.3	85.8	87.8	87.5	87.6	86.0	84.2	85.0
Qwen-14B+lev+cos+lcs-中	76.3	72.6	74.4	78.1	77.2	77.6	80.2	80.0	80.1	78.2	76.6	77.4
Qwen-14B+lev+cos+lcs-英	80.8	76.7	78.7	81.7	80.9	81.3	83.2	83.0	83.1	81.9	80.2	81.0
Qwen−14B+lev+cos+lcs-中/英	83.7	79.6	81.6	85.9	84.9	85.4	87.1	86.8	87.0	85.6	83.8	84.7

漏洞报告中关于漏洞影响库的信息并不准确清晰，因此单独使用中文报告和英文报告进行漏洞影响库识别的结果也不准确，而使用多数据源(中文报告和英文报告)相互补充可以一定程度缓解该问题。例如对于CVE-2023-22491，它的漏洞影响库是npm仓库的“gatsby-transformer-remark”库，但它的中文报告只提及了“gatsby”，而它的英文报告中则明确提到该漏洞影响了“gatsby-transformer-remark”插件，因此单纯基于中文报告无法准确识别出该漏洞的影响库，而同时基于中英文报告则可以正确识别。再例如对于CVE-2023-25313，它漏洞库影响是Composer仓库中WWBN组织的“AVideo”库，但它

的英文报告中只显式给出了“AVideo”，因而只基于英文报告识别的结果不准确，识别结果为“avideo-server”，事实上该软件库并不存在；而它的中文报告中漏洞的标题便是“WWBN AVideo 命令注入漏洞”，清晰给出了该漏洞影响的组织和软件库，因而同时基于中英文报告可以正确识别出漏洞影响库为“WWBN/AVideo”。再例如对于CVE-2023-25313，它的漏洞影响库是Composer仓库中wintercms组织的“winter”库，它的英文漏洞描述中大量阐述了该漏洞的利用原理和触发条件，非常干扰模型对漏洞影响库的识别，并且在参考链接里提到了storm仓库，导致模型只基于英文报告错误地将漏洞影响库识别为“storm/storm”；而它的中文报告中则较为简洁地提到了WinterCMS平台和winter库，因而同时基于中英文报告则可以正确识别出漏洞的影响库为“wintercms/winter”。多数据源（中英文报告）相互补充，可以更好地从漏洞报告识别漏洞的影响库，构建更全面的安全防护网。

总结来说，以中英文报告相互补充为输入，通过微调后的Qwen1.5-14B模型从漏洞报告进行漏洞影响库识别，然后使用cosine相似度加最长公共子字符串相似度进行幻觉消除的方法在漏洞影响库识别任务上的效果最佳，平均$F1$分数达到0.85，因此该方法为本文提出的基于大语言模型的多来源漏洞影响库识别方法的最终实例，后续提到的本文方法如无特殊说明均代指该方法。

5.3.3　识别不同编程语言包管理器中的漏洞影响库

表5展示了本文提出的方法和基线方法对于不同编程语言包管理器中影响库的识别效果。

首先从方法的角度进行比较，本文提出的方法对于五种主流包管理器（占数据集分布的92%）的漏洞影响库识别上有四种全面占优，即在k=1，2，3上$F1$分数均为最优，而只在Maven库识别上当k=2时弱于基于中文漏洞报告的Chronos，但$F1$分数也仅落后0.5%，而在k=1和3时本文的方法在$F1$分数上分别领先3.2%和1.1%，因此在对Maven漏洞库的识别总体上仍为本文方法略微占优。在对PyPI、Composer、NPM和Golang库的识别上，本文提出的方法相较于基线方法都有着明显的优势，$F1$分数在k=1时均领先基线方法超过10%. 总体而言，基线方法更侧重于识别数据集中出现较多的Maven库（占数据集分布的36%），但在识别其他包管理器库上的效果欠佳，而本文提出的方法则更加全面，不仅关注训练中占比较多的Maven库，在其他包管理器库的识别上也有较高的性能保障。

从包管理器的角度来看，所有方法对于Maven库的识别效果相较于其他包管理器均为最差。Maven库的命名长度较长，描述更详细，例如“org.apache.felix.healthcheck.webconsoleplugin”库，它的描述中含有组织名、项目名、模块名和功能名，模型完整准确地识别出该库难度较大。此外Maven库数量繁多，命名结构相对模块化，因而有很多名称十分相似的库，例如“wiremock-standalone”库和“wiremock-jre8-standalone”库十分相似，模型从多个相似的库中准确识别出漏洞影响库难度较大。例如对于CVE-2023-40809，模型识别出的结果为“org.apache.sling.resource-mapper”，但该库并不存在，真正的影响库为“org.apache.sling.resourcemerger”库，再例如对于CVE-2023-38905，模型识别的结果为“jeecg-boot-common”库，但事实上漏洞的影响库为“jeecg-boot-parent”库，因此相比于其他包管理器，准确识别Maven中的漏洞库难度更大，防护效果相对较差。对于NPM包管理器，模型的识别效果在k=1时与其他包管理器相近，但在k=2和3时效果更差，这是因为关于NPM库的有些漏洞会影响多个库，但模型只识别到了其中的一个。造成这个现象的原因主要有三点：(1)漏洞报告中只提到了一个库的信息。例如对于CVE-2023-40014，漏洞报告报告只提及了“openzeppelin contracts”，因而模型只识别出了“openzeppelin/contracts”库，但事实上该漏洞还影响了“openzeppelin/contracts-upgradeable”库。(2)漏洞报告中更侧重提及其中一个库的信息。例如对于CVE-2023-35165，漏洞报告报告虽然提到了“aws-cdk-lib”和“aws-cdk/aws-eks”这两个影响库，但却反复提及“AWS CDK”，导致模型最终只识别出了“aws-cdk-lib”库。(3)漏洞影响了一个主包中的多个子包，但漏洞报告中只提到了主包相关的信息。例如对于CVE-2023-38507，漏洞报告报告只提到了该漏洞影响的主包“strapi”，因而模型只识别出了“strapi”，但具体地，该漏洞影响了“strapi”主包中的“admin”和“plugin-users-permissions”两个子包，模型均未能正确识别。对于PyPI、Composer和Golang这三个包管理器，它们的库数量相对较少，库的命名方式更扁平化，因而相似名称的软件库也相对较少，同时它们的漏洞报告大多只影响单一软件库，因而漏洞影响库识别方法在这三个包管理器上效果更优，当k=3时，本文方法在这三个包管理器上的$F1$分数均在0.90以上，基线方法如Chronos的$F1$分数也都达到0.75。

表5 不同包管理器上的漏洞影响库识别效果

k	方法	Maven			PyPI			Composer			NPM			Golang		
		P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$
1	LightXML-BERT-中	24.8	23.6	24.2	50.7	45.5	48.0	64.2	62.4	63.3	41.0	36.2	38.5	31.8	30.2	31.0
	LightXML-RoBERTa-中	21.3	20.1	20.7	47.8	42.3	44.9	60.5	58.7	59.6	40.1	35.3	37.6	29.5	28.0	28.7
	LightXML-XLNet-中	25.3	24.1	24.7	52.2	46.7	49.3	60.9	59.4	60.1	40.5	35.8	38.0	29.1	27.5	28.3
	Chronos-w/o-DE-中	64.3	61.1	62.7	74.3	71.9	73.1	67.9	64.4	66.1	65.8	62.6	64.2	67.3	63.0	65.1
	LightXML-BERT-英	42.8	41.0	41.9	50.4	48.3	49.3	48.3	46.3	47.3	45.5	43.7	44.6	48.6	46.5	47.6
	LightXML-RoBERTa-英	38.4	36.6	37.5	42.2	39.7	40.9	41.0	39.2	40.0	42.8	41.3	42.0	43.6	41.7	42.7
	LightXML-XLNet-英	43.1	41.3	42.2	51.0	48.6	49.8	47.6	45.4	46.5	48.2	46.1	47.1	50.0	47.9	48.9
	Chronos-w/o-DE-英	64.1	60.8	62.4	71.4	69.1	70.2	69.0	62.8	65.7	68.0	63.9	65.9	66.8	62.3	64.5
	Chronos-英	63.5	60.0	61.7	73.2	70.2	71.6	71.6	65.4	68.3	64.9	60.7	62.7	65.5	61.4	63.4
	Qwen-cos-lcs-中/英	**66.9**	**64.9**	**65.9**	**89.9**	**80.8**	**85.1**	**92.6**	**89.6**	**91.1**	**90.1**	**84.1**	**87.0**	**93.6**	**89.8**	**91.7**
2	LightXML-BERT-中	29.1	28.9	29.0	54.6	52.5	53.5	74.2	73.8	74.0	41.4	40.3	40.9	32.5	32.5	32.5
	LightXML-RoBERTa-中	23.1	22.9	23.0	51.2	49.1	50.1	72.3	72.0	72.1	39.9	38.7	39.3	29.8	29.8	29.8
	LightXML-XLNet-中	27.5	27.3	27.4	53.8	51.7	52.8	74.4	74.0	74.2	43.0	41.9	42.4	29.8	29.8	29.8
	Chronos-w/o-DE-中	**72.7**	**71.9**	**72.3**	83.2	82.1	82.6	76.1	75.1	75.6	72.7	71.7	72.2	75.5	74.3	74.9
	LightXML-BERT-英	45.1	44.9	45.0	52.6	52.3	52.5	50.2	49.9	50.0	48.9	48.4	48.6	50.0	48.9	49.5
	LightXML-RoBERTa-英	40.6	40.5	40.5	45.7	45.3	45.5	43.0	43.0	42.9	44.8	44.4	44.6	46.8	45.6	46.2
	LightXML-XLNet-英	45.8	45.6	45.7	54.0	53.4	53.7	49.3	49.1	49.2	50.9	50.2	50.5	50.9	49.8	50.3
	Chronos-w/o-DE-英	72.4	71.6	72.0	79.8	78.9	79.3	78.4	76.7	77.6	73.4	71.7	72.5	78.0	76.8	77.4
	Chronos-英	70.9	70.0	70.5	80.5	79.3	79.9	79.2	77.5	78.3	72.5	70.6	71.5	79.3	78.1	78.7
	Qwen-cos-lcs-中/英	72.0	71.6	71.8	**91.1**	**87.1**	**89.1**	**93.3**	**93.3**	**93.3**	**86.5**	**86.2**	**86.3**	**94.1**	**93.8**	**93.9**
3	LightXML-BERT-中	31.6	31.6	31.6	55.3	55.3	55.3	75.8	75.8	75.8	43.8	43.7	43.7	33.6	33.6	33.6
	LightXML-RoBERTa-中	26.3	26.3	26.3	53.0	53.0	53.0	74.7	74.7	74.7	40.2	40.1	40.1	31.4	31.4	31.4
	LightXML-XLNet-中	31.4	31.4	31.4	54.2	54.2	54.2	75.1	75.1	75.1	43.4	43.2	43.3	31.1	31.1	31.1
	Chronos-w/o-DE-中	76.4	76.0	76.2	85.5	84.9	85.2	82.3	82.1	82.2	75.3	74.9	75.1	78.0	77.7	77.8
	LightXML-BERT-英	46.3	46.2	46.2	54.3	54.2	54.3	51.8	51.6	51.7	50.8	50.4	50.6	50.6	49.7	50.2
	LightXML-RoBERTa-英	42.2	42.2	42.2	47.8	47.7	47.8	44.8	44.7	44.8	46.8	46.4	46.6	48.1	47.1	47.6
	LightXML-XLNet-英	46.5	46.5	46.5	54.8	54.5	54.6	51.0	51.0	51.0	50.5	50.2	50.4	52.1	51.1	51.6
	Chronos-w/o-DE-英	75.3	75.0	75.2	83.7	83.2	83.4	82.5	82.3	82.4	76.2	75.2	75.7	83.1	82.9	83.0
	Chronos-英	74.4	74.1	74.3	85.6	85.1	85.4	82.4	82.3	82.3	75.7	74.6	75.1	81.8	81.6	81.7
	Qwen-cos-lcs-中/英	**77.3**	**77.2**	**77.3**	**92.1**	**92.1**	**92.1**	**93.3**	**93.3**	**93.3**	**86.5**	**86.5**	**86.5**	**95.0**	**95.0**	**95.0**

6 讨 论

本节讨论应用大语言模型进行漏洞影响库识别的经验，漏洞影响库的数据分布情况，本研究的局限性以及未来可能的改进方向。

6.1 大语言模型的使用

6.1.1 大语言模型微调

研究表明，相较于大语言模型参数的规模，指令微调对于提升大模型在特定任务上的表现更为有效，本文也尝试过使用Qwen-1.5-7B作为基础大模型，其最终的F1得分相较于本文提出的方法（使用Qwen-1.5-14B作为基础大模型）只相差不到5%，而对于同一个大语言模型是否在漏洞报告数据集上进行指令微调，二者F1得分相差超过15%。而对于大语言模型全量参数进行指令调优开销巨大，例如Qwen-1.5-14B的参数超过140亿，对其以bfloat16的精度全量微调，该任务在2张NVIDIA A100-PCIE-40GB（共80 G显存）上无法运行，而使用LoRA参数微调后，该任务在1张NVIDIA A100-PCIE-40GB上即可正常运行，且训练速度在理论上相较于全量微调提升了数十倍，只需要两天，训练的参数量也减少至原来的0.05%。因此对大语言模型在领域数据上进行微调，是一种成本可接受的性能提升方法。

另一方面，当模型在训练时仅需更新d维度参数就可以达到与更新全部网络参数相当的效果时，d就是该模型的本质维数[74]，该本质维数里的参数

是真正对于模型提升性能有价值的参数，而大语言模型通常具有较低的本质维数，远低于其拥有的庞大参数，这为微调的性能保障提供了理论支持。大语言模型的局部微调方法主要有三类：Adapter微调、Prefix微调和LoRA微调。Adapter微调会在模型的某些层之间插入小的神经网络模块（adapters），这会增加大语言模型的网络层数，增加训练和推理的时间，Prefix微调会在输入中加入特殊的前缀，微调难度更高且会减少有效信息的长度，而LoRA微调只需要训练两个可插拔的低秩矩阵，既不会增加训练时间，也不会减少有效信息的输入长度，因此考虑成本和性能通常会选择使用LoRA来微调大语言模型。研究表明，LoRA微调相较于全量参数微调几乎不会造成性能损失[75]。因此LoRA微调是一种高效提升大语言模型在特定任务上性能的方法。

此外，任务数据集的质量对大语言模型的微调效果也至关重要，数据集质量的重要性远高于数量，数千条高质量的数据即可显著提升大模型在特定任务上的性能。数据的质量主要体现在标签上，因为大语言模型在训练过程中会将原本的生成内容逐步向标签内容靠近，因此需要确保标签内容质量较高，即符合预期输出。本文为了确保漏洞报告对应的漏洞影响库质量，并非通过从网页手动检索的方式收集漏洞的影响库，而是从学术文献和安全网站收集，这些信息来源中公布的漏洞及其影响的软件库都经过研究人员或安全专家的审查，在准确性和全面性上具有较高的的保障。本文提出方法的最终效果也验证了该数据集的质量较高，可以有效地提升基础大模型在漏洞影响库识别上的表现。本文尝试直接使用Qwen-1.5-14B从漏洞报告识别漏洞的影响库，不仅含有大量的无关输出，而且模型对于漏洞报告的分析整合能力较差，在k=1，2，3上的平均$F1$分数只有0.51，而使用微调后的Qwen-1.5-14B模型的识别结果可以达到0.68。实验结果表明微调对于提升大语言模型在特定任务上的性能效果显著。

6.1.2　输出结果后处理

虽然微调阶段限制了大语言模型的输出，要求其直接以列表的形式输出漏洞的影响库，但在实际应用阶段，仍有少量样本本未严格遵循指令，有的样本输出的每个漏洞影响库自成一个列表（例如“[io.netty：netty-codec-http2] [io. netty：netty-codechttp] [io.netty：netty-codec-quic]...”），有的样本输出中仍含有解释性文本（例如“['folders'] 单个组织存在漏洞的组件：folder 存在漏洞的包和组件：Jenkins Plugin Folders 6.846.v23698686f0f6及之前版本”），这些情况都需要判断（字符串模式匹配）并采取相应的处理操作，将模型的输出转化为一个由漏洞影响库组成的列表，以保障模型的可用性并便于后续的自动化处理。

此外，对大语言模型输出结果的后处理操作对提升性能也很关键，大语言模型的输出会存在幻觉问题，在本文的任务中输出的漏洞影响库可能不存在，但其输出结果往往很接近真实的漏洞影响库。例如对于CVE-2023-27604，模型输出的漏洞影响库是“apache-airflow-providers-sqoop”，但事实上该软件仓库并不存在，真正的漏洞影响库是Apache Airflow 组织的“apache-airflow-providers-sqoop”库。因此如果直接将大模型的识别结果作为漏洞影响库，则可能会输出不存在的软件库，而如果将大模型的识别结果与真实的软件库列表进行精确匹配，则会忽视很多存在细小偏差的结果，造成漏报，降低安全防护效果。因此本文采用基于文本相似度的方法对大模型的识别结果进行后处理，基于多种文本相似度算法将大模型的结果向真实的软件库列进行映射，在消除大模型幻觉的同时减少漏报，提升识别效果。基于文本相似度的方法可以在大模型的输出结果与真实漏洞影响库存在一定偏差时进行修正。例如，对于CVE-2023-26512，它的真实影响库为Maven 仓库中的“eventmesh-connector-rabbitmq”库，大模型的预测结果为“eventmesh-rabbitmq-connector”，虽然大模型的结果与真实的漏洞影响库间存在差异，但二者具有较高的语义相似性，因而使用cosine相似度可以将大模型的结果有效修正为“eventmesh-connector-rabbitmq”。再例如对于CVE-2023-39523，它的真实影响库为Pip仓库中的“scancodeio”库，而大模型的预测结果为“scancode”，二者只相差“io”两个字母，编辑距离较小，使用Levenshtein相似度则可以将大模型的结果有效修正为“scancodeio”。

本文尝试将微调后的Qwen-1.5-14B模型的输出结果与真实软件库列表进行精确比配，实验结果表明在k=1，2，3上的平均$F1$分数只有0.68，而经过基于文本相似度匹配的后处理操作后在k=1，2，3上的平均$F1$分数可以达到0.85。这说明了输出结果后处理对于增强基于大语言模型的方法至关重要。

6.2 漏洞影响库的数量

本文收集了9,260份中/英漏洞报告,每份报告对应的漏洞影响库的数量分布如图7所示,约90%的漏洞报告都只对应1个漏洞影响库,6%的漏洞报告对应2个漏洞影响库,3%的漏洞报告对应3个漏洞影响库,只有2%的漏洞报告对应超过3个漏洞影响库。由于98%的漏洞报告的漏洞影响库数量都不超过3个,因而实验选取的k=1,2,3足以涵盖绝大多数的漏洞报告。同时接近9成的漏洞报告都只有1个影响库,因而k=1时方法的性能与实际使用的情况最相关,也最能体现方法的实际使用价值。

漏洞影响库的数量分布也反映出漏洞报告只会报告漏洞直接影响的软件库,即发生漏洞的软件库,而很少包含依赖该漏洞库的软件库,因而漏洞影响库的数量通常不超过3个。但在软件供应链的开发范式下,一个软件会依赖大量第三方软件库,因而更全面的防护措施需要先从漏洞报告识别出漏洞的直接影响库,然后从该漏洞直接影响的软件库出发,查找依赖该库的软件库,这些库由于依赖关系,也可能受到该漏洞的影响,这样将会提供更加全面立体的软件供应链安全防护。

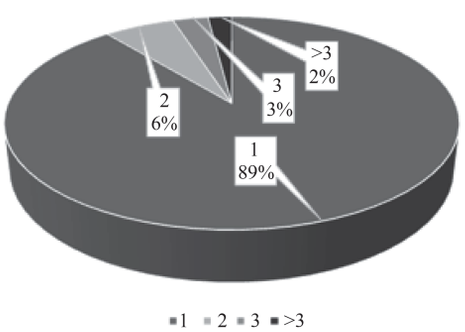

图7 每份漏洞报告对应漏洞影响库的数量

6.3 研究局限性

本文中漏洞报告的数据来源只包含NVD和CNNVD,而未纳入其他数据源如Rapid7[76]、X-Force[77]等。本文选择NVD和CNNVD漏洞报告是因为(1) NVD和CNNVD由官方进行维护,漏洞报告数量多且质量较高;(2)与之前研究[23-24,26]保持一致,因此使用NVD和CNNVD漏洞报告对方法进行评估是合理且可行的。若要将数据源扩大至更多漏洞报告平台,则需根据新的漏洞报告的数据特征重新设计训练模板,然后对大模型重新进行微调,使其可适用于更多来源的漏洞报告。

此外,本文提出的漏洞影响库方法基于大语言模型,相较于传统的基于XML模型的识别方法,所需要的资源开销更高。对于时间开销,本文方法的时间占用主要集中在训练阶段,需要数十小时,而在测试阶段对于每份漏洞报告只需要数秒即可完成识别,可以满足企业或安全公司对漏洞报告的识别速度要求。对于硬件开销,本文依赖的Qwen1.5-14B模型如果进行全量参数调优需要占用超过80 G显存,但使用LoRA参数微调可将显存占用降低至34 G,而使用Q-LoRA参数微调可将显存占用进一步降低至18 G,可以在大部分硬件设备上完成部署。因此综合来看,虽然本文方法的资源开销相对基线方法更高,但仍在可接受的范围内,同时可以提供更好的识别效果。此外,大语言模型虽然具有大量的预训练知识,在预训练阶段见过很多软件库,但大模型的知识有时间限制,例如本文依赖的Qwen1.5-14B基模型的预训练知识截至到2024年初,而由于不断有新的软件库发布,这些软件库也可能被曝出存在安全漏洞从而成为漏洞影响库,对于这些新出现的漏洞影响库,大模型的预训练知识中并不包含这些库,因而识别能力会出现一些降低。针对这一问题,可以定期对基座大模型进行升级,更新为能力更强,预训练知识更广的大模型,不断提升本文方法的漏洞影响库识别效果。

6.4 改进方向

根据对实验结果的分析与讨论,本文总结出三个对本文方法的改进方向。

RQ1的实验结果表明,大语言模型相较于XML模型从漏洞报告识别漏洞影响库的能力更强。但对于一些描述不清晰、噪声信息干扰较大的漏洞报告,Qwen1.5-14B的信息整合提取能力仍存在不足,例如对于CVE-2023-25313,它漏洞库影响是Maven仓库中的“wiremock-webhooks-extension”库,虽然它漏洞报告中的漏洞描述里已经给出了“WireMock Webhooks Extension”,但由于漏洞描述里大量介绍了该漏洞的利用方式,导致模型未能正确识别出该库,识别结果为“wiremock-studio”。因此未来可以通过对基座模型进行升级,更新为具有更强信息分析抽取能力的大模型来提升本文方法的效果。

RQ2的实验结果表明,中英文漏洞报告相互补充作为输入的识别效果优于单一使用中文或英文漏洞报告。例如对于CVE-2023-25313,它漏洞库影响是Composer仓库中WWBN组织的“Avideo”库,但它的英文报告中只显式给出了“AVideo”,因而只

基于英文报告识别的结果不准确，识别结果为“avideo-server”；而它的中文报告中漏洞的标题为“WWBN AVideo 命令注入漏洞”，清晰给出了该漏洞影响的组织和软件库，因而基于中英文报告相互补充则可以正确识别出漏洞影响库为“WWBN/AVideo”。由于漏洞报告中的信息可能并不完整，多来源相互补充则可以提供更多漏洞影响库的相关信息，进而提升识别效果。因此未来可以考虑纳入更多的数据源作为补充输入，例如将Rapid7[76]、X-Force[77]等来源的漏洞报告与现有的NVD和CNNVD漏洞报告相结合，补充更多有价值的信息，同时重新构建训练模板并微调模型，不断提升漏洞影响库的识别效果。

RQ3的实验结果表明，相较于其他包管理器，漏洞影响库识别方法在Maven包管理器上效果较差。这是因为Maven库的命名长度较长，描述更详细，命名结构模块化，有很多相似库名，因而准确识别Maven库的难度较大。例如对于CVE-2023-40177，它漏洞库影响是Maven仓库中的“xwiki-platform-appwithinminutes-ui”库，虽然漏洞报告中提到了“XWiki Platform”、“AppWithinMinutes”和“custom displayer”等信息，但本文方法并未能准确识别出影响的Maven库，识别结果为“xwiki-platform-app-within-minutes”，与真正的漏洞影响库很相似。因此针对Maven包管理器，未来可以将大模型的识别结果作为初判结果，筛选出与其名称相近的k个Maven库，然后通过Maven社区的安全信息（例如有些Maven库会关联CVE漏洞，有些组织如Spring会定期发布安全公告）来辅助进一步确认，最终识别出真正受该漏洞影响的软件库。

7 效度威胁

（1）内部效度（internal validity）。实验的内部效度威胁可能来自大语言模型的预训练知识以及对于软件库列表的使用。大语言模型在预训练中可能已经见过大量漏洞报告，但由于漏洞报告来源于NVD、CNNVD等漏洞信息网站，而漏洞影响库则来源于Snyk等安全网站，因此大语言模型在预训练中并未同时见过漏洞报告及其影响库，也并未建立二者之间的关联关系，而漏洞报告本就是所有方法的输入，因而大模型在预训练阶段见到漏洞报告并不会对实验结果造成偏差。大模型在幻觉消除阶段需要先收集并构建软件库列表，而基线方法均基于XML模型，也需要提前构建软件库列表，然后将漏洞报告映射到软件库列表中，因此使用软件库列表也不会影响实验的公平性。此外，本文对基线方法采用和之前工作[23,26]一样的参数设置和数据预处理流程，确保实验评估及结果的公平可靠。

（2）结论效度（conclusion validity）。为了避免可能存在的结论效度威胁，本文实验的漏洞报告收集来源为权威的NVD和CNNVD，漏洞影响库的收集来源为学术文献、Github Advisory和Snyk，这些平台提供的漏洞影响库都经过安全研究者或安全专家的评审，最终形成的实验数据集规模也超过之前的工作[23-24,26]。实验没有评估中/英文报告相互补充对于基线方法的效果，这是因为①RQ1中已经证明了在使用相同数据源的情况下，本文的方法优于基线方法；②之前的工作只关注英文NVD报告，多数据源（中文和英文漏洞报告）相互补充也是本文方法的一部分。实验的主体实施流程和评价指标与之前的工作保持一致，并从多个角度对本文的方法进行评估，从而确保结论的正确性。

（3）外部效度（external validity）。本文提出的方法在CNNVD和NVD两个权威漏洞网站提供的漏洞报告上进行了验证，对于新的数据源，本文的方法只需要按照新数据源的格式重新构建训练模板进行微调即可适配。同时本文方法使用的大语言模型为Qwen1.5-14B，随着大语言模型能力的不断提升，可以对本文方法的基础大模型进行升级，进一步提升漏洞影响库的识别效果。因此本文的方法和结论都具有较高的泛化性。

8 总结与展望

漏洞报告是软件供应链安全中的关键信息，从漏洞报告中自动识别漏洞的影响库可以有效地帮助企业减少外部软件引入的风险。针对当前研究只关注NVD英文漏洞报告导致准确率较低且忽略了自动化方法对不同包管理器中漏洞影响库的识别效果存在差异等问题，本文提出了一个基于大语言模型的多来源漏洞影响库识别方法，使用中/英文报告相互补充进行输入增强，在漏洞报告数据集上对大模型进行局部参数微调然后使用文本相似度算法消除大模型的幻觉，并在多个包管理器上进行评估。实验使用了9260份中文/英文漏洞报告及其影响库作为数据集，涵盖了Maven、PyPI、Composer等多种主流包管理器。结果表明本文方法相较于基线方法能

够更有效地从漏洞报告识别漏洞的影响库，对于Maven库的识别性能相比于其他包管理器较低，而对于Golang、Composer、PyPI和NPM库则可以提供更好的安全防护。

未来工作将在以下几个部分开展：(1)在更多数据源上验证本文方法的识别效果；(2)增加对漏洞影响范围的探究，从漏洞报告识别出的漏洞库出发，探究其对下游依赖的影响；(3)将探究企业对于漏洞影响库识别的专有场景(例如企业只关心自己依赖的第三方库)并设计定制化方法。

参 考 文 献

[1] Derr E, Bugiel S, Fahl S, et al. Keep me updated: An empirical study of third-party library updatability on android// Proceedings of the 2017SIGSAC Conference on Computer and Communications Security. New York, USA, 2017: 2187-2200

[2] Li M, Wang W, Wang P, et al. Libd: Scalable and precise third-party library detection in android market//Proceedings of the39th International Conference on Software Engineering. Buenos Aires, Argentina, 2017: 335-346

[3] Liang G, Wu Y, Wu J, et al. Open source software supply chain for reliability assurance of operating systems. Journal of Software, 2020, 31(10): 3056-3073 (in Chinese)
(梁冠宇，武延军，吴敬征，等人.面向操作系统可靠性保障的开源软件供应链.软件学报，2020，31(10):3056-3073)

[4] Li S, Liu J, Wang S, Tian H, et al. Survey on dependency conflict problem of third-party libraries. Journal of Software, 2023, 34(10): 4636-4660 (in Chinese)
(李硕，刘杰，王帅，田浩翔，等人.第三方库依赖冲突问题研究综述.软件学报，2022，34(10):4636-4660)

[5] Kula R G, German D M, Ouni A, et al. Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration. Empirical Software Engineering, 2018, 23: 384-417

[6] Wang Y, Chen B, Huang K, et al. An empirical study of usages, updates and risks of third-party libraries in java projects//Proceedings of the2020 International Conference on Software Maintenance and Evolution. Adelaide, Australia, 2020: 35-45

[7] Black Duck Software, BridgeNorth. The ninth annual future of open sourcesurvey. Industry Report, 2015. Available: https://gfoss.eu/the-ninth-annual-future-of-open-source-survey

[8] Lin J, Liu B, Sadeh N, et al. Modeling Users' mobile app privacy preferences: Restoring usability in a sea of permission settings//Proceedings of the10th Symposium on Usable Privacy and Security. Menlo Park, USA, 2014: 199-212

[9] Wang Y, Wen M, Liu Z, et al. Do the dependency conflicts in my project matter//Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Lake Buena Vista, USA, 2018: 319-330

[10] Vasilakis N, Karel B, Roessler N, et al. BreakApp: Automated, flexible application compartmentalization// Proceedings of the2018 Network and Distributed Systems SecuritySymposium. San Diego, USA, 2018: 1-15

[11] Kula R G, Ouni A, German D M, et al. On the impact of micro-packages: An empirical study of the npm javascript ecosystem. arXiv preprint arXiv:1709.04638, 2017

[12] Liang S, Bracha G. Dynamic class loading in the Java virtual machine//Proceedings of the 1998 ACMSIGPLAN Conference on Object-OrientedProgramming Systems, Languages and Applications.Vancouver, Canada, 1998, 33(10): 36-44

[13] Prabhu Y, Kag A, Harsola S, et al. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising//Proceedings of the2018 World Wide Web Conference. Lyon, France, 2018: 993-1002

[14] Imtiaz N, Khanom A, Williams L. Open or sneaky? Fast or slow? Light or heavy? Investigating security releases of open source packages. IEEE Transactions on Software Engineering, 2022, 49(4): 1540-1560

[15] Imtiaz N, Thorn S, Williams L. A comparative study of vulnerability reporting by software composition analysis tools// Proceedings of the 15th International Symposium on Empirical Software Engineering and Measurement. Bari, Italy, 2021: 1-11

[16] Zahan N, Zimmermann T, Godefroid P, et al. What are weak links in the npm supply chain//Proceedings of the44th International Conference on Software Engineering: Software Engineering in Practice. Pittsburgh, USA, 2022: 331-340

[17] Hiesgen R, Nawrocki M, Schmidt T C, et al. The race to the vulnerable: Measuring the log4j shell incident//Proceedings of the6thNetwork Traffic Measurement and Analysis Conference. Enschede, The Netherlands, 2022: 1-9

[18] Ali S, Anantharaman P, Smith S W. Armor within: Defending against vulnerabilities in third-party libraries//Proceedings of the 2020 IEEE Security and Privacy Workshops. San Francisco, USA, 2020: 291-299

[19] Mao T, Wang X, Chang R, et al. Software supply chain analysis techniques for Java ecosystem. Journal of Software, 2023, 34(6): 2628-2640 (in Chinese)
(毛天宇，王星宇，常瑞，等人.面向Java语言生态的软件供应链安全分析技术.软件学报，2023，34(6):2628-2640)

[20] Tang W, Xu Z, Liu C, et al. Towards understanding third-party library dependency in c/c++ ecosystem//Proceedings of the 37th International Conference on Automated Software Engineering. Rochester, USA, 2022: 1-12

[21] Tang W, Wang Y, Zhang H, et al. Libdb: An effective and efficient framework for detecting third-party libraries in binaries//Proceedings of the 19th International Conference on Mining Software Repositories. Pittsburgh, USA, 2022: 423-434

[22] Zhan Q, Pan S, Hu X, et al. Survey on vulnerability awareness of open source software. Journal of Software, 2024, 35(1): 19-37 (in Chinese)
(詹奇，潘圣益，胡星，等人.开源软件漏洞感知技术综述.软件学

报，2023，35(1):1-19)

[23] Haryono S A, Kang H J, Sharma A, et al. Automated identification of libraries from vulnerability data: Can we do better//Proceedings of the 30thInternational Conference on Program Comprehension. Pittsburgh, USA，2022: 178-189

[24] Chen Y, Santosa A E, Sharma A, et al. Automated identification of libraries from vulnerability data//Proceedings of the 42nd International Conference on Software Engineering: Software Engineering in Practice. Seoul, Republic of Korea, 2020: 90-99

[25] Lin Y, Li Y, Gu M, et al. Vulnerability dataset construction methods applied to vulnerability detection: A survey// Proceedings of the52nd Annual International Conference on Dependable Systems and Networks Workshops. Baltimore, USA, 2022: 141-146

[26] Lyu Y, Le-Cong T, Kang H J, et al. Chronos: Time-aware zero-shot identification of libraries from vulnerability reports// Proceedings of the 45th International Conference on Software Engineering. Melbourne, Australia, 2023: 1033-1045

[27] Chinthanet B, Kula R G, McIntosh S, et al. Lags in the release, adoption, and propagation of npm vulnerability fixes. Empirical Software Engineering, 2021, 26: 1-28

[28] Zhuge J, Gu L, and Duan H. An investigation into the underground industry chain of China's internet information security. Information Security and Communications Privacy, 2012, 9:1-57 (in Chinese)
(诸葛建伟，谷亮，段海新. 中国互联网信息安全地下产业链调查. 信息安全与通信保密，2012，9: 1-57)

[29] Bai J, Bai S, Chu Y, et al. Qwen technical report. arXiv preprint arXiv:2309.16609, 2023

[30] Yang S, Li M, Xu L. Named entity recognition based on large language model and instruction tuning//Proceedings of the 5th International Conference on Big Data & Artificial Intelligence & Software Engineering. Wenzhou, China, 2024: 336-343

[31] Cai Y, Sun H, Huang H Y, et al. Assessing the performance of chinese open source large language models in information extraction tasks. arXiv preprint arXiv:2406.02079, 2024

[32] Foo D, Yeo J, Xiao H, et al. The dynamics of software composition analysis. arXiv preprint arXiv:1909.00973, 2019

[33] Wu J, Xu Z, Tang W, et al. Ossfp: Precise and scalable c/c++ third-party library detection using fingerprinting functions// Proceedings of the 45th International Conference on Software Engineering. Melbourne, Australia, 2023: 270-282

[34] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018

[35] Kombrink S, Mikolov T, Karafiát M, et al. Recurrent neural network based language modeling in meeting recognition// Proceedings of the 12th Annual Conference of the International Speech Communication Association. Florence, Italy, 2011, 11: 2877-2880

[36] Gao J, Lin C Y. Introduction to the special issue on statistical language modeling. ACM Transactions on Asian Language Information Processing, 2004, 3(2): 87-93

[37] Kasneci E, Seßler K, Küchemann S, et al. ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences, 2023, 103: 102274

[38] Chen M, Tworek J, Jun H, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

[39] Zhao W X, Zhou K, Li J, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 2023

[40] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need//Proceedings of the 31st Conference on Neural Information Processing Systems. Long Beach, USA, 2017, 30: 1-11

[41] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners//Proceedings of the2020 Advances in Neural Information Processing Systems, 2020, 33: 1877-1901

[42] Bommasani R, Hudson D A, Adeli E, et al. On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258, 2021

[43] Wei J, Tay Y, Bommasani R, et al. Emergent abilities of large language models. arXiv preprint arXiv:2206.07682, 2022

[44] Gururangan S, Marasović A, Swayamdipta S, et al. Don't stop pretraining: Adapt language models to domains and tasks. arXiv preprint arXiv:2004.10964, 2020

[45] Wu S, Irsoy O, Lu S, et al. Bloomberggpt: A large language model for finance. arXiv preprint arXiv:2303.17564, 2023

[46] Saunshi N, Malladi S, Arora S. A mathematical exploration of why language models help solve downstream tasks. arXiv preprint arXiv:2010.03648, 2020

[47] Radford A, Wu J, Child R, et al. Language models are unsupervised multitask learners. OpenAI Technical Report, 2019. Available: https://storage. prod. researchhub. com/uploads/papers/2020/06/01/language-models.pdf

[48] Fedus W, Zoph B, Shazeer M, et al. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. Journal of Machine Learning Research, 2022, 23(120): 1-39

[49] Rae J W, Borgeaud S, Cai T, et al. Scaling language models: Methods, analysis & insights from training gopher. arXiv preprint arXiv:2112.11446

[50] Thoppilan R, De Freitas D, Hall J, et al. Lamda: Language models for dialog applications. arXiv preprint arXiv: 2201.08239, 2022

[51] Zhang S, Dong L, Li X, et al. Instruction tuning for large language models: A survey. arXiv preprint arXiv: 2308.10792, 2023

[52] Zhang Y, Li Y, Cui L, et al. Siren's song in the AI ocean: A survey on hallucination in large language models. arXiv preprintarXiv:2309.01219, 2023

[53] Adlakha V, BehnamGhader P, Lu X H, et al. Evaluating correctness and faithfulness of instruction-following models for question answering. Transactions of the Association for Computational Linguistics, 2024, 12: 775-793

[54] Liu T, Zhang Y, Brockett C, et al. A token-level reference-free hallucination detection benchmark for free-form text generation. arXiv preprint arXiv:2104.08704, 2021

[55] Min S, Krishna K, Lyu X, et al. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. arXiv preprint arXiv:2305.14251, 2023

[56] Muhlgay D, Ram O, Magar I, et al. Generating benchmarks for factuality evaluation of language models. arXiv preprint arXiv: 2307.06908, 2023

[57] Li J, Cheng X, Zhao W X, et al. Halueval: A large-scale hallucination evaluation benchmark for large language models. arXiv preprint arXiv:2305.11747, 2023

[58] Kaddour J, Harris J, Mozes M, et al. Challenges and applications of large language models. arXiv preprint arXiv: 2307.10169, 2023

[59] Yang Y, Zhou X, Mao R, et al. DLAP: A Deep learning augmented large language model prompting framework for software vulnerability detection. arXiv preprint arXiv: 2405.01202, 2024

[60] Gao L, Dai Z, Pasupat P, et al. Rarr: Researching and revising what language models say, using language models// Proceedings of the 61st Annual Meeting of the Association for ComputationalLinguistics. Toronto, USA, 2023:16477-16508

[61] Levenshtein V I. Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics doklady, 1966, 10(8): 707-710

[62] Salton G, Buckley C. Term-weighting approaches in automatic text retrieval. Information Processing & Management, 1988, 24(5): 513-523

[63] Weiner P. Linear pattern matching algorithms//Proceedings of the 14th Annual Symposium on Switching and Automata Theory. Iowa City, USA, 1973: 1-11

[64] Gupta N, Bohra S, Prabhu Y, et al. Generalized zero-shot extreme multi-label learning//Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. New York,USA, 2021: 527-535

[65] Chen T, Li L, Shan B, et al. Identifying Vulnerable Third-Party Java Libraries from Textual Descriptions of Vulnerabilities and Libraries. arXiv preprint arXiv:2307.08206, 2023

[66] Chen T, Li L, Zhu L, et al. VulLibGen: Generating names of vulnerability-affected packages via a large language model// Proceedings of the 62nd Annual Meeting of the Association for ComputationalLinguistics. Bangkok, Thailand, 2024, 1: 9767-9780

[67] Wu S, Song W, Huang K, et al. Identifying affected libraries and their ecosystems for open source software vulnerabilities// Proceedings of the 46th International Conference on Software Engineering. New York, USA, 2024: 1-12

[68] Hendrycks D, Burns C, Basart S, et al. Measuring massive multitask language understanding. arXiv preprint arXiv: 2009.03300, 2020

[69] Huang Y, Bai Y, Zhu Z, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models// Proceedings of the Advances in Neural Information Processing Systems. New York, USA,2023, 36: 62991-63010

[70] Li H, Zhang Y, Koto F, et al. Cmmlu: Measuring massive multitask language understanding in chinese. arXiv preprint arXiv:2306.09212, 2023

[71] Snyk database. https://security.snyk.io/disclosed-vulnerabilities

[72] GitHub advisory database. https://github.com/advisories

[73] Chatbot Arena Leaderboard Updates (Week 4), https://lmsys.org/blog/2023-05-25-leaderboard/, 2023,5,25

[74] Li C, Farkhoor H, Liu R, et al. Measuring the intrinsic dimension of objective landscapes. arXiv preprint arXiv: 1804.08838, 2018

[75] Hu E J, Shen Y, Wallis P, et al. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021

[76] Rapid7 database. hhttps://www.rapid7.com/db/

[77] X-Force database. https://exchange. xforce. ibmcloud. com/activity/list

XU Jin-Wei, Ph. D. candidate. His research interests include software supply chain security, code review and software process security, etc.

ZHOU Xin, Ph. D., assistant researcher. His research interests include software supply chain security, vulnerability detection, DevSecOps, empirical software engineering and natural language processing, etc.

YANG Yan-Jing, Ph. D. candidate. His research interests include AI robustness, source code security and software supply chain security, etc.

LI Xiao-Kang, M. S. His research interests include software supply chain security and empirical software engineering.

YU Hao-Feng, B. S., engineer. His research interests include software supply chain security and large language model.

YANG Lan-Xin, Ph. D., assistant researcher. His research interests include code review, empirical software engineering, and software supply chain security, etc.

ZHANG He, Ph. D., professor. His research interests include software engineering, DevOps, software supply chain security, empirical software engineering, software security and blockchain security, etc.

WU Yong-Hang, M. S., engineer. His research interests include intelligent software engineering and development of large language models.

Background

This article focuses on the task of identifying affected libraries from vulnerability reports, which is a critical aspect of software supply chain security. Vulnerability disclosure platforms such as the NVD regularly publish vulnerability reports to help security practitioners and users stay informed about the security status of software. However, these reports often do not clearly specify the libraries affected by security vulnerabilities, requiring security experts to manually identify them—a process that is both time-consuming and labor-intensive. Therefore, automating the identification of affected libraries from vulnerability reports can provide more efficient protection for softwaresupply chain security.

Recently a growing number of studies have focused on the automatic identification of affected libraries from vulnerability reports. These works formulate the task as an XML problem and train dedicated XML models to solve it. However, existing approaches are limited to NVD reports, overlooking other valuable sources of vulnerability information. They also ignore how the performance of these approaches varies across different packaging managers. To address these limitations, this article proposes a large language model-based approach for identifying affected libraries from multiple sources. Our approach enhances input representation through mutual complementation of Chinese and English reports, and is evaluated across multiple packaging managers, demonstrating its effectiveness.

Our research group has extensive experience in software supply chain security and software vulnerability analysis. We have previously conducted research on risk detection in the development processes of third-party software repositories, credibility assurance of software bills of materials (SBOMs), and automated vulnerability detection. Our work has been published in leading international conferences and journals, including ICSE, JSS, and JSEP. These achievements provide a strong foundation for this study.

This work was supported by the Natural Science Foundation of Jiangsu Province (No. BK20241195), the National Natural Science Foundation of China (No. 62202219, No. 62302210), the Open Project of Key Laboratory of Industry and Information Technology Ministry for Software Fusion Application and Testing Verification (No. RFT20250301), the Science and Technology Development Program of Two Districts in Xinjiang Province, China (No. 2024LQ03004), and the Beijing Economic and Technological Development Area's 'Cabbage Heart Project'.