软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(9):4018−4036 [doi: 10.13328/j.cnki.jos.006879]

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

源码处理场景下人工智能系统鲁棒性验证方法*

杨焱景[1], 毛润丰[1], 谭 睿[1], 沈海峰[2], 荣国平[1]

[1](南京大学 软件学院, 江苏 南京 210093)

[2](Discipline of Information Technology, Peter Faber Business School, Australian Catholic University, Sydney NSW 2060, Australia)

通信作者: 毛润丰, mrf@smail.nju.edu.cn

摘 要: 人工智能 (artificial intelligence, AI) 技术的发展为源码处理场景下 AI 系统提供了强有力的支撑. 相较于自然语言处理, 源码在语义空间上具有特殊性, 源码处理相关的机器学习任务通常采用抽象语法树、数据依赖图、控制流图等方式获取代码的结构化信息并进行特征抽取. 现有研究通过对源码结构的深入分析以及对分类器的灵活应用已经能够在实验场景下获得优秀的结果. 然而, 对于源码结构更为复杂的真实应用场景, 多数源码处理相关的 AI 系统出现性能滑坡, 难以在工业界落地, 这引发了从业者对于 AI 系统鲁棒性的思考. 由于基于 AI 技术开发的系统普遍是数据驱动的黑盒系统, 直接衡量该类软件系统的鲁棒性存在困难. 随着对抗攻击技术的兴起, 在自然语言处理领域已有学者针对不同任务设计对抗攻击来验证模型的鲁棒性并进行大规模的实证研究. 为了解决源码处理场景下 AI 系统在复杂代码场景下的不稳定性问题, 提出一种鲁棒性验证方法 (robustness verification by Metropolis-Hastings attack method, RVMHM), 首先使用基于抽象语法树的代码预处理工具提取模型的变量池, 然后利用 MHM 源码攻击算法替换变量扰动模型的预测效果. 通过干扰数据和模型交互过程, 观察攻击前后的鲁棒性验证指标的变化量来衡量 AI 系统的鲁棒性. 以漏洞预测作为基于源码处理的二分类典型场景为例, 通过在 3 个开源项目的数据集上验证 12 组 AI 漏洞预测模型鲁棒性说明 RVMHM 方法针对源码处理场景下 AI 系统进行鲁棒性验证的有效性.

关键词: 源码结构化分析; 源码对抗攻击; AI 系统鲁棒性验证

中图法分类号: TP18

中文引用格式: 杨焱景, 毛润丰, 谭睿, 沈海峰, 荣国平. 源码处理场景下人工智能系统鲁棒性验证方法. 软件学报, 2023, 34(9): 4018–4036. http://www.jos.org.cn/1000-9825/6879.htm

英文引用格式: Yang YJ, Mao RF, Tan R, Shen HF, Rong GP. Robustness Verification Method for Artificial Intelligence Systems Based on Source Code Processing. Ruan Jian Xue Bao/Journal of Software, 2023, 34(9): 4018–4036 (in Chinese). http://www.jos.org.cn/1000-9825/6879.htm

Robustness Verification Method for Artificial Intelligence Systems Based on Source Code Processing

YANG Yan-Jing[1], MAO Run-Feng[1], TAN Rui[1], SHEN Hai-Feng[2], RONG Guo-Ping[1]

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

[2](Discipline of Information Technology, Peter Faber Business School, Australian Catholic University, Sydney NSW 2060, Australia)

Abstract: The development of artificial intelligence (AI) technology provides strong support for AI systems based on source code processing. Compared with natural language processing, source code is special in semantic space. Machine learning tasks related to source

* 基金项目: 国家自然科学基金 (62072227, 62202219); 国家重点研发计划 (2019YFE0105500); 江苏省重点研发计划 (BE2021002-2); 南京大学计算机软件新技术国家重点实验室创新项目 (ZZKT2022A25); 海外开放课题 (KFKT2022A09)
本文由“AI 软件系统工程化技术与规范”专题特约编辑张贺教授、夏鑫博士、蒋振鸣副教授、祝立明教授和李宣东教授推荐.
收稿时间: 2022-09-05; 修改时间: 2022-10-13; 采用时间: 2022-12-14; jos 在线出版时间: 2023-01-13
CNKI 网络首发时间: 2023-07-05

code processing usually employ abstract syntax trees, data dependency graphs, and control flow graphs to obtain the structured information of codes and extract features. Existing studies can obtain excellent results in experimental scenarios through in-depth analysis of source code structures and flexible application of classifiers. However, for real application scenarios where the source code structures are more complex, most of the AI systems related to source code processing have poor performance and are difficult to implement in the industry, which triggers practitioners to consider the robustness of AI systems. As AI-based systems are generally data-driven black box systems, it is difficult to directly measure the robustness of these software systems. With the emerging adversarial attack techniques, some scholars in natural language processing have designed adversarial attacks for different tasks to verify the robustness of models and conducted large-scale empirical studies. To solve the instability of AI systems based on source code processing in complex code scenarios, this study proposes robustness verification by Metropolis-Hastings attack method (RVMHM). Firstly, the code preprocessing tool based on abstract syntax trees is adopted to extract the variable pool of the model, and then the MHM source code attack algorithm is employed to replace the prediction effect of the variable perturbation model. The robustness of AI systems is measured by observing the changes in the robustness verification index before and after the attack by interfering with the data and model interaction process. With vulnerability prediction as a typical binary classification scenario of source code processing, this study verifies the robustness of 12 groups of AI vulnerability prediction models on three datasets of open source projects to illustrate the RVMHM effectiveness for robustness verification of source code processing based on AI systems.

Key words: code structure analysis; code adversarial attack; AI system quality evaluation

机器学习理论和技术的发展为 AI 系统进行各种源码文本分析的复杂任务提供了数据和模型层面的强有力支撑. 在诸多关键领域, 由于 AI 模型优异的表现, 一些基于 AI 的软件系统也正在成为不可或缺的组成部分. 但是多数 AI 系统为数据驱动, 且可解释性较差, 容易受到数据干扰. 尽管在训练场景下有良好的表现, 但在真实场景中, 由于可能存在噪音数据、恶意用户甚至是攻击者, 系统在生命周期的各个阶段都可能面临着不同程度的安全风险, 导致无法提供正常的服务[1]. 如果关键安全漏洞识别软件因为模型遭受攻击或欺骗而检测失败, 将会对软件开发和使用造成极大的隐患[2]. 而现有的对于漏洞预测模型的评价体系还是仅仅关注模型的在其特定拆分的训练和测试集上的指标表现, 疏忽了对于真实世界数据集的泛化评估[3]. 同时有研究指出大多数现有方法的训练和测试数据包含重复数据 (高达 68%), 实验表现夸大了报告的结果[4]. 所以实际应用当中 AI 系统的鲁棒性表现是保证软件正常运行的基础, 优异的指标也需要建立在稳定的、鲁棒的模型表现基础之上[5].

但是 AI 软件系统的开发方式也较传统软件系统开发方式不同. AI 系统具有数据驱动的特性 (通过学习观测数据上的模式, 并对新的未知数据做出预测)[6], 其过程中数据、模型、权重的初始化和模型优化训练过程的随机性导致系统在构建时存在大量不确定性, 并使软件测试和衡量软件系统复杂度都具有一定的挑战性[7]. 这就导致现有的鲁棒性评估方法不适用于基于 AI 的软件系统. 在一项调查中近半数的受访工程师感到难以衡量基于 AI 的软件系统的质量[8].

在自然语言处理及图像识别领域, 已有不少研究使用对抗攻击的方法来验证模型的鲁棒性[9]. 然而在源码处理领域内, 目前缺少研究使用对抗攻击对于该领域内的 AI 系统进行鲁棒性验证. 在目前的评价体系当中基于源码处理的各类任务中, 输入大量数据时, 多数 AI 系统在实验场景下的综合表现甚至超过了人类[10−14], 但在真实复杂代码场景下表现却不稳定[4]. 这反映了目前研究对模型的评价体系缺少对应泛化性能的验证. 目前针对源码处理的扰动方法是可以对目前的模型使用造成扰动[15], 但是该方法使用的过多的模型优化过程的梯度信息, 并不适用于测试阶段对于模型的黑盒鲁棒性验证. 需要设计方法对源码处理场景下 AI 系统鲁棒性进行验证并将鲁棒性作为重要的筛选标准之一.

为了避免 AI 系统验证过程复杂的问题, Li 等人[7]将 AI 软件系统测试拆解为 3 个部分测试并进行了实证研究. 我们参考该实证研究拆解源码处理场景下 AI 系统, 重要组件如图 1 所示: (1) 基于源码处理的机器学习模型. (2) 系统搭建所需的驱动数据. (3) 项目部署应用的实现框架. 其中我们使用现有的前后端交互框架进行模型的开发和部署, 在模型的应用层面我们需要使用目前机器学习的支持库 (如 TensorFlow, Keras 等) 进行模型的搭建. 许多基于机器学习的 AI 软件系统都是从大型数据集中学习数据分布特征实现目标功能, 所以使用机器学习方法的项目成功与否很大程度上取决于数据的可用性、质量和管理[16]. 数据组件是驱动整个 AI 系统的核心, 用于训练和测试模型. AI 系统的功能都是通过数据的高维特征关系反向传播到网络权重形成的. 数据组件和机器学习模型

的交互质量决定了 AI 软件系统的核心功能完善程度[17]. 由于交互过程的重要性, 如图 1 所示本文设计鲁棒性验证方法主要针对验证系统拆解后的 AI 模型组件和驱动数据组件交互过程, 并以此来衡量 AI 系统的鲁棒性. 该问题的目前存在主要难点有: (1) 暂缺明确的框架去验证源码处理场景下 AI 系统鲁棒性. (2) 缺少实际的量化指标衡量机器学习模型鲁棒性[18]. (3) 一般对抗攻击方式可能造成源码样本数据失效. 现有研究多数采用对抗扰动方式验证模型和数据交互的鲁棒性[19]. 然而由于源码语义空间的特殊性, 自然语言处理领域常用的文本黑盒对抗攻击方式可能会替换影响代码语义结构的关键符号, 导致源码语义完全不同甚至导致编译失败, 并不能够形成有效的对抗样本. 我们参考自然语言处理领域的研究设计验证框架, 引入基于 Metropolis-Hastings 采样的源码攻击算法构建对抗样本的采样空间, 保留关键语义符号, 在不破坏源码编译结构的前提下进行对抗攻击.

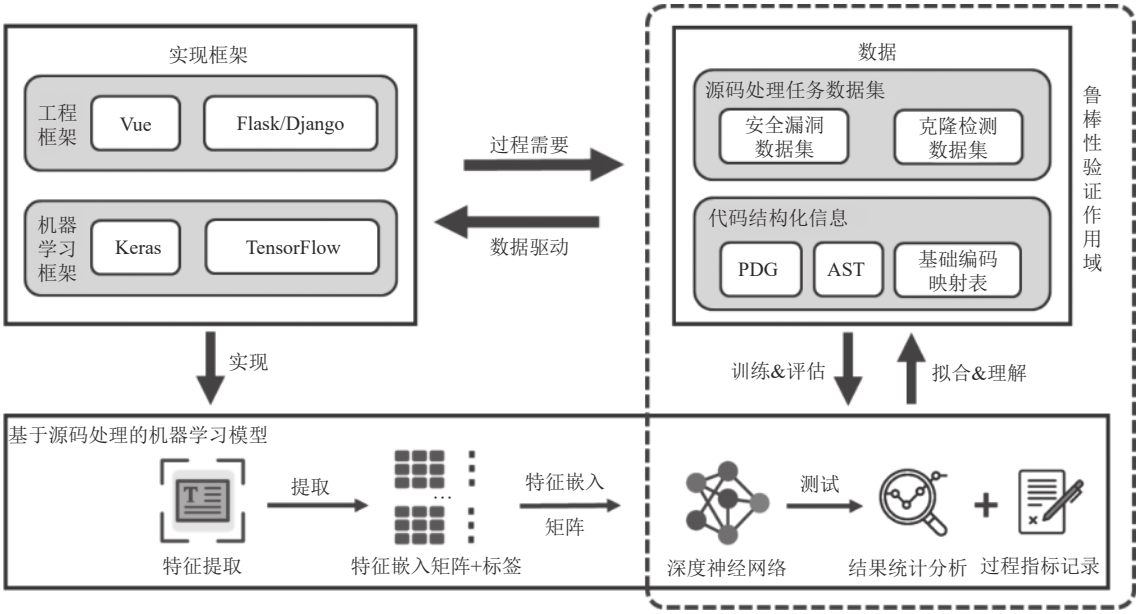

图 1　实践中源码处理场景下 AI 系统组件框架

源码处理当中有多种类别的机器学习任务 (如: 源码克隆检测, 源码补全, 函数命名等) 大多数都是基于 n 阶 Markov 假设构建语言模型进行代码的高维特征嵌入提取之后进行机器学习形成对应的 AI 模型. 针对源码处理场景下 AI 系统在复杂代码场景下的不稳定性问题, 本文选取检测模型和数据集都相对充分的漏洞预测场景作为研究的主要实验场景并提出漏洞预测场景下 AI 系统的鲁棒性验证方法 (robustness verification by Metropolis-Hastings attack method, RVMHM), 通过验证系统拆解后的 AI 模型组件和驱动数据组件交互的鲁棒性来衡量系统的鲁棒性. RVMHM 首先提取每个源码样本的变量表. 然后筛选分类平面附近的样本作为待攻击样本, 利用基于 Metropolis-Hastings 采样的源码攻击算法进行变量采样替换生成对抗样例干扰模型的预测效果. 相较于本文选取的基线工作 (Metropolis-Hastings modifier, MHM) 对抗样本生成算法[20]在漏洞预测场景下存在由于采样扰动方式的限制导致扰动效率缓慢, 以及无法直接处理真实项目中源码切片的问题, RVMHM 框架体系内的源码攻击算法可以通过设置模型的超参数使得模型优先攻击一些易受到干扰的源码样本, 优化 RVMHM 中源码攻击算法在鲁棒性验证体系下的攻击效率. 同时相较于 MHM 提取变量的方式, 本文利用更为通用的源码处理工具 CodeSensor[21] 使得 RVMHM 可以有效地处理漏洞预测场景下的源码切片, 并且只会替换目标源码切片样本的变量名, 不会干扰到样本的语义空间. RVMHM 使得选取的基于 Metropolis-Hastings 采样的源码攻击算法更加适合 AI 系统的鲁棒性验证框架. 通过观察攻击前后方法中设计的各项鲁棒性验证指标上的变化可以对系统鲁棒性进行验证. 通过在漏洞预测场景下的相关实验结果, 展现该方法对于目前基于 n 阶 Markov 假设的源码处理任务的有效性.

本文第 1 节介绍漏洞预测场景下 AI 系统鲁棒性验证方法的相关工作和研究现状. 第 2 节介绍本文所使用和参考的有关技术和知识, 包括 Metropolis-Hastings 采样方法和针对文本的黑盒采样攻击方式. 第 3 节介绍本文构建的漏洞预测场景下 AI 系统鲁棒性验证方法 RVMHM. 第 4 节在 3 个真实项目的数据集 (FFmpeg, LibPNG, LibTIFF) 上进行对比实验说明 RVMHM 在鲁棒性验证上的有效性, 并对攻击算法的超参数进行灵敏度分析. 最后总结全文.

1 源码处理场景下人工智能系统鲁棒性验证方法的相关工作

鲁棒性一般指的是系统 (程序) 在面对输入集变化时, 表现出结果输出集稳定的一种性质. 当前许多开发团队都在调查中表明 AI 的系统组件更难进行鲁棒性评估工作[17]. Alexander 等人提出一套审核方法 evAIa 通过问卷审查 QA 活动的方式, 让 AI 系统的开发透明化, 工程化. 并且该方法通过 ISO25010 印证了其在系统鲁棒性改善上的有效性[22]. 近年来随着对抗攻击在机器学习各领域的兴起[23], Gui 等人[9]提出模型 TextFlint 对最自然语言处理领域内先进的基于深度学习的模型进行了大规模的鲁棒性实证评估. 几乎所有的模型在收到对抗攻击后, 在诸多任务上 (包括情感分类、命名实体识别和自然语言推理等任务) 都表现不同程度的指标波动, 以此来评估模型的鲁棒性, 并且提出将鲁棒性作为自然语言处理模型中重要的指标加入模型的评价体系当中.

目前基于源码分析的绝大多数任务都使用了机器学习方法, 虽然机器学习方法目前在计算机的各个领域都存在很多突出贡献, 但是在真实场景中很多系统中的实用性却不尽如人意. 造成 AI 系统实际应用效果不佳的原因之一就是应用前缺乏对 AI 系统组件中模型的鲁棒性的重视和深入探讨, 导致模型只能在特定场景数据上表现良好, 在模型的效果评测中也仅关心在特定测试集上的性能. 在本文实验选取的漏洞预测场景下, Chakraborty 等人[4]对目前表现较好的模型做了评估, 发现其中 VulDeePecker[24]在报告中实现了 86.9% 的精度. 然而在真实世界的数据集中使用模型 VulDeePecker 时, 由于缺少验证数据和模型之间交互的鲁棒性, 导致其泛化精度受到数据选择的干扰降低到 11.12%, 再训练后, 精度变为 17.68%. 同时该研究还指出目前最先进的漏洞预测模型都是脆弱的, 虽然训练模型学习如何区分漏洞, 但训练范式并没有明确关注增加漏洞和非漏洞样本之间的分离. 因此, 只要稍加扰动, 分类就会变得不稳定[4]. Yamaguchi 等人[25]指出在真实场景下的漏洞要复杂得多, 需要对控制流、数据流、优势关系和代码元素之间的其他类型的依赖关系进行推理, 才能很好地表达程序和提取的特征嵌入之间的语义差距, 而训练集和真实场景下代码语义复杂性的差距, 也造成了在复杂的真实场景下应用模型表现十分不稳定.

这些扰动可以分为噪声扰动和对抗攻击, 其中噪声扰动是不规则无规律的, 无法具体研究. 而对抗攻击是一种试图用恶意输入来欺骗或误导基于机器学习的模型的技术, 该技术在安全领域也得到了广泛的研究. 对抗性规避攻击[26]是最常见的对抗性机器学习攻击类型之一, 发生在机器学习过程的测试阶段, 攻击者试图通过操纵测试数据来逃避检测系统, 从而导致错误的模型分类. Nicholas 等人[27]说明神经网络在面对对抗性实例时很脆弱, 并且评估调查了最近设计的 10 个检测方案得出对抗性的例子比之前认为的更难察觉, 提出了评估防御对抗性扰动的指导原则. Papernot 等人[28]提出了针对黑盒神经网络模型的攻击方法, 该方法对于不同的机器学习模型都有很好的攻击效果. 这种基于黑盒网络的设计使得攻击方式更适合实际生产环境中使用. 在针对源码处理的多种任务当中, 对抗样本就受到编程语言的结构化格式和语法限制. Zhang 等人[20]指出由于代码的语义空间具有离散性, 基于源码的对抗攻击必须遵从一定的编程语言的规则, 否则将会不可通过相应的编译, 进行不了任何一步工作. 所以模型中采用的对抗攻击的方式就会被限制在采样对抗的方式下. Alzan 等人[29]提出 GA 模型采用遗传算法对源码处理模型分类器进行采样攻击, 进一步的研究将梯度信息引入到对抗样本的实例化生成的过程中. Zhang 等人[20]针对源码多分类匹配模型提出 MHM 源码生成对抗模型, 采用了 Markov 随机采样算法从源码中提取变量符号, 对每个例子中的变量名进行替换, 以达到生成对抗实例的目的. Wei 等人[30]提出通过 10 个变异算子来自动生成有效的、语义上保持不变的源代码示例用于模型的测试评估, 同时该团队提出了基于神经元覆盖的方法来指导研究中测试评估样本的生成. Chen 等人[31]提出了 KUAFUDET, 一种二阶段对抗学习增强方法, 通过从训练集中选择和提取特征的训练阶段, 以及利用第 1 阶段训练的分类器的识别阶段, 利用对抗攻击学习提高检测模型的表现. Shu 等人[32]提出了集成漏洞预测模型 Omni. 该模型利用集成防御的思想, 通过超参数优化等方法创建与攻击者预期模

型 (即目标模型) 距离较远的候选模型集成学习漏洞特征并预测, 提高了模型的抗干扰能力.

2 源码处理场景下人工智能系统鲁棒性验证方法的相关技术和知识

本文提出的方法主要基于 Metropolis-Hasting 采样方法和针对文本的黑盒攻击方式 TextBugger. Metropolis-Hasting 采样方法可以获取到复杂目标分布的采样集合, 我们使用这种采样方法去近似采样逼近样本的对抗攻击. TextBugger 可以对文本属性的样本进行黑盒的对抗攻击, 以此来扰动模型的正常处理过程. 下面就相关知识予以介绍.

2.1 Metropolis-Hastings 采样方法

对抗攻击在理论上可以被处理成 3 种形式: (1) 最大化的优化问题[19]; (2) 梯度上升优化的扰动[33]; (3) 采样近似逼近对抗性攻击[29]. 但是由于代码语义空间的特殊性, 我们只能通过采样的方法近似逼近我们理想的对抗样本的分布. 但是由于在自然条件下, 许多分布并不能够通过简单的基本常见分布运算得到, 所以需要对其进行蒙特卡洛采样. 其中一种常见的方法就是接受-拒绝采样, 如果目标分布 $p(x)$ 太复杂, 在程序中没办法直接采样, 那么我们就需要设定一个既定已知分布 $q(x)$ (例如: 高斯分布), 然后按照一定的方法拒绝某些样本, 以达到近似采样 $p(x)$ 的目的.

在 MCMC (Markov chain Monte Carlo) 采样中, 一般情况下目标平稳分布 $\pi(x)$ 和某一个马尔科夫链状态转移矩阵 Q 不满足细致平稳条件. 此时就通过引入新的变量 α 改造细致平稳条件的等式如下:

$$\pi(i)Q(i,j)\alpha(i,j)=\pi(j)Q(j,i)\alpha(j,i) \tag{1}$$

要使得细致平稳条件成立就必须让 α 满足以下两个条件.

$$\begin{cases}\alpha(i,j)=\pi(j)Q(j,i)\\ \alpha(j,i)=\pi(i)Q(i,j)\end{cases} \tag{2}$$

由此可以得到满足对应满足目标矩阵的 P 可以通过任意一个 Markov 状态转移矩阵 Q 乘 α 得到. 这里我们设定 α 为接受率, 取值在 [0, 1] 之间, 即可执行接受-拒绝采样过程. 但是对于采样来讲, 如果 α 取值过小就会导致采样效率过低. 由于等式两边同时扩大并不会影响细致平稳条件成立的条件, 所以 Metropolis-Hasting 改良对应的细致平稳条件如下.

$$\alpha(i,j)=\min\left\{\frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)},1\right\} \tag{3}$$

然后我们通过改良后的细致平稳条件计算接受-拒绝采样的接受率, 然后细化采样过程.

首先通过输入任意 Markov 状态转移矩阵 Q, 平稳分布 $\pi(x)$, 最大状态转移次数 n_1, 采样需要的样本数 n_2. 然后从任意已知简单概率分布中采样得到初始状态 x_0.

从 t=0 循环至最大迭代次数 n_1+n_2-1, 每一个循环中流程如下.

(1) 从条件概率分布 $Q(x|x_t)$ 中采样得到样本 x_*.

(2) 从均匀分布中采样 $u\sim uniform[0,1]$.

(3) 如果 $u<\alpha(x_t,x_*)=\min\left\{\frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)},1\right\}$, 则接受转移 $x_t\to x_*$, 即 $x_{t+1}=x_*$. 如果条件不满足则不接受转移, 重新执行该轮次采样, 即 $t=\max(t-1,0)$.

最后执行完得到的样本集 $(x_{n1},x_{n1+1},\ldots,x_{n1+n2-1})$ 即为我们求得的平稳分布对应的采样集合.

2.2 针对文本的抽样黑盒攻击方式

对抗攻击大体可以分为白盒和黑盒攻击. 由于白盒的攻击方式及只存在模型的自测和验证阶段, 并不适用于模拟实际生产环境的攻击测试. TextBugger[34]作为一个抽样黑盒的攻击方式, 用于生成文本对抗样本. 其优势如下: (1) 有效, 超出从前的模型. (2) 隐蔽, 保留了正常文本的特点. (3) 高效, 运算速度是文本长度的次线性. 由于在黑盒场景下, 没有衡量攻击损失函数的梯度信息. TextBugger 采用如下过程进行采样攻击.

第 1 步: 将文档进行语句级分割, 逐句作为输入, 查看模型的预测结果. 从而过滤掉那些对标签预测不重要的单句, 剩下的语句也使用预测置信度进行排序. 第 2 步: 寻找语句级当中重要的词. 利用去除前后的预测置信度变化, 衡量词语在预测过程中的影响大小. 第 3 步: TextBugger 生成. 对于字母级扰动, 只要随意变动单词中字母的顺序, 就能轻易让模型将单词判定成未识别词, 从而达到攻击的效果. 对于单词级扰动, 可以在单词嵌入层寻找最接近的单词. 在一些字母的变化或使用语义句法相似词替换后, 对抗样本与原样本在感官层面是相似的. 据此提出 5 种对抗样本生成方法: (1) 在单词中间插入空格. (2) 随机删除除了开始和末尾的字母. (3) 交换单词中除了开头和结尾的两个字母. (4) 替换看起来相似的字母. (5) 使用情境感知空间中距离最近的 k 个单词来进行替换. 将候选词生成的对抗样本输入模型, 得到对应类别的置信度, 选取置信度下降最大的词. 如果替换掉单词后的对抗样本与原样本的语义相似度大于阈值, 对抗样本生成成功. 如果未大于阈值, 则选取下一个单词进行修改.

3 漏洞预测场景下 AI 系统鲁棒性验证方法 RVMHM

由于漏洞预测场景下的主题任务多数属于二分类任务, 我们刻画该任务如下所示.

首先我们将实验的使用数据集 D 定义为:

$$D=\{(x_i,y_i)\}_{i=1}^{N},\ \ y_i=0 \text{ or } 1 \tag{4}$$

其中, x_i 是经过模型需要处理后的源码片段编码, 预处理方式可以采用字符解析序列, 抽象语法解析树和程序结构化图特征 (如: 控制流图) 等形式. y_i 是数据集对应预先设置好的向量化标签. 我们分别用 $D(t)$, $D(v)$ 和 $D(e)$ 表示用于训练, 验证和测试的数据集.

$$\zeta(D^{(t)}|C)=-\frac{1}{N^{(t)}}\sum_{i=1}^{N^{(t)}}\sum y_i\odot\log C(x_i) \tag{5}$$

其中, 深度检测模型 $C(x)$ 通过训练获取到数据的高维非线性变换特征, 然后通过最小化特征映射到标签的损失作为目标函数, 然后反向传播误差 ζ 得到具有一定泛化能力的检测模型.

我们提出该场景下 AI 系统验证方法 RVMHM, 框架如图 2 所示, 首先使用基于 AST 的源码解析工具 CodeSensor[21]对源码进行解析并提取每个样本的变量采集表. 由于漏洞预测场景下的数据集多数存在不平衡, 以及数据分布分散的特性[4], 文本级别的采样扰动可能会对模型对于该样本的预测概率有所干扰, 但是由于二分类任务映射函数的特殊性, 可能最终经过公式 (5) 优化后的模型做出的类别判断并没有发生改变, 即存在以下过程.

$$\hat{x}=A(x),C(\hat{x})\neq C(x),y(\hat{x})=y(x) \tag{6}$$

其中, A 代表生成对抗样本的方式, $C(x)$ 表示对应类别的预测概率, $y(x)$ 表示对应最后的预测类别结果. 即对应类别的预测概率受到干扰, 但是由于数据与主题任务的特殊性, 保证语义完整性的微小文本级别扰动并没有使最终的预测结果发生变化, 存在大量的无效扰动.

相较于如果对每个样本进行扰动会导致验证系统工作效率缓慢, 验证系统不能及时反馈一个最有效的评估结果, RVMHM 增加了筛选优先攻击对象的方法避免了在有限时间当中产生很多无效攻击尝试, 优先去攻击那些最有可能扰动的样本提高了限定时间内的扰动性能. 所以我们通过样本特征构建攻击范围的筛选规则, 选取最有可能被干扰样本进行对抗性扰动.

通过筛选构建待攻击对象集合, 从中采样候选和目标变量标识集, 通过 MHM 源码对抗攻击算法采样筛选替换的变量名形成一次一阶 Markov 转换过程, 计算替换后采样概率和该次转换的接受率 α, 根据接受率判断是否执行该次 Markov 转换, 最后再使用分类模型验证是否是一次成功的对抗攻击. 最后结合这些对抗样本对模型进行测试, 观察模型在设计好的鲁棒性验证指标上的变化来衡量系统的鲁棒性. RVMHM 可以将对抗样例生成的过程、鲁棒性验证的结果进行分析和展示.

3.1 数据预处理

(1) 提取基于抽象语法树的源码解析文件

在进行漏洞预测场景下, 现有研究通常采用基于自然语言处理的方式, 然而字符集的识别过程并不能表示完

整的程序结构. 程序当中一些特殊的符号表达的意思也不能够简单利用同级别的字符顺序含义所表示, 如循环结构标志 (while, for), 条件判断结构 (if, else), 或者变量名称. 此类特殊字符存在循环、跳跃或反复调用结构. 本文选取基于抽象语法树的源码分析工具 CodeSensor 进行源码解析工作. 由图 3 可知 CodeSensor 生成的解析文件清晰地表示了源码语义的结构化信息, 每个字符的类别在程序中可以被快速解析. 如变量 a 和 b 无需编写复杂的变量识别程序, 只需要去读取 CodeSensor 解析的结构, 找出 decl 和 param 对应的单位即可快速地提取对应源码样本的变量.

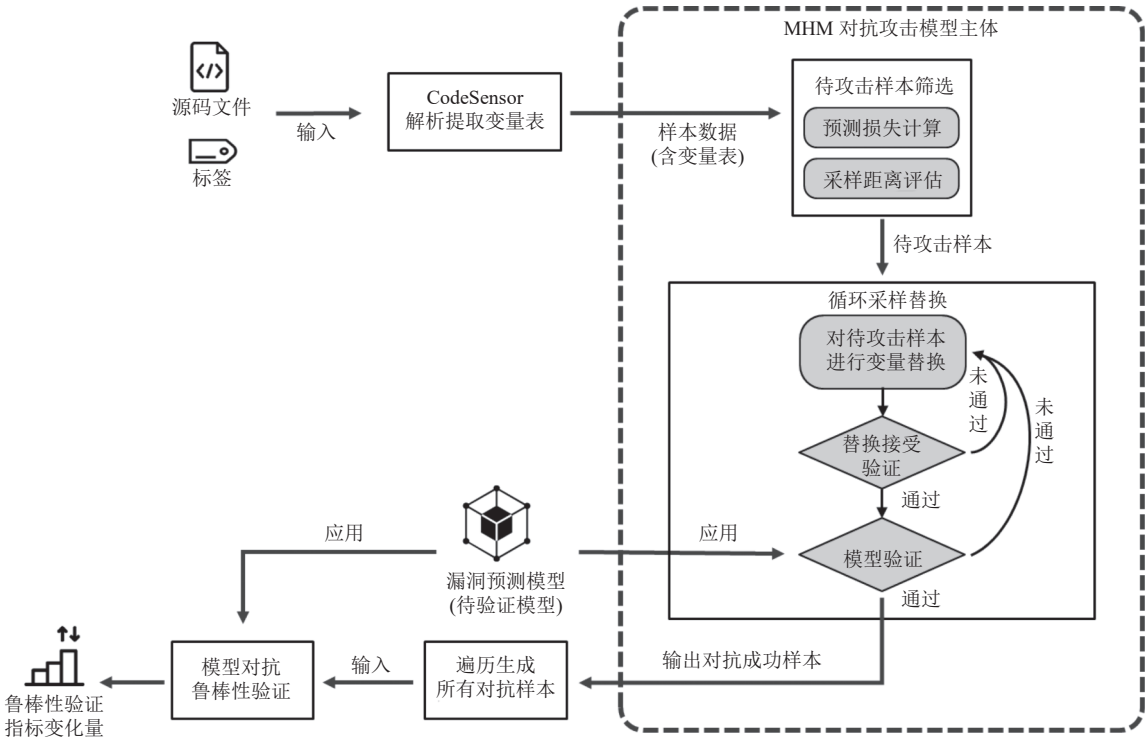

图 2 RVMHM 总体框架图

```
1  int test(int a){
2     int b = a * 2;
3     if(b>0){
4        int c = b - a;
5      }
6     printf("c is %d\n",c)
7     return c;
8  }
```

#Type	Path1	Path2	Depth	Value1	Value2
func	1:0	8:0	0	int	test
params	1:8	0:0	1		
param	1:9	1:14	2	int	a
stmnts	0:0	7:12	1		
decl	2:4	2:17	2	int	b
op	2:10	0:0	2	=	
water	2:12	0:0	2	a	
op	2:14	0:0	2	*	
water	2:16	0:0	2	2	
if	3:4	5:6	2	(b > 0)	
cond	3:7	3:9	3	b > 0	
water	3:7	0:0	4	b	
op	3:8	0:0	4	>	
water	3:9	0:0	4	0	
stmts	3:11	5:6	3		
water	3:11	0:0	4	{	
decl	4:8	4:21	4	int	c
op	4:14	0:0	4	=	
……					

图 3 对应源码及 CodeSensor 解析结果

(2) 提取每个样本的变量表

根据第 1 步源码的解析工作, 找出每个源码样本的变量. 然后进行编码工作对所有样本构建位置编码, 找出对应变量字符的编码. 为每个源码样本的变量和对应编码构建映射关系作为变量采集表. 所有的变量采集表合并可以形成对应的整个数据集的变量采集池. 变量提取的预处理工作方便之后利用变量进行源码对抗攻击.

3.2 基于 Metropolis-Hastings 采样的源码对抗攻击算法

基于 Metropolis-Hastings 采样的攻击算法通过构造对抗攻击目标样本的随机采样近似生成干扰目标主题任务的对抗样本. 基于源码的分析处理任务是一个近似一阶 Markov 链的过程[20], 但是由于源码空间的离散性, 所以针对这个过程, 我们无法直接去衡量或者知道如何在源码空间领域进行最优化问题梯度及优化的目标函数, 利用梯度信息是非常困难的. 这使得基于优化和基于梯度的方法难以实现, 所以只能通过近似采样去逼近理想的对抗攻击.

如果只是对于一般的语言模型, 可以使用类似 TextBugger[33]模型的方法, 随机盲目替换样本中的符号, 构造形成对应的预测模型的对抗样本. 虽然基于源码处理的漏洞预测模型会利用自然语言处理领域内的方法, 提取源码样本的语义信息. 但是代码本身符号的所表述的语义关系不能够简单地利用相同字符间顺序进行语义表述. 如图 4, 通过利用 Pythontutor 工具[35]生成随机盲目替换后的源码执行过程. 不难发现, 如果利用 TextBugger 模型随机盲目改变符号构建对抗样本, 尽管会使得原有语义和原本样本差距变大造成对抗, 但是会改变程序的执行轨迹. 甚至可能导致源码样本无法通过编译过程, 这样的对抗样本从执行的检测过程上来看就是无效样本.

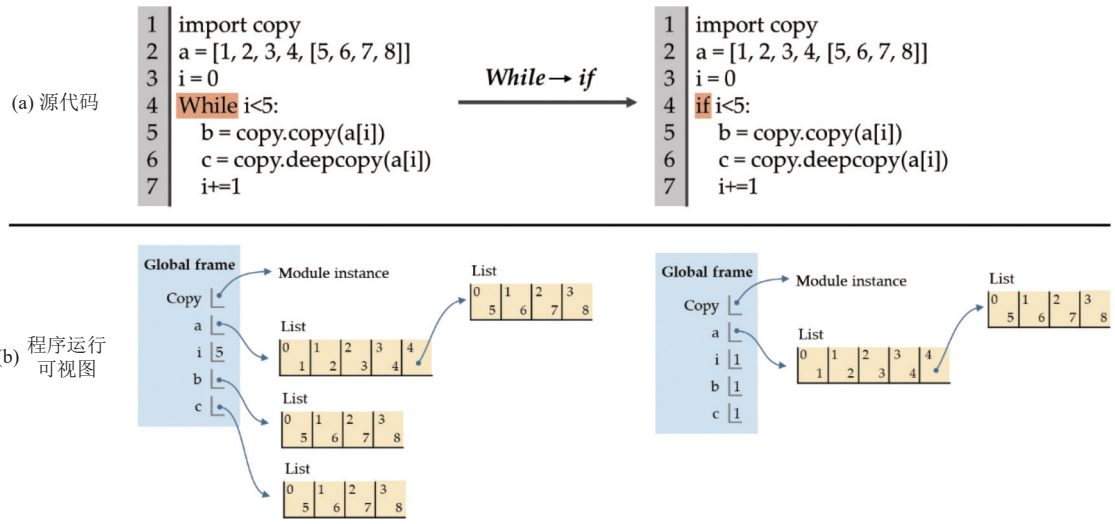

图 4 错误替换会影响程序的执行过程

为了提高对抗源码生成算法工作的效率, 算法利用待验证模型的预测层, 计算样本到分类检测平面的预测损失, 本文采用 L_2 范式衡量该损失, 对待测试的源码进行初次筛选. 参考自然语言处理类别其他的基于文本的攻击过程, 攻击检测平面周围的样本, 将更有可能攻击成功.

$$X_{\text{待攻击样本}} = \{x_1, x_2, \ldots\}\left(x_n \sim \left\{x \middle| \frac{|x\cdot \vec{n}|}{|\vec{n}|} < range\right\}\right) \tag{7}$$

其中, X 表示待攻击样本的集合, *range* 表示设定的筛选攻击范围, 超过这个距离的样本将不会收到攻击. 通过攻击迭代次数上限 *maxiter* 限制算法的运行时间, 达到最短时间攻击最多样本的目标. 获取待攻击源码样本集合之后, 利用待攻击样本的源码变量采集表挑选变量进行采样替换, 在不影响程序执行过程的情况下构建对抗样本.

算法把对抗性实例生成看作是一个抽样问题. 首先定义对抗性样本为给定一个训练良好的主题模型 C 和一个标记的数据对 (x, y), 其中 C 正确地将 x 分类到 y, 对抗例子集对 x 的定义如下.

$$A(x)=\{\hat{x}|\hat{x}\in\varepsilon\wedge\forall i\in I, E(i|\hat{x})=E(i|x)\wedge\arg\max(y)\neq\arg\max C(\hat{x})\} \tag{8}$$

对抗性攻击看作抽样的过程, 目标概率分布 可以被定义为:

$$\pi(x)\propto(1-C(x)[y])\cdot X_1\ldots X_k \tag{9}$$

$C(x)[y]$ 使用主题模型 C 预测到 y 这类的概率, 对应 $X_1\ldots X_k$ 是对目前任务的若干约束: 如语法约束, 词法约束等. 之后基于 Metropolis-Hastings 采样方法进行变量名的采样更新. MHM 变量名采样替换算法, 首先会利用受害者模型 C 选取漏洞预测场景下易受到攻击的样本. 然后将一对正确分类的数据对 (x, y), 输出一系列对抗样例 . 这些对抗样例应该具备如下特征: (1) 能够误导主题模型 C. (2) 与源码中的语义空间一致, 即不会出现编译性质的错误. (3) 在给定输入和输出结果之后能够与原样例输出结果一致, 即不会改变源码本身所表示的语义.

该采样算法是一种基于经典的 Markov 转换链的蒙特卡洛采样方法. 给定分布和转换方法, M-H 采样方法可以从分布中采样出期望的样本. 对于漏洞预测场景下, MHM 算法在一次迭代过程中包括 3 个阶段.

阶段 1: 选择源命名标识符 s: 收集源码片段 x 中所有变量和函数的定义和声明, 形成集合 $S(x)$. 然后从中提取源标识符 s, 每个原标识符被提取的概率相等.

阶段 2: 选择目标命名标识符 t: 目标标识符从候选标识符中产生. 候选标识符集 $T(x)$ 是从整体词汇集 V 中生成的. $T(x)$ 中的元素必须满足上述标识符的词汇规则, 且不能出现在 $S(x)$ 中. 这两个约束保证标识符重命名后的源码段仍然满足对抗子集中定义. 重命名的标识符来自构建整体变量标识全集但不属于当前源码样本标识集的其他变量标识符, 整体变量标识符是包含于整体词汇集 V 的. 这种变量替换即可以保证目标源码仍然可以通过原有的项目编译, 而且替换的词汇不会引入新的未知变量作为毫无意义的干扰噪声.

同时阶段 1 和阶段 2 会生成一个转换方案将 x 中的 s 重命名为 t, 以此来形成 x' . 那么该过程的转移概率就可以被定义为:

$$Q(x'|x)\propto\Gamma\{s\in S(x)\wedge t\in T(x)\}\cdot P_{s(x)}(s)\cdot P_{T(x)}(t)\ \text{ where }\ S(x)\cap T(x)=\varnothing \tag{10}$$

其中, $P_{s(x)}(s)$ 和 $P_{T(x)}(t)$ 是从 $S(x)$ 和 $T(x)$ 中分别采样得到 s 和 t 的均匀分布. 而且要求 $S(x)$ 和 $T(x)$ 是毫无交集的分布.

阶段 3: 根据转移概率 Q 该次转换的接受率 α . 对于每一次迭代替换过程, 该算法会基于转移分布 $Q(x'|x)$ 提出一个转移方案. 该方案被接受通过的概率如下:

$$\alpha(x'|x)=\min\{1,a^*\}=\min\left\{1,\frac{\pi(x')Q(x|x')}{\pi(x')Q(x'|x)}\right\}=\min\left\{1,\frac{(1-C(x')[y])\cdot P_{s(x')}(s)\cdot P_{T(x')}(t)}{(1-C(x)[y])\cdot P_{s(x)}(s)\cdot P_{T(x)}(t)}\right\} \tag{11}$$

如果通过检验, 那么样本就会由 x 转换到 进行采样. 否则算法就会停留在当前状态不进行采样.

整体算法可以被描述为算法 1.

算法 1. MHM 源码对抗攻击算法.

输入: 分类模型 model; 筛选为攻击对象的源码样本数据对 (x, y); 最大迭代次数 *maxiter*; 候选判别个数 *n_candidate*; 接受阈值;

输出: 对抗攻击样本, 附加对抗样本的测试数据集.

1. 初始化 $x0$ (源码材料→embedding), t=1, *count*=0

While (是否遍历攻击对象)

2. 进行第 *count* 个对象攻击

   While t < *maxiter* (迭代次数):

3. 通过扫描源码生成对应的标识符候选集合 S

4. 在 S 中进行采样

5. 通过源码扫描生成代替换候选集合 T

6. 从 T 中进行采样 (由于约束, 以上采样步骤采样的时候所有的候选单位被采样的概率是相等的)

7. 执行替换
8. 在正向转移概率 Q 和反向转移概率 Q' 的基础上计算执行替换接受率
9. 根据接受阈值和接受率的大小关系反馈是否执行 Markov 转换链
10. If 执行转换链
使用模型 C 对其进行验证
If $\arg\max(y) \neq \arg\max C(\hat{x})$ 即攻击成功 break
Else
不执行转换链, 退回更换前的采样状态, 重新进行采样
11. t=t+1
end While
12. $count$ = $count$+1
end While

利用 MHM 变量名采样换名算法计算转换接收概率, 然后根据接受率和接受阈值的大小关系筛选能够执行转换的样本形成新的源码对抗样本. 整个变量替换产生对抗样本的过程是一个累计替换、叠加影响的过程, RVMHM 在执行的过程中将替换的变量条目和当时变量的取值情况详细记录, 方便进行进一步的攻击日志分析和后续研究.

3.3 多指标鲁棒性验证

经过对于鲁棒性的调查研究, 现有多数的评价模型还是通过展示攻击前后, 或更换输入集前后模型参数, 通过各项指标上的变化量, 综合展示模型的鲁棒性. 所以在本文中挑选了以下 3 个指标, 通过其变化量综合评估模型鲁棒性.

(1) $F1$-$score$

$F1$-$score$ 是统计学中用来衡量二分类模型精确度的一种指标, 用于测量不均衡数据的精度. 它同时兼顾了分类模型的精确率 $precision$ 和召回率 $recall$. $F1$-$score$ 可以看作是模型精确率和召回率的一种加权平均, 它的最大值是 1, 最小值是 0.

$$F1\text{-}score = 2\frac{recall \times precision}{recall + precision} \tag{12}$$

(2) $AUPRC$

由于漏洞预测的问题在网络检测方向上, 可以被抽象成一个二分类的问题. 所以衡量二分类器本身好坏与否也非常重要. 在综合考虑漏洞预测场景大部分数据集不平衡的特性, 我们选取 (area under precision recall curve, $AUPRC$)[36] 用于评估模型工作的可靠性. 在这些不平衡或倾斜的数据集上, PRC 相较于 (receiver operating characteristic, ROC) 可以突出 ROC 曲线中丢失的性能差异[37]. 所以绝大多数算法在面对不平衡数据集时选取 PRC 作为性能的一般衡量标准. 在本文中我们使用黎曼和的形式近似计算 $AUPRC$.

$$AUPRC = \int PRC = \sum_{i=2}^{n} (recall_i - recall_{i-1}) \times precision_i \tag{13}$$

(3) 存在漏洞类的相关指标

由于漏洞预测场景下的数据集大多数都存在由不平衡的情况. 也符合该场景真实运行所得的数据集, 毕竟漏洞大多数存在隐蔽性和欺骗性. 如果存在很明显的漏洞, 很容易就被发现且及时修复了. 所以存在漏洞类别的数据量要远远小于无漏洞类别的数据量.

$$RecallVul = \frac{TP_{\text{Vulner}}}{TP_{\text{Vulner}} + FN_{\text{Vulner}}} \tag{14}$$

所以本文更加关注如上所示漏洞预测类别的相关指标, 这些指标的变化将会更好地体现漏洞预测模型的性能.

4 实验设计与结果分析

实验设计主要展示了漏洞预测场景下, 基于机器学习的漏洞预测模型 (2 种特征提取算法和 6 种检测网络) 在 3 个真实项目的数据集上进行鲁棒性验证的结果. 通过对比扰动方法 MHM[20]进行实验. 相较于基线 MHM 扰动算法无差别的攻击每一个源码样本, 会产生大量文本级别无效的扰动过程, RVMHM 在使用 M-H 采样源码攻击分布时会通过源码置信度的分布优先去攻击那些最有可能被扰动的源码样本. 提高了漏洞预测场景下验证模型鲁棒性的效率. 同时 RVMHM 采用了更加一般的基于抽象语法树的源码处理工具 CodeSensor, 相较于 MHM 原始扰动算法, 使得 M-H 采样攻击成功作用于无法单独进行编译代码切片.

实验结果说明了本文提出的鲁棒性验证模型 RVMHM 的有效性. 通过结果展示在真实数据集上相关指标表现说明尽管参与验证的漏洞预测模型在训练场景下能够取得较好的指标表现, 但是在真实场景下 (即包含了对抗攻击和噪声扰动的检测情况下) 模型的表现并不稳定[38]. 本文涉及的实验工作的将在 GitHub 仓库 (https://github.com/Yang-Yanjing/Robustness_for_SVD.git) 开放源码获取以供读者后续科研以及鲁棒性验证使用.

4.1 实验数据

为了尽可能模拟模型在真实场景下的表现情况, 在对比前人研究的情况下[21,39], 本文参考漏洞预测场景已经存在的研究, 专注于对鲁棒性验证方法效果进行评估, 我们选取了漏洞类别与无漏洞类别不平衡比例小于 1:50 的数据集以减少不平衡数据集对于实验预测模型训练的影响. 根据上述原则选取 3 个在 GitHub 上存在的真实的项目作为实验数据集: (1) FFmpeg; (2) LibPNG; (3) LibTIFF. FFmpeg 是一套可以用来记录、转换音频、视频, 利用内置算法将它们转化成计算机可以操作、修改的数据流. 同时该项目提供了录制、转换以及流化音视频的完整解决方案. LibPNG 是一款基于 C 语言编写的底层的读写 PNG 文件的库, 该项目不受到平台的限制, 利用该项目可以获取 PNG 文件的每一行像素, 方便图像处理算法的进一步实施. LibTIFF 是一个用来读写标签图像文件格式的库, 项目以源码方式分发, 并且构建不受平台限制, 利用该项目可以很方便地利用命令行就能处理 TIFF 文件. 通过收集这些项目对应有缺陷的版本, 并对源码进行函数切片形成实验数据集, 具体数据集情况如表 1.

表 1 数据集信息汇总

数据集	漏洞函数源码文件总数	无漏洞函数源码文件总数
FFmpeg	5 553	250
LibPNG	578	46
LibTIFF	732	124

实验所使用的漏洞预测模型架构如图 5 所示由特征提取部分和深度网络预测部分构成.

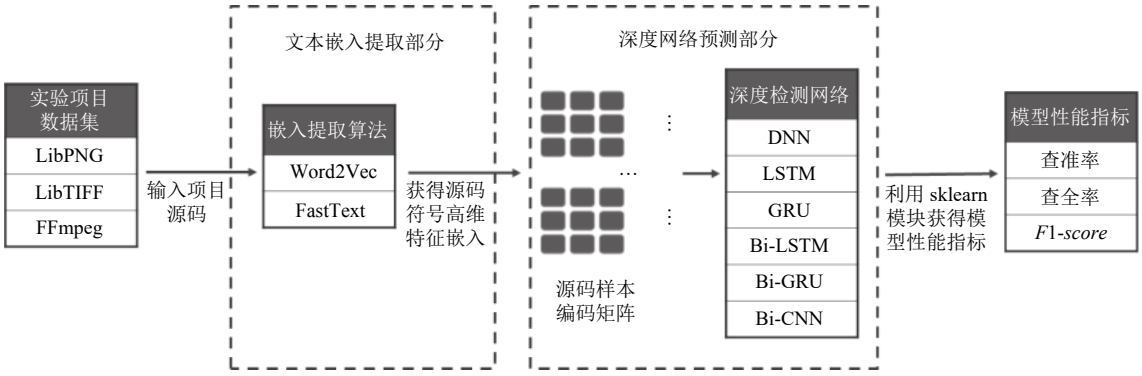

图 5 实验使用漏洞预测模型基本架构

4.2 漏洞预测模型对抗鲁棒性验证

在数据集 LibPNG, LibTIFF, FFmpeg 上本文挑选了表现较为优良的嵌入算法和检测网络组合成 12 组模型进

行扰动攻击, 选取攻击方法设置超参数为表 2 所示. 为了保证模型在统计意义上相较于基线算法的优势, 本文参考文本攻击工作[20,29]和观察重复性实验攻击记录的差异, 设计对每组漏洞预测模型进行 10 次对抗性攻击, 取攻击后指标变化量的平均值作为实验的统计结果.

表 2 MHM 模型超参数设计

数据集	执行时间 (min)	最大攻击迭代次数 (*maxiter*)	候选替换样本个数 (*n_candidate*)	攻击范围 (*range*)
LibPNG	4	300	30	0.2
LibTIFF	6	200	30	0.2
FFmpeg	10	200	30	0.1

因为 LibPNG 相对于其余两组数据集, 对应的数据量相对较少, 所以综合考虑攻击时间, 本文选取 LibPNG 的最大迭代次数 *maxiter* 相对较大, 因为 FFmpeg 的数据量庞大, 所以本文相对于其他两个数据集设置对应 FFmpeg 的筛选攻击范围 *range* 相对较小. 执行时间根据数据集大小综合选定, 避免验证执行时间过长导致效率偏低.

如表 3 所示, 为数据集 LibPNG 上各模型经过 MHM 源码攻击模型的扰动之后, 在鲁棒性指标评价上的变化, 可以看到原本高准确率的分类模型, 攻击之后都存在有一定的指标浮动, 鲁棒性普遍存在一定的影响. 由于 LibPNG 数据集本身数据量的少和正负类别数据不平衡的特性, 部分数据可能趋向 0. 通过详细筛查攻击成功的对象发现此类变化是符合实验过程的.

表 3 LibPNG 数据集上模型鲁棒性验证实验结果

嵌入提取算法	扰动算法	检测网络	*F1-score*		*AUPRC*		*RecallVul*	
			攻击前后	变化量	攻击前后	变化量	攻击前后	变化量
FastText	RVMHM	Bi-GRU	0.88→0.87	−0.01	0.476→0.474	−0.002	0.44→0.43	−0.01
		Bi-LSTM	0.88→0.87	−0.01	0.476→0.473	−0.003	0.44→0.43	−0.01
		DNN	0.92→0.9	−0.02	0.607→0.563	−0.044	0.44→0.33	−0.11
		GRU	0.95→0.94	−0.01	0.844→0.838	−0.005	0.33→0.32	−0.01
		LSTM	0.9→0.9	0.00	0.417→0.411	−0.006	0.56→0.55	−0.01
		Text-CNN	0.91→0.87	−0.04	0.934→0.901	−0.033	1.0→0.89	−0.11
	MHM	Bi-GRU	0.88→0.88	0.00	0.476→0.476	0.000	0.44→0.43	−0.01
		Bi-LSTM	0.88→0.88	0.00	0.476→0.476	0.000	0.44→0.44	0.00
		DNN	0.92→0.92	0.00	0.607→0.607	0.000	0.44→0.44	0.00
		GRU	0.95→0.94	−0.01	0.844→0.844	0.000	0.33→0.33	0.00
		LSTM	0.9→0.89	−0.01	0.417→0.417	0.000	0.56→0.56	0.00
		Text-CNN	0.91→0.87	−0.04	0.934→0.902	−0.031	1.0→0.89	−0.11
Word2Vec	RVMHM	Bi-GRU	0.94→0.93	−0.01	0.450→0.362	−0.089	0.11→0.1	−0.01
		Bi-LSTM	0.88→0.87	−0.01	0.372→0.364	−0.007	0.22→0.21	−0.01
		DNN	0.81→0.79	−0.02	0.611→0.606	−0.006	0.89→0.89	0.00
		GRU	0.9→0.88	−0.02	0.394→0.346	−0.048	0.33→0.22	−0.11
		LSTM	0.9→0.89	−0.01	0.449→0.442	−0.007	0.44→0.43	−0.01
		Text-CNN	0.69→0.55	−0.14	0.080→0.067	−0.013	0.33→0.32	−0.01
	MHM	Bi-GRU	0.94→0.94	0.00	0.450→0.448	−0.002	0.11→0.11	0.00
		Bi-LSTM	0.88→0.88	0.00	0.372→0.372	0.000	0.22→0.21	−0.01
		DNN	0.81→0.81	0.00	0.611→0.606	−0.006	0.89→0.88	−0.01
		GRU	0.9→0.9	0.00	0.394→0.394	0.000	0.33→0.33	0.00
		LSTM	0.9→0.89	−0.01	0.449→0.443	−0.006	0.44→0.43	−0.01
		Text-CNN	0.69→0.49	−0.20	0.080→0.062	−0.019	0.33→0.33	0.00

通过观察图 6 和表 3 可以发现在该数据集下, 可以发现普遍我们的 RVMHM 比现有先进的扰动算法 MHM 在性能上较好, 同时可以发现一些较为先进的网络如 Text-CNN 在面对扰动时变化量较大, 可以说明该检测网络虽然在实验阶段其表现良好, 但是在面对对抗攻击时波动较大, 鲁棒性较差.

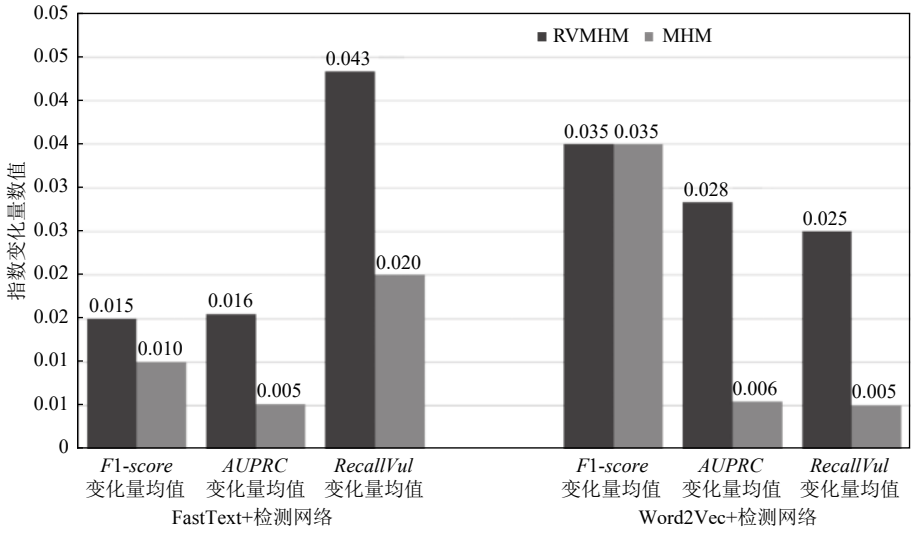

图 6 LibPNG 数据集上 RVMHM 与 MHM 验证指标波动对比

如表 4 所示, 为数据集 LibTIFF 上各模型经过 RVMHM 验证之后, 在鲁棒性指标评价上的普遍存在较大变化, 如图 7 所示相比于 MHM 扰动算法, RVMHM 在该数据集上对模型的 3 个鲁棒性指标扰动更加明显.

表 4 LibTIFF 数据集上模型鲁棒性验证实验结果

嵌入提取算法	扰动算法	检测网络	*F*1-*score*		*AUPRC*		*RecallVul*	
			攻击前后	变化量	攻击前后	变化量	攻击前后	变化量
FastText	RVMHM	Bi-GRU	0.86→0.85	−0.01	0.485→0.405	−0.080	0.19→0.15	−0.04
		Bi-LSTM	0.81→0.77	−0.04	0.639→0.627	−0.012	0.59→0.58	−0.01
		DNN	0.94→0.91	−0.03	0.836→0.817	−0.019	0.67→0.63	−0.04
		GRU	0.85→0.82	−0.03	0.420→0.389	−0.031	0.37→0.3	−0.07
		LSTM	0.77→0.74	−0.03	0.542→0.254	−0.289	0.63→0.04	−0.59
		Text-CNN	0.81→0.75	−0.06	0.491→0.432	−0.059	0.56→0.44	−0.12
	MHM	Bi-GRU	0.86→0.86	0.00	0.485→0.479	−0.006	0.19→0.15	−0.04
		Bi-LSTM	0.81→0.79	−0.02	0.639→0.632	−0.006	0.59→0.59	0.00
		DNN	0.94→0.91	−0.03	0.836→0.817	−0.019	0.67→0.63	−0.04
		GRU	0.85→0.83	−0.02	0.420→0.396	−0.024	0.37→0.33	−0.04
		LSTM	0.77→0.76	−0.01	0.542→0.530	−0.012	0.63→0.59	−0.04
		Text-CNN	0.81→0.74	−0.07	0.491→0.386	−0.105	0.56→0.37	−0.19
Word2Vec	RVMHM	Bi-GRU	0.88→0.81	−0.07	0.550→0.454	−0.096	0.41→0.22	−0.19
		Bi-LSTM	0.82→0.8	−0.02	0.670→0.669	−0.001	0.56→0.55	−0.01
		DNN	0.91→0.87	−0.04	0.881→0.871	−0.010	0.89→0.81	−0.08
		GRU	0.82→0.74	−0.08	0.335→0.254	−0.081	0.26→0.04	−0.22
		LSTM	0.79→0.71	−0.08	0.553→0.498	−0.055	0.63→0.37	−0.26
		Text-CNN	0.75→0.59	−0.16	0.220→0.146	−0.074	0.11→0.0	−0.11
	MHM	Bi-GRU	0.88→0.85	−0.03	0.550→0.506	−0.045	0.41→0.3	−0.11
		Bi-LSTM	0.82→0.79	−0.03	0.670→0.617	−0.052	0.56→0.48	−0.08
		DNN	0.91→0.87	−0.04	0.881→0.872	−0.009	0.89→0.85	−0.04
		GRU	0.82→0.79	−0.03	0.335→0.302	−0.032	0.26→0.15	−0.11
		LSTM	0.79→0.76	−0.03	0.553→0.535	−0.018	0.63→0.56	−0.07
		Text-CNN	0.75→0.59	−0.16	0.220→0.146	−0.074	0.11→0.0	−0.11

如表 5 所示, 为数据集 FFmpeg 上各模型经过 MHM 源码攻击模型的扰动之后, 在鲁棒性指标评价上的变化, 因为 FFmpeg 采集的样本数 3 个样本中最高的, 对应的也更能反应各个模型在真实情况下, 收到对抗攻击

扰动后, 模型指标的变化量, 以此来反映各组模型的鲁棒性. 由于真实模型的应用场景中存在着大量的扰动条件, 模型优良表现更需要建立在鲁棒稳定的基础之上. 图 8 也体现在真实大量的数据集上我们算法较基线算法的优势.

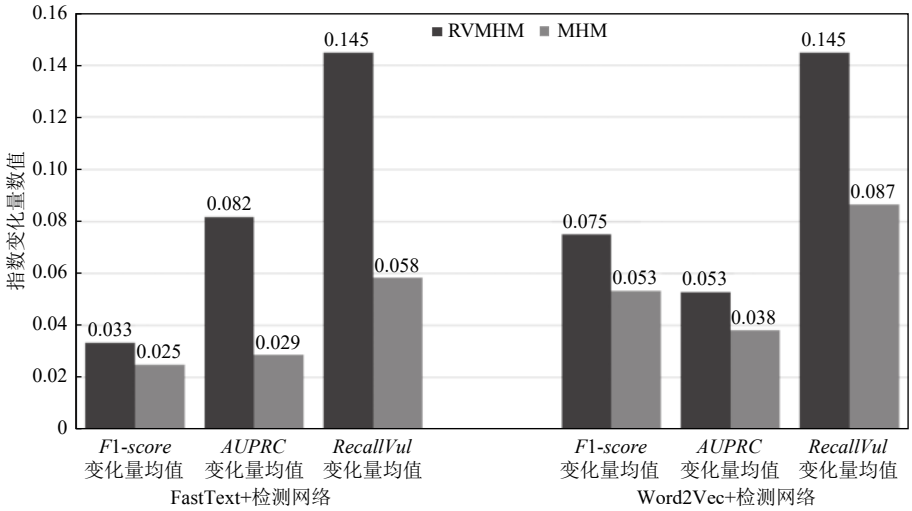

图 7 LibTIFF 数据集上 RVMHM 与 MHM 验证指标波动对比

表 5 FFmpeg 数据集上模型鲁棒性验证实验结果

嵌入提取算法	扰动算法	检测网络	*F1-score*		*AUPRC*		*RecallVul*	
			攻击前后	变化量	攻击前后	变化量	攻击前后	变化量
FastText	RVMHM	Bi-GRU	0.91→0.9	−0.01	0.236→0.231	−0.005	0.5→0.48	−0.02
		Bi-LSTM	0.91→0.88	−0.03	0.291→0.277	−0.014	0.69→0.62	−0.07
		DNN	0.98→0.97	−0.01	0.814→0.809	−0.005	0.65→0.58	−0.07
		GRU	0.93→0.87	−0.06	0.380→0.323	−0.057	0.71→0.56	−0.15
		LSTM	0.92→0.9	−0.02	0.359→0.331	−0.029	0.73→0.6	−0.13
		Text-CNN	0.92→0.89	−0.03	0.305→0.287	−0.019	0.56→0.55	−0.01
	MHM	Bi-GRU	0.91→0.9	−0.01	0.236→0.235	−0.001	0.5→0.49	−0.01
		Bi-LSTM	0.91→0.9	−0.01	0.291→0.290	−0.001	0.69→0.68	−0.01
		DNN	0.98→0.98	0.00	0.814→0.812	−0.002	0.65→0.62	−0.03
		GRU	0.93→0.92	−0.01	0.380→0.371	−0.009	0.71→0.71	0.00
		LSTM	0.92→0.92	0.00	0.359→0.359	0.000	0.73→0.73	0.00
		Text-CNN	0.92→0.89	−0.03	0.305→0.287	−0.019	0.56→0.55	−0.01
Word2Vec	RVMHM	Bi-GRU	0.91→0.9	−0.01	0.236→0.232	−0.004	0.5→0.5	0.00
		Bi-LSTM	0.91→0.88	−0.03	0.291→0.277	−0.014	0.69→0.62	−0.07
		DNN	0.98→0.97	−0.01	0.814→0.809	−0.005	0.65→0.58	−0.07
		GRU	0.93→0.86	−0.07	0.380→0.321	−0.059	0.71→0.56	−0.15
		LSTM	0.92→0.9	−0.02	0.359→0.331	−0.029	0.73→0.6	−0.13
		Text-CNN	0.92→0.89	−0.03	0.305→0.287	−0.019	0.56→0.56	0.00
	MHM	Bi-GRU	0.91→0.9	−0.01	0.236→0.235	−0.001	0.5→0.49	−0.01
		Bi-LSTM	0.91→0.9	−0.01	0.291→0.290	−0.001	0.69→0.69	0.00
		DNN	0.98→0.97	−0.01	0.814→0.812	−0.002	0.65→0.62	−0.03
		GRU	0.93→0.92	−0.01	0.380→0.369	−0.011	0.71→0.71	0.00
		LSTM	0.92→0.92	0.00	0.359→0.356	−0.004	0.73→0.73	0.00
		Text-CNN	0.92→0.89	−0.03	0.305→0.287	−0.019	0.56→0.55	−0.01

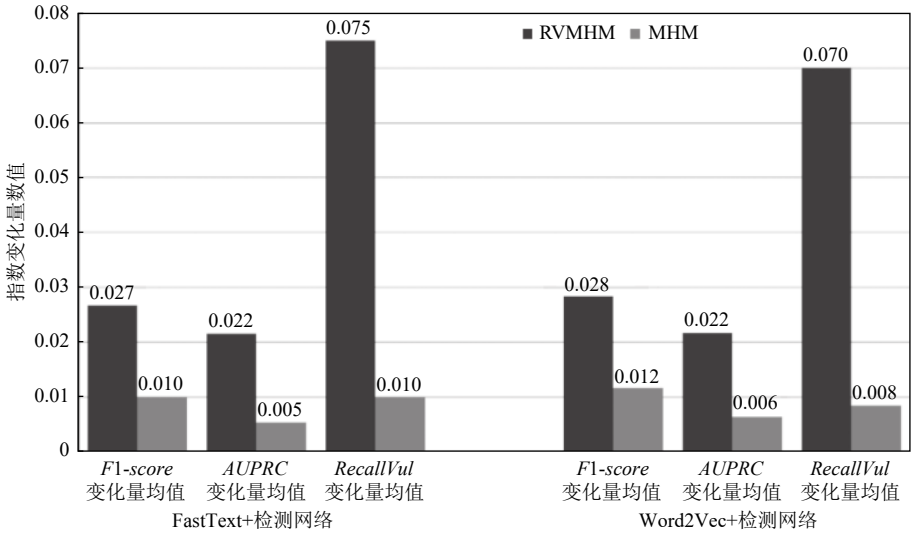

图 8 FFmpeg 数据集上 RVMHM 与 MHM 验证指标波动对比

由于不同的漏洞预测模型使用的网络和特征提取方式不同, 源码样本进入预测模型后编码得到高维特征嵌入向量也有所区别, 与其他对应的源码文本对抗工作[19,20,29]相同, RVMHM 在尝试进行变量名替换之后对于不同模型影响的性能差异也有所不同, 是一个文本对抗工作普遍存在的差异现象. 同时我们关注到 RVMHM 在面对复杂大型的数据集上可以获得更加稳定的扰动效果, 这是由于 RVMHM 在筛选攻击对象之后, 如果对应数据集足够充分, 进行 M-H 采样空间也较大, 攻击成功率就会更高.

为了进一步展示 RVMHM 相较于 MHM 提升量的统计显著性, 考虑无法确定 RVMHM 生成数据的分布, 我们选择非参数检验方法 Wilcoxon test 进行显著性检验, 并使用 Cohen's d 方法计算他们之间的差异性效应量.

如表 6 所示, 可以发现在非参数的统计检验 Wilcoxon-test 当中, 除了在数据集 LibPNG 上计算 *F*1-*score* 对应的 *p*-*value* 值较大 (*p*-*value*>0.1), 综合考虑检验对应的效应量, 可以发现其他数据都满足统计显著性, 可以说明 RVMHM 在其他各项指标上的表现是优于 MHM 方法的.

表 6 不同数据集中方法差异性统计显著性检验指标

检验指标	数据集	*F*1-*score*	*RecallVul*	*AUPRC*
p-*value* (Wilcoxon-test)	LibPNG	0.113	0.020	0.010
	LibTIFF	0.025	0.062	0.063
	FFmpeg	0.011	0.005	0.003
Effect-size	LibPNG	0.052	0.553	0.824
	LibTIFF	0.365	0.603	0.577
	FFmpeg	1.072	1.542	1.093

而 LibPNG 上对应 *F*1-*score* 检验异常值 (0.113), 关注其对应的 Effect-size 同时也非常小, 说明对应检验不具有实际应用意义. 同时关注表 4 原数据集差异, 由于 LibPNG 数据集样本量较小的原因, 导致 RVMHM 在 *F*1-*score* 上没有相较于 MHM 提升很大.

4.3 MHM 攻击模型超参数灵敏度分析

对于本文的对抗攻击, 前文所述有 3 个影响算法流程的超参数, 分别是最大攻击迭代次数 *maxiter*, 候选替换样本个数 *n_candidate*, 筛选攻击范围 *range*. 我们将进行实验对算法设定的超参数进行灵敏度分析. 对应最大攻击迭代次数会影响算法的执行效率, 候选样本替换个数会影响算法一次筛选替换样本的窗口大小, 筛选攻击范围会影响算法选择对抗攻击样本的方法. 考虑到 FFmpeg 的数据量较为充分, 本文选取特征提取算法 FastText 组合预测网络 Text-CNN

的组合模型上在该数据集上进行参数灵敏度分析. 灵敏度分析图例当中横坐标轴进行灵敏度分析的超参数迭代次数和攻击范围变化范围, 纵坐标轴表示鲁棒性验证指标 *F*1-*score*、*RecallVul*、*AUPRC* 这 3 个指标的变化范围.

如图 9 可以发现相关指标随着最大迭代次数变化, 在最大迭代次数到达 240 之后相关指标变化不明显, 再扩大攻击迭代次数对攻击效果的影响不大, 但是会消耗更多的时间, 算法执行的效率明显降低. 因为尽管该样本被选取为 RVMHM 对应的扰动对象, 但是有些样本由于自身的高维特征结构在分类检测空间中较为稳定不会被文本级别的词汇替换进行扰动, 扩大攻击迭代次数在一定范围内可以达到更好的性能, 但是多余的攻击次数并不能够产生对应成比例的扰动性能.

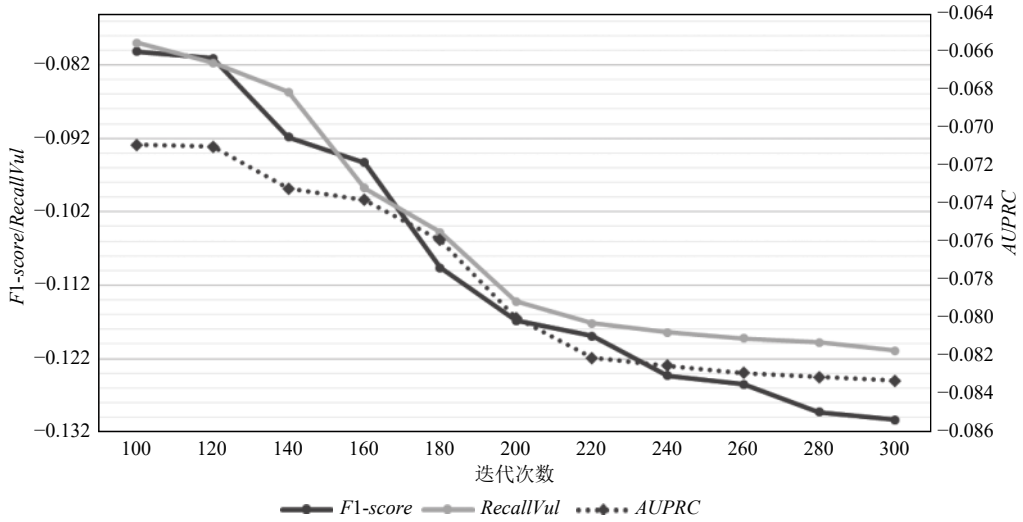

图 9 鲁棒性评估指标变化量随超参数最大攻击迭代次数的变化关系

如图 10 可以发现在扩大了攻击范围之后明显参与攻击的样本更加多, 但是成功攻击的个数并不是随着攻击范围的增加而正比增加的, 在扩大到 0.15 之后对于算法的影响效果逐渐降低. 一定程度扩大攻击范围能够使得 RVMHM 从目标数据集中选取更多的源码样本, 扩大了攻击目标的数量. 但是扩大筛选范围后多余的攻击样本攻击成功率低, 造成不必要的性能消耗. 通过实验表明候选变量个数在实验中对于攻击指标的影响不明显, 但是在实验过程中我们发现超参数候选变量个数会影响到攻击算法单次攻击过程执行的效率.

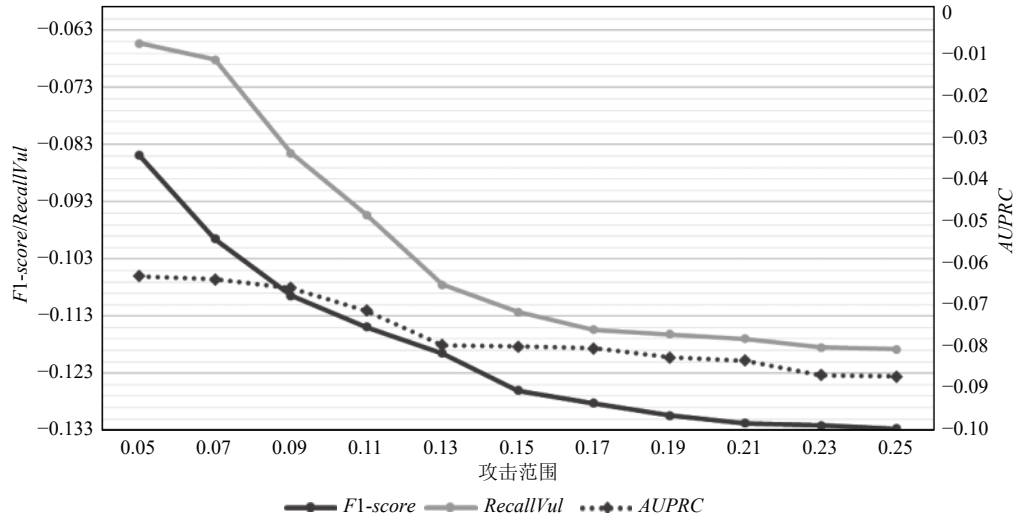

图 10 鲁棒性评估指标变化量随超参数攻击范围的变化关系

5 总结与展望

针对源码处理场景下 AI 系统在复杂代码场景下表现不稳定的问题, 本文选取数据和模型研究都较为充分的漏洞预测场景进行鲁棒性验证, 并提出一种漏洞预测场景下 AI 系统的鲁棒性验证方法 RVMHM. 该方法可以为漏洞预测的研究工作落实到实际应用提供重要的参考评价指标. RVMHM 通过使用 CodeSensor 源码分析工具对每一个样本建立抽象语法树, 提取源码数据文件的变量表, 通过设定条件筛选对抗攻击源码样本然后使用源码对抗攻击算法, 对参与测试的样本进行对抗攻击生成对抗样本, 观察攻击前后的模型相关鲁棒性指标变化衡量 AI 系统组件中预测模型和驱动数据之间交互的鲁棒性. 本文涉及的工作挑选了数据条件较为成熟的漏洞预测场景进行实验. 由于主题任务会影响模型读取的方法的构造, 以及相关样本筛选方法和扰动算法超参数的选择, 所以本文目前只在漏洞预测场景下进行实验说明了该方法在漏洞预测模型鲁棒性验证上的有效性. 但是目前多数源码处理场景的任务都是基于 n 阶 Markov 假设构建的语言模型, 该类假设下会利用字符之间的相关关系 (相邻字符, 附加了抽象语法树的字符顺序) 对代码语义进行特征识别, 对字符进行的合法扰动就存在一定概率会干扰到对应的模型应用过程.

本文在 3 个公开的数据集 FFmpeg, LibPNG, LibTIFF 上对 12 组模型进行了鲁棒性验证, 可以发现在 RVMHM 验证框架的扰动下, 其模型鲁棒性或多或少都存在一定的影响. 我们建议研究者们在筛选模型的时候不仅需要观察模型本身在数据集上的表现, 也需要将对抗扰动下鲁棒性指标的浮动, 纳入模型的评价筛选体系当中. 同时通过生成的对抗样本对模型进行对抗性训练提升模型的鲁棒性是很多工作正在进行的重要研究方向, 在 RVMHM 的对抗扰动过程中, 也会存储生成的对抗样本, 以供之后的工作以及 RVMHM 的使用者设计辅助任务对模型进行进一步的对抗训练以提高模型的鲁棒性.

在实验过程中, 本文的 MHM 源码对抗攻击算法在以下几个部分可以开展后续的研究.

(1) 将多种先进的攻击算法纳入 RVMHM 鲁棒性验证框架体系当中, 通过多种算法的综合扰动结果给出对应鲁棒性指标变化的综合评判结果.

(2) 针对筛选对抗攻击样本对象的时候, 目前采用的是简单的 L_2 范式差距来进行对象筛选, 之后可以采用启发式的算法进行对抗样本的筛选.

(3) 使用字词距离或者影响大小作为标准筛选优先参与对抗的样本, 去采样拟合连续对抗问题中对于梯度方向上的扰动, 可能会提高攻击效率.

(4) 使用生成的对抗样本对当前参与测试的模型进行对抗性训练, 观察模型的各项指标的效果变化.

(5) 实验结果目前可以表明 RVMHM 在面对不同嵌入提取算法和深度检测网络的组合上存在性能的差异. 之后的工作会尝试控制实验变量研究扰动算法在嵌入提取算法和深度检测网络上的细节差异, 以及扰动算法在漏洞预测场景下对于不同检测方法组合的扰动可解释性.

References:

[1] Ji SL, Du TY, Li JF, Shen C, Li B. Security and privacy of machine learning models: A survey. Ruan Jian Xue Bao/Journal of Software, 2021, 32(1): 41–67 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6131.htm [doi: 10.13328/j.cnki.jos.006131]

[2] 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]

[3] Johnson B, Song Y, Murphy-Hill E, Bowdidge R. Why don't software developers use static analysis tools to find bugs? In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). San Francisco: IEEE, 2013. 672–681. [doi: 10.1109/ICSE.2013.6606613]

[4] Chakraborty S, Krishna R, Ding Y, Ray B. Deep learning based vulnerability detection: Are we there yet? IEEE Trans. on Software Engineering, 2022, 48(9): 3280–3296. [doi: 10.1109/TSE.2021.3087402]

[5] Kuwajima H, Yasuoka H, Nakae T. Engineering problems in machine learning systems. Machine Learning, 2020, 109(5): 1103–1126. [doi: 10.1007/s10994-020-05872-w]

[6] Giray G. A software engineering perspective on engineering machine learning systems: State of the art and challenges. Journal of Systems and Software, 2021, 180: 111031. [doi: 10.1016/j.jss.2021.111031]

[7] Li SY, Guo JQ, Lou JG, Fan M, Liu T, Zhang DM. Testing machine learning systems in industry: An empirical study. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Pittsburgh: IEEE, 2022. 263–272.

[doi: 10.1145/3510457.3513036]

[8] Ishikawa F, Yoshioka N. How do engineers perceive difficulties in engineering of machine-learning systems?—Questionnaire survey. In: Proc. of the 7th IEEE/ACM Joint Int'l Workshop on Conducting Empirical Studies in Industry (CESI) and the 6th Int'l Workshop on Software Engineering Research and Industrial Practice (SER&IP). Montreal: IEEE, 2019. 2–9. [doi: 10.1109/CESSER-IP.2019.00009]

[9] Gui T, Wang X, Zhang Q, *et al*. TextFlint: Unified multilingual robustness evaluation toolkit for natural language processing. arXiv:2103.11441, 2021.

[10] Lin GJ, Zhang J, Luo W, Pan L, Vel OD, 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]

[11] Feng HT, Fu XT, Sun HY, Wang H, Zhang YQ. Efficient vulnerability detection based on abstract syntax tree and deep learning. In: Proc. of the 2020 IEEE INFOCOM-IEEE Conf. on Computer Communications Workshops (INFOCOM WKSHPS). Toronto: IEEE, 2020. 722–727. [doi: 10.1109/INFOCOMWKSHPS50562.2020.9163061]

[12] Lee YJ, Choi SH, Kim C, Lim SH, Park KW. Learning binary code with deep learning to detect software weakness. In: Proc. of the 9th KSII Int'l Conf. on Internet (ICONI) Symp. 2017. 245–249.

[13] 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]

[14] 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]

[15] Yefet N, Alon U, Yahav E. Adversarial examples for models of code. Proc. of the ACM on Programming Languages, 2020, 4: 162. [doi: 10.1145/3428230]

[16] Polyzotis N, Roy S, Whang SE, Zinkevich M. Data management challenges in production machine learning. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. Chicago: ACM, 2017. 1723–1726. [doi: 10.1145/3035918.3054782]

[17] Amershi S, Begel A, Bird C, DeLine R, Gall H, Kamar E, Nagappan N, Nushi B, Zimmermann T. Software engineering for machine learning: A case study. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Montreal: IEEE, 2019. 291–300. [doi: 10.1109/ICSE-SEIP.2019.00042]

[18] Ji SL, Du TY, Deng SG, Cheng P, Shi J, Yang M, Li B. Robustness certification research on deep learning models: A survey. Chinese Journal of Computers, 2022, 45(1): 190–206 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2022.00190]

[19] Carlini N, Wagner D. Towards evaluating the robustness of neural networks. In: Proc. of the 2017 IEEE Symp. on Security and Privacy (SP). San Jose: IEEE, 2017. 39–57. [doi: 10.1109/SP.2017.49]

[20] Zhang HZ, Li Z, Li G, Ma L, Liu Y, Jin Z. Generating adversarial examples for holding robustness of source code processing models. Proc. of the 2020 AAAI Conf. on Artificial Intelligence, 2020, 34(1): 1169–1176. [doi: 10.1609/aaai.v34i01.5469]

[21] Liu SG, Lin GJ, Han QL, Wen S, Zhang J, Xiang Y. DeepBalance: Deep-learning and fuzzy oversampling for vulnerability detection. IEEE Trans. on Fuzzy Systems, 2020, 28(7): 1329–1343. [doi: 10.1109/TFUZZ.2019.2958558]

[22] Poth A, Meyer B, Schlicht P, Riel A. Quality assurance for machine learning–an approach to function and system safeguarding. In: Proc. of the 20th IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). Macao: IEEE, 2020. 22–29. [doi: 10.1109/QRS51102.2020.00016]

[23] Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y. Generative adversarial nets. In: Proc. of the 27th Int'l Conf. on Neural Information Processing Systems. Montreal: MIT Press, 2014. 2672–2680.

[24] 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.

[25] Yamaguchi F, Golde N, Arp D, Rieck K. Modeling and discovering vulnerabilities with code property graphs. In: Proc. of the 2014 IEEE Symp. on Security and Privacy. Berkeley: IEEE, 2014. 590–604. [doi: 10.1109/SP.2014.44]

[26] Biggio B, Corona I, Maiorca D, Nelson B, Šrndić N, Laskov P, Giacinto G, Roli F. Evasion attacks against machine learning at test time. In: Proc. of the 2013 European Conf. on Machine Learning and Knowledge Discovery in Databases. Prague: Springer, 2013. 387–402. [doi: 10.1007/978-3-642-40994-3_25]

[27] Carlini N, Wagner D. Adversarial examples are not easily detected: Bypassing ten detection methods. In: Proc. of the 10th ACM Workshop on Artificial Intelligence and Security. Dallas: ACM, 2017. 3–14. [doi: 10.1145/3128572.3140444]

[28] Papernot N, McDaniel P, Goodfellow I, Jha S, Celik ZB, Swami A. Practical black-box attacks against machine learning. In: Proc. of the 2017 ACM on Asia Conf. on Computer and Communications Security. Abu Dhabi: ACM, 2017. 506–519. [doi: 10.1145/3052973.3053009]

[29] Alzantot M, Sharma Y, Elgohary A, Ho BJ, Srivastava M, Chang KW. Generating natural language adversarial examples. arXiv: 1804.07998, 2018.

[30] Wei MS, Huang YC, Yang JQ, Wang JJ, Wang S. CoCoFuzzing: Testing neural code models with coverage-guided fuzzing. arXiv: 2106.09242, 2021.
[31] Chen S, Xue MH, Fan LL, Hao S, Xu LH, Zhu HJ, Li B. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. Computers & Security, 2018, 73: 326–344. [doi: 10.1016/j.cose.2017.11.007]
[32] Shu R, Xia TP, Williams L, Menzies T. Omni: Automated ensemble with unexpected models against adversarial evasion attack. Empirical Software Engineering, 2022, 27(1): 26. [doi: 10.1007/S10664-021-10064-8]
[33] Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. arXiv:1412.6572, 2014.
[34] Li JF, Ji SL, Du TY, Li B, Wang T. TextBugger: Generating adversarial text against real-world applications. arXiv:1812.05271, 2018.
[35] Guo PJ. Online python tutor: Embeddable web-based program visualization for cs education. In: Proc. of the 44th ACM Technical Symp. on Computer Science Education. Denver: ACM, 2013. 579–584. [doi: 10.1145/2445196.2445368]
[36] Boyd K, Eng KH, Page CD. Area under the precision-recall curve: Point estimates and confidence intervals. In: Proc. of the 2013 European Conf. on Machine Learning and Knowledge Discovery in Databases. Prague: Springer, 2013. 451–466. [doi: 10.1007/978-3-642-40994-3_29]
[37] Goadrich M, Oliphant L, Shavlik J. Gleaner: Creating ensembles of first-order clauses to improve recall-precision curves. Machine Learning, 2006, 64(1): 231–261. [doi: 10.1007/s10994-006-8958-3]
[38] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R. Intriguing properties of neural networks. arXiv:1312.6199, 2014.
[39] 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]

附中文参考文献:

[1] 纪守领, 杜天宇, 李进锋, 沈超, 李博. 机器学习模型安全与隐私研究综述. 软件学报, 2021, 32(1): 41–67. http://www.jos.org.cn/1000-9825/6131.htm [doi: 10.13328/j.cnki.jos.006131]
[18] 纪守领, 杜天宇, 邓水光, 程鹏, 时杰, 杨珉, 李博. 深度学习模型鲁棒性研究综述. 计算机学报, 2022, 45(1): 190–206. [doi: 10.11897/SP.J.1016.2022.00190]

杨焱景(1999－), 男, 博士生, CCF 学生会员, 主要研究领域为 AI 系统鲁棒性, AI 系统安全.

沈海峰(1971－), 男, 博士, 教授, 博士生导师, 主要研究领域为软件工程, 人机交互, 以人为本的人工智能, 仿真与可视化.

毛润丰(1996－), 男, 博士生, 主要研究领域为 DevSecOps, 软件安全, 漏洞预测.

荣国平(1977－), 男, 博士, 副研究员, CCF 专业会员, 主要研究领域为软件过程, DevOps, AIOps.

谭睿(2001－), 女, 硕士生, 主要研究领域为机器学习, 软件缺陷预测.