

这是一个将基于图形的学习与用于代码漏洞检测的自动数据收集相结合的存储库副本。

本文的白玫瑰研究在线URL：

http://eprints.whiterose.ac.uk/168594/

版本：接受的版本



**第条：**

Wang，H，Ye，G，Tang，Z等（另有6位作者）（接受：2020年）结合基于图形的

通过自动数据收集学习代码漏洞检测。IEEE

信息取证和安全交易。ISSN 1556-6013（出版中）



©2020 IEEE。允许个人使用此材料。在任何当前或未来媒体上的所有其他用途，包括出于广告或宣传目的重新印刷/再版本材料、创作新的集体作品、转售或再分配给服务器或列表，或在其他作品中重复使用本作品的任何受版权保护的部分，必须获得IEEE的许可。

**重新使用**

存放在白玫瑰在线研究中心的物品受版权保护，除非另有说明，否则保留所有权利。它们可以下载和/或打印用于私人学习，或国家版权法允许的其他行为。出版商或其他权利持有人可允许进一步复制和重复使用全文版。白玫瑰研究在线记录上的许可证信息表明了这一点。

**拆除**

如果你认为网上白玫瑰的内容违反了英国法律，请通过电子邮件通知我们。eprints@whiterose.ac.uk包括记录的URL和撤回请求的原因。

eprints@whiterose.ac.uk https://eprints.whiterose.ac.uk/

# 将基于图形的学习与自动数据收集相结合用于代码漏洞检测

王焕婷、叶桂新、汤展勇、黄申伟、谭松芳、丁一芳、冯燕松、卞立中、王郑[[1]](" \l "_ftn1" \o ")*,*∗1.∗1*,* 2 31456*,*

1*.* 中国西北大学；2、中国南部科技大学；

3*.* 阿里巴巴达摩学院；4.中国北京大学；

5*.* 支付宝（杭州）信息技术有限公司。；6、利兹大学，U. K.

*摘要*-本文提出了一个新的学习框架，用于构建漏洞检测模型。基金会利用图形神经网络（GNN）的进步，开发了一种新的基于图形的学习方法，以捕获和推理程序的控制、数据和调用依赖关系。与之前将程序视为序列或非类型图的工作不同，Fundered学习并操作程序源代码的图形表示，其中单个语句通过关系边连接到其他语句。通过捕获程序语法、语义和流，Fundered为下游软件漏洞检测任务找到了更好的代码表示。为了提供足够的训练数据来构建有效的深度学习模型，我们将概率学习和统计评估相结合，从开源项目中自动收集高质量的训练样本。这提供了许多现实生活中的易受攻击代码训练样本，以补充标准漏洞数据库中有限的易受攻击代码样本。1

我们申请资金从程序源代码中识别功能级别的软件漏洞。我们使用C、Java、Swift和Php编写的程序对大型真实数据集进行评估，并将其与六种最先进的代码漏洞检测模型进行比较。实验结果表明，在不同的评估环境中，受资助的方法显著优于其他方法。

*索引项*-软件漏洞、代码漏洞检测、深度学习、深度图神经网络

一、导言

软件漏洞导致了许多系统攻击[1]和数据泄露事件[2]。机器学习是构建工具和模型以识别常见软件漏洞的可行方法。它的工作原理是首先从训练样本中学习指示易受攻击程序的潜在模式。然后，机器学习模型可以应用于新的软件项目，以识别潜在的易受攻击代码，这些代码显示出与训练数据中的易受攻击样本类似的模式。现在有大量证据表明，机器学习技术在检测常见代码漏洞或bug方面可以超过专家制定的规则[3]。

最近的研究利用深度学习（DL）对程序结构进行推理，以识别源代码中潜在的软件漏洞[4、5、6、3、7]。与…相比

这项工作部分由中国国家自然科学基金（NSFC）在资助协议61972314, 61672427和61872294，陕西省的国际合作项目下的资助协议2019KW-09和2020KWZ-013，蚂蚁金融科学资助项目和阿里巴巴创新研究计划。

通讯作者：汤占勇(zytang@nwu.edu.cn)及

王郑（z。wang5@leeds.ac.uk))

\*王焕婷和叶桂新是第一作者。

经典的机器学习技术，DL的优点是不需要专家参与手动调整程序结构的表示；相反，它会自动从训练样本中捕获并确定它们。

现有的基于DL的程序建模方法通常使用递归神经网络（RNN），如长短时记忆（LSTM）或其变体[5,6,8,3,7]。这些方法的工作原理是将源代码及其相应的程序结构（如抽象语法树（AST））视为一系列标记。然而，LSTM是为序列[9]设计的，不适合对程序的结构良好的控制和数据流进行建模。因此，以前基于LSTM的方法只能捕获源代码文本的浅表结构，无法利用程序结构丰富且定义良好的语义。如我们的评估所示，现有的类似LSTM的方法通常精度较低，要么漏掉漏洞，要么给出了绝大多数假阳性结果。

为了更好地对复杂的代码结构进行建模——传统上，在代码分析的编译器中，这些结构被表示为图形结构[10]——我们需要一种可以直接操作代码的图形表示并从中学习的方法。这样做将允许学习框架保存并推理大部分控制和数据流信息，以捕获许多软件漏洞的基本代码结构。例如，要检测use afterfree漏洞，我们需要知道在多个执行路径上何时何地分配缓冲区。

我们介绍了一种更好的代码结构建模方法。Fundered操作程序源代码的图形表示，具有学习和聚合多个代码关系的能力。它通过利用最近提出的门控图神经网络（GGNNs）[11]来实现这一点。通过直接操作图形表示，图形神经网络（GNN）在社交网络[12]和知识图[13]甚至编译二进制文件[14]中取得了惊人的成功。虽然GNN提供了一个很好的起点，但将其应用于开发实用且高效的软件漏洞检测框架并非易事。由于标准GNN在具有非类型化边的单个图形表示上运行，它无法区分控制流和数据流信息。然而，这些信息对于捕获易受攻击和有缺陷的代码模式至关重要。正如我们的评估所表明的，当忽略不同的代码关系时，最近一项使用普通GNN[15]的工作与LSTM替代方案相比，在代码漏洞检测方面给出了微小的改进。

它扩展了GNN区分和建模多个代码关系（包括数据、控件、操作顺序和操作数值）的能力。这是通过首先在不同的关系图中编码不同的代码关系，然后使用可学习的、特定于关系的函数在关系图中传播和聚合信息来实现的。通过将输入程序表示为具有显式控制和数据流或语法信息的多个关系图，我们的新图模型捕获了比以前基于GNN的方法更丰富的程序内关系[15]。这种更丰富的关系集提高了模型学习有用程序表示的能力，从而提高了下游代码漏洞检测任务的性能。正如我们将在后面展示的，通过使用GGNN来建模和区分丰富的代码关系，我们的方法明显优于其他基于图形的方法。

虽然我们新的GNN扩展为学习代码表示提供了潜在的强大功能，但它的潜力只能通过足够的训练数据来释放。典型的DL算法需要数百万个示例来学习有效的模型[16]，但现实生活中脆弱训练样本的稀缺性是一个常见问题[3]。缺乏训练数据限制了机器学习检测模型的质量，因为对于典型的高维程序空间，这些模型的训练数据非常稀疏。以前的一些方法通过为编译器测试生成程序来解决这个问题[17]。然而，合成程序有两个明显的缺点。它们受到用于生成程序的语法、模板或模型的影响，可能无法反映现实生活中程序的多样性和演变模式。因此，通过合成数据学习的模型很难推广。

我们解决脆弱性培训数据稀缺问题的解决方案是在开源项目中利用丰富的历史信息。我们通过使用离线训练的模型来预测哪个代码提交用于修补代码漏洞来实现这一点。然后，与漏洞相关的提交（即，为代码漏洞提供补丁的代码修订版）用于从补丁提交之前的版本中定位易受攻击的源代码片段。

然而，将这一高层次的想法转化为一个实用的数据收集系统并非易事。收集大量高质量易受攻击的代码样本是一项挑战，因为我们需要排除与软件漏洞无关的提交（例如，用于增强性能或功能但不修复漏洞的代码提交）。如果不能在我们的训练样本中排除这些良性代码片段，将会混淆机器学习算法。同时，由于收集的数据量很大，要求开发人员手动检查所有训练样本是不切实际的。因此，我们必须找到更好的措施来确保所收集数据的质量，并且只在必要时要求开发人员介入。

之前关于脆弱训练样本收集的工作[18,19,20]的特点是一刀切的假设。他们使用一个单一的整体模型来定位与漏洞相关的提交。这些方法往往无法检验模型是否符合当前输入，或者另一个模型是否会表现得更好。然而，众所周知，机器学习对不确定性很脆弱。当面对一种前所未有的情况时，机器学习技术通常会以很高的概率给出答案。不确定情况下的高概率通常会导致较差的预测结果——在我们的例子中，这将在训练数据中引入噪声，并恶化学习模型的质量。

为了确保培训数据的质量，我们采取了不同的方法，采用“专家混合”方案[21]从开源项目收集培训数据。我们的方法通过使用多个预测模型（称为专家）和仅使用我们对模型输出有高度信心的预测（或专家建议）来减少数据噪音。为了评估每个建议的置信度（或确定性），我们应用保形预测[22]来测量单个模型给出的预测的统计有效置信度。通过这种方式，我们只使用高度自信的建议。鉴于这种测量预测置信度的能力，我们只要求开发人员检查低置信度预测，以提供基本事实，然后作为额外的训练数据，随着时间的推移改进数据收集模型。我们表明，我们的混合专家方法提高了收集的训练数据的质量，从而产生了性能更好的漏洞检测模型。这还提供了一种方案，可以在开发人员最少参与的情况下逐步、持续地更新数据收集模型。

我们通过将其应用于检测程序源代码中的功能级漏洞[2]，展示了其优势。我们彻底评估了用C、Java、Php和Swift编写的代码提交历史和易受攻击程序的大型真实数据集。我们比较了六种最先进的（SOTA）基于学习的软件缺陷或漏洞检测方法[4,5,16,6,3,15]，以及五种用于自动漏洞代码样本收集的SOTA方法[18,19,23,20,24]。实验结果表明，通过发现更多的代码漏洞和较低的误报率，在评估设置中，Fund始终优于竞争方法。

贡献。本文是第一篇：

•展示如何开发用于漏洞检测的多关系门控图神经网络（第四节）；

•结合概率学习和统计评估，开发“专家混合”方法，以解决脆弱训练代码样本的不足（第五节）；

•利用跨编程语言传输学习到端口漏洞检测模型（第VII-D节）；

二、出身背景

## A.问题范围

这是一个用于代码漏洞检测的通用学习框架。在这项工作中，我们从源代码中识别漏洞。可以预测给定的函数或方法是否包含潜在的漏洞以及漏洞的类型。在这里，目标函数可以调用标准库调用和用户定义函数。注意我们的意图

1 attr\_value=（char\*）malloc（attr\_len+1）；2.

3 **否则如果**（！strcmp（attr\_name，“dateadded”））4 { 5   ae->date\_added=atoi（属性值）；6       自由（属性值）；7 }

8 **其他的**

### 9          自由（属性值）；



图1。GitHub的良性代码示例。VULDEEPECKER、VULDEEPECKER和LIN都对包含缓冲区属性值“双自由”漏洞的代码进行了错误分类。*µ等*



图2。缓冲区属性值的控制和数据流如图1所示。行号在每个矩形框中给出。

不是为了发现一种新的漏洞。相反，我们希望检测一段新的、看不见的代码是否包含与训练数据集中看到的代码模式类似的易受攻击的代码模式。因此，在检测常见的、反复出现的软件漏洞（或bug）时，它非常有用。为此，我们的工作重点是检测常见漏洞枚举（CWE）数据库中定义的常见漏洞（或弱点），而不是关注性能问题等非易受攻击的漏洞。

## B.对流量敏感方法的需求

为了显示对控制和数据流建模的需要，请考虑图1中给出的良性代码示例。

VULDEEPECKER[5]、VULDEEPECKER[6]和LIN[3]是SOTA漏洞检测模型。它们基于双向LSTM（BiLSTM）[25]——一种序列深度学习模型。在本例中，他们都错误地对包含双重自由漏洞的代码进行了分类。这种误报的根本原因是序列模型必须线性化代码结构，并将其视为令牌的序列，这忽略了控制流分歧（见图2）。因此，他们认为动态缓冲区attr\_值（第9行）在第6行释放后将再次释放。*µ等*

对于本例，我们希望通过使用流敏感决策模型来捕获目标程序的控制和数据流。如果我们能够做到这一点，那么我们可以推断第9行的缓冲区attr\_值被释放到不同的执行路径中，因此不会导致双重释放漏洞。对于更一般的情况，我们需要捕获语法信息（当用顺序表示法交换图形时），以及控制流和数据流，或者可能对下游处理任务至关重要的任何其他代码关系。Fund旨在通过扩展最近提出的GNN架构来提供此类功能。

三、 我们的方法概述

基金由两个关键组成部分组成。第一种是基于GNN的模型，用于识别潜在的软件漏洞

表一

GITHUB提交示例



|  |  |  |
| --- | --- | --- |
| 消息 | 添加空检查以避免空指针访问。 | 检查err when partial==NULL没有意义，因为partial==NULL意味着无错误地成功获取分支。 |

*4个添加行，2个删除行*

*3个添加行，2个删除行*

-sap ctx− >csa原因=原因；

-如果（错误）+如果（sap ctx）

-去清理；

补丁+sap ctx− >csa原因=原因；

+如果（错误）

-hdd ap ctx− >...

+互斥解锁（ei）− >...

+如果（hdd ap ctx− > sap上下文+转到清理+hdd ap ctx− >...



在源代码级别。第二个是一个自动框架，用于从开源存储库（即本文中的GitHub）收集易受攻击的代码样本，为学习漏洞检测模型提供额外的培训数据。

## A.软件漏洞检测

我们的漏洞检测模型基于本文提出的新GNN。该模型将目标函数的源代码作为输入。接下来，它通过结合从抽象语法树（AST）和程序控制和依赖图（PCDG）中提取的信息来构造程序图。程序结构以有向图的形式呈现，其中语句、标识符和直接值是图节点，两个节点之间的直接关系（例如父子关系、数据或控制流等）记录为边。由于一对节点之间可能存在多个关系（或边），我们使用关系图来记录每种类型的关系（见第IV-C2节）。关系图的节点连通性被编码为程序图矩阵。我们的GNN采用程序矩阵和初始节点表示来学习称为嵌入的代码表示，这些代码表示被表示为数值向量。代码嵌入被传递到下游神经网络进行预测。

该检测模型通过使用CVE和SARD等标准软件漏洞数据库中的训练数据集，以及从开源存储库收集的示例进行离线训练。然后，经过训练的模型可以应用于任何“新的、看不见的”项目。与之前的工作[4,5,3,16,6,15]不同，我们的多关系GNN可以更好地捕获多个代码关系，从而获得更准确的检测结果。我们在Sec中描述了漏洞检测模型。四、

## B.培训数据收集

为了从GitHub等开源存储库中收集易受攻击的代码样本，我们开发了一个自动数据收集框架。该框架旨在提供真实世界的易受攻击代码样本，以补充标准易受攻击数据库（如通用漏洞和暴露（CVE）和软件保证参考数据集（SARD））中的可用代码样本。我们的框架使用一组预测模型或专家，每个模型或专家独立预测代码提交是否为软件项目的前一版本中的代码漏洞提供补丁。通过识别脆弱性相关代码提交，我们可以检查带来的更改

|  |
| --- |
|  |
| F |

预言

       代码增强AST漏洞矩阵

     片段检测模型

图3。代码漏洞检测。我们的检测模型在功能级别将目标代码段的AST和CDFG作为输入。

通过补丁来定位以前版本的哪些代码段可能导致漏洞。然后将识别的代码段（本工作中的一个函数或方法）用作易受攻击的代码[3]训练示例。这种自动数据收集框架使我们能够建立一个真实生活项目的大型训练数据集。

从定制的表I中考虑两个提交

Linux内核托管在GitHub上。第一次提交（C1）修复了空指针漏洞，第二次提交（C2）修复了性能问题，但没有修复漏洞。对于检测易受攻击的代码，应从易受攻击的训练样本中排除从第二次提交中提取的代码。然而，现有的方法（VCCFINDER[18]和ZHOU[23]）可能会错误地将表I中的第二个版本标记为与漏洞相关的提交，因为提交消息包含关键字“check”和“NULL”。为了避免这样的错误，我们采用共形预测（CP）来量化每个专家模型的预测（或推荐）的提交（或推荐）的置信度（或凭据），并且仅在我们相信模型的结果时考虑预测。CP的使用帮助我们提高了收集数据的质量。我们在第二节中描述了这个培训数据收集框架。五、*等*

路线图。在接下来的两个部分中，我们首先在Sec中描述我们基于GGNN的代码漏洞检测模型。然后在第二节中介绍我们的训练数据收集模型。五、

四、 检测代码漏洞

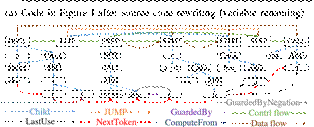
在本节中，我们将介绍基于GGNN的代码漏洞检测模型。我们首先在第二节中概述我们的模型。IV-A.然后，我们在第二节中解释模型结构。在描述如何将程序结构组织为模型的图形输入之前，请参见第IV-B节。IV-C.接下来是第二节中对我们模型的培训和学习过程的详细描述。IV-D，秒。IV-E和Sec。最后，我们将在第二节讨论模型的可解释性问题。静脉注射。

## A.我们的检测模型概述

图3描述了我们的检测模型的工作流程，该模型将目标程序的源代码（即函数）作为输入。我们使用标准的编译器解析器构造代码的AST。我们使用额外的控制和数据流以及令牌序列等顺序信息来扩展AST。扩展的AST表示为有向多重

1 a=（char\*）malloc（b+1）；2.

### 3 **否则如果**（！strcmp（c，“dateadded”）{4  d->d0=atoi（a）；5        免费（a）；6 }否则{7    免费（a）；8 }



（b） 增强AST

图4。规范化代码（a）和扩展AST（b）。

图，其中语句、代码块或值是图节点，直接关系（例如，两个节点之间的父子关系和其他关系）记录为边。由于一对节点之间可能存在多个关系，我们使用关系图来记录每种类型的关系（总共九个关系）。关系图的节点连通性被编码为邻接矩阵。

## B.GNN模型结构

在我们最近的工作[26]的基础上，我们扩展了门控图神经网络（GGNN）[27]，以模拟从源代码中提取的多个代码关系。我们的GGNN由四个基于选通递归单元（GRU）[28]的堆叠嵌入模型组成，因此它可以在关系图中包含更高阶的邻域。它利用关系图的邻接矩阵和初始节点表示来学习全局嵌入向量，然后将其传递到标准全连接网络，使用softmax层进行分类。

## C.图形表示法

*1) 代码预处理：*作为预处理步骤，我们使用编译器解析器使用一致的命名方案重写变量名。这一步确保程序中细微的语义差异（如变量名的选择）不会影响标记嵌入的选择（第IV-D节）。图4（a）显示了应用于图1所示示例的源代码重写。

*2) 程序图：*我们的程序图由包含语法节点（即语言语法中的非终结符，例如if语句或函数声明的AST节点）和语法标记（标识符名称和常量值等终结符）的AST构建而成。标准AST只有子边，用于编码两个AST节点之间的父子关系。为了捕获额外的语法、数据和控制信息，我们按照[29]中描述的方法向AST中添加了八种额外类型的边。我们对附加边的描述如下。

数据和控制流。我们将从PCDG提取的数据和控制路径集成到AST。

守卫着。我们使用GuardeBy边将变量的每个AST标记连接到变量的封闭保护表达式。例如，对于图4（a）中的if语句，我们将d和free（a）中的GuardeBy边添加到对应于！strcmp（）。这可能有助于确定错误的操作数顺序[16]。

跳我们使用跳转边将变量与控制依赖项连接起来。GuardedBy和Jump边缘允许我们记录发散控制流的关系。这种关系对于捕获漏洞的控制和数据流模式非常重要，如图1中给出的“双重自由”示例和“CWE-413：不正确的资源锁定”。

计算自。对于每个赋值，=expr，我们使用ComputedFrom Edge连接到表达式中出现的所有变量标记。此边缘捕获使用变量或缓冲区的位置，可用于检测“空指针解引用”等漏洞。*五、五、expr*

下一个。由于标准AST子级父边不会对语法节点的子级产生顺序，因此我们添加NextToken边以将每个语法标记连接到其后续标记。这用于捕获语句的操作码和操作数顺序。此类信息对于“CWE-404：不当资源关闭或释放”等漏洞类型非常有用，因为它捕获了API使用和释放的顺序。

LastUse和LastLexicalUse。我们使用LastUse边连接同一变量的所有用法，以捕获变量的用法，其中一个特例是if语句中的变量，我们使用LastLexicalUse边连接这类变量。例如，对于图4（a）中的if语句，我们将链接循环头中出现的c及其使用位置。通过记录变量或缓冲区上次使用的时间，这种关系有助于识别“doublefree”等漏洞。图4（b）显示了处理图4（a）中给出的代码后的增强AST。

*3） 关系图：*我们将增强AST的关系存储在单独的关系图中——上述九种关系中的每一种都有一个关系图。在本文中，关系图是一个有向图G=<V，E>，其中包含AST节点V和边E，它们指示两个节点之间存在给定的关系。我们使用邻接矩阵来记录每个关系图的边连接。对于每条边，我们还分别添加一条后向边（通过转置邻接矩阵），使边的数量和边类型增加一倍。这些后向边有助于在关系图之间传播信息。

## D.图节点表示

我们使用word2vector网络将每个程序图的节点（例如Stmt）和标记映射到数值的嵌入向量[30]。其思想是构造一个向量空间，使得在源代码中类似上下文中找到的单词在向量空间中彼此非常接近。用于将单词和标记映射到值的嵌入表和word2vector由训练代码语料库中的节点类型和从训练程序中收集的标记组成。由于变量名、函数名和常量值可以是任意长度，我们将它们编码为标记（即字母、符号和数字）。

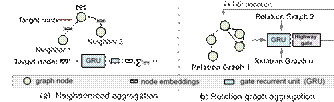


图5。通过聚合邻居（a）和同一节点（b）的其他关系图中的信息来学习节点嵌入。初始节点嵌入使用word2vec网络生成。

为了获取类型信息，我们将变量、常量和函数的（返回）类型的嵌入（例如整数变量的int）与AST节点名称表示连接起来，并将其传递到线性层，以获得图中每个节点的初始表示。

## E.学习多关系图

给定邻接矩阵和初始节点嵌入，我们的多关系GNN在关系图上生成100个特征的全局一维嵌入。

*1) 社区聚合：*与所有GNN一样，我们使用邻域聚合方案（图5a）来更新节点嵌入。图节点的100维嵌入向量由嵌入层通过递归聚合和转换其相邻节点的表示向量来计算。节点通过将其当前状态（即嵌入向量）作为消息发送给边缘上的所有邻居来交换信息。在每个节点上，消息被聚合，然后用于在下一个嵌入层（即下一次迭代）更新关联的节点表示。在重复更新节点状态的固定迭代次数后，使用读出函数将节点状态聚合为单个嵌入向量。*高压五、*

*2) 多关系建模：*与标准GNN不同，我们的模型跨多个关系图传播和聚合信息。如图5b所示，我们首先通过使用可学习的、特定于关系的函数，通过邻域聚合计算单个关系图的新图状态来实现这一点。然后，我们应用GRU单元来聚合和更新关系图中相同节点的状态。形式上，我们使用前向传播来更新顶点的状态，*五、*，来获取一个新状态：

*htv*+1 ：=GRU（h，tvX）∈ （W）ℓ ∗ htu）(1)

*ℓ* （u，v）A*ℓ*

其中是节点和之间的定向边。GRU是可学习的参数。初始节点状态是使用word2vec创建的，如第节所述。静脉注射。*A.ℓ U五、Wℓ H*0*五、*

受自然语言处理[31]中使用公路闸门[32]控制噪声传播的工作的启发，我们还在GGNN中采用分层公路闸门：

|  |  |
| --- | --- |
|  | (2) |
| *H*′（t+1）=t（h（））•hv（t+1）+（1）− T（h（））∗ h（）*五、五、T五、T五、T* | (3) |

其中是对层t+1的输入，并获得一个新状态；是一个S形函数是元素级乘法；*H*(*五、T*) *H*′(*五、T*+1) *西格玛WT*（t）和是门的权重矩阵和偏置向量*T*（h（vt））分别。

*3） 读数：*在我们跨多个嵌入层执行邻域聚合过程之后，我们将为每个令牌获得另一组嵌入。为了表示整个程序，我们使用一个读出函数来连接所有邻域聚合迭代和嵌入层的图形表示，以形成一个输出向量，作为关系图的全局程序表示：*汞MGi*

*汞*=CONCAT

式中=0,1，。。。，n、 是邻域聚合迭代。给定单个节点嵌入，该读出函数生成关系图的全局嵌入。*TM*

## F.训练GNN

我们的GGNN是使用来自标准漏洞数据库（本工作中的CVE和NVD）的训练样本以及使用我们的数据收集框架收集的开源代码示例进行离线训练的，如第。V.然后，学习的模型可以应用于看不见的程序。

我们在分批训练样本上训练我们的GGNN，其中每批由正样本和负样本组成。由于我们的目标是最小化两个概率分布（预测和实际）之间的距离，我们选择交叉熵损失作为目标函数。该函数被证明非常适合我们的GNN使用的sigmoid和softmax激活函数[33]。我们使用小批量随机梯度下降（SGD）和Adam算法[34]，学习率为0.001。当损失小于0.005或达到最大100个训练周期时，训练终止。我们将在第二节中展示。VII-F4，我们的漏洞检测模型的训练开销与其他DNN方案相当。由于培训是离线进行的，因此是一次性成本，对最终用户没有影响。

## G.受资助项目的直觉

和大多数机器学习技术一样，DNN就像一个黑匣子[35]，对于我们的方法来说也是如此。DNN工作机制的模型解释和理论分析仍然是一个突出的挑战，不在本工作范围内。Xu等人[36]表明，在区分图形结构方面，GNN与Weisfeiler-lehman测试[37]具有相同的功能。在高层——如图4（b）所示——GNN遵循邻域聚合策略，包括聚合层和组合层，用于迭代更新

通过聚合其邻居的表示来创建节点。对于图1所示的代码示例，Fundered通过聚合其相邻AST节点malloc和free来学习变量a的分配和释放。在读出阶段，学习到的信息将被映射到一个数字向量中，以允许在程序控制和数据流图中对变量a执行学习操作。

五、培训数据收集

为了为我们的GNN模型提供大量高质量的培训数据，我们通过构建数据收集工具来利用开源项目中的可用数据。

|  |
| --- |
|  |
| <\_> |

犯罪

图6。我们的数据标记模型由多个单独的分类器组成。对于由组合分类器给出的预测，我们量化其置信度，该置信度用于过滤低置信度的预测。*PC*

我们的数据收集工具的核心是一组专家模型，用于预测代码修订是否易受攻击。本质上，我们成立了一个“专家委员会”，由几个具有代表性的分类器组成（第V-A节）。每个专家模型将从提交消息和提交之间的代码更改中获得的一组特征（表II）作为输入（参考表I）。然后，它会预测目标代码版本是否为漏洞提供补丁。所有专家模型都使用带标签的训练样本进行离线训练。然后，经过训练的模型可以应用于任何新的、不可见的、来自不可见项目的代码提交（第V-D节）。以秒计。VII-E1，我们将我们的“专家混合”方法与使用单一单一模型的替代建模技术进行比较。

## A.专家模型的混合

图6描述了我们的混合模型，它由五个分类器组成：支持向量机（SVM）、随机森林（RF）、k近邻（KNN）、逻辑回归（LR）和梯度增强（GB）。我们使用这些模型是因为它们在之前的工作中被证明是有用的[20,23,19,24]，但其他模型也可以添加到我们的专家委员会中。

与之前直接利用预测的工作[18][19][20]不同，我们首先应用CP评估给定输入的单个分类器的凭证，以过滤出具有高度不确定性的预测。然后，我们使用多数投票方案来汇总剩余的预测结果。

我们描述了如何按照监督学习的三步过程建立和使用专家模型：（1）训练数据生成；（2） 模特训练；（3） 使用模型。

## B.专家模型的培训数据

*1） 收集代码修订培训样本：*我们使用相同的训练数据集来训练每个专家模型。训练数据来自两个来源。第一个包含CVE[38]和国家漏洞数据库（NVD）[39]中报告的提交日志和补丁。第二个包含从GitHub上托管的开源项目中提取的提交日志和补丁，如表I中给出的示例。

CVE和NVD的日志已经与已知漏洞关联，可以直接使用。为了从GITHUB收集数据，我们考虑了1000个最高级的项目，其中的C语言是java语言。我们选择C和Java，因为它们是最流行的编程语言之一。然而，我们的数据收集框架通常是适用的，也可以应用于其他编程语言。我们应用一组从[23]扩展的正则表达式（RE）规则来选择可能与漏洞相关的提交。

为了简化提取漏洞代码样本的过程，我们当前的实现只考虑代码

表二

用于标记与漏洞相关的提交的功能。

|  |  |
| --- | --- |
| 类别 | 特征 |
| 项目质量和活动 | （1） #星星；（2） #承诺；（3） #释放；（4） #贡献者；（5） 贡献率；（6） #分支机构 |
| 代码提交描述 | 提交消息； |
| 代码补丁 | 代码更改； |

一次修改一个源文件的修订。收集初始代码修订样本后，我们手动检查收集的数据，以确定代码中报告的漏洞是否已在CVE中发布。如果以前在CVE中报告过已识别的漏洞，我们将使用CVE编号与公共CVE描述建立链接。否则，我们将手动提取包含漏洞、提交日志和问题报告（如果有）的代码段。我们手动标记所有通过RE规则的代码修订是否与漏洞相关。然后，我们使用标记的样本作为训练数据。在这项工作中，我们使用3000多个手动标记的代码提交（来自以C或Java为主要语言的项目）来训练专家模型（另请参见第V-C3节和第VI-D节）。我们强调，这种手动检查过程只需要执行一次，就可以对模型进行培训，并且学习的模型可以用于收集更多的样本。因此，培训费用可以摊销。

*2） 特征提取：*建立一个好的机器学习模型的一个关键方面是找到正确的特征来描述输入。在这项工作中，我们使用表2中给出的三种类型的特性来捕获开源项目的质量和代码提交的目的。直观地说，提交消息描述了代码修订的原因，无论它是否与漏洞修复相关，以及漏洞的类型。开源项目的质量越高，代码提交消息可能就越严格和有意义；而一个积极开发的项目更有可能定期修补漏洞。

提交消息和修改后的代码语句被映射到一个嵌入向量中，使用预先训练好的word2vec网络[40]。生成的嵌入与项目质量和活动的特征值一起构成聚合特征向量。

## C.专家模型培训

*1) 培训个人专家模型：*训练数据用于确定每个专家模型的最优超参数。每个训练样本由一个数值特征向量和一个标签组成，该标签指示代码修订样本是否用于修复代码漏洞。对于训练，我们只需向专家模型提供训练数据，并执行其内部监督学习算法。

*2) 信心评估：*我们还应用CP来捕获类别标签（即与漏洞相关或不相关）的“陌生度”（称为不一致性度量）以供输入。为此，我们使用特定于模型的不合格函数（x，y，h）来估计模型的不合格分数。我们使用PyCP[41]中给出的默认方法特定不符合函数。直观地说，特征空间中的异常模式*Y十、A.H*



图7。收集和标记开源代码样本。

专家模型给出的不符合项得分将高于更常见的模式。

为了计算统计置信度，我们留出10%的模型训练数据作为校准集（不用于训练专家模型）。我们通过应用函数离线计算校准分数*A.*对于校准集中的每个实例，使用模型为每个类标签给出的概率（）。给定一个新的输入，我们使用函数计算一致性分数。然后我们计算一个p值，如下所示：*NypHYxn*+1*艾因P*+1*A.pvxn*+1

计数



*pv*= 计数{i∈ {1，…，n+1}:yi=y}*P*

(5)

这里，如果p值很小（接近其下限1/（n+1）），则预测非常不一致（异常值）。如果它很大（接近其上限1），那么预测是非常一致的。我们只考虑一个预测值大于1的预测值。−c、 其中是一个可配置的显著水平（在本研究中，根据经验设置为0.3）。*C*

*3） 培训费用：*培训专家模型的时间主要由培训数据收集和标记决定。在这项工作中，我们使用自动脚本收集GitHub修订版样本只需不到三天的时间（主要受GitHub帐户每天可以发出的请求数量限制），以及两名不到两天的付费注释员，通过交叉引用提交消息、代码更改和问题报告，手动检查和标记用于培训的收集的代码修订。相比之下，训练分类器和调整训练数据比率的时间可以忽略不计（使用多核服务器不到一个小时）。由于培训只进行一次，因此是一次性成本。

## D.使用专家模型

一旦我们了解了专家模型，我们就可以使用它们来预测代码提交是否为漏洞提供了补丁。图7描述了收集和标记开源代码样本的过程。我们使用GitHub API自动抓取并获取排名靠前的项目的代码提交。然后，我们应用RE规则（见第V-B1节）选择可能与漏洞相关的代码提交。

标记代码提交。对于通过重新筛选的代码修订，我们应用离线培训的专家模型来预测代码修订是否与代码漏洞修复相关。我们使用特征提取器处理收集的代码提交日志、补丁和项目相关信息，以形成特征向量（如第V-B2节所述）。给定特征值，每位专家都会预测代码修订是否与漏洞修复相关。为了在多个分类器（专家）之间达成共识，我们应用CP来估计每个专家输出的不合格分数。我们保留不合格分数高于置信水平的输出（见第V-C2节）。然后，我们根据剩余产出的简单多数票达成最终共识。

表三

用于评估漏洞检测的数据集。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 来源 | 语言 | #沃恩。类型 | #样品 | #阳性样本 |
|  | C | 30 | 90,954 | 45,477 |
| SARD和NVD | JAVA | 14 | 29,512 | 14,756 |
|  | Php | 5 | 17,578 | 8789 |
| github | C  敏捷的 | 10  5 | 10,400 2,506 | 5,200 1253 |

表四

代码修订历史数据集。

|  |  |  |  |
| --- | --- | --- | --- |
| 来源 | 语言 | #承诺 | #与易受攻击性有关的犯罪 |
| github | C  JAVA | 5,718  2,195 | 2,573  1,796 |
| 液 | JAVA | 1,787 | 804 |
| ZVD | C/C++ | 3,422 | 1,540 |

提取代码样本。对于通过重新筛选的每个代码提交，我们使用代码更改来定位修补函数的前一个版本。然后，我们提取该函数的代码，并将其与专家委员会给出的标签（易受攻击或不易受攻击）相关联。

持续学习。使用CP评估专家信心的优点之一是，随着时间的推移，我们可以使用可信度较低的样本来改进专家模型。这样做可以让我们随着时间的推移不断改进专家模型。以秒计。VII-G2，我们将展示如何利用持续学习来改进我们的数据收集框架。

六、 实验装置

## A.评价数据集

我们在两类数据集上进行评估。我们在用四种源语言编写的代码样本上评估我们的漏洞检测模型（第四节）：C、Java、Php和Swift。我们测试了混合编程的专家方法（SEC.V），我们的数据收集工具的核心，在代码修订历史的项目，使用C，C++和java作为主要的编程语言。

用于漏洞检测的数据集。表III给出了该数据集的详细信息，该数据集在函数级共包含150950个样本，源语言为C、Java、Php和Swift。我们一半的样本是阳性（易受攻击）代码样本。我们将范围限制在CWE 2019中定义的前5至前30个最危险的软件错误（例如，“缓冲区溢出”、“越界读/写”、“空指针解引用”）。我们从SARD[42]、NVD[39]和GitHub上托管的开源项目构建了这个数据集。与之前的工作[5]一样，我们使用SARD和NVD提供的补丁版本作为负面（或无漏洞）代码示例。同样，对于从GitHub收集的易受攻击的样本，我们应用相应的补丁提交来获得无漏洞版本。我们的测试样本包含具有数千行代码的真实代码样本。更多信息和示例可从补充文件中找到。补充文档给出了评估中使用的每种CWE类型的分布，以及我们的测试用例示例。

代码修订历史记录。从表IV可以看出，该数据集包括GitHub的6713个漏洞相关代码修订版，以及SAP[43]和ZVD[44]数据集。GitHub提交的4369个漏洞相关文件中，

2071个通过CVE和NVD链接建立；剩下的2298个项目来自GitHub上1000个最受欢迎的项目，其中C和Java是主要编程语言。对于后者，我们手动检查并标记承诺，以确定基本事实。请注意，SAP和ZVD数据集已经包含负面样本（即，从发现漏洞相关代码提交的同一项目中随机选择的与漏洞无关的提交）。我们使用相同的方法从GitHub获得负面提交。具体来说，我们保留已通过重新筛选但通过手动检查发现与漏洞无关的代码提交。总的来说，我们总共有13122个代码提交示例，包含与漏洞相关的提交和不相关的提交。

## B.竞争办法

对于漏洞检测，我们比较了六种相关方法：VULDEEPECKER[5]、VULDEEPECKER[6]、LIN[3]、VUDDY[4]、DEEPBUGS[16]和DEVIGN[15]；前三个基于BiLSTM，VUDDY使用哈希函数发现易受攻击的代码克隆，DEEPBUGS使用前馈神经网络进行错误检测，而DEVIGN使用标准GNN[45]对具有非类型AST边的图形表示进行操作。除了VULDEEPECKER之外，所有竞争方案都会做出一个二进制决策来预测代码是否包含bug或漏洞。*µ等µ*

对于数据收集，我们比较了五种SOTA数据收集方法：VCCFINDER[18]、SABETTA[20]、VULPECKER[19]、ZVD[24]和ZHOU[23]。*等等*

## C.执行

我们使用Tensorflow v.1.8[46]实现了基于GNN的漏洞检测模型，并使用Python scikit学习包实现了数据收集模型[47]。为了构建AST，我们对Java使用了coomy[48]，对Swift使用了ANTLR[49]，对C/C++使用了Php和Joern[50]。我们使用14核2.4 GHz Intel Xeon CPU和NVIDIA 2080Ti GPU在多核服务器上训练和测试所有方法。

## D.评价方法

模型评估。除非另有说明，我们使用五重交叉验证来评估各自数据集上的所有方法。该标准方法评估预测模型的泛化能力。

业绩报告。我们使用四个更高即更好的指标：

*精确*：正确标记的案例与测试案例总数的比率。

*精确*：正确预测的样本与预测具有特定标签的样本总数的比率。该指标回答了“在所有标记为与漏洞相关的代码修订中，有多少是真正正确的？”。高精度表示低假阳性率。

*回忆起*：正确预测的样本与属于某一类别的测试样本总数的比率。该指标回答了“在所有易受攻击的测试样本中，有多少实际被标记为易受攻击？”。高召回率意味着低假阴性率。

|  |
| --- |
| 图8。在检测具有前30个CWE漏洞的C函数时，平均准确率为92%。 |

*F1成绩*: 精确性和召回率的调和平均值，计算为。当测试数据的脆弱性类型分布不均时，它非常有用。

我们报告了交叉验证折叠中上述评估指标的几何平均值，因为它被广泛视为比算术平均值更可靠的性能指标[51]。

七、实验结果

我们评估结果的重点是：

•对于软件漏洞检测（第VII-a节和第VII-B节），平均而言，获得资助的软件漏洞检测准确率为92%。

•在检测（第VII-C节）和收集（第VII-E节）易受攻击代码方面，Fund优于所有竞争方法。

•我们对基金的工作机制进行了详细分析（第VII-G节和第VII-F节）。

## A.总体结果

在本实验中，我们应用Fund检测了具有前30个CWE漏洞类型的C函数的漏洞，并在Sec中对其他语言进行了评估。VII-D.在本实验中，我们使用标准漏洞数据库和数据收集工具收集的训练样本来训练我们的检测模型。图8报告了每种漏洞类型的评估指标。Fund成功识别了大多数易受攻击的样本，平均准确度和精密度为92%。高精度降低了误报率，这在实践中很重要，因为误报结果浪费了开发人员进行验证的时间。

Fundered还具有较高的召回率，F1得分为0.94（最高为0.99），这表明它的假阴性率较低，很少漏检漏洞。由于word2vec模型（用于初始节点嵌入，请参见第IV-D节）在捕获API误用令牌方面的准确性较低，因此，Fundered对某些CWE类型（如CWE-400和CWE-369）的准确率低于90%（但仍高于80%）。在未来，我们可以通过一个更强大的语言模型来增强我们的能力。最后，我们注意到，在其他竞争方法失败的情况下，我们可以从现实世界的复杂代码中识别漏洞。

## B.对大型代码库的评估

我们向五个开源项目申请资金，这些项目也在之前的研究中进行了评估[5,18,52,53]。表V列出了软件版本和功能级漏洞的数量。我们在培训中排除了这些项目中的代码，以确保培训后的模型在“看不见的”项目中进行测试。

图9总结了每个项目成功识别的单个漏洞，左侧的条形图

表五

五个开源项目的评估数据集。

|  |  |  |  |
| --- | --- | --- | --- |
| 不 | 项目 | 版本 | #沃恩。 |
| 1 | FFmpeg | v3。4.1，v3。4.2，v4。0.1，v4。1. | 12 |
| 2 | ImageMagick | v7。0.6，v7。0.8 | 14 |
| 3 | Linux内核 | v4。19，v4。20，v5。0，v5。4. | 16 |
| 4 | OpenSC | v1。8.2，v0。19 | 8 |
| 5 | rdesktop | v1。8.2 | 6 |

side是成功发现的漏洞总数。在这里，黑色符号表示通过模型识别漏洞，圆圈表示漏洞在NVD或CVE中报告，而正方形表示漏洞未在两个数据库中报告，也称为“silentlypatched漏洞”[5]。

通过在56个漏洞中识别出53个漏洞，包括11个“愚蠢修补”的漏洞，召回率为0.95，Fund优于所有竞争方法。通过对图形表示进行操作，DEVIGN以更高的整体精度优于序列模型，显示了基于GNN的模型的优势。然而，DEVIGN也未能检测到序列模型成功的12个漏洞，因为在用序列图表示法换取非类型图表示法时，它牺牲了许多语法和语义关系。通过识别13个漏洞，Fund比DEVIGN提高了33%的准确性和召回率。此外，Fundered还发现了所有其他模型都无法检测到的四个漏洞。在FFmpeg、ImageMagick和Linux内核中，每种情况下都有一个无法检测到但可以被其他人识别的情况。这些情况下，漏洞是由滥用API参数造成的。这些模式不是由

提供资金。这个问题可以通过使用更好的语言嵌入模型来解决，这是我们未来的工作。

## C.单个数据集的漏洞检测

现在，我们使用交叉验证在表III中的单个数据集上评估我们的漏洞检测模型。

*1） 标准数据集的评估：*本实验将所有方法应用于来自SARD和NVD的C函数。在里面

秒。VII-D，我们将评估扩展到Java、Php和Swift。

图16和图10显示，在漏洞检测方面，FUNDED提供了最佳的整体性能。由于检测模型的局限性，VUDDY和DEEPBUGS的检测精度较低。使用BiLSTM时，VULDEEPECKER和VULDEEPECKER对少量漏洞类型有效，但对某些类型（如CWE-469、CWE-676和CWE-834）的准确率低于50%，并可能导致大量误报（即低精度）。通过利用一组丰富的手动标记*µ*

|  |
| --- |
| 图9。每种方法为每个开源项目识别的漏洞数量。实心符号表示成功检测到的漏洞，圆圈表示漏洞在NVD或CVE中报告，而正方形表示漏洞未在标准数据库中报告。提供资金 |

成功检测到比其他漏洞更多的漏洞。



                精确回忆F1成绩

图10。对标准漏洞数据库的评估。最小-最大条形图显示了不同漏洞类型的性能。



                精确回忆F1成绩

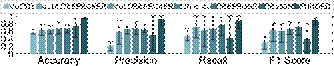
图11。对GitHub样本的评估。这项技术提供了最好的准确性、回忆力和F1成绩。

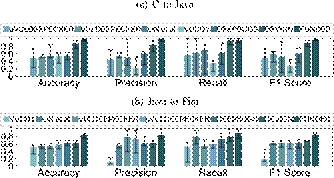
从训练数据来看，林书豪是表现最好的比赛方法。然而，其基于BiLSTM的模型未能检测到一些易受攻击的案例。对于带有CWE-834漏洞的测试用例，LIN仅成功发现了52.6%的漏洞样本。DEVIGN的总体准确率排名第二，但这只意味着比VULDEEPECKER的微小改进不到2%，并且优于LIN DEVIGN，而且在CWE-138和CWE-754等几种漏洞类型中，其性能也较低，而其他模型的准确率超过80%。*等等µ等*

对于大多数漏洞类型，在所有评估指标中，FUNDED优于所有其他方法。在少数几种类型的漏洞中，基金会漏掉一个可由性能最佳的替代方法检测到的漏洞样本。大多数这种情况是因为我们预先训练好的word2vec网络没有捕获特定的语言关键字，例如sizeof。这可以通过向网络词汇表中显式添加重要关键字来改进，这样word2vec就可以在单词（而不是令牌）级别直接建模这些关键字的语义[54]。总的来说，FUNDED是唯一一个平均准确率超过90%的方案，并且在整个评估指标中具有最佳的整体性能。

*2） 对GitHub数据集的评估：*现在，我们将实验扩展到从GitHub收集的C函数。在这个实验中，我们在SARD-NVD数据集上进行训练，并在GitHub数据集上进行测试。图11报告了不同指标的结果，其中最小-最大条显示了不同漏洞类型的差异。正如预期的那样，来自标准数据库的训练数据不能完全代表现实生活中看到的易受攻击的代码样本。因此，我们看到GitHub代码样本的准确性有所下降。总的来说，FUNDED在准确性、回忆力和F1成绩方面表现最佳。虽然VUDDY的精确度最高，但召回率却低得多。这是因为尽管

VUDDY的误报率最低，它在检测漏洞方面过于受限——它漏掉了85%以上的漏洞





（c） c到斯威夫特

图12。应用迁移学习为一种新的编程语言移植一个检测模型。

测试样本。LIN是准确度方面表现最好的替代模型，但其F1分数比Fund低20%，这表明Fund在假阳性和阴性阳性之间取得了更好的平衡。*等*

## D.跨语言学习

之前在其他领域的工作表明，针对不同任务在类似输入上训练的神经网络通常具有有用的共性[55]。我们观察到，由神经网络的起始层抽象出来的输入特性基本上与任务无关。相比之下，在网络的最后几层学习到的信息更专门用于特定任务。我们的工作利用这一观察结果，重用从一种语言学习到的网络的这些部分，以加快学习一种新语言的速度。此外，通过在代码级别对程序图结构而不是表面层信息进行建模，我们的方法也可以更好地捕获跨编程语言的易受攻击的代码结构。这表明，我们可以利用从另一种语言学习到的易受攻击代码模式的知识，为一种新的编程语言移植一个检测模型。实现这一点的技术被称为转移学习[56,57]。由于我们使用相同的网络结构，迁移学习是通过复制为一种语言构建的模型的权重来初始化另一种语言的网络来实现的。然后，我们像往常一样使用目标语言的一小组训练样本来训练模型。

1 ...

### 2 **如果返回**（inputStream！=null）{3       输入流。close（）；4 } 5 （fileOutputStream！=null）{6      错误的7 }

8 fileOutputStream。close（）；9假；10 ...**回来**



（a） 使用CWE-775的GitHub的Java代码示例

1       ...

2       **如果一会儿**（$userfile！=FALSE）{3        （$user=fgetcsv（$userfile））=

**错误的**)

### 4  **如果**（$user[0]==$username）5      返回TRUE；6       返回FALSE；7 }

8

9 **fclose**（$userfile）；10 ...



（b） 来自Github的带有CWE-775的Php代码示例

图13。C代码（a）和Php代码（b）都包含“不当资源关闭”漏洞。尽管这些程序是用两种不同的语言编写的，并且来自两个不同的开源存储库，但它们具有相似的控制和数据流，导致了一个共同的漏洞。

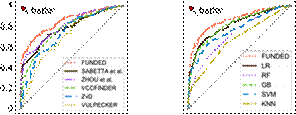
为了说明图形级知识如何跨编程语言重用以用于代码漏洞检测，请考虑表13中的两个代码段，它们来自托管在GITHUB上的两个不同的库。这两个代码示例都包含一个不当的资源关闭漏洞，程序在其有效生命周期结束后不会释放文件处理程序。具体来说，如果fileOutputStream不为null，图13中的Java函数不会关闭文件处理程序，如果userfile为True，Php方法也不会关闭文件处理程序。通过对导致漏洞的微妙程序结构进行建模，我们的方法可以重用从Java培训样本中学习到的知识，为Php程序建模相同的漏洞，反之亦然。

在这个实验中，我们首先训练一种语言的基线模型。然后，我们应用迁移学习，使用交叉验证将基线模型移植到另一种语言。我们考虑了三种跨语言设置：C到java，java到PHP，C到SWIFT，其中第一个是基线模型被训练的语言，第二个是要面向的新语言。图12显示，通过提供跨语言设置和评估指标的最佳性能，Fundered可以更好地利用先验知识来检测新编程语言的软件漏洞。这是因为它为易受攻击的代码模式捕获了许多与语言无关的信息。此功能对于缺少训练样本的语言或库非常有用。

## E.评估数据收集

现在，我们使用交叉验证对表IV中给出的代码修订数据集评估我们的数据收集框架。

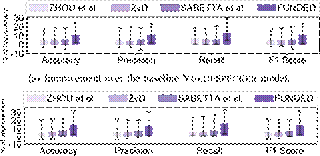
*1） 与其他数据标记方法相比：*图14显示了本研究中使用的不同数据收集方法和个人专家模型（第V-A节）的ROC曲线。ROC图绘制了真阳性率（TPR）



                                          8                                                                                  8

（a） 公司。收集数据。方法。（b） 公司。给个别专家

图14。真阳性率（TPR）和假阳性率（FPR）的ROC曲线。我们将其与其他自动数据收集框架（a）和专家委员会（b）中使用的单个专家模型进行比较。使曲线更靠近左上角的方法表明性能更好。



（b） 对VULDEEPECKER基准模型的改进。*µ*

图15。使用GitHub的额外标记训练样本时，性能比基线代码检测模型有所提高。“最小-最大”栏显示了前10个CWE漏洞类型的改进范围。

针对不同分类阈值下的假阳性率（FPR），其中阳性样本指的是脆弱性相关提交。降低分类阈值（即，更高的FPR）会增加将更多样本标记为漏洞相关的可能性，从而增加真阳性和假阳性。在有意义的FPR阈值（例如，小于0.5）下，通过提供比其他方法更接近左上角的曲线，FUNDED提供了最佳的整体性能。

*2） 收集的培训数据的影响：*在本实验中，我们评估开源项目样本是否有助于学习更好的检测模型。为了隔离我们的GNN模型的影响，我们测试通过数据收集方法收集的训练样本是否可以改善模型的基线

VULDEEPECKER[5]和VULDEEPECKER[6]。我们使用*µ*

基于SARD的训练数据集[42]学习基线模型。接下来，我们将通过资助和其他方法[23,20,24]收集的额外400个GitHub代码样本（具有相等的正负分割）纳入训练数据集，以学习第二个优化模型。然后，我们将基线模型和优化模型应用于Github的1000个C测试样本，其中一半样本是易受攻击的代码。为了进行公平比较，我们确保测试数据只包含在基线模型的训练数据中看到的相同CWE类型。

图15通过使用数据收集方法报告了性能改进。“最小-最大”栏显示了不同漏洞类型的改进范围，负值表示性能下降。使用GitHub提供的额外训练数据可以改进基线模型。FUNDED在所有评估指标中提供了最佳且一致的改进，表现优于竞争对手

|  |
| --- |
| 图16。标准漏洞数据库上排名前30位的CWE漏洞类型的检测精度。 |

表六

预测脆弱性类型的前三位准确性。

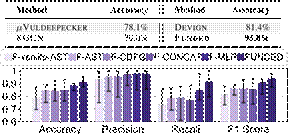


图17。比较受资助项目的实施变体。我们的实现提供了最佳的整体性能。

方法通过更高质量的培训数据，至少提高4.6%（高达10.2%）。对于被标记为脆弱性相关的200个GitHub样本，受资助专家模型的混合精度超过90%，而其他样本的精度较低，介于60.1%和81.6%之间。一些竞争模型给出的错误标记示例也可能对结果性能产生负面影响。这个实验表明，需要有一个精确的数据标记模型来收集额外的、高质量的训练示例。

基金提供的正是这种能力。

## F.漏洞检测模型分析

*1） 预测漏洞类型：*到目前为止，我们已经应用了一个二进制决策来预测一段代码是否包含漏洞。在本实验中，我们扩展了对漏洞类型的预测。我们将其与VULDEEPECKER进行了比较，VULDEEPECKER是竞争方案中唯一的多类漏洞检测模型，但我们也扩展了两个基于GNN的变体：RGCN[58]和DEVIGN到多类预测。我们使用来自SARD、NVD和GitHub的C数据集（见表三）进行评估。*µ*

表六列出了四种方法的前三名准确度。该指标检查前三名预测标签（根据预测概率排序）之一是否与测试样本的基本事实相符。这实际上意味着开发人员只需要验证三个潜在的漏洞。正如预期的那样，我们看到从二进制预测到多类预测的准确性有所下降。然而，FUNDED仍然是表现最好的模型，也是唯一一个精度超过90%的模型。2） 实施选择的影响：我们使用SARD数据集（它隔离了我们的数据收集方法的影响）比较了几个实施变量。第一种变体称为F-vanilla-AST，在标准AST上运行（没有第IV-C节中描述的附加边缘）。第二种变体称为F-AST，在增强的AST上运行，但不具有控制和数据流边缘。第三种变体称为F-CDFG

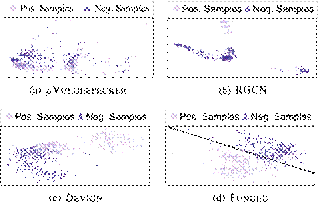


图18。不同方法给出的程序嵌入空间。在将代码样本映射到一个空间中时，更有效的方法是，在这个空间中，可以绘制一个更独特的边界来分隔易受攻击的代码样本和良性代码样本。

在增强的AST上，但只有额外的控制和数据流边缘。第四个变量称为F-CONCAT，它学习每个关系图的各个嵌入，然后将它们连接起来进行预测。最后一种变体被称为F-MLP，它使用多层感知器（MLP）层来学习和聚合单个关系图的嵌入[59]，但没有注意和高速公路入口。

图17显示，使用标准AST不足以建模程序结构。通过使用控制和数据流信息增强AST，F-AST或F-CDFG适度地将F-AST的准确度提高了5%。然而，单独使用AST或CDFG是不够的，因为两者的准确率都低于85%。F-CONCAT的性能也低于FUNDED，这表明简单地结合关系图的嵌入效果较差。这个实验强调了利用和聚合附加控制和数据流信息的重要性。通过对多关系图进行操作，F-MLP是性能最好的竞争方法，显示了多关系学习的巨大优势。通过采用注意机制和使用高速公路入口，在建模长依赖时最小化梯度递减问题，Fund进一步改进了F-MLP。

*3) 程序嵌入空间：*为了说明学习的程序表示，我们可视化了测试数据的嵌入空间。直观地说，一个有效的分类器应该将测试输入映射到一个空间中，在这个空间中，正类和负类之间可以画出不同的决策边界。

图18显示了一个经过训练的BiLSTM（VULDEEPECKER、VULDEEPECKER和LIN使用）、Devgn（一个标准GNN）、RGCN和Fund如何将来自NVD、SARD和GitHub的1000多个C函数测试样本（具有相等的正负分割）映射到嵌入空间。为了清晰起见，我们应用了t-SNE[60]，一种可视化技术*µ等*

图19。培训费用（a）和准确性（b）。最小-最大条形图显示了评估设置之间的差异。

技术，将多维嵌入空间投影到二维空间。与其他方法相比，Fundered（图18b）更有效地将测试样本映射到空间，其中大多数样本可以通过二进制决策边界分离。结果表明，该方法能更有效地提取漏洞检测的基本程序结构。

*4) 模型培训费用：*图19显示了不同漏洞检测模型的训练开销和实现的准确性。通过将所有方法应用于本工作中使用的最大训练数据集来测量训练时间，该数据集由来自SARD、NVD和Github的101354个C函数组成，如表III所述。当损失在20个连续训练周期内没有改善，或在估值集上达到95%的准确度时，训练终止，或符合已发布实施中给出的终止标准。这项实验是在一台多核服务器上进行的，使用的是台式NVIDIA 2080Ti GPU。

在我们的训练数据集上，所有模型可以在两小时内收敛。DEEPBUGS的训练时间最少，因为它使用一个简单的前馈神经网络。然而，DEEPBUGS的精确度最低，因为该模型不足以捕获复杂的代码结构。剩下的DNN方法需要比DEEPBUGS更长的训练时间，但它们的检测精度要高得多。与其他DNN方法相比，该方法需要更长的训练时间，因为该模型可以跨多个关系图学习。然而，受资助的DNN模型的训练开销仍然与其他DNN模型相当（在两小时内），但它产生了更好的检测精度。我们强调，模型培训是离线进行的，是一次性成本。此外，我们的方法可以在几秒钟内做出预测，因此对最终用户的影响可以忽略不计。

## G.数据收集框架分析

*1) 信心评估：*本实验评估了我们的CP函数（见第V-C节）成功检测到数据标记预测错误的频率。我们将我们的方法应用于代码修订历史数据集（表IV）。当专家模型给出错误的预测时，我们的CP方案成功捕获了91%的输入，并且具有9%的低误报率（即，当CP模型认为分类器是错误的，但它不是时）。请注意，我们还可以应用CP来进一步改进我们的检测模型。我们发现，当我们的检测模型对易受攻击的代码给出错误预测时，未优化的CP能够捕获80%的情况。

*2) 持续学习：*在这个实验中，我们检查是否可以使用CP标记的预测的地面真值标签来更新分类器。这样做是为了预测

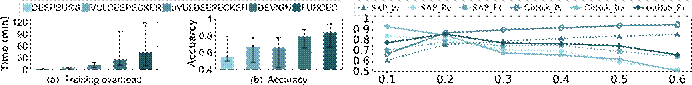
  
不合格分数

图20。随着SAP和Github数据集上不符合性阈值的增加，我们的数据标签模型的性能如何变化。

当p值大于0.7（见第V-C2节）时，我们手动检查代码修订以获得基本事实。通过仅对5%的预测失误样本使用基本事实来更新分类器，我们的数据标记模型可以在测试数据上实现超过90%的F1分数。这意味着初始标签模型的性能提高了35%以上。因此，使用CP作为置信度度量提供了一种逐步改进数据收集框架的实用方法。

*3) 不合格阈值的影响：*图20显示了不合规阈值如何影响我们的数据标签模型对从GitHub和SAP数据集收集的代码提交的性能。较大的不符合阈值有助于降低假阳性率，从而提高准确率，但也会对回忆和F1成绩产生负面影响。在这项工作中，我们使用0.3作为不合格阈值，因为它提供了最佳的整体性能。

八、讨论和未来的工作

这是第一次尝试使用GNNs和CP进行代码漏洞检测。当然，未来的工作和进一步的改进是有余地的。我们在这里讨论几点

模型稳健性。机器学习模型可能遭受对抗性攻击，精心构建的数据样本可能导致训练模型的偏差和异常行为。然而，我们认为，由于以下几个原因，基金会不太容易受到这个问题的影响。首先，我们使用从大型开源存储库收集的代码样本来提高培训数据集的覆盖率和多样性。在开发人员和用户没有注意到的情况下，将对抗性样本注入数千个排名靠前、积极维护的开放源代码项目是极不可能的，即使这是可以实现的，也将付出巨大的努力。因此，攻击的复杂性很高。其次，由于我们的数据收集框架使用多个模型，对模型发起攻击通常会成倍增加附加的复杂性，因为对抗性示例通常与目标模型紧密耦合[61]。最后，我们的CP模型可以通过使用基于阈值的分析来过滤对抗性样本，该分析被证明在防御对抗性样本方面是有效的[62,63]。*NN*

模型能力。GNNs的能力受到模型深度和宽度的限制[64]。我们的工作目标是功能级别的漏洞检测，我们发现我们的GGNN在学习程序结构方面是足够的。然而，我们设想，要为更大的项目建模，将需要更大、更深的GGNN。为了训练一个更大的神经网络，通常还需要大量的训练数据，我们的数据收集框架将对这些数据非常有用。应用神经架构搜索技术[65]来找到正确的神经网络结构也很有趣。我们将此作为未来的工作。

语言模型。在这项工作中，我们使用word2vec通过学习表面级语法信息来初始化节点嵌入（第IV-D节）。预先训练好的word2vec模型最初是为自然语言处理而设计的，没有经过调整以利用编程语言的良好定义结构。因此，它可能会错过一些重要的语言关键字。在这项工作中，我们试图通过在网络词汇表中明确添加一些重要关键字来规避这个问题。这种变通方法需要人工干预，因此不能很好地适应新的编程语言。为建模程序语言结构而设计的更好的语言模型可以消除专家的参与，从而缓解这个问题。我们未来的工作将探索一种专门为code2vec等程序源代码建模而构建的语言模型[66]。

模型可解释性。一般来说，机器学习技术存在依赖黑匣子的问题。DNN能力和边界的理论分析目前是一个活跃的研究领域[36,64,67]。为基金的基本工作机制提供理论证明是我们未来的工作。了解模型未能产生预期结果的一种方法是训练一个可解释的模型（或所谓的替代模型），如线性回归器，以近似基本黑箱模型的预测[66]。

九、 相关工作

我们的工作建立在过去代码漏洞分析、机器学习和软件工程的基础上。我们利用图神经网络的最新发展，对程序图结构进行建模，以学习易受攻击的代码模式，并使用所学知识检测代码漏洞。我们的工作还利用集成学习从现实生活中的开源项目中收集训练样本，为训练代码漏洞检测模型提供额外的数据。

经典方法。软件漏洞检测的早期工作依赖于专家精心编制的规则[68]。然而，构建高质量的规则并非易事，因为这需要大量专家参与。符号执行[69,70]通过利用符号值并在源代码上分析它们在程序控制流图上的使用，避开了手工编制规则的需要，但它不能扩展到大型程序，并且经常出现高误报[71]。

基于深度学习的漏洞检测。我们的工作是最近基于DL的软件漏洞检测工作的一部分[5,6,3,72]。有关该领域的全面综述，请参见[7]。之前在该领域的工作将源代码或AST视为一个连续字符串，并且经常忽略程序的结构信息。我们的工作利用并扩展了GGNN，以更好地模拟具有多种关系的程序。为了提供足够的训练数据，最近的一些工作开发了预测模型，以便从代码修订历史中自动提取易受攻击的代码样本[43,20,24]。受这些近期努力的启发，Fund将概率学习和统计评估相结合，自动从开源项目中提取易受攻击的代码样本，显著提高了提取样本的质量。

图形神经网络。GNN在处理诸如挖掘社交网络[12]、实体对齐[13]和二进制相似性检测[14]等任务的图形数据结构方面显示了有希望的结果。DEVIGN[15]与我们的工作关系最为密切。与DEVIGN不同的是，我们的方法可以同时模拟多个代码关系，并且更有效地进行跨语言学习。本文介绍的GNN扩展了我们最近使用多关系图对程序结构建模的工作[26]。新的GNN通过扩展AST编码来捕获额外的代码关系和类型信息，并使用GRU和高速门来建模长期依赖关系，从而提高了性能[26]，如第节所示。VII-F2。

基于机器学习的软件开发。人们对将机器学习技术应用于软件开发越来越感兴趣[73]。现有的方法解决了各种开发任务，包括模糊测试[74,75]、检测代码克隆[76,4,19,77]、改进漏洞资金的静态分析[78,79]、修复程序[80]、缺陷预测[81,82]、攻击检测[83]和处理漏洞报告[18,23,24]。基金会建立在过去的基础上，但质量与这些研究不同。

十、结论

我们提出了一种基于图学习的新方法，用于学习代码漏洞检测模型。基金会扩展了标准图神经网络，对多个代码关系进行建模，这些关系对于漏洞检测的程序结构建模至关重要。为了提供足够的培训数据，Fundered将概率学习和统计评估相结合，自动从开源存储库中收集易受攻击的代码样本。我们在大型真实数据集的功能级别上应用程序来检测源代码漏洞。实验结果表明，在评估指标方面，受资助的方法显著优于各种竞争方法。

推荐人

[1] 孙立军、张建军、林巴、高世强、张立业和项耀英，“数据驱动的网络安全事件预测：一项调查”，《IEEE通信调查与教程汇编》，第21卷，第2期，第1744-1772页，2018年。

[2] F.Wu，J.Wang，J.Liu和W.Wang，“深度学习的漏洞检测”，2017年第三届IEEE国际计算机与通信会议论文集。IEEE，2017，第1298-1302页。

[3] 林国强、张建军、罗文华、潘立军、O.De Vel、P.Montague和Y.Xiang，“通过学习多领域知识库发现软件漏洞”，IEEE可靠和安全计算事务集，2019年。

[4] S.Kim、S.Woo、H.Lee和H.Oh，“Vuddy：易受攻击代码克隆发现的可扩展方法”，发表于2017年IEEE安全与隐私研讨会论文集。IEEE，2017，第595-614页。

[5] 李志强，邹德强，徐世强，欧X，金H，王S，邓Z，钟Y，“Vuldeepecker：基于深度学习的漏洞检测系统”，NDSS会议录，2018年。

[6] D.Zou，S.Wang，S.Xu，Z.Li和H.Jin，“vuldeepecker：一个基于深度学习的多类漏洞检测系统”，IEEE可靠和安全计算事务集，2019年。*µ*

[7] 林胜文，韩秋林，张建军，项耀军，“基于深度神经网络的软件漏洞检测：综述”，《IEEE会议录》，第108卷，第10期，第1825-1848页，2020年。

[8] 李志强，邹德强，徐世强，金海华，朱耀强，陈志强，“Sysevr：一个利用深度学习检测软件漏洞的框架”，arXiv预印本arXiv会议录，2018年，1807.06756。

[9] S.Hochreiter和J.Schmidhuber，“长期-短期记忆”，《神经计算学报》，第9卷，第8期，第1735-1780页，1997年。

[10] D.J.Kuck，R.H.Kuhn，D.A.Padua，B.Leasure和M.Wolfe，“依赖图和编译器优化”，第八届ACM SIGPLAN-SIGACT编程语言原理研讨会论文集，1981年，第207-218页。

[11] 吴志强、潘世华、陈福荣、龙国荣、张志强和菲利普，“图形神经网络的综合调查”，IEEE神经网络和学习系统学报，2020年。

[12] H.Gao，Z.Wang和S.Ji，“大规模可学习图卷积网络”，第24届ACM SIGKDD国际知识发现与数据挖掘会议论文集，2018年，第1416-1424页。

[13] Wu Y.Wu，X.Liu，Y.Feng，Z.Wang和D.Zhao，“为实体对齐联合学习实体和关系表示法”，arXiv预印本arXiv会议录：1909.09317，2019年。

[14] Xu，C.Liu，Q.Feng，H.Yin，L.Song和D.Song，“基于神经网络的图形嵌入用于跨平台二进制代码相似性检测”，2017年ACM SIGSAC计算机与通信安全会议论文集，2017年，第363-376页。

[15] Zhou，S.Liu，J.Siow，X.Du和Y.Liu，“设备：通过图形神经网络学习综合程序语义的有效漏洞识别”，神经信息处理系统进展论文集，2019年，第10197-10207页。

[16] M.Pradel和K.Sen，“Deepbugs:基于名称的错误检测的学习方法”，《编程语言ACM会议录》，第2卷，no.OOPSLA，第1-252018页。

[17] C.Cummins、P.Petoumenos、A.Murray和H.Leather，“通过深度学习实现编译器模糊化”，第27届ACM SIGSOFT软件测试与分析国际研讨会论文集，2018年，第95-105页。

[18] H.Perl、S.Dechand、M.Smith、D.Arp、F.Yamaguchi、K.Rieck、S.Fahl和Y.Acar，“Vccfinder：发现开源项目中的潜在漏洞以协助代码审计”，《第22届ACM SIGSAC计算机和通信安全会议记录》，2015年，第426-437页。

[19] 李泽民、邹德明、徐世华、金海华、齐海华和胡志强，“Vulpecker：基于代码相似性分析的自动漏洞检测系统”，载于2016年第32届计算机安全应用年会论文集，第201-213页。

[20] A.Sabetta和M.Bezzi，“安全相关承诺自动分类的实用方法”，发表于2018年IEEE国际软件维护和发展会议（ICSME）论文集。IEEE，2018年，第579-582页。

[21]R.A.Jacobs，M.I.Jordan，S.J.Nowlan和G.E.Hinton，“本地专家的自适应混合”，《神经计算学报》，第3卷，第1期，第79-87页，1991年。

[22]G.Shafer和V.Vovk，“保形预测教程”，《机器学习研究杂志》，第9卷，3月号，第371-4212008页。

[23]Y.Zhou和A.Sharma，“从提交消息和错误报告中自动识别安全问题”，载于2017年软件工程基础第11次联合会议记录，2017年，第914-919页。

[24]X.Wang、K.Sun、A.Batcheller和S.Jajodia，“检测“0天”漏洞：oss中秘密安全补丁的实证研究”，发表于2019年第49届IEEE/IFIP可靠系统和网络国际会议（DSN）论文集。IEEE，2019年，第485-492页。

[25]黄志强、徐文华和余克强，“用于序列标记的双向lstm crf模型”，arXiv预印本arXiv论文集：1508.0199112015。

[26]G.Ye，Z.Tang，H.Wang，D.Fang，J.Fang，S.Huang和Z.Wang，“通过基于多关系图的学习进行深层程序结构建模”，第111-123页，2020年。

[27]Y.Li，D.Tarlow，M.Brockschmidt和R.Zemel，“门控图序列神经网络”，ICLR学报，2015年。

[28]K.Cho，B.Van Merrienboer，C.Gulcehre，D.Bahdanau，¨F.Bogares，H.Schwenk和Y.Bengio，“使用rnn编码器-解码器学习短语表示用于统计机器翻译”，2014年。

[29]M.Allamanis、M.Brockschmidt和M.Khademi，“学习用图形表示程序”，ICLR会议记录，2018年。

[30]T.Mikolov，I.Sutskever，K.Chen，G.S.Corrado和J.Dean，“单词和短语的分布式表示及其组成性”，摘自《神经信息处理系统的进展》，2013年，第3111-3119页。

[31]A.Rahimi，T.Cohn和T.Baldwin，“通过图卷积网络进行的半监督用户地理定位”，摘自《美国公民自由协会学报》，2018年。

[32]R.K.Srivastava、K.Greff和J.Schmidhuber，“公路网络”，arXiv预印本arXiv会议录：1505.003872015。

[33]Z.Zhang和M.Sabuncu，“用于训练带有噪声标签的深层神经网络的广义交叉熵损失”，《神经信息处理系统进展学报》，2018年，第8778-8788页。

[34]D.P.Kingma和J.Ba，“亚当：随机优化的方法”，arXiv预印本arXiv会议录：1412.69802014。

[35]F.帕斯夸尔，黑匣子协会。哈佛大学出版社，2015年。

[36]K.Xu、W.Hu、J.Leskovec和S.Jegelka，“图形神经网络有多强大？”ICLR会议记录，2018年。

[37]B.Weisfeiler和A.A.Lehman，“将一个图简化为一个标准形式，并在此过程中产生一个代数”，《Nauchno Technicheskaya Informatsia学报》，第2卷，第9期，第12-16页，1968年。

[38]“常见漏洞和暴露（CVE）”https://cve.斜接。org/。

[39]“国家脆弱性数据库（NVD）”https://nvd.nist.gov.

[40]Y.Goldberg和O.Levy，“word2vec解释：推导mikolov等人的负采样单词嵌入方法，”arXiv预印本arXiv会议录：1402.37222014。

[41]V.N.Balasubramanian、A.Baker、M.Yanez、S.Chakraborty和S.Panchanathan，“Pycp：一个开源的符合形预测工具包”，摘自IFIP人工智能应用与创新国际会议记录。斯普林格，2013年，第361-370页。

[42]NIST，“软件保证参考数据集项目”，https://samate。nist。gov/SRD/。

[43]“SAP数据集”https://github.com/SAP/脆弱性评估kb/tree/master/MSR2019。

[44]“Zvd数据集”https://github.com/SecretPatch/Dataset.

[45]Y.Li，D.Tarlow，M.Brockschmidt和R.Zemel，《门控图序列神经网络》，2016年。

[46]“TensorFlow”https://www.tensorflow.org/.

[47]“Scikit学习：预测数据分析工具”，https://Scikit学习。组织。

[48]“烟尘：分析和转换Java应用程序的框架，”http://sable.github.io/soot/.

[49]“ANTLR（另一种语言识别工具），”https:

//www.antlr。org/。

[50]“Joern（C/C++的开源代码查询引擎），”https:

//乔恩。io/。

[51]W.Ertel，“关于加速的定义”，欧洲并行体系结构和语言国际会议论文集。斯普林格，1994年，第289-300页。

[52]A.Younis、Y.Malaiya、C.Anderson和I.Ray，“害怕还是不害怕这是个问题：存在漏洞的易受攻击函数的代码特征”，载于2016年第六届ACM数据和应用程序安全与隐私会议录，第97-104页。

[53]S.Ognawala、R.N.Amato、A.Pretschner和P.Kulkarni，“通过成分分析发现的漏洞的自动评估”，发表于《第一届Symbiosis机器学习和软件工程国际研讨会论文集》，2018年，第16-25页。

[54]T.Mikolov，S.Kombrink，L.Burget，J.Cernockˇy和S.Khu-`danpur，“递归神经网络语言模型的扩展”，发表于2011年IEEE声学、语音和信号处理国际会议录（ICASSP）。IEEE，2011，第5528-5531页。

[55]M.Long，H.Zhu，J.Wang和M.I.Jordan，“具有联合适应网络的深度迁移学习”，摘自《国际机器学习会议记录》。PMLR，2017年，第2208-2217页。

[56]潘世杰和杨问，“关于迁移学习的调查”，《IEEE知识与数据工程学报》，第22卷，第10期，第1345-1359页，2009年。

[57]L.Torrey和J.Shavlik，“转移学习”，摘自《机器学习应用和趋势研究手册：算法、方法和技术》。IGI global，2010年，第242-264页。

[58]M.Schlichtkrull、T.N.Kipf、P.Bloem、R.Van Den Berg、I.Titov和M.Welling，“用图卷积网络建模关系数据”，欧洲语义网会议。斯普林格，2018年，第593-607页。

[59]G.Ye，Z.Tang，H.Wang，D.Fang，J.Fang，S.Huang和Z.Wang，“通过基于多关系图的学习进行深层程序结构建模”，载于ACM并行体系结构和编译技术国际会议论文集，2020年，第111-123页。

[60]L.v.d.Maaten和G.Hinton，“使用t-sne可视化数据”，《机器学习研究杂志》，第9卷，第11期，第2579-26052008页。

[61]D.Hendrycks和K.Gimpel，“检测对抗性图像的早期方法”，arXiv预印本arXiv会议录：1608.00530，2016年。

[62]S.Kokalj Filipovic，R.Miller和G.Vanhoy，“射频深度学习中的对抗性示例：检测和物理鲁棒性”，发表于2019年IEEE全球信号和信息处理会议记录（GlobalSIP）。IEEE，2019年，第1-5页。

[63]D.Meng和H.Chen，“磁铁：对抗对抗性示例的双管齐下的防御”，摘自2017年ACM SIGSAC计算机与通信安全会议记录，2017年，第135-147页。

[64]A.Loukas，“神经网络无法学习的图形：深度与宽度”，arXiv预印本arXiv论文集：1907.031992019。

[65]C.Liu，B.Zoph，M.Neumann，J.Shlens，W.Hua，L.-J.Li，L.Fei Fei，A.Yuille，J.Huang和K.Murphy，“渐进式神经架构搜索”，欧洲计算机视觉会议记录，2018年，第19-34页。

[66]M.T.Ribeiro、S.Singh和C.Guestrin，“我为什么要相信你？”解释任何分类器的预测，”第22届ACM SIGKDD国际知识发现和数据挖掘会议论文集，2016年，第1135-1144页。

[67]R.Sato，“关于图形神经网络表达能力的调查”，arXiv预印本arXiv论文集：2003.040782020。

[68]“Findbugs”http://findbugs.sourceforge.net/.

[69]C.Cadar和K.Sen，“软件测试的符号执行：三十年后”，《ACM通信会议录》，第56卷，第2期，第82-902013页。

[70]D.A.Ramos和D.Engler，“受限符号执行下：真实代码的正确性检查”，载于《第24届{}{}会议录》，2015年，第49-64页。*USENIX安全研讨会(USENIX保安（15）*

[71]K.Wang和Z.Su，“学习混合的精确语义程序嵌入”，PLDI学报，2020年。

[72]Y.Li，S.Wang，T.N.Nguyen和S.Van Nguyen，“通过基于上下文的代码表示学习和基于注意的神经网络改进错误检测”，《编程语言ACM会议录》，第3卷，no.OOPSLA，第1-30页，2019年。

[73]Z.Wang和M.O&apos;Boyle，“编译器优化中的机器学习”，《IEEE学报》，第106卷，第11期，第1879-19012018页。

[74]P.Godefroid，H.Peleg和R.Singh，“学习与模糊：输入模糊化的机器学习”，发表于2017年第32届IEEE/ACM国际自动化软件工程会议录（ASE）。IEEE，2017，第50-59页。

[75]Y.Chen，D.Mu，J.Xu，Z.Sun，W.Shen，X.Xing，L.Lu和B.Mao，“Ptrix:cots二进制文件的高效硬件辅助模糊化”，摘自2019年亚洲计算机与通信安全大会论文集，2019年，第633-645页。

[76]M.White、M.Tufano、C.Vendome和D.Poshyvanyk，“用于代码克隆检测的深度学习代码片段”，发表于2016年第31届IEEE/ACM国际自动化软件工程会议记录（ASE）。IEEE，2016，第87-98页。

[77]T.Unruh、B.Shastry、M.Skoruppa、F.Maggi、K.Rieck、J.P.Seifert和F.Yamaguchi，“利用有缺陷的教程进行大规模web漏洞发现”，发表于2017年第11届会议录。*USENIX攻击性技术讲习班(胡特17)*

[78]K.Heo、H.Oh和K.Yi，“机器学习引导的选择性非健全静态分析”，发表于2017年IEEE/ACM第39届软件工程国际会议（ICSE）论文集。IEEE，2017，第519-529页。

[79]S.A.Gorski III和W.Enck，“Arf：识别android系统服务中的再授权漏洞”，摘自《第12届无线和移动网络安全与隐私大会论文集》，2019年，第151-161页。

[80]Z.Huang，D.Lie，G.Tan和T.Jaeger，“使用安全属性生成漏洞补丁”，发表于2019年IEEE安全与隐私研讨会论文集。IEEE，2019年，第539-554页。

[81]A.N.林、A.T.阮、H.A.阮和T.N.阮，

“将深度学习与信息检索相结合，以本地化缺陷报告中的缺陷文件（n），《2015年第30届IEEE/ACM自动化软件工程国际会议记录》（ASE）。IEEE，2015，第476-481页。

[82]S.Wang，T.Liu和L.Tan，“自动学习缺陷预测的语义特征”，2016年IEEE/ACM第38届国际软件工程会议（ICSE）论文集。IEEE，2016，第297-308页。

[83]X.Shu，D.Yao，N.Ramakrishnan和T.Jaeger，“大跨度程序行为建模和攻击检测”，ACM隐私和安全事务会议录（TOPS），第20卷，第4期，第1-28页，2017年。

[[1]](" \l "_ftnref1" \o ")FUNDED=流敏感漏洞代码检测。

[[2]](" \l "_ftnref2" \o ")代码和数据可从以下网址获得：https://github.com/HuantWang/FUNDED尼索

[[3]](" \l "_ftnref3" \o ")在这项工作中，与漏洞相关的代码提交=为漏洞提供修复的代码修订日志，而易受攻击的代码=包含CWE中定义的漏洞类型的代码段。我们强调，识别与漏洞相关的代码提交与从源代码中识别漏洞有着根本的不同，因为我们可以利用提交消息和两次提交之间的代码更改等附加信息来帮助完成前一项任务。