基于脆弱性检测的深度学习模型的实证研究

本杰明斯廷霍克
爱荷华州立大学艾姆斯分校，美国爱荷华州。edu

马哈布尔拉赫曼博士
爱荷华州立大学
艾姆斯，爱荷华州，美国
mdrahman@iastate.edu

理查德贾尔斯
爱荷华州立大学
艾姆斯，爱荷华州，美国
rdjiles@iastate.edu

魏乐
爱荷华州立大学
艾姆斯，爱荷华州，美国
weile@iastate.edu

第45届软件工程国际会议（ICSE）| 978-1-6654-5701-9/23/$31.00©2023IEEE|DOI: 10.1109/ICSE48619.2023.00188

摘要：代码的深度学习（DL）模型最近在漏洞检测方面取得了很大的进展。在某些情况下，基于dl的模型的表现优于静态分析工具。虽然已经提出了许多伟大的模型，但我们还对这些模型没有很好的理解。这限制了模型的健壮性、调试和部署的进一步发展。在本文中，我们在2个广泛使用的漏洞检测数据集上调查并复制了9个最先进的（SOTA）深度学习模型： Devign和MSR。*我们调查了三个领域的6个研究问题，即模型能力、训练数据和模型解释。*我们通过实验证明了一个模型的不同运行之间的可变性和不同模型的输出之间的低一致性。我们研究了针对特定类型的漏洞进行训练的模型，并与同时针对所有漏洞进行训练的模型进行了比较。我们探索了DL可能认为“难”处理的程序类型。我们研究了训练数据大小和训练数据组成与模型性能的关系。最后，我们研究了模型的解释，并分析了模型用于进行预测的重要特征。我们相信，我们的发现可以帮助更好地理解模型的结果，为准备训练数据提供指导，并提高模型的鲁棒性。我们所有的数据集、代码和结果都可以在https://doi.org/10.6084/m9.figshare.20791240.

指标术语-深度学习，脆弱性检测，实证研究

Ⅰ.介绍

近年来，深度学习漏洞检测工具近年来取得了很好的成果。最先进的（SOTA）模型报告了0.9 F1评分[15]，[35]，并优于静态分析器[6]，[12]。研究结果令人兴奋，因为深度学习可能会为软件保证带来变革性的变化。因此，IBM、谷歌和亚马逊等行业公司对此非常感兴趣，并投入了大量资金来开发此类工具和数据集[19]、[27]、[30]、[46]。

深度学习漏洞检测虽然很有前途，但尚未达到计算机视觉和自然语言处理的水平。我们的大部分研究都集中在尝试一种新出现的深度学习模型，并使其适用于像Devign或MSR数据集[13]，[27]，[47]这样的数据集。然而，我们对模型本身知之甚少。g.，该模型能够/不能很好地处理什么类型的程序，我们是应该为每种漏洞类型构建模型，还是应该为所有漏洞类型构建一个模型，什么是良好的训练数据集，以及该模型拥有什么信息过去是用来做决定的。了解这些问题的答案可以帮助我们更好地开发、调试和在实践中应用这些模型。但考虑到深度学习的黑盒本质，这些问题很难回答。本文并不是要为这些问题提供一个完整的解决方案，而是对这些目标的一种探索。

在本文中，我们调查并复制了一组SOTA深度学习漏洞检测模型，并构建了研究问题和研究来理解这些模型，目的是提炼经验教训和指导方针，以便更好地设计和调试未来的模型。据我们所知，这是第一篇系统地研究和比较了各种SOTA深度学习模型的论文。在过去，查克拉博蒂等人。[7]探索了目前现有的四种模型，如VulDeePecker [24]、SySeVR [23]和Devign [47]，并指出，使用合成数据训练的模型在真实测试集上的精度较低，模型使用变量名等虚假特征来进行预测。

*我们构建了我们的研究问题，并将其分为三个领域，即模型能力、训练数据和模型解释。*具体来说，我们的第一个目标是了解深度学习处理漏洞检测问题的能力，特别是关于以下研究问题：

RQ1模型对漏洞检测结果是否一致？一个模型的不同运行和不同模型之间的变量是什么？

RQ2某些类型的漏洞是否更容易检测？我们应该为每种类型的漏洞建立模型，还是应该建立一个可以检测所有漏洞的模型？

RQ3具有某些代码特性的程序是否很难被当前模型正确预测，如果是，这些代码特性是什么？

我们的第二项研究集中在训练数据上。我们的目的是了解训练数据的大小和项目组成是否会以及如何会影响模型的性能。具体来说，我们构建了以下研究问题：

增加数据集的大小是否有助于提高漏洞检测的模型性能？

1558-1225/23/$31.00©2023IEEE2237
DOI 10.1109/ICSE48619.2023.00188

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

RQ5训练数据集中的项目组合如何影响模型的性能？

最后，我们的第三个研究领域是模型整合。我们使用SOTA模型解释工具来调查：

RQ6用于预测的模型有什么源代码信息？这些模型对重要的特征是否一致？

为了回答这些研究问题，我们调查了SOTA的深度学习模型，并在其原始数据集上成功复制了11个模型（见第二节）。这些模型使用了不同的深度学习体系结构，如GNN、RNN、LSTM、CNN和变形金刚。为了比较这些模型，我们设法让9个模型与Devign和
MSR，两个流行的数据集。我们选择了这两个数据集
因为(1)这两个数据集都包含真实世界的项目和漏洞；(2)大多数模型在其论文中使用Devign数据集进行评估和调整；(3)MSR数据集包含310个项目，其数据有关于漏洞类型的注释，这是研究我们的RQs所需要的。我们通过精心设计的实验（第三节）和对威胁的考虑（第四节）发现了我们的6个RQs的发现。总之，我们的研究贡献包括：

1)我们对深度学习漏洞检测模型进行了全面的调查。
2)我们提供了一个复制包，包括11个SOTA深度学习框架的训练模型和数据集；
3)我们设计了6个RQs来理解模型能力、训练数据和模型解释；
4)我们构建了rq的研究和实验结果；和
5)我们准备了有趣的例子和数据来进一步研究模型的可解释性。

II .对模型及其繁殖情况的调查

为了收集SOTA的深度学习模型，我们研究了2018年至2022年的论文，并使用了微软的CodeXGLUE排行榜[1]以及IBM的缺陷检测D2A排行榜[2].我们使用了所有我们能找到的开源模型，并成功地复制了11个模型。在我们的数据复制包中给出了模型的完整列表和我们未能复制一些模型的原因。

如表一所示，复制的模型涵盖了各种深度学习架构。Devign [47]和ReVeal [7]在属性图[47]上使用了GNN，这些属性图[47]集成了控制流、数据依赖关系和AST。ReGVD [31]在令牌上使用了GNN。*代码2Vec在AST上使用了多层感知器（MLP）*。虚拟定位器[22]和SySeVR [23]是基于RNN和Bi-LSTMs的序列模型。最近的深度学习检测使用了预先训练好的变压器，包括CodeBERT [14]，VulBERTa-CNN [16]，VulBERTa-MLP，PLBART [2]和LineVul [15]

1https://microsoft.吉图布。io/CodeXGLUE
2https://ibm.吉图布。io/D2A

表一：11个复制模型

模型	年	架构	数据集
Devign [47]	2019	GNN, 属性图	邪恶的
ReVeal [7]	2021	GNN, 属性图	邪恶的，报复的
ReGVD [31]	2022	GNN，令牌	邪恶的
CodeBERT [14]	2020	Transformer	邪恶的
VulBERTa-CNN [16]	2021	变压器，CNN	秃鹰，德雷珀，小牛肉μVulDeePecker，德维恩，D2A
VulBERTa-MLP [16]	2021	变压器，MLP	
PLBART [2]	2021	Transformer	邪恶的
LineVul [15]	2022	Transformer	MSR
Code2Vec [3]	2021	MLP, AST	邪恶的
SeSyVR [23]	2018	递归神经网络	萨德，NVD
VulDeeLocator [22]	2020	双LSTM	萨德，NVD

对于我们的rq，我们使用了Devign [47]和MSR [13]数据集。我们研究了这11个模型在其原始论文中使用的数据集，见表一中的数据集。我们发现，Devign数据集已经对11个模型中的8个进行了评估和调优。它是一个平衡的数据集，由大致相同数量的脆弱和非脆弱的示例组成，总共有27,318个数据点（每个示例也被称为一个数据点）。LineVul使用的是MSR数据集，这是一个最近可用的数据集。它是一个不平衡的数据集，由10,900个脆弱的示例和177,736个非脆弱的示例组成。这些示例用它们的源项目和共同弱点枚举项（CWE）进行标记，后者指示了漏洞的类型。我们利用数据集的这些特性来处理一些rq。

我们使用模型的原始数据集和设置复制了模型的结果，如表2所示。A、P、R和F的列表示深度学习漏洞检测中常用的指标，包括准确性、精度、查全率和F1。我们的复制结果能够计算出与原始论文相比2%的差异。例外是ReVeal，作者证实了我们的结果修复了原始论文中的数据泄漏错误，以及Devign[3]复制由沙特尔等人[7]发布，因为原始的Devign代码不是开源的。

为了实现模型的比较，我们改进了模型的实现，以同时支持Devign和MSR数据集。*在对RQs进行实验时，我们排除了vulde定位器和SeSyVR，因为它们不能轻易地对Devign和MSR数据集进行修改*。因此，我们使用了其余的9个模型来研究rq。

III .研究问题和发现

我们将研究问题组织为三个领域，即模型能力、训练数据和模型间-

3https://github.com/saikat107/Devign

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

表二：在其原始数据集上的模型复制。繁殖结果报告为3个随机种子的平均值。‘- ’指出，在他们的论文中没有报道过一个指标的结果。这些数字是按百分比计算的。

模型	论文结果				我们的复制品			
	A	P	R	F	A’	P’	R’	F’
邪恶的	59	54	63	57	56	50	71	59
雷维尔	63	57	75	64	53	48	71	56
ReGVD	63	–	–	–	62	62	46	52
代码BERT	62	–	–	–	64	59	54	55
VulBERTa-CNN	64	–	–	–	64	60	59	59
VulBERTa-MLP	65	–	–	–	63	60	58	59
普尔巴特	63	–	–	–	62	58	59	59
LineVul	–	97	86	91	99	96	88	92
代码2Vec	62	–	–	–	59	55	58	57
SeSyVR	98	90	92	90	94	88	84	86
VulDee定位器	99	98	–	97	98	99	96	98

*tation.*分别见第III-A至III-C节。对于每个RQ，我们展示了动机、研究设置和我们的发现。

A.深度学习模型的能力

RQ1模型对漏洞检测结果是否一致？模型的不同运行的变化是什么
跨不同的模型？

动机：我们知道，当使用不同的随机种子时，深度学习模型的性能可能会在不同的训练运行中有所不同。在这个RQ中，我们的目标是测量漏洞检测，这种可变性实际存在多少。此外，我们还希望发现在不同的深度学习模型和具有相似架构的模型之间存在多少一致性。我们希望我们的发现能够告知开发人员和研究人员，在这些工具所报告的数字背后可能存在的不确定性。

研究设置：我们在Devign数据集的同一个训练/有效/测试分区上使用3个不同的随机种子来训练模型。我们使用这个数据集是因为几乎所有的模型都对它调整了它们的超参数。我们测量了稳定输入的百分比——一个对所有3个随机种子具有相同的二进制标签的输入。然后，我们比较了各个模型之间的稳定输入，以衡量它们的一致性。

结果：在表三中，我们报告了整个数据集在所有稳定测试下的稳定输入的百分比和在稳定测试下的稳定输入的百分比。*我们还报告了3个种子（在测试数据集上）在标准测试下的f1分数的变化*
F1.

我们的结果显示，平均34.9%的测试数据（30.6%的总数据）报告了不同的预测，依赖于在训练中使用的种子。处理属性图的GNN模型排名前2位；特别是对于ReVeal，对于50%的测试数据，它的输出在运行之间发生变化。与GNN和变压器模型相比，代码2Vec报告的可变性最小。有趣的是，我们发现不稳定的输入与更不正确的预测有关——稳定的在所有种子中，输入总共有19%的错误预测，不稳定的有47%。

虽然许多例子报告了不同运行之间的不同预测，但我们发现F1测试分数没有那么大的变化，其标准差为2。平均9。也就是说，对于大多数型号，我们预计会有95%的性能
当对多个随机种子进行测量时，测量值应在高于或低于平均性能的5.8%的范围内。

表三：Devign数据集上3个随机种子的可变性

模型	稳定的所有	稳定试验	Stdev测试-F1
雷维尔	55%	50%	2.73
邪恶的	57%	55%	2.24
VulBERTa-MLP	60%	58%	3.13
普尔巴特	72%	67%	1.03
LineVul	72%	67%	3.46
代码BERT	72%	69%	2.78
VulBERTa-CNN	74%	71%	2.60
ReGVD	74%	72%	7.33
代码2Vec	89%	77%	0.78

表四显示，深度学习模型学习了不同的分类器，只有7%的测试数据（和7%的总数据）被所有模型同意。3个GNN模型同意了20%的测试示例（总计25%），而3个性能最好的变压器（LineVul、PLBART和VulBERTa-CNN）同意了34%的测试数据（总计44%的测试数据）。但当我们比较所有5种变压器型号时，只有22%的测试示例（总共29%）是一致的。*不同模型之间的低一致性意味着，当没有地面真实标签时，作为不同模型进行比较的差异测试方法可能有有限的用途。*

表四：不同模型之间的一致性

模型	同意的所有	约定的试验
所有9款车型	7%	7%
所有3个GNN模型	25%	20%
前3个变压器型号	44%	34%
所有5个变压器型号	29%	22%

RQ2某些类型的漏洞是否更容易检测？我们是应该为每种类型的漏洞构建模型，还是应该构建一个可以检测所有漏洞的模型？

动机：在传统的软件保证技术中，如程序分析，我们使用不同的算法来检测不同的漏洞。某些类型，e。g.，无限循环比其他类型更难检测，例如，内存泄漏，因为一个需要跟踪符号值和关于循环的推理，而另一个只需要检查空闲内存是否在分配后被调用。在这个RQ中，我们感兴趣的是了解，对于深度学习漏洞检测器，某些类型的漏洞是否比其他漏洞更容易检测。考虑不同类型的

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

漏洞有不同的语义和根本原因，我们还希望获得一些见解，关于我们是否应该为每种类型的漏洞建立一个模型或一般的漏洞（不将它们划分为类型），就像当前大多数工作所做的那样。

研究设置：在这里，我们研究了基于漏洞类型的模型，因此我们使用了MSR数据集。MSR数据集中的示例用CWE进行了注释[4].使用这些CWE类型，我们将这些漏洞分为5类，即缓冲区溢出、值错误、资源错误、输入验证错误和特权升级，如表V中的漏洞类型所示。我们的标准是：(1)每一组都包含具有相似的根本原因和语义的bug，以及(2)每一组都有一个足够大的数据集来有效地训练模型。列Total列出了从MSR数据集收集的示例的数量，包括所有具有CWE注释的补丁示例。

具体来说，缓冲区溢出是由于读缓冲区边界之外的内存造成的。g，CWE125“出界读”和CWE-787“出界写”。完整的CWE列表给出了我们的映射到5组。值误差包括CWE190“整数溢出”、CWE-369“除以零”和CWE-682“错误计算”的例子。这种错误是由于通过数据处理或算术操作传播不正确的值而造成的。资源错误是由于错误地释放或使用内存或文件指针等资源而引起的，通常使用类型状态分析[36]检测到。CWE的例子包括CWE-415“双自由”和CWE-404“资源关闭不当”。输入验证错误是由于使用外部输入而没有验证其是否正确/良性而造成的，例如，CWE134“使用外部控制的格式字符串”和CWE-89“在SQL命令中使用的特殊元素的不当中和”（“SQL注入”）。它们通常使用污染分析[40]进行检测。最后，特权升级是由于缺少适当的权限检查和允许未经授权的实体执行特权命令或查看特权数据而造成的，例如CWE-264：“权限、权限和访问控制”和CWE-255：“凭据管理错误”。

表五：五种类型的漏洞

漏洞类型	合计	CWE示例
缓冲区溢出	37,291	CWE125, CWE-787
值错误	15,126	CWE190, CWE-369
资源错误	33,748	CWE-415, CWE-404
输入验证错误	25,514	CWE134, CWE-89
权限升级	32,749	CWE-264, CWE-255

我们将每种bug类型的数据集划分为具有80%/10%/10%比率的训练、有效和测试数据集。我们分别使用5组bug，以及a结合模型与所有的bug类型进行训练以进行比较。这反映了现实世界中的场景，即使用特定漏洞类型进行训练的模型可能更集中，但组合后的模型可以使用更多的数据进行训练。我们报告了每个模型的样本类型性能和跨样本类型性能。当训练数据和测试数据具有相同的bug类型时，相同bug类型的性能报告测试F1分数。当训练数据和测试数据有不同的错误类型时，跨错误类型的性能报告测试F1分数。

在这个实验中，VulBERTa-CNN、VulBERTa-MLP和一些Code2Vec的模型没有报告有效的结果，因为它们总是对测试数据预测相同的类。

发现：我们在图1中展示了结果。条形图报告相同类型设置的F1分数，圆圈报告跨错误类型的性能。每个bug类型都与4个跨bug类型测试集相关联，因此每个条形图有4个圆，除了组合模型有5个圆，每个圆表示组合模型上的bug类型运行测试。

通过分析相同的错误类型的性能，我们发现模型并不总是同意哪个类型的漏洞是最容易，并且不同类型的漏洞在不同的模型中获得了最好的F1分数。有趣的是，输入验证和资源错误（橙色和红色条）报告的性能通常低于其他类型。相反，与其他类型相比，缓冲区溢出和值错误（蓝色和紫色条）通常报告了更好的性能。在传统的程序分析中，这些类型很难被检测到，因为它们需要跟踪变量值，有时还需要对循环进行推理。

Devign和ReVeal在属性图上使用了GNN体系结构。他们的条形图显示了相似之处。对于其余的模型，资源错误（红色条）显示了5种漏洞类型中最低的性能。一种可能性是，资源分配和空闲可以在代码中位于很远的位置，而变压器模型不能捕获如此长期的依赖关系。另一种可能性是，这种错误覆盖了各种资源，而训练数据集可能没有为每个资源包含足够的数据，以便模型提取模式。

与具有特定漏洞类型的模型训练的模型相比，组合模型（棕色条）通常性能较差，但对于某些漏洞类型，如输入验证和资源错误，组合模型可以表现得更好。例如，对于CodeBERT，组合模型比其他5个模型获得了更高的f1。棕色条中的圆圈显示，特权升级和资源错误报告的准确性相对较低，但对于所有漏洞类型，组合模型比特定类型的模型报告了更好的性能。

通过分析跨漏洞类型的性能，我们发现在大部分时间内，除了LineVul外，跨漏洞类型检测报告的性能要低得多，这意味着不同的漏洞类型代表了不同的数据分布。LineVul似乎可以很好地处理值错误。应用模型时

4https://cwe.米特尔。org

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

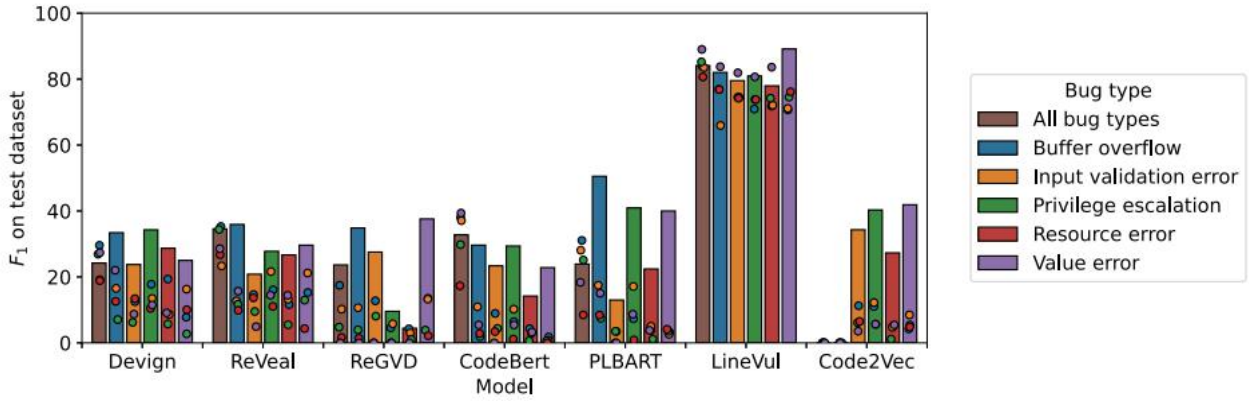


图1：相同不同类型和跨不同类型的性能。条形图表示相同类型的性能；圆圈表示彼此类型的跨类型性能。

使用其他漏洞进行训练后，值错误报告的性能高于相同错误的性能。

RQ3具有某些代码特性的程序是否更难被当前的漏洞检测模型正确预测，如果是，这些代码特性是什么？

动机：在这里，我们研究了我们是否可以描述那些不能很好地预测和对深度学习模型“困难”的程序，以及不同的模型是否对这些困难达成一致。知道我们不能处理的项目为我们未来的工作改进提供了一个很好的目标。在程序分析中，我们知道某些特性很难处理，比如循环和指针。我们想知道这些特性是否对深度学习也很困难。

研究设置：在第一步中，我们准备了一个代码特性列表以进行调查。我们认为与程序分析工具比较哪些程序类型很难处理是很有趣的。因此，我们的方法是列出对程序分析很重要的代码特性，然后检查它们是否也对深度学习工具产生了影响。

我们总共获得了12个代码特性。有些是控制流相关，e。*g.，while，fof，如果，调用和切换以及无条件的中断，继续，返回；有些是数据结构和指针相关，如数组和指针（包括字段访问）；最后一些是辅助结构，如注释和宏。基于这个特征列表，我们应用了树的保姆*[5]使用解析器来计算每个函数中出现的代码特性的频率。

*为了理解某些代码特征是否会使深度学习模型更难预测，我们使用了一个多元逻辑回归（LR）模型(见等式*1)，将代码特性与一个函数被正确预测的可能性联系起来。如果一个具有某些代码特征的函数更有可能被正确预测，我们认为它更容易进行深度学习，反之亦然。给定一个具有特定特征组合的函数，等式中的Y1报告了预测的概率

5https://树保姆。吉图布。io/tree-sitter/

深度学习模型将会正确地预测它。xi是函数中每个代码特性的计数。βiS是从数据中学习到的系数。每个βi与一个代码特性x关联i.当βi为负值，该值为βi* xi降低了预测的正确性概率，所以我们称这些特征对模型很困难。同样地，具有正系数的代码特性也被称为是容易的。与此同时，一个大的βi这意味着代码特性x的计数的增加i大大增加了正确预测的预测概率，反之亦然。

$$Y = \sigma\left(\sum_i \beta_i * x_i + \beta_0\right) \quad (1)$$

我们根据在验证集上的预测对LR模型进行训练，然后使用训练后的LR模型从测试集中找到困难/简单的例子。量化在测试集的一个例子的困难，我们在LR模型中使用logit输入的sigio函数：！(x) =工 βi* xi.

我们把这个数量的否定表示为难度分数。难度分数较高的函数比难度分数较低的函数更有可能被错误预测。为了评估这个LR模型的有效性，我们在测试集中选择了顶部和底部10%的例子，并根据它们的难度分数进行排序。然后，我们通过比较模型在我们选择的简单和困难的数据集上的性能来评估我们的LR模型的预测。

需要注意的是，最初我们尝试了统计显著性检验和基于相关性的方法来将代码特征与模型精度联系起来。然后，我们意识到这些方法一次只考虑一个特征。例如，具有3个循环、400个指针和500个宏的函数表现良好，而具有300个循环、300个指针和100个宏的函数表现不好。我们不能得出性能差异是由循环、指针还是宏引起的结论。另一方面，LR模型同时合并多个代码特征，并考虑这些特征组合的效果。

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

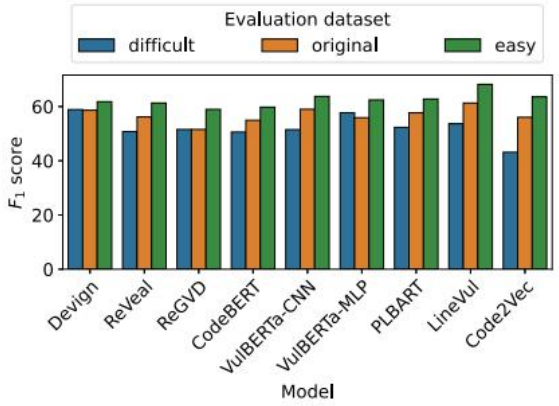


图2：根据LR模型难度评分选择的评价集的比较性能，在Devign数据集上平均超过3个随机种子。“原始”是在原始测试集上的性能，如表二所示。

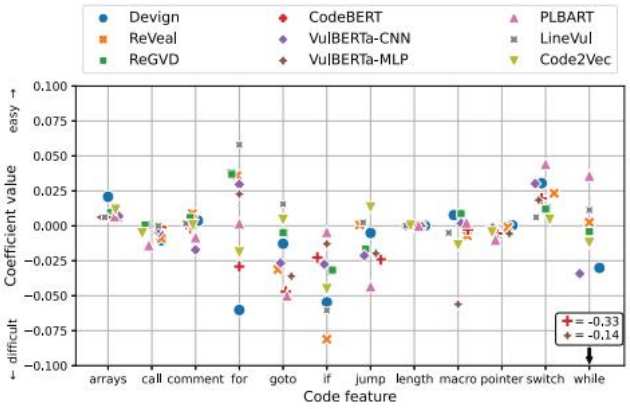


图3：基于来自Devign数据集的稳定例子训练的LR模型的系数。

结果：图2显示，所有9个模型在简单数据集上的表现都比在困难数据集上的表现更好。所有模型的易/难性能之间的平均差异为10.3%。对于大多数模型（9个模型中的7个），原始测试集的性能介于困难集和容易集的性能之间。这些结果表明，LR模型和难度分数对于为深度学习模型选择困难和简单的例子是有效的。

图3绘制了为每个深度学习模型训练的LR模型中每个代码特征的系数。我们发现，所有模型的特征调用、长度和指针的点都被分组在一起，这意味着所有的模型都同意这些特征的重要性。有趣的是，所有这些点都位于0附近，这表明这些特征并没有产生很大的影响。另一方面，功能，goto，如果，跳转，切换，而改变不同的模型。这些都是与控制流相关的结构。

我们还观察到，对于所有的模型，数组和开关都与正系数相关，而对于大多数模型，如果，goto，而虽然属于负范围。特别是，如果所有模型的系数都为负。在一个程序中，if、goto和的高数量表明其高循环复杂度[29]，这种类型的程序对程序分析也具有挑战性。特别是，像Devign这样的基于属性图的模型使用了控制流信息，因此，对于、goto、跳转和当的特性都是负的（困难的）是有意义的。

B. 培训数据

增加数据集的大小是否有助于提高漏洞检测的模型性能？

动机：难以获得高质量的漏洞检测数据。在过去，我们使用了自动标记方法，如静态分析和挖掘更改提交。这些方法可能会引入不正确的标签[7]。我们还使用了手动标签[47]，它生产[25]很慢。这个研究问题有助于我们理解目前可用的数据集是否足够大来训练模型，以及增加数据集的大小是否可以显著提高模型的性能。

*研究设置：为了调查这个RQ，我们结合了Devign和MSR数据集，即不平衡数据集。*由于Devign中使用的项目与MSR中使用的项目有重叠，因此我们排除了82个匹配提交id的重复示例。这总共生成了一个包含194,285个示例的数据集。*一些已发表的模型最初是在平衡模型上调整的，如Devign，所以我们还构建了一个包含45363个例子的平衡数据集，采用MSR中所有脆弱的例子，然后随机抽样相同数量的非脆弱的例子*。对于每个数据集，
我们将10%的数据作为所有模型的测试集。然后，我们准备了10%，20%，90%，100%的其余数据来训练10个模型，以观察f1评分如何...
当数据集的大小增加时，测试集就会发生变化。此外，我们准备了两个占总数据的1%和5%的小数据集，来实验模型能够学习一些东西所需的最小数据量是多少。

发现：我们在图4中总结了我们的结果。图4a和图4b也显示了类似的趋势。一般来说，当我们添加更多的数据时，所有模型的性能都有所提高。然而，改善并不显著。比较100%的数据与10%的数据，当我们取平衡数据集的所有模型的平均值时，测试集上的F1报告没有差异。
对于不平衡的数据集，F1评分的值提高了0。平均16。

在图4a中，在这些模型中，当我们每次多添加10%的数据时，只有LineVul显示出一致的改进。当我们增加训练数据集时，所有其他模型都会波动，这表明增加的数据并不总是能带来好处。例如，对于VulBERTa-MLP，F1值平均下降为0。1.在增加数据集大小的过程中。该模型的性能训练率为100%

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

比用10%的数据集训练的模型的性能低0.08。似乎还有其他的因素
而不是数据集的大小在性能方面发挥了非常重要的作用。在图4b中，ReVeal是随着增加的数据集改进最大的模型。在100%的数据集上，ReVeal正在追赶最佳的模型LineVul。然而，Devign与在属性图上使用GNN的架构类似，但它并没有显示出额外数据的这种好处。

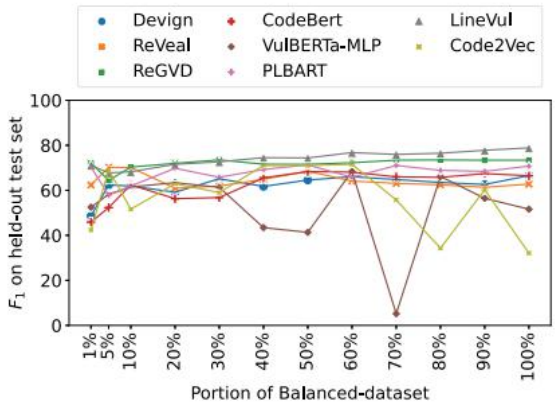
基于小数据集的模型实验
筛选1%和5%的总数据)显示，令人惊讶的是，除了CodeBERT外，我们只使用5%（约2268个数据点和1134个脆弱的例子），就可以使模型达到良好的性能。当使用不平衡的数据进行学习时，转折点就会稍晚一些。例如，ReGVD和CodeBERT需要占总数据的50%（96.4 k）和30%（57.8 k）。其他模型需要大约5-10%的数据（9.7-19.4k）才能达到一个高点。有趣的是，这个数据集有10.8%的脆弱示例；也就是说，模型需要大约1048-2095个易损示例来实现良好的性能——与平衡数据集的设置相当。

RQ5训练数据集中的项目组合如何影响漏洞检测模型的性能？

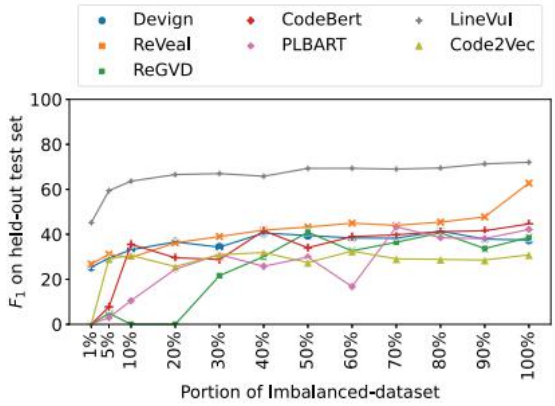
动机：在这个RQ中，我们的目的是进一步了解如何构建一个良好的漏洞检测训练数据集。具体来说，我们感兴趣的是知道训练数据集中项目的多样性是否有帮助。我们也感兴趣的了解是否不同的项目确实代表不同的分布，当测试和训练数据来自相同的项目，模型可以明显表现更好，当训练和测试数据来自不同的项目，模型是否可以概括看不见的项目。

研究设置：我们为本研究设计了两个实验。在第一个实验中，我们准备了一个非多样化的训练数据集和一个多样化的训练数据集，并将训练后的模型与同一测试集上的两个数据集进行了比较。在MSR数据集中，我们发现Chrome包含76k个例子，是所有310个项目中最大的。我们使用它作为非多样化的数据集。我们在5倍交叉验证设置中进行了这个实验，以消除在选择项目时可能存在的潜在偏差。对于每一次折叠，我们从MSR数据集中随机抽取10 k个例子作为测试集。然后，我们排除了Chrome和测试集中使用的项目，并从其余的项目中随机抽取了总共76 k个示例（与Chrome的数量相同）。不同数据集中的平均项目数量为50.6个，跨越5倍。

*在第二个实验中，我们准备了一个混合项目设置，其中测试集与训练集分开，而不考虑源项目，训练集和测试集中的一些例子可能来自相同的项目。*这是我们大多数深度学习论文所在的背景



平衡数据集的(a)结果。100%的数据集大小为= 45,363。



在不平衡数据集上的(b)结果。100%的数据集大小为= 194,285。

图4：当模型通过增加训练数据集进行训练时，保留测试集的F1分数。

评估。*我们还构建了一个跨项目设置，其中测试集示例必须来自不同的项目，而不是代表训练集的项目。*这种设置有助于我们理解，当我们使用没有看到测试项目的现成训练的训练深度学习漏洞检测模型时，我们是否会有显著的性能下降。

我们还在5倍交叉验证设置中使用了MSR数据集。对于每一个折叠，我们首先构建一个跨项目设置的测试集，包括来自随机选择的项目的所有例子，直到集合包含至少10k个例子。因为每个项目都有不同数量的示例，所以结果集略大于10k个示例。然后，我们通过将剩余的示例随机划分为测试集（10 k）、验证集（10 k）和训练集（剩余的示例，约158 k）集，为混合项目设置构建了一个测试集。我们使用158 k个训练示例和10 k个验证集对模型进行训练，然后在跨项目设置和混合-的测试集上运行它

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

项目设置。

结果：第一次实验的结果如图5所示。我们使用箱线图总结了5倍交叉验证的结果。令我们惊讶的是，我们发现，对于所有的模型，与只包含Chrome的训练集相比，多样化的训练集并没有提供任何好处。事实上，6个模型中有5个在非多样性数据上报告了更高的中位数表现。

在第二个实验中，图5b显示，在所有模型中，混合项目的表现明显优于交叉项目(F1得分的平均值和最大差异为0。分别为11和0.32)。这意味着查看来自一个项目的数据确实可以帮助预测来自同一项目的其他数据。与直接使用的已经训练过的现成模型相比，脆弱性检测可以从定制的训练模型中大大受益。对于LineVul，5倍报告了在交叉项目设置中非常不同的性能，这表明给定一个目标测试集，一些项目比其他项目更有用的培训。这些结果还表明，模型可以学习从数据集的项目特定属性中检测漏洞，如样式、语言特性或命名约定。我们相信，这推动了进一步的研究，因果检测的错误的泛化。

C.深度学习内部人员

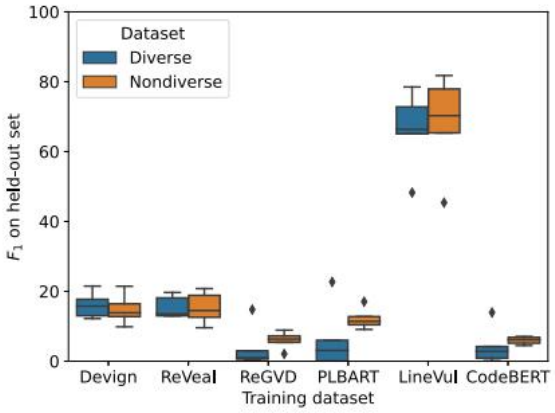
RQ6这些模型使用什么代码信息来进行预测？这些模型对重要的特征是否一致？

动机：最近的深度学习漏洞检测器已经取得了高性能，e。g.，LineVul报告的F1评分为91%。我们想知道为什么这些工具可以表现良好，以及模型是否使用了漏洞的语义的任何方面来做决策。例如，为了检测缓冲区溢出，一个基于语义的程序分析工具可以识别有关字符串长度和缓冲区大小的依赖语句和原因。我们还想调查不同的模型是否同意什么是重要的特征。

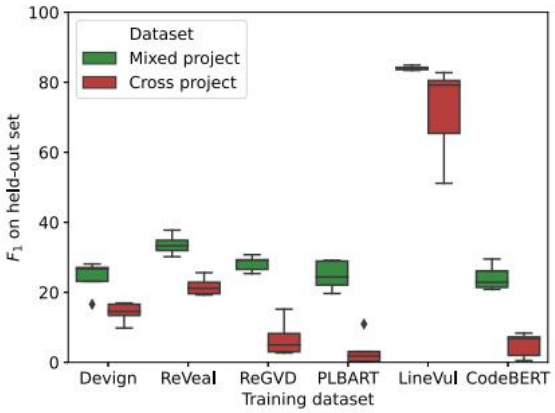
研究设置：我们调查了一套SOTA深度学习解释工具，特别是对于GNN和变压器架构。我们使用GNN解释器[44]和ReGVD，LIT [33]、[37]、[39]表示LineVul、VulBERTaCNN、VulBERTa-MLP、CodeBert和PLBART，在我们研究的所有工具中，这两种工具对我们的模型最有效。在我们的数据复制包中，我们

记录了为什么其他模型不能与GNNExcranerer和LIT以及我们尝试过的其他智能工具一起工作的原因。

为了解释这些模型，GNNExcranerer和LIT都提供了分数来衡量代码特性的重要性。GNNExcranerer为图中的每条边给出分数，LIT为程序中的每个标记给出分数。*为了比较GNNExcranener和LIT报告的结果，我们对工具的输出执行了以下归一化。*对于GNNExcranener，我们通过取其所有入事件边的平均分数来计算每个节点的分数



(a)多样化vs。非多样化性能



(b)混合项目vs。跨项目绩效

图5：对培训数据中的项目组成的研究。条形图表示均值f1，区间表示标准差。

在[21]中完成。节点中的每个标记都将使用此分数。对于GNNExcrener和LIT，我们通过汇总文献[15]的分数来计算每个源代码行的分数。*对于测试数据集中的每个示例，我们选择了得分前10个最高的行，如在[15]，[21]中所做的那样，以形成重要的特征集，表示表示*

作为我。我们认为这10行是模型用于做出决策的最重要的代码特征。

为了度量两个重要特征集的相似性I_A和我B由模型A和模型B报告，我们计算了交点$I_{ab}= I_A \cap I_B$.*我们还使用Jaccard索引[20]作为另一个度量，定义为：*

$$J(I_A, I_B) \quad \frac{|I_A \cap I_B|}{|I_A \cup I_B|} \qquad (2$$

为了报告相似性，我们首先计算I_{ab}和$J(I_A, I_B)$，然后我们取I的平均值ab和$J(I_A, I_B$各自地

我们从以下几组中取样例子

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

检查：1）实例脆弱，模型检测正确（正确）；2）实例非脆弱，模型预测为脆弱（假阳性）；3）实例脆弱，模型预测为非脆弱（假阴性）。

结果：在表六中，我们报告了每个模型对的重要特征集的相似性。结果表明，Linevul和ReGVD在所有模型对之间有最大的重叠。在10行重要特征中，两个模型平均共享6.88行。我们发现，有趣的是，尽管模型在个别预测上可能有很多分歧（见表四），但它们使用的代码信息是重叠的。所有的模型对都至少有3行共同的重要特征。Devign是唯一基于属性图的GNN模型，与其他模型的重叠程度较低，与PLBART的重叠程度最低，平均为3.38行。与其他变压器型号相比，PLBART使用了不同的变压器架构，并且也报告了与其他变压器型号的低重叠。

我们还使用与表六相同的方法分析了修正后的预测例子。我们发现重要的特征集对修正后的例子有更多的重叠。g.，Linevul和ReGVD仍然有最多的共同点，在重要的特性集中共享7.29行。

表六：由I测量的每两个模型之间的重要特征集的相似性ab（以蓝色形式报道）和J（IA,IB穿黑衣服最大值和最小值为粗体。

模型	L-Vul	编解码器	普尔巴特	邪恶的	ReGVD	V-CNN	V-MLP
L-Vul	–	0.58	0.31	0.38	**0.60**	0.32	0.40
编解码器	6.84	–	0.37	0.35	0.50	0.31	0.40
普尔巴特	4.08	4.89	–	**0.24**	0.27	0.30	0.28
邪恶的	4.90	4.54	**3.38**	–	0.42	0.27	0.30
ReGVD	**6.88**	5.86	3.58	5.36	–	0.33	0.40
V-CNN	4.20	4.06	3.88	3.87	4.35	–	0.30
V-MLP	4.83	4.74	3.71	3.95	4.83	3.97	–

从我们的人工检查中，我们观察到模型通常突出显示for、if和while的代码行，以及函数签名作为重要特性。这些模型还经常突出显示逻辑集、元集和元集的内存操作，以及打印包含错误或打印错误的错误信息的行。为了证实这一观察结果，我们使用这些关键字对代码进行了分析，并在表7中报告了结果。使用错误
例如，在不失一般性的情况下，前7行报告

表七：经常突出显示的代码特性

模型	错误	打印	分派	为了	memset	memcpy	在…期间	如果
L-Vul	.035	.021	.017	.042	.003	.003	.005	.115
编解码器	.027	.015	.015	.032	.004	.002	.004	.111
普尔巴特	.025	.012	.011	.034	.003	.001	.005	.114
邪恶的	.023	.012	.014	.069	.002	.002	.007	.147
ReGVD	.034	.027	.022	.038	.004	.005	.005	.132
V-CNN	.024	.014	.011	.028	.002	.002	.004	.116
V-MLP	.031	.018	.015	.020	.003	.003	.004	.095
函数	.010	.010	.010	.026	.002	.002	.004	.108
I/F	2.79	1.78	1.52	1.47	1.39	1.21	1.12	1.10

```
1静态int asf_read_ext_content（asf_read_ext_content上下文，C→指南参数*g）// (3)
2 {
3个f背景*asf=->priv_data;
4 AVIOContext *pb = s->pb;
5 uint64_t大小= avio_rl64（pb）；//(7)
6 uint16_t nb_desc = avio_rl16（pb）；//(8)
7 int i，ret；
8为（i = 0；i < nb_desc；i++）{
9 uint16_t name_len，类型，val_len；//(5)
10个uint8_t*名称= NULL；
11个name_len = avio_rl16（pb）；
12如果(！name_len )
13返回AVERROR_INVALIDDATA；//(6)
14个名字是= av_malloc（name_len）；
15如果(！名字
16返回平均数（ENOMEM）；//(9)
17 avio_get_str16le（pb，name_len，名称，//(4)
18 name_len )；//(2)
19型= avio_rl16（pb）；//（10）
20个val_len = avio_rl16（pb）；
21 if（（ret =进程_元数据（s，name，C→name_len，val_len，type，&s->元数据））C→< 0）//(1)
22 ret =进程_元数据（s，名称，name_len，C→val_len，type，&s->元数据）；
23 av_freep(&name);
24如果（ret < 0）
25返回的人员数量；26 }
27 }
```

清单1：通过LineVul已成功检测到内存泄漏。LIT报告的前10行用黄色突出显示。

每个模型的重要特征集中发生错误的概率（重要特征集中的误差总数/重要特征集中的总行数）。在第1行Func行中，我们展示了在一个函数中发生错误的概率（错误的总数/一个函数中的总行数）。通过比较这两者，我们发现在重要特征集中发生错误的概率是在程序中平均发生错误的概率的2.79倍，如第I/F行所示。这意味着错误最好被选择到重要的特征集中。在所有的特征中，错误、打印和alloc的比例最高。

我们的第二个观察结果是，变压器的模型
有时，他们会在没有看到根本原因的情况下做出预测。这是因为转换器模型采用固定大小的输入，并且一些代码，有时包括根本原因，是被截断了。有趣的是，这些模型仍然能够正确地预测一个函数在高f1分数下是否脆弱。

第三，我们检查了所有模型所遗漏的漏洞，并研究了用于检测这些漏洞的重要特征集。我们发现这些漏洞是非常特定于应用程序的。遗漏这些bug可能是因为没有足够的训练数据来处理这种类型的bug。

在清单1中，我们展示了一个示例，其中的预测是正确的，但所使用的特征不是因果关系的。该示例包含一个内存泄漏漏洞，并且LineVul预测该示例易受攻击。在第14行分配给名称的内存从未被释放。第23行的补丁显示了修复。LIT报告的重要特征集（前10行）是

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

```
1  静态intdecode_frame(自动转换上下文*avctx,
2                                空白数据，int*got_帧，C→数据数据
                                 包*avpkt) //（1）
3  {
4      //  . . . 10行
5      bytestream2_init(&s->gb, avpkt->data,
       C→ avpkt->size); // (5)
6      如果（（ret = ff_tdecode_header（&s->gb，&le，
       C→&off）））{ // (6)
7          av_log（avctx，AV_LOG_ERROR，“无效的TIFFC→标头\n”
                ）; // (3)
8          返回ret；
9      }其他如果（off>=UINT_MAX-14||avpkt->大小<C→off + 14）{ //
       (4)
10         av_log（avctx，AV_LOG_ERROR，“IFD偏移量为C→大于图像
           大小\n”）; // (2)
11         返回AVERROR_INVALIDDATA;
12     }
13     年代->=;
14 // TIFF_BPP不是必需的标记，默认为C→1 //（10）
       s->bppcount = s->bpp = 1; // (9)
15     采用s->光度法测量= TIFF_PHOTOMETRIC_NONE; // (7) s->compr
16     = TIFF_RAW; // (8)
17     //    . . . 140行
18
```

清单2：由于虚假的特性，非脆弱的代码被预测为脆弱的代码。

用黄色突出显示。我们可以看到，它包括我们已经讨论过的“模式”，包括第1行的函数签名，包含错误的行，例如，第13行和第16行，以及第21行的if语句。我们还看到，对于这个项目，变量name_len是重要的，并且包含了多次。但是，这10行中没有一个覆盖第14行的内存分配，这对于理解这个bug很重要。

这个示例表明，模型试图捕获漏洞的模式，而不是对值进行推理，并且很难捕获代码中的长期语义依赖关系。但是，我们也在其他示例中观察到，有时，突出显示为重要的控制结构和内存语句（见表七）可能是该漏洞的依赖语句的一部分，因此它们对于检查bug的根本原因很有用。

清单2显示了一个非脆弱函数的示例，LineVul错误地将其预测为易损函数。该模型突出显示了函数签名（第2行）、“错误”行（第7和10行）、初始化例程（第5行）、如果（第6和9行）和字段分配（第1517行），并预测该函数很脆弱。它表明，基于这些结构的模式做出决策可能会导致错误。

增值对有效性的威胁

我们的观察结果是来自于模型和数据
可用的集合，可能不适用于深度学习漏洞检测。我们同时使用了平衡数据集（Devign）和不平衡数据集（MSR）来减轻这种威胁。这两个数据集都包含了真实世界中的错误。大多数模型在评估中都使用了Devign，所以我们需要它来重现模型（见表I）。然而，我们的数据集可能仍然不能代表现实世界
漏洞分布。我们包含了所有我们可以找到和复制的模型。

RQ2中的分组存在偏见，因为不同的研究人员可能会以不同的方式划分脆弱性类型。在这里，拥有领域知识的两位作者分别检查了CWE列表，并讨论并同意了分组。为了减轻特定项目组成可能带来的偏差，RQ5进行了5倍交叉验证。对于RQ6，我们选择了SOTA模型解释工具；然而，这样的技术可能并不能完美地识别模型所使用的重要特征。RQ2、RQ4和RQ5的实验要求模型与我们定制的模型不附带的定制数据一起工作。我们对我们观察到的任何可疑数据尝试了不同的随机种子，e。g.，当一个模型报告了所有的0或1时。当调优不能解决这些问题时，我们在结果中排除了这些模型。

V.相关工作

一些工作已经对基于机器学习的漏洞检测模型进行了实证研究。查卡波西等人。[7]研究了4 DL模型，研究了合成数据集、数据重复和数据不平衡等问题，并指出了虚假特征的使用，然后利用这些特征来改进其模型设计。唐等人。[38]的目的是确定哪些神经网络结构、向量表示方法、符号化方法是最好的。他们调查了2个模型。Mazuera-Rozo等。[28]评估了1个浅层和
2个关于二元分类和bug类型（非二元）分类的深度模型。在我们完成了我们的研究后，我们发现了两个相关的实证研究。林等人。[26]对9个软件项目评估了6个DL模型的泛化。班等人。[4]评估了6个机器学习模型（其中1个是一个神经网络）和3个软件项目，并研究了两种bug类型的训练。单个错误类型。

近年来，许多漏洞检测模型被提出了多种架构，如MLP[9]，RNN[22]-[24]，[45]，CNN [34]，[42]，变压器[2]，[11]，[12]，[14]-[16]，[32]，[41]，和GNN[5]-[8]，[10]，[17]，[18]，[21]，[31]，[35]，[43]，[47].例如，Devign在属性图[1]上使用了门控图神经网络。LineVul [15]在大量不同的开源项目中使用了一个预先训练过的变压器模型。ReVeal [7]应用SMOTE来解决数据不平衡问题和三重损失，以学习最大限度地分离脆弱和非脆弱的代码。

在这些论文中，大多数模型都是在非分布数据上进行评估的，其中训练集可以包含与测试集重叠的项目和bug类型。罗素等人。[34]，Li等。[24]和Xu等人。[43]训练他们的模型来检测特定类型的漏洞，所有人都发现一些漏洞比其他漏洞更困难。Hin等。[18]在跨项目设置中评估了他们的模型，一次保留一个项目，发现性能略有下降。大多数模型评估在F1等指标上比较了不同的基线，但没有量化预测的一致性。据我们所知，

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

我们的工作是第一次尝试来描述模型不能很好地预测的程序和代码特征。

VI .结论和未来的工作

为了理解深度学习脆弱性检测模型，我们对6个研究问题进行了实证研究。我们的实验表明，平均而言，34.9%的测试数据在运行之间有不同的预测，只有7%的预测在9个模型中是一致的。基于特定类型的漏洞检测通常比为所有漏洞构建的模型执行得更好。模型的性能不会随着数据集的增加而显著提高，并且对于平衡和不平衡的数据集，模型在开始使用大约1k个脆弱的示例时表现良好。我们开发了一个逻辑回归模型，它可以找到模型难以正确预测的程序。解释工具显示，模型使用共同特征进行预测，每前10行共有3.38-6.88行。我们报告了模型经常突出显示为重要特性的代码模式。在未来的工作中，我们计划进一步研究这些模式。

罗马数字 7致谢

我们感谢匿名审稿人提供的宝贵反馈。我们感谢高洪阳为我们的实验提供的计算资源。我们感谢李琦讨论了一个实验指标。这项研究部分是
由美国支持。S.美国国家科学基金会（NSF）颁发，奖项编号为1816352。

参考文献

[1]乔恩。https://github.com/octopus-platform/joern.

[2] Wasi Uddin艾哈迈德，沙特克拉博邦，白沙基雷，和张开伟。对项目的理解和生成的统一预培训，
2021.

[3] Uri·阿隆，梅塔尔·齐尔伯斯坦，奥默·利维和埃兰·雅哈夫。学习代码的分布式表示。*过程ACM程序。长的*，3（POPL），2019年1月。

[4]班欣博、刘石刚、陈超、蔡家隆。一种在软件漏洞检测中对深度学习特征的性能评估。
并发性与计算：实践与经验，31（19），2019年10月。

[5]、曹思聪、孙小分队、波丽丽、英伟、李斌。BGNN4VD：构建双向图神经网络。*信息与软件技术，136：106576,8月*
2021.

[6]曹思聪、孙小分队、波丽丽、吴荣信、李斌、道传奇。基于流程敏感图神经网络的内存相关脆弱性检测。2022.

[7] Saikat，拉胡尔·克奎师那，丁杨瑞博和白沙克雷。基于深度学习的漏洞检测：我们已经存在了吗？《IEEE《软件工程学报》，第48(9)页：3280-3296,2022年。

[8]肖成、王浩宇、华嘉怡、徐国爱、隋穗。利用深度图神经网络静态检测软件漏洞。《ACM的软件工程和方法论学报》，30(3)：38：1-38：33,2021年4月。

[9]大卫科英布拉，索菲亚雷斯，瑞阿布鲁，科琳娜普斯雷努，和哈坎埃尔多格姆斯。关于使用源代码的分布式表示来检测C安全漏洞，2021年。

[10]伊丽莎白·迪内拉，戴汉军，资阳李，梅奈克，乐歌，王可。希望遗憾：学习图转换来检测和修复程序中的bug。*2020年学习代表国际会议。*

丁[11]·杨瑞博、卢卡·布拉蒂、索拉布·普贾尔、亚历山德罗·莫拉里、白沙基雷和查克拉博蒂。从程序对比中学习源代码的（差异）相似性，2021年。

[12]杨瑞博·丁，孙尼佳，郑云会，吉姆·拉雷多，亚历山德罗·莫拉里，盖尔·凯泽，和白沙基雷。天鹅绒：一种新的集成学习方法，以自动定位脆弱的语句，2022。

[13]嘉豪范、李毅、王少华、田恩。阮。一个包含代码更改和cve摘要的C/C++代码漏洞数据集。在*第17届采矿软件存储库国际会议的会议记录，MSR '20，第508-512页，纽约，纽约，美国，2020年。*美国计算机协会

[14]张音峰、郭大亚、唐杜宇、南段、冯晓程、明公、临军寿、秦兵、刘婷、江大新、周明。CodeBERT：一个用于编程和自然语言的预训练模型。CoRR，abs/2002.08155,2020年。

[15]傅迈克尔和查克里特。LineVul：一种基于转换器的线级脆弱性预测。*2022年，IEEE/ACM第19届采矿软件存储库国际会议（MSR），第608-620页，2022页。*

[16] Hazim Hanif和Sergio马菲斯。VulBERTa：针对漏洞检测的简化源代码预训练。*2022年国际神经网络联合会议（IJCNN），第1-8页，2022年。*

[17]文森特J。赫伦多恩，查尔斯萨顿，里沙布辛格，马尼亚提斯和大卫比伯。源代码的全局关系模型。在
学习代表国际会议，2020年。

[18]大卫欣，安德烈，陈华明和M。阿里巴巴。使用图神经网络进行声明级漏洞检测，
2022.

[19] IBM。项目代号，2021年。

[20]保罗Jaccard。高山地区植物区系的分布情况。1.*新植物学家，11(2)：37-50,1912年。*

[21]李毅，王少华、田南。阮。具有细粒度解释的漏洞检测。*在第29届ACM欧洲软件工程基础会议和研讨会上，ESEC/FSE 2021，第292-303页，纽约，纽约，美国，2021年。*美国计算机协会

李[22]、邹庆、徐寿怀、陈朝旭、朱亚伟、金。一种基于深度学习的细粒度漏洞检测器。IEEE关于可靠和安全计算的交易，19(4)：2821-2837,2022年7月。

李[23]、邹庆、徐寿怀、金、朱亚伟、陈朝旭。SySeVR：一个使用深度学习来检测软件漏洞的框架。*IEEE关于可靠和安全计算的交易，19(4)：2244-2258,2022年7月。*

李[24]、邹庆、徐寿怀、新余Ou、金、王素娟、邓志军、钟玉义。虚拟者：基于深度学习的漏洞检测系统。*在2018年网络和分布式系统安全研讨会的论文集中。*互联网协会，2018年。

[25]林将军、盛文、韩长、张军、杨翔。使用深度神经网络的软件漏洞检测：一项调查。IEEE论文集，108（10）：1825-1848,2020年10月。会议名称：IEEE的会议记录。

[26]林冠军、魏晓、张利宇、商高、中航泰、张军。基于神经的深度漏洞发现不神秘化：数据、模型和性能。*神经计算与应用程序，33（20）：13287-13300,2021年10月。*

[27]帅路，郭大亚，任硕，黄俊杰，阿列克谢·斯维亚特科夫斯基，安布罗西奥·布兰科，科林·B。克莱门特、黎明排水沟、江大新、唐杜宇、葛李、周立东、林军寿立俊、周龙长、米歇尔·图法诺、明公、明周、南段、孙达里山、邓邵坤、傅胜宇、刘树杰。codexl：用于代码理解和生成的机器学习基准数据集。CoRR，abs/2102.04664,2021年。

[28]亚历杭德罗·马祖拉-罗佐，阿纳玛里亚·莫吉卡-汉克，马里奥·利纳雷斯夫斯奎斯和加布里埃尔·巴沃塔。浅或深？利用深度学习检测脆弱性的实证研究。*2021年，IEEE/ACM第29届项目理解国际会议（ICPC），第276-287页，2021年5月。*ISSN： 2643-7171。

[29] T.J.麦凯布盖尔语姓氏的英语形式复杂度量。IEEE《软件工程学报》，SE-2(4)：308-320,1976年。

[30]深心灵。竞争编程，2022年。

[31]阮、阮、阮、阮、阮、丁平。ReGVD：重新审视图形神经

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。

用于漏洞检测的网络。*2022年，IEEE/ACM第44届国际会议*
全国软件工程会议：同伴会议程序
（ICSE-同伴版），第178-182页，2022页。
[32] Long Phan，河川，丹尼尔·勒，河阮，詹姆斯·阿尼巴尔，亚历克
佩尔特基安和叶燕芳。CoTexT：多任务学习
变压器，2021年。
[33]马可图利奥里贝罗，萨米尔辛格，和卡洛斯格斯特林。“我为什么要
相信你吗？”：解释任何分类器的预测。KDD ‘16页
1135-1144，纽约，纽约，美国，2016年。计算协会
机械。
[34]，丽贝卡·罗素，路易斯·金，雷·汉密尔顿，托莫·拉佐维奇，雅各布
哈尔，奥兹德米尔，保罗·埃林伍德和马克·麦康利。自动化的
使用深度表示的源代码漏洞检测
学习*第17届IEEE国际会议*
《中国学习与应用》，ICMLA 2018，第757-762页，2019。
出版商： IEEE。
[35]子华、王俊峰、刘胜利、方志阳、杨开元、
和顾兆泉。一种基于代码漏洞检测的方法
关于异构源代码级的中间表示。*秒和*
通勤。网络。，2022，jan 2022.
[36]罗伯特E。斯特罗姆和苏拉也门尼。类型状态：编程
提高软件可靠性的语言概念。*IEEE事务*
关于软件工程学，SE12(1)：157-171,1986。
[37] Mukund，[37]，和奇琪颜。公理属性
深度网络。*在第34届国际会议会议记录上*
关于机器学习-第70卷，ICML‘17页，第3319-3328页。
JMLR。org，2017.
[38] Gaigai唐、莲晓孟、王辉强、任双阴、强
王、林杨、曹伟鹏。神经学的比较研究
自动软件漏洞检测的网络技术。
在2020年关于软件的理论方面的国际研讨会上
工程（TASE），第1-8页，2020年12月。
[39]伊恩·坦尼，詹姆斯·韦克斯勒，贾斯米金·巴斯廷斯，托尔加·博鲁克巴西，安迪
科宁，塞巴斯蒂安·格尔曼，姜艾伦，马希马·普什卡尔纳，凯里
拉德博，艾米丽瑞夫和安袁。语言的可解释性
工具：可扩展的，交互式的可视化和分析的NLP模型。
在2020年自然经验方法会议的论文集上
语言处理：系统演示，第107-118页，在线，
2020年10月。计算语言学协会。
[40] Omer Tripp，马可·皮斯托亚，斯蒂芬·J。芬克，马努·斯里德哈兰，和奥姆里
韦斯曼TAJ：对web应用程序的有效污染分析。*自动控制模块*
《通知》，44(6)：87-97,2009年6月。
王[41]鑫、王亚生、飞美、周平邑、姚万、肖
刘、李李、吴浩、刘金、新疆。SynCoBERT：语法
代码表示的引导性多模态对比预训练，
2021年9月。arXiv:2108.04556 [cs].
吴[42]月明、邹庆、窦贤、杨伟、徐多、
海金。VulCNN：一个受图像启发的可伸缩的漏洞检测
体系*在第44届国际软件会议的会议集*
工程，ICSE ‘22，第2365-2376页，纽约，纽约，美国，5月
2022.美国计算机协会
徐[43]嘉喜、艾军、刘静宇、石涛。ACGDP：增强
基于代码图的软件缺陷预测系统。*电器和电子工程师学会*
可靠性交易，第1-10页，2022页。会议名称：IEEE
交易的可靠性。
[44]雷克斯·英，迪伦·资产阶级，家璇你，马林卡·齐特尼克，和法律上
莱斯科维奇。一个事后解释图表的工具
神经网络。CoRR，abs/1903.03894,2019年。
张[45]吉安、许王、张洪宇、孙海龙、王凯旋、
刘徐东。一种基于该算法的一种新的神经源代码表示方法
抽象语法树。*2019年IEEE/ACM第41届国际会议*
关于软件工程（ICSE），第783-794页，蒙特利尔，QC，加拿大，
2019年5月。电器和电子工程师学会
[46]云会，郑，索拉布·普贾尔，燃烧刘易斯，卢卡·布拉蒂，爱德华
爱泼斯坦、杨波来、拉雷多、亚历山德罗·莫拉里和钟苏。D2a:
一个为基于ai的漏洞检测方法构建的数据集
圆周分析。*在ACM/IEEE第43届国际会议记录上*
软件工程会议：实践中的软件工程，
ICSE-SEIP ‘21，纽约，纽约，美国，2021年。计算协会
机械。
周[47]亚勤、刘上清、萧景开、杜晓宁、刘杨。
偏离：通过全面的学习来有效地识别漏洞
通过图神经网络的程序语义。*神经的进步*
信息处理系统，2019年32：1-11日。

授权许可使用仅限于： NANKAI大学。从2024年6月17日00：35：01从IEEE Xplore下载。限制适用。