**从VulDeePecker出发对AI漏洞挖掘技术讨论**

1. **研究背景**

随着社会数字化发展，软硬件逐渐在我们的生产生活发挥出了极为重要的作用，而软件也成为社会基础建设的一个重要部分。

互联网行业的蓬勃发展，带动了软件数量急速的增长。就Github一个开源软件平台，按照其报告，其上的软件项目数量级就超过了一亿。并且由于项目的开发特点，软件经常会出现大量的版本升级与迭代，这进一步造成了软件数量的膨胀。但是软件数量的增加和技术的进步，并不意味着我们能够对其完全信任。相反的，软件数量膨胀的贡献者更多是开发能力有限的企业、团队甚至是个人，存在着大量的能力限制和开发周期环节缺失，这直接加剧了软件漏洞的产生。

软件漏洞是由软件设计、开发或配置中的错误导致的缺陷实例，可利用该漏洞违反某些明确或隐含的安全策略。软件漏洞的产生源于代码编写人员的编写失误、设计疏漏，或者是代码克隆、第三方库调用造成的，这些都是难以避免的。因此软件项目开发中，漏洞的挖掘与修补以及后期的维护就成了软件项目中很大的一部分。

目前，工业环境下的软件漏洞挖掘大多交由人工实现，以尽可能地提高软件出厂时地完备性。但是这就存在着高成本，低覆盖率地问题。基于人工的漏洞挖掘的效果直接由专家先验知识与经验决定，因此具有较高的人工门槛；由于人工检测的系统性有限，因此没有办法像软件一样实现高覆盖率；由于漏洞类型复杂且多样，完整的漏洞检测需要大量的时间成本和人力成本，去做很多重复劳动。自动化软件测试工具为了解决以上三个问题就应运而生，它虽然具有较高地开发门槛，但却极大地降低了漏洞挖掘地人力成本，并提高了检测地覆盖率，在工业界也有很广泛地运用。

但是现行地动态或静态的自动化漏洞挖掘工具仍然存在着规则依赖导致的泛化性低下的问题，并且很显然无法实现新兴漏洞的检测。为此，研究者们将时下发展迅速且能力出众的人工智能与漏洞挖掘进行结合，希望借此解决传统漏洞挖掘技术的缺陷。

1. **深度学习漏洞挖掘框架概述**

为了能够更好地对深度学习漏洞挖掘进行讨论，本文将从VulDeePecker出发来叙述现在漏洞挖掘的发展现状与后续方向。VulDeePecker是由华中科技大学的Li于2018年在DNSS会议上发表的全球第一个深度学习漏洞挖掘模型，在漏洞挖掘领域是一个基石工作。

如同VulDeePecker的工作流程，目前的基于深度学习的漏洞挖掘技术都遵循着一下的六个步骤：

1. 数据集获取：数据集虽然说是深度学习最基础的工作需求，但是这反而也导致了数据集对学习效果的决定性作用。漏洞的数据集存在着数量有限、分布不均且样本比例极度失衡、漏洞的覆盖性有限等问题，可信且足量的数据集能够提高模型对漏洞特征的学习水平；
2. 代码预处理：代码预处理是一种针对直接来自数据集的代码的过滤技术。通过对代码进行去冗余、剪枝、归一化等预处理行为，降低来自代码编写习惯、项目差异带来的代码差异度；
3. 代码表示：由于程序语言相对于其他文本语言更加复杂，存在着大量的结构信息、语义信息以及流信息。不同的表示形式能够承载不同结构信息与语义信息，通过使用合适的代码表示能够提高所需信息的存储；
4. 表示嵌入：由于深度学习模型能够支持的输入有限，因此需要在最小的信息损失代价下，将代码表示嵌入为向量或图的邻接矩阵等形式；
5. 模型训练：不同的学习模型存在着各自的优势，适用于提取不同信息，借此让模型更好地去获取序列的前后文信息、图的结构信息等；
6. 模型检测：使用训练出来的模型，对数据集中分离出来的测试集进行测试，来证明框架的有效性和能力；
7. **基于VulDeePecker的关键技术讨论**

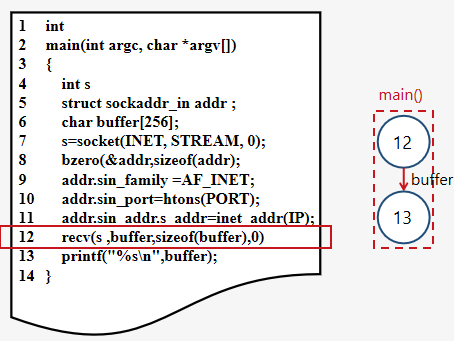
基于2.1中所述的深度学习漏洞挖掘基本方案，本节将从每个步骤对VulDeePecker框架进行详细的描述。其中，由于该漏洞挖掘框架需要使用敏感函数等具体语义信息，因此VulDeePecker会先进性代码表示，然后再进行源代码预处理。

**3.1 代码表示**

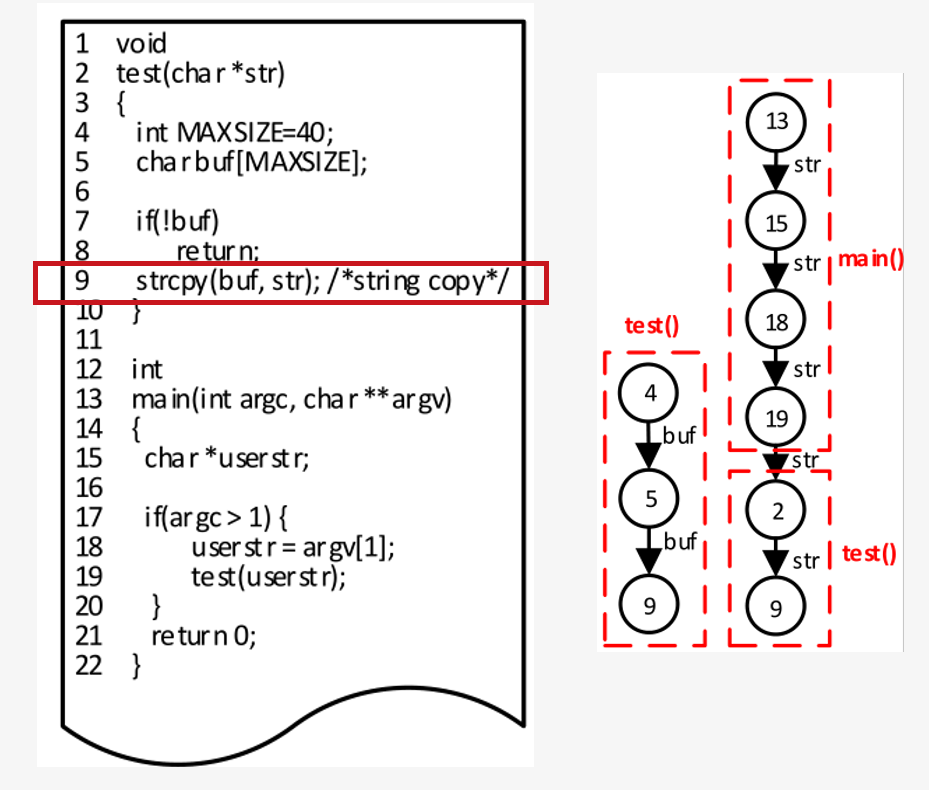
VulDeePecker在代码表示阶段主要做出的贡献是进行了漏洞的精准切片，并将code gadget生成引入了漏洞挖掘框架。众所周知，漏洞在代码中的存在比例一般都是很小的，一段高危代码中仍然是以大面积的中性代码和一个或多个漏洞点所组成的。基于以上的规律分析，如果直接将代码进行表示的嵌入用于模型训练，那么模型显而易见会学习到大量的中性代码特征，而忽略了漏洞点的特征，造成部分过拟合现象，导致大量的假阳性的出现。

VulDeePecker的检测范围比较局限，仅关注API/库调用的漏洞点，并针对缓冲区溢出漏洞和资源管理漏洞。基于以上的两点关注策略，研究者提取了相关API/库调用，作为兴趣点，并根据规则将之分为了前向调用。

前向调用是从直接从外部输入（如命令行、程序、套接字或文件）接收一个或多个输入，而不能从本地获取参数的函数调用。因此它的参数受到函数内语句的影响较小，框架会更加关注受输出参数影响的语句，不当参数值会对敏感点后续的语句功能造成影响。根据以上结论，针对前向调用的每一个参数在调用时和调用后提取各提取一个数据依赖切片。例如，int recv(socket, \*buf, len, flags)就是一个典型的前向调用，如图可以提取其输出参数buffer的切片12->13；



后向调用是不直接从程序运行环境接收任何外部输入，而是从传参、本地参数等作为输入的函数调用。因此它的参数能直接受到函数内语句的影响，从而造成API/库调用间接引发的错误。框架会更加关注影响参数值的语句。根据以上结论，针对后向调用的每一个参数在调用时和调用前各提取一个数据依赖切片。例如，char\* strcpy(buf, str)就是一个典型的后向调用，如图可以提取其输入参数buf的切片4->5->9，以及输入参数str的切片13->15->18->19->2->9；



在VulDeePecker中，代码仍然是以代码行依次排列形成的代码块形式存在的。根据上述我们获得的前向切片和后向切片，我们可以获取code gadget。对同一个key point，我们将其所有的切片按照源代码顺序拼接，并删除重复行。最后得到的部分代码行组成的序列就作为该框架中的代码表示。

3.2 代码预处理

在完成敏感点的提取后，代码中的语义信息不再被需要了。因此在该框架中先进行了代码表示，然后再对代码表示的结果进行预处理。VulDeePecker框架主要应用了冗余代码（注释、非ASCII字符）修剪、函数定义归一化、参数定义的归一化三种策略，来降低代码编写的差异性。其中归一化指的是将同一类token使用“相同的关键词+编号”来替换原有的多样化命名。比如针对所有用户定义的变量，均用“VAR”和序号进行组合替换原有的命名。

同时在这一步中，为了能够更好地帮助最后模型训练，还要为每一个已经获取的code gadget进行标签标记，其中标记遵循不信任原则，只有有直接叙述是无漏洞或存在漏洞点的，才认为是可以使用的数据集。其余的数据集均会被划分在未知集中。

3.3 表示嵌入

对预处理过的code gadget，我们仍需要让它映射为更加扁平的张量形式。

首先我们需要对所有的语句进行拆分，让以语句行为单元的长序列拆转化为以短token为单元的长序列。以strcpy(buf, str)；语句行为例，我们就会将其完全拆分那为[“strcpy”, “(“, “buf”, “,”, “str”, “)”, “;”]共7个小token。很显然token不仅长度不一，并且形式多样，无法直观提取语义信息，因此我们需要进一步对token进行嵌入。

我们针对每一个拆分出token使用word2vec进行词嵌入。Word2vec是一种十分有效的词嵌入手段，甚至能够再嵌入的同时提取出词对应的部分语义信息。它包含了CBOW（continuous bag of words）和Skip-gram两个模型，分别是由上下文预测中间词的模型和由中间词猜测上下文的模型，通过学习，就可以将独热表示的词转化为一个定长的词向量。虽然十分有效，但它仍然存在着无法处理多义词和语义局限于窗口的问题。

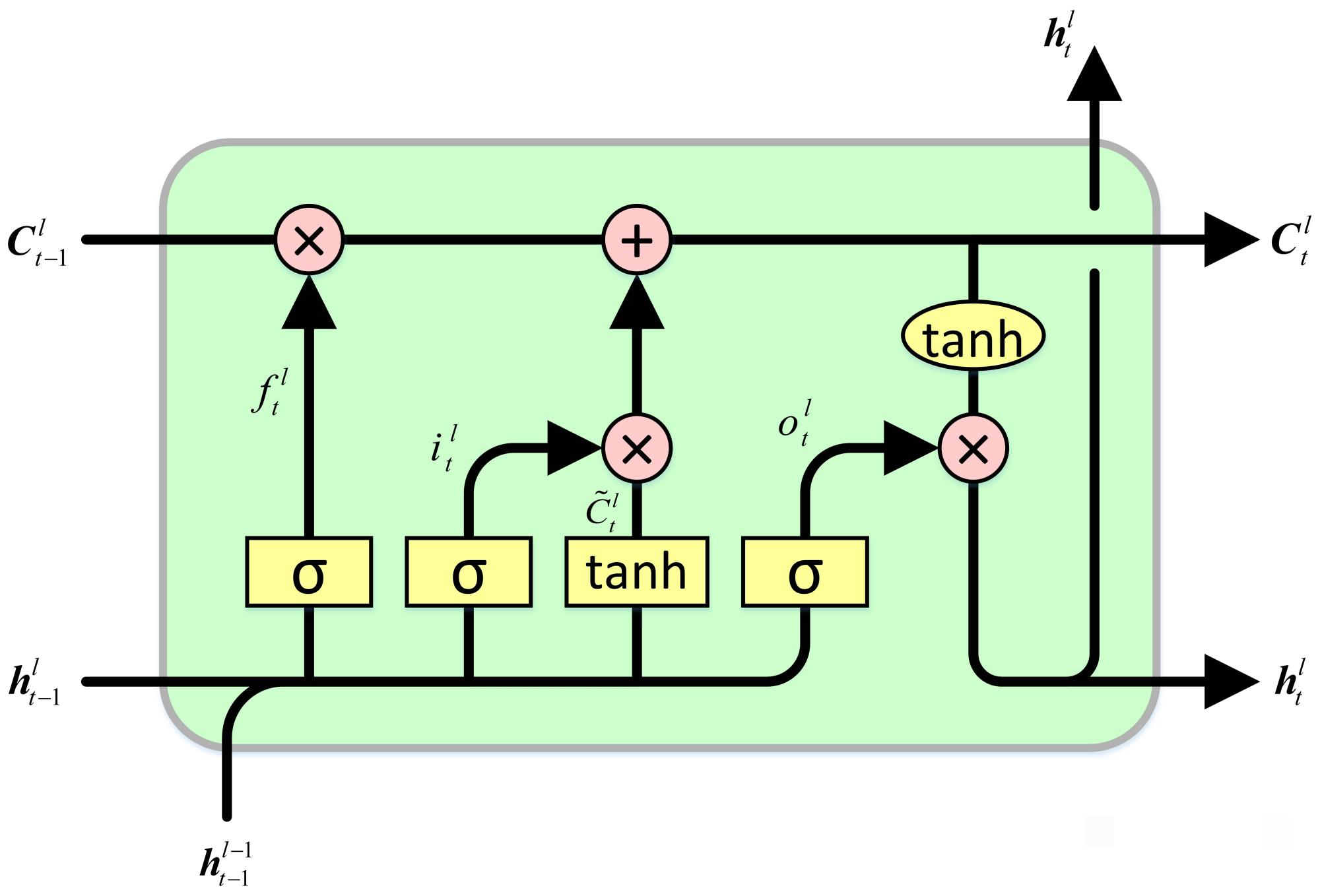
使用word2vec，我们就可以将每一个token都嵌入为一个定长的向量。将所有定长的向量直接按序拼接就可以得到我们最后模型所需的代码表示嵌入。

3.4 模型训练

由3.3节的叙述可知，检测目标的源代码或用于训练的源代码被嵌入成为了一个长数字向量，因此我们学习它特征的模型要善于提取长向量的特征。BLSTM（双向长短期记忆）递归神经网络就是一个很好的选择。

BLSTM是从LSTM发展出来的一种针对常向量的上下文学习模型，我们首先从LSTM开始介绍：

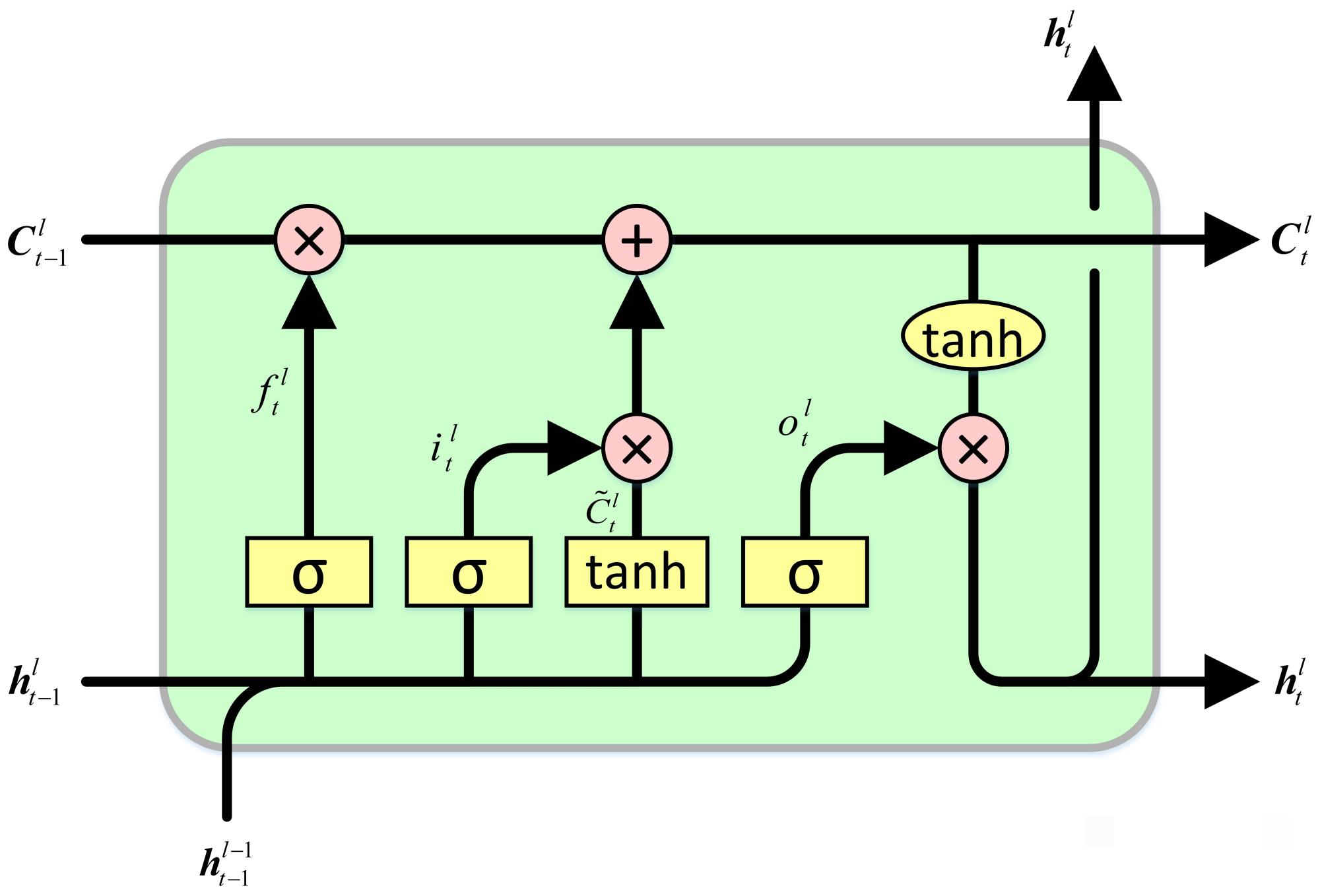
LSTM（长短期记忆）递归神经网络如下图，是一种以cell为单位的链式深度神经网络模型。每一个cell都会接受上一个cell的输出和cell状态共两个参数。根据这两个参数和cell内部的各类参数矩阵，获得输出门O、遗忘门F以及I&C，来选择性地继承上一个cell的特征并提取本cell的特征，从而做到记忆长向量的前向信息。



而我们通过上文的叙述能很容易发现，cell的输出仅仅会收到本节点的输入和上一节点的信息的影响。因此LSTM无法同时学习上下文的信息。为了解决这个问题，研究者想到了一个很直接的解决办法，将一个正向LSTM和反向LSTM直接结合，以此达到同时记忆上下文的目的。这也就是在VulDeePecker中使用的BLSTM。

BLSTM（双向长短期记忆）递归神经网络如下图所示，是两个LSTM正反交错结合在一起的，对同一输入的LSTM的cell结构，其输出通过直接拼接或者softmax函数等方案结合得到一个综合的输出。

最终通过对源代码嵌入结果进行BLSTM进行训练，得到一个漏洞的分类器，可用于识别有无漏洞。

3.5 数据集选择和结果分析

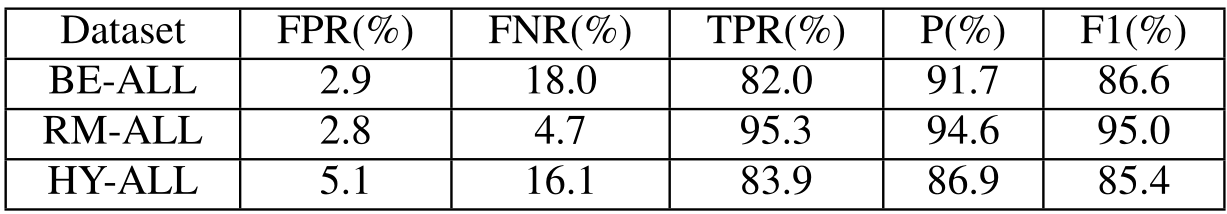
在Li的文章中VulDeePecker主要从NVD（National Vulnerability Database）和SARD（Software Assurance Reference Dataset）两个数据集中获取训练集与测试集。VulDeePecker在NVD中选取了CWE-199缓冲区溢出错误共520个和CWE-399资源管理错误共320个，在SARD中选取了CWE-199缓冲区溢出错误共8122个和CWE-399资源管理错误共1729个。

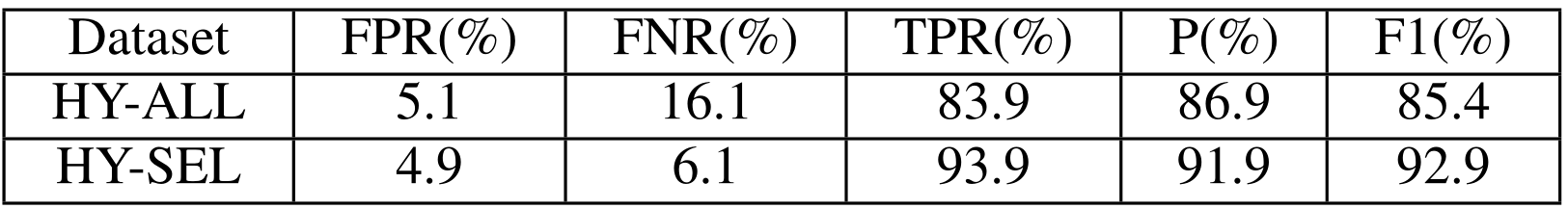
本文主要通过回答以下三个问题，对VulDeePecker进行了评测：

1. VulDeePecker是否能够同时处理多种漏洞；
2. VulDeePecker是否能够通过专家数据集预选择来实现性能的提高；
3. VulDeePecker与现行的漏洞挖掘系统相比性能水平如何；

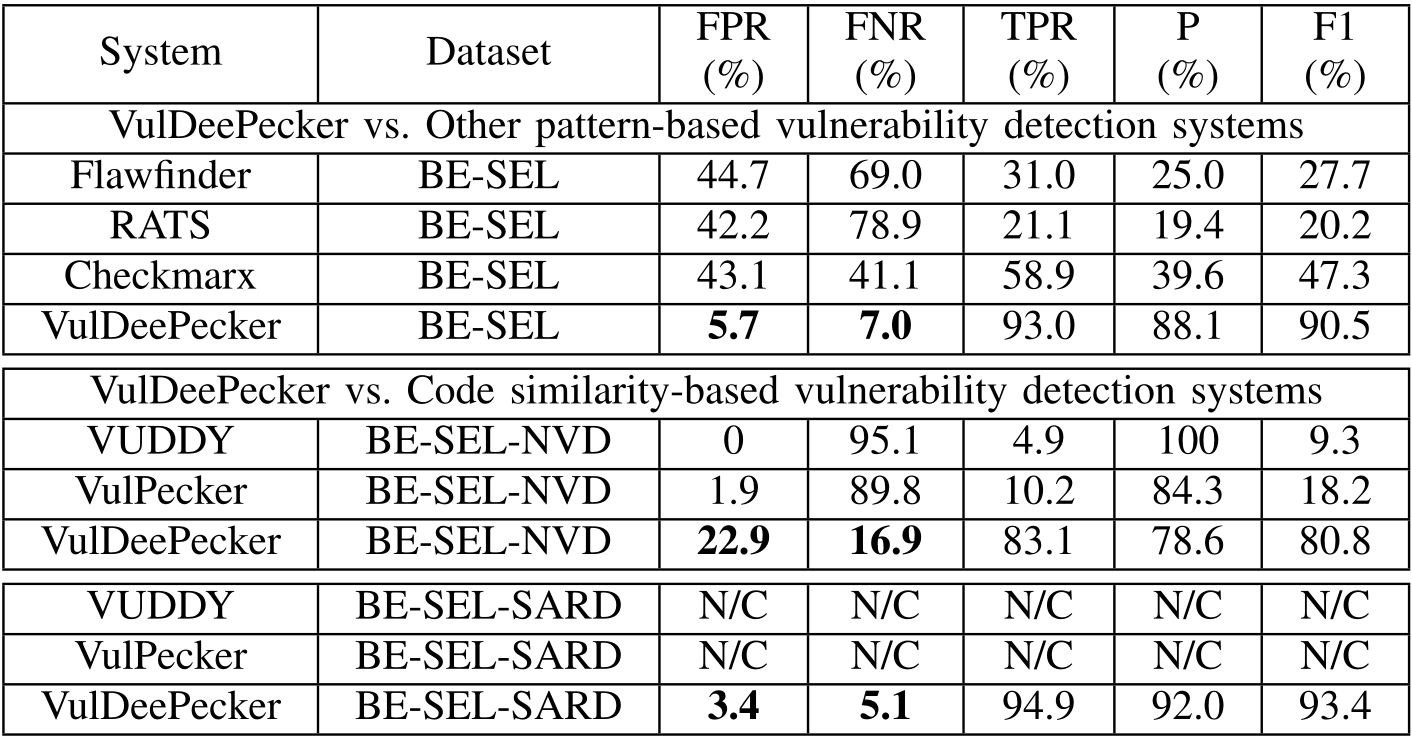
针对第一个问题，研究人员通过同时训练CWE-199类型漏洞和CWE-399类型漏洞，最终获得了如下图的结果，发现虽然有一定的性能削减，但是F1值仍能在混合模式下达到85.4%（相对单独学习缓冲区溢出漏洞仅降低了1.2%），说明该框架能在低漏洞类型数下依旧保持着客观的正确率。

针对第二个问题，研究者通过对单个类型漏洞下、混合模式下的数据集进行了再挑选，选择出了一批更贴近的源代码，来强化模型对漏洞特征的学习。通过最后的检测，发现框架的F1有了近8%的提升。这里展示在混合模式下的ALL数据集训练效果和SEL选择数据集效果的比较，结果如下图所示。





针对第三个问题，研究者选取了有效的传统漏洞挖掘框架再相同数据集上进行检测，通过比较检测结果来证明VulDeePecker的有效性。结果如下图所示，可以看到在一定程度的FPR牺牲下，它换取了极大的FNR降低，获得了极大的F1值提高。



1. **后续工作与研究发展展望**

4.1 后续工作

在VulDeePecker框架出现之后，给予了该领域研究者很多启发，并在此基础上做出了很大的改进。由于预处理和代码嵌入模型比较简单，并且模式已经发展的比较成熟，因此改进的主体集中在了代码表示和模型选择上。

4.1.1 代码表示的改进

首先是代码表示部分，后续工作在切片获取、代码表示形式以及测信道信息获取上做出了很多改进。

在切片获取方面，原本框架存在两个弊病。第一，代码敏感点类型十分局限，并且还要人工的选择，并甄别其前向性和后向性。对此μVulDeePecker框架直接简单考虑了API/库调用，VulDetector则通过分析漏洞修补类diff文件的行差异，自动化获取敏感漏洞点。第二，仅进行前向切片和后向切片会引起许多漏洞的信息获取不全，例如需要联系上下文才能在malloc前后发现缓冲区溢出漏洞是否存在。为此μVulDeePecker和SySeVR等工作都将切片形式转化为了前后向切片结合的模式，以同时学习漏洞点的上下文。

在代码表示形式方面，原框架中是直接应用了代码行组成的代码块。在该形式中，原框架只提取了数据依赖关系，来确定切片方案，导致嵌入的结果具有较严重的扁平化。这造成了大量结构信息和控制信息的损失。为了解决这个问题，SySeVR通过基于CFG和DFG结合生成的PFG提取切片信息，然后再结合原有的代码行输出SeVC；VulDeeLocator则使用代码中间表示IR，实现了跨文件和跨函数的CFG、DFG生成，进一步获取切片。

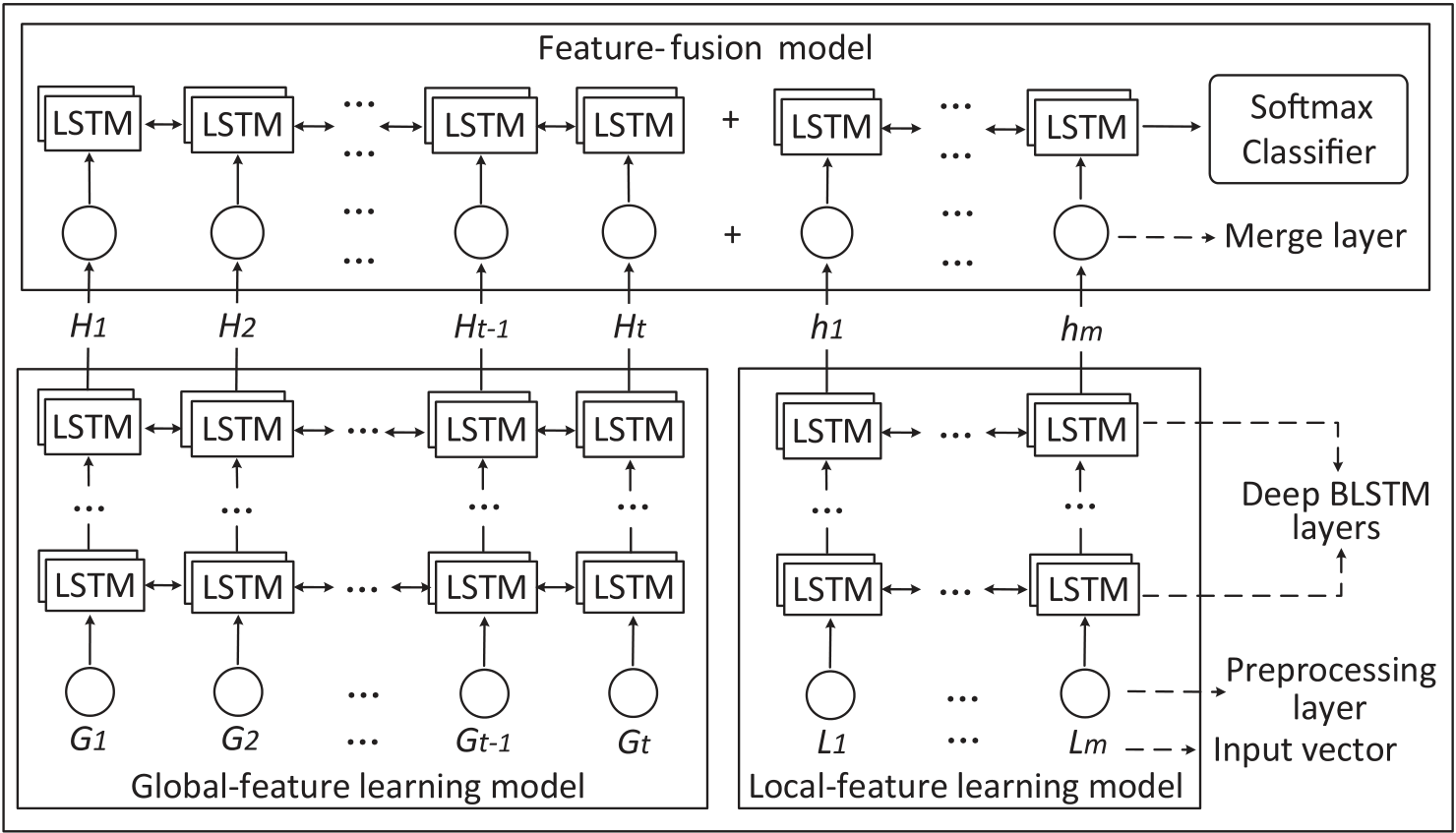
在测信道信息提取方面，原框架是不存在此类功能的，为了让模型能够更加精确地学习到漏洞本身的特征，许多后续工作都通过注意力机制的加入，丰富检测语义信息与结构信息，来提高正确率。在VulDeeLocator框架中，研究者设置了一个基于敏感点和固定上下文窗口的扁平注意力机制，在有漏洞的数据中，在漏洞窗口内代码行的注意力为1，窗口外的代码行的注意力为0。在无漏洞的数据中，所有代码行的注意力均为1。在VulDetector框架中，实现了一个基于敏感点距离的负相关的注意力机制；

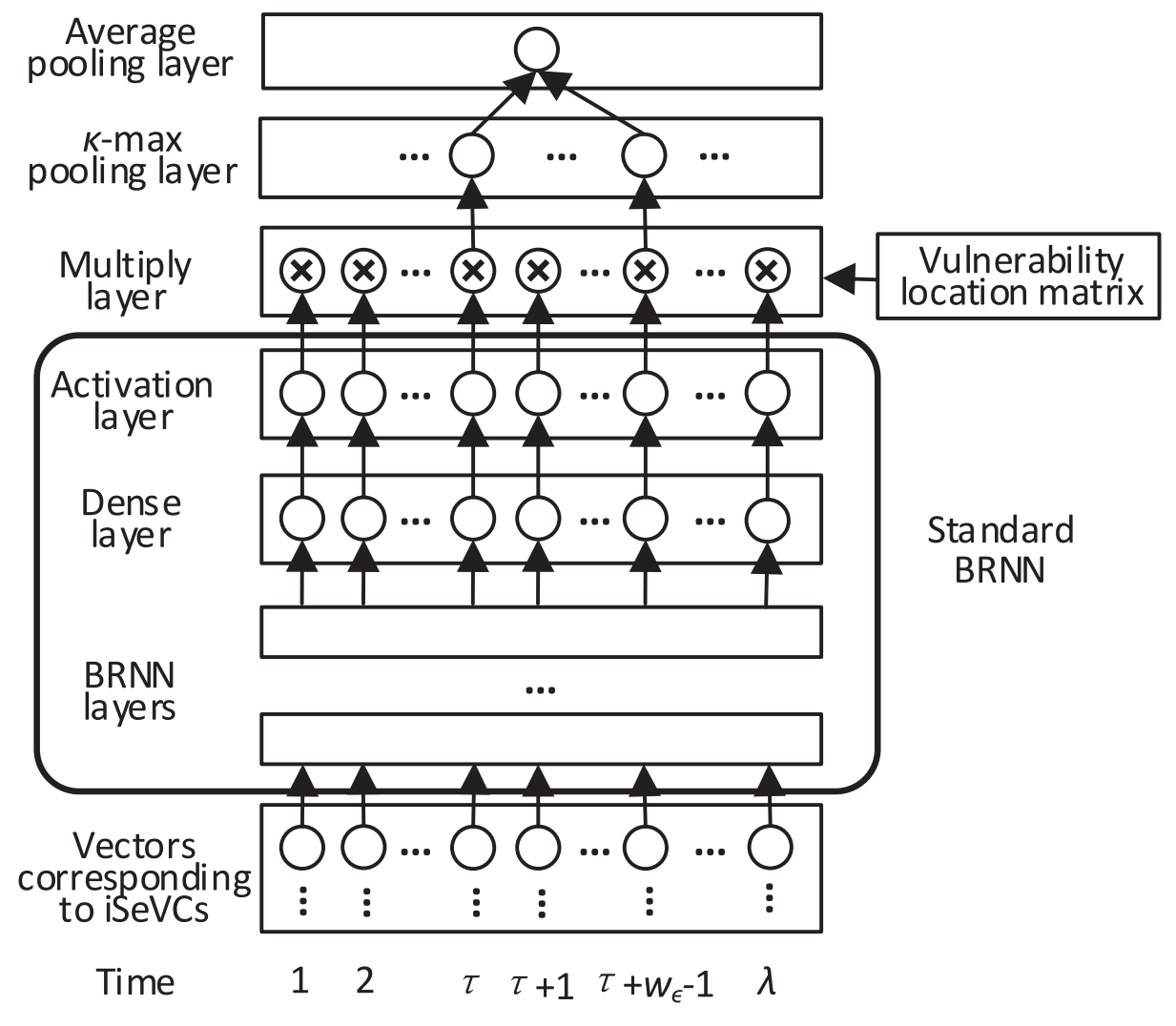
4.1.2 模型选择的改进

模型的选择上，BLSTM已经是一种比较优秀的长向量学习模型了。但为了更好地适配底层的代码表示，并且增加更多模型改进带来的能力附加，研究者们还是在BLSTM的基础上，进行了许多改进和修正。

在μVulDetector框架中，为了适配距离负相关的注意力机制，原有的简单BLSTM被扩充了一个三BLSTM组成的大框架。首先由一个BLSTM学习原本代码的语义特征与结构特征，这些在文章中被称为全局特征，然后由另一个BLSTM实现注意力的特征提取，这被成为本地特征。再由一个BLSTM对拼接的全局特征和本地特征进行学习，通过softmax分类器以达到最终的效果，实现了注意力机制的加入。

在VulDeeLocator中，为了能够让网络能够有更强的能力，研究者对BRNN进行了结构上的添加与改进。该框架主要想通过网络结构的变化，实现粒度的细化、注意力机制的增加以及映射的简便化。由于BRNN本身就拥有一定的映射能力，因此研究者在BRNN的基础上增加了乘法层来实现注意力机制的，通过漏洞位置矩阵来实现扁平注意力机制的融入。并且通过一个最大均值的池化层，让输出粒度减小，相对提高输出精度。





4.2 研究发展展望

未来漏洞挖掘技术的发展依旧十分艰难，但是仍然有许多研究者在不同的方向上努力突破。从第三章的步骤中，我们就可以发现，主要可以改进的流程有数据集获取、代码表示、表示嵌入以及模型选择共四部分。

在代码表示方面，仍有许多改进的余地。

在表示形式上，未来发展方向更加倾向于使用复杂的代码表示来增加代码信息采集率，以提高模型中保留的结构信息、语义信息和控制信息的获取，比如FUNDED中就是用了一个增量化的AST树，来包含更多的语义信息和控制信息。但是模型能够承载的结构复杂度仍然是有限的，目前能接受最复杂的输入也只是图神经网络的矩阵级别的张量，因此还需要考量表示的复杂性和可嵌入性。同时，为了在基本没有可解释性的深度神经网络中，能够更加精确地实现定位，代码表示也在向细粒度的方向发展。

在切片方面，为了能让模型更加专注于漏洞本身的特征学习，切片技术应运而生，以提高学习源数据中漏洞代码占比。目前这一方向现在研究者已经做的很不错了，能够有效地自动化提取敏感点和基于敏感点的双向切片。但是仍然存在切片的依据和切片和范围仍然需要进一步的精进。

在侧信道方面，为了在结构信息、语义信息和控制信息之外能够获取更多信息来提高正确率，许多框架加入了注意力机制。

在模型选择上，模型也在向着新兴化、复杂化发展。在拥有记忆能力、适配嵌入结果这一前提下，模型在努力丰富自己的结构来容纳更多侧信道信息，来提高检测能力。同时通过引入新兴模型，能够将新兴模型的特性迁移到漏洞挖掘中，研究者期盼着通过使用这些新特性能够带来更多的收益。比如能否将元学习模型的任务迁移能力应用到漏洞挖掘领域，让高数据集量的漏洞类训练可以辅助低数据集量的漏洞类检测。

在代码嵌入方面，能否通过更先进的NLP技术实现词嵌入、句嵌入，甚至代码段嵌入，来提高框架的检测能力。本文中提到的VulDeePecker的有效性，其实主要就来源于word2vec这一强大的模型给予代码保留语义信息的定长嵌入的能力。因此能否通过使用更先进的嵌入方式，来提高漏洞挖掘有效性，也是研究的方向之一。

在数据集获取方面为了能提供更有现实价值、更大体量、正反样本比例合理且漏洞覆盖率更大的数据库给模型进行训练，针对漏洞数据集的研究也至关重要。为了提高漏洞的数据集体量，现在的研究方向主要包括以下三种：

1） 基于现实漏洞模式实现具有深度的自动化漏洞植入系统、漏洞代码生成系统的开发，成为了现在十分重要但困难的研究方向。继SARD数据集以后，就很少有一个能够被大家广泛认可的人工生成的漏洞数据集。

2）为了避免人工漏洞生成的简单化、特征重复等问题，研究者开始考虑实现自动化的漏洞收集办法，例如现在正在完善的基于补丁分类器和diff文件的漏洞代码自动提取系统；

3） 在以上方案都不太方便成立时，数据增强也是一种在框架中十分有效的方案。现行的方案有很多，例如SMOTE及其后续工作，但现在仍没有基于漏洞挖掘的特化数据增强技术，如有效的代码变异技术，这也是一个数据集平衡化，复杂化的突破口。