基于跨模态协同表示学习的二进制代码相似性检测方法

杨宏宇[1,2],王云龙[2],胡 泽[1],成 翔[3,4]

(1. 中国民航大学安全科学与工程学院,天津 300300;2. 中国民航大学计算机科学与技术学院,天津 300300;
3. 扬州大学信息工程学院,江苏扬州 225127;4. 中国民航大学民航飞联网重点实验室,天津 300300)

摘 要: 二进制代码相似性检测(Binary Code Similarity Detection,BCSD)技术能够在无源代码的情况下检测二进制文件内在的安全威胁,在软件成分分析、漏洞挖掘等软件供应链安全领域中广泛应用. 针对现有BCSD方法普遍忽略程序实际执行信息和局部语义信息,导致汇编指令语义表示学习效果不佳、特征提取模型的训练资源消耗过大以及相似性检测性能较差等问题,提出一种基于跨模态协同表示学习的二进制代码相似性检测方法(Cross-Modal coordinated Representation Learning for binary code similarity detection,CMRL). 首先,提取汇编指令序列和编程语言片段语义间的对应关系并构建一个对比学习数据集,提出一种面向二进制代码的汇编指令-编程语言协同表示学习方法(Assembly code-Programming language Coordinated representations Learning method,APECL),将源代码的高层次语义作为监督信息,通过对比学习任务使汇编指令编码器APECL-Asm与编程语言编码器生成的特征表示在语义空间中对齐,提升APECL-Asm对汇编指令的语义表示学习效果. 然后,设计一种基于图神经网络的二进制函数嵌入向量生成方法,通过语义结构感知网络对APECL-Asm提取到的语义信息和程序实际执行信息进行融合,生成函数嵌入向量. 最后,通过计算函数嵌入向量之间的余弦距离对二进制代码进行相似性检测. 实验结果表明,与现有方法相比,CMRL对二进制代码相似性检测的Recall@1指标提升8%~33%;针对代码混淆场景下的相似性检测任务,CMRL的Recall@1指标衰减幅度更小,具有更强的抗干扰能力.

关键词: 二进制代码相似性检测;跨模态;协同表示学习;语义结构感知网络;深度神经网络

基金项目: 国家自然科学基金(No.62201576,No.U1833107);江苏省基础研究计划自然科学基金青年基金(No.BK20230558)

中图分类号: TP309.5 **文献标识码:** A **文章编号:** 0372-2112(2025)04-1279-14

电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20240769

Binary Code Similarity Detection Method Based on Cross-Modal Coordinated Representation Learning

YANG Hong-yu[1,2], WANG Yun-long[2], HU Ze[1], CHENG Xiang[3,4]

(1. *School of Safety Science and Engineering, Civil Aviation University of China, Tianjin* 300300, *China*;
2. *School of Computer Science and Technology, Civil Aviation University of China, Tianjin* 300300, *China*;
3. *School of Information Engineering, Yangzhou University, Yangzhou, Jiangsu* 225127, *China*;
4. *Key Laboratory of Civil Aviation Flight Networking, Civil Aviation University of China, Tianjin* 300300, *China*)

Abstract: Existing binary code similarity detection (BCSD) methods often overlook the actual execution information and local semantic details of programs, leading to suboptimal performance in assembly code semantic representation learning, high training resource consumption, and poor similarity detection performance. To address these issues, this paper proposes a cross-modal coordinated representation learning method (CMRL) for binary code similarity detection. First, we extract the semantic correspondence between assembly instruction sequences and programming language fragments to construct a contrastive learning dataset. We then propose an assembly code-programming language coordinated representation learning method (APECL), which uses the high-level semantics of source code as supervisory information. Through contrastive learning tasks, we align the feature representations of the APECL-Asm encoder and the programming language encoder

收稿日期:2024-08-21;修回日期:2025-01-20;责任编辑:孙瑶

in the semantic space, thereby enhancing the semantic representation learning capability of APECL-Asm for assembly instructions. Next, we design a graph neural network-based method for generating binary function embedding vectors. This method uses a semantic structure-aware network to fuse the semantic information extracted by APECL-Asm with the actual execution information of the program, generating function embedding vectors for similarity detection. Experimental results show that compared to existing methods, CMRL improves the Recall@1 metric for binary code similarity detection by 8%~33%. Additionally, in the context of code obfuscation, CMRL exhibits stronger resilience, with less degradation in the Recall@1 metric.

Key words: binary code similarity detection; cross-modal; coordinated representation learning; semantic structure-aware networks; deep neural network

Foundation Item(s): National Natural Science Foundation of China (No.62201576, No.U1833107); Youth Fund Project of Jiangsu Provincial Basic Research Program Natural Science Foundation (No.BK20230558)

1 引言

随着开源项目的增多,软件开发的模块化趋势日益显著,利用第三方组件或源码构建软件的代码重用现象非常普遍. 然而,在代码重用过程中,大量的代码和动态链接库未经过充分的安全审计,导致软件中隐藏着大量漏洞. 存在安全漏洞的组件被复用后,其负面影响的范围也会成倍增加. Synopsys 在 2021 年审计的 2 409 个开源项目中,97% 的项目包含第三方代码,其中的 81% 包含已知漏洞[1]. 近期,XZ 后门漏洞(CVE-2024-3094)影响 Linux/Unix 系统中处理“. xz”和“. lzma”文件的命令行压缩工具 XZ Utils,由于 SSH 底层依赖 liblzma 等组件,该漏洞导致几乎所有的 Linux 系统可能遭受供应链攻击[2]. 所以,如何在软件供应链安全场景下检测二进制文件的内在安全威胁,已成为当前软件安全领域的一个难题.

二进制代码相似性分析方法通过检测两段二进制代码之间的相似程度,识别代码重用和组件复用,在补丁分析、已知漏洞搜索、软件剽窃检测、恶意软件检测、软件供应链分析等任务中得到广泛应用,辅助软件开发方在没有软件供应商参与合作以及缺少源代码的情况下对二进制文件进行安全性检测. 在软件安全领域中,二进制代码相似性分析(Binary Code Similarity Analysis,BCSA)被认为是最有效的解决方法之一,并已成为学术界和工业界的研究热点[3].

基于汇编指令表示学习的 BCSD 方法通过嵌入向量编码二进制代码的语义信息,并使用向量距离衡量函数间的相似性. 例如,Asm2Vec[4]和 INNEREYE[5]基于 Word2Vec 学习汇编指令的指令嵌入表示,PalmTree[6]和 VulHawk[7]使用基于 Transformer 的双向编码器表示技术(Bidirectional Encoder Representations from Transformers,BERT),在大规模的汇编指令上进行无监督训练并使用语言模型对汇编指令进行建模.

近年来,受到对比语言-图像预训练(Contrastive Language-Image Pre-training,CLIP)[8]等多模态表示学习方法的启发,研究人员将二进制代码视为一种模态,探索不同模态语义表示之间的一致性,并在此基础上进行二进制代码表示学习的研究. 例如,Wang 等人[9]提出一种将自然语言解释作为监督信息的二进制代码表示学习方法(Contrastive Language-Assembly Pre-training,CLAP). 尽管现有二进制代码相似性分析方法在相似性检测性能上有所提升,但是仍存在以下局限性:

(1)基于自然语言处理技术(Natural Language Processing,NLP)的汇编指令建模方法通常仅考虑指令的顺序及其之间的关系,忽略了程序实际执行的信息(如控制流和数据流). 这种局限性造成模型难以深入理解二进制代码的实际语义,导致在处理复杂的代码结构时表现不佳. 此外,编译器优化会对代码的结构和指令顺序产生显著影响,在不同的编译优化设置下,相同功能的代码生成不同的汇编指令序列. 由于现有方法难以应对这些变化,导致在不同优化级别下的相似性检测性能不理想.

(2)现有方法将整个二进制函数视作整体,并将其与对应的自然语言描述进行匹配. 这种粗粒度的训练策略忽略了函数的局部细节和具体指令之间的差异,导致模型对语义信息的获取不够细致和准确,在处理复杂的代码结构和逻辑时效果不佳. 此外,当使用不同的语言风格描述相同的代码功能时,自然语言的多样性和模糊性会增加学习的难度. 上述问题导致模型对汇编指令语义特征的学习效率低下,需要依赖大量的训练数据弥补其不足.

为解决以上问题,本文提出一种基于跨模态协同表示学习的二进制代码相似性检测方法(Cross-Modal coordinated Representation Learning for binary code similarity detection,CMRL). 通过汇编指令-编程语言协同表示学习方法提取二进制函数的局部语义特征信息,并结合二进制程序的实际执行信息生成特征融合图,使用语义结构感知网络融合两种特征信息,生成高质量的二进制函数特征嵌入向量用于相似性检测,有效

提升二进制代码相似性检测性能. 本文主要贡献如下:

(1)提出一种面向二进制代码的汇编指令-编程语言协同表示学习方法(Assembly code-Programming language Coordinated representations Learning method, APECL),将源代码的语义作为监督信息,设计对比学习任务,将汇编指令编码器APECL-Asm与编程语言编码器生成的特征表示在语义空间中对齐,通过优化温标交叉熵损失函数(Normalized Temperature-scaled cross entropy Loss, NT-Xent Loss),最大化汇编指令与编程语言正样本对之间的互信息,显著增强APECL-Asm对汇编指令的语义表示学习能力.

(2)构建一个高质量的汇编指令-源代码协同表示学习训练数据集,其中包含130万对汇编指令和与其语义一致的编程语言代码片段. 与自然语言解释相比,作为更加规范的表达方式,源代码能够提供更统一和准确的监督信息,从而降低模型在对比学习过程中所需的训练数据数量,提高模型对汇编代码细粒度语义信息的获取能力,从而有效提升跨模态对比学习过程中的训练效率和语义表示学习效果.

(3)设计一种基于图神经网络的二进制函数嵌入向量生成方法(Graph neural network-based Binary Function Embedding generation Method, GBFEM). 将函数基本块的语义特征与函数控制流图的结构特征相结合生成特征融合图(Semantic-Structural feature fusion Graph, SSG),通过语义结构感知网络生成高质量的二进制函数特征嵌入向量用于相似性检测,显著提升本文方法的二进制相似性检测性能.

2　相关工作

自二进制相似性分析问题于1999年提出以来[10],随着技术发展和研究的不断深入,二进制代码相似性分析问题的解决方法包括基于代码逻辑特征的方法、基于机器学习的方法和基于语言模型的方法.

(1)基于代码逻辑特征的方法. 此类方法依赖领域专家的知识和经验,从二进制代码中提取代码块的指令数量、常量值、基本块数量等统计信息,基本块的控制流图结构、数据依赖关系等逻辑特征,使用上述特征度量二进制代码之间的相似性. discovRE[11]和Bindiff[12]使用统计特征作为度量依据,Genius[13]、Multi-MH[14]使用二分图匹配和最大公共子图同构算法对转换为控制流图的二进制代码进行相似性搜索,COP[15]和BinSim[16]使用符号执行和约束求解器检查二进制代码片段的等价性. 此类方法具有较好的可解释性,但是在大规模二进制程序的相似性分析中难以达到精度和效率平衡.

(2)基于机器学习方法. 此类方法的核心思想是将二进制函数映射到数值向量中,通过向量之间的距离表示不同二进制函数之间的相似性. Genius[13]首先将嵌入向量用于二进制代码相似性分析,之后是VulSeeker[17]、CVSSA[18]和Gemini[19],上述4种方法为函数的属性控制流图(Attributed Control Flow Graph, ACFG)生成图嵌入向量. 随后出现DeepBinDiff[20]和Codee[21]、Asm2Vec[4]、Trex[22]、SAFE[23]等不依赖手动特征选择并使用神经网络自动学习生成指令序列嵌入的方法,此类方法使用自注意力神经网络,能够自动提取二进制代码中用于相似性检测的关键特征;此外,此类方法直接从汇编代码中学习嵌入表示,避免了手动特征提取过程中产生的偏差和遗漏,提升了检测模型的鲁棒性. 由于现有基于机器学习的BCSD方法过度依赖CFG特征,导致检测模型在代码结构变化情况下表现不稳定. 为解决这一问题,Wang等人[24]提出一种基于Delta-CFG的数据增强方案,用于扩充训练数据并降低模型对CFG特征的依赖. 在上述方法中,部分方法[13,17~19]仅使用手动选择的特征表示基本块,忽略了汇编指令的语义信息;部分方法[4,20,21]没有使用完整的控制流信息,忽略了二进制函数的实际执行信息. 这些问题导致上述方法无法准确提取二进制代码的语义信息,在相似性检测中表现不佳.

(3)基于语言模型的方法. 此类方法针对汇编指令的特点设计语言模型和预训练任务进行二进制代码表示学习. 近年来,使用大规模未标记语料库和自监督训练任务的预训练模型在自然语言处理领域十分流行. 研究人员考虑到包括汇编指令在内的编程语言与自然语言的相似性[25],将自然语言处理技术用于二进制代码表示学习. PalmTree[6]在BERT模型的基础上针对汇编指令长距离数据依赖的特点设计三个预训练任务,使模型获取二进制指令的内部格式信息、上下文控制流信息和数据流信息. VulHawk[7]基于RoBERTa模型设计中间表示函数模型(Intermediate Representation Function Model, IRFM)从二进制代码的中间表示上学习语义表示. CLAP[9]基于RoBERTa模型,通过对比学习预训练任务将二进制函数和自然语言解释进行对齐,从而提升模型对二进制代码的表示能力. Xu等人[26]通过评估每条指令对模型分类结果的影响检测出高分类重要性但低语义重要性的指令,提升BCSD模型的泛化性能. jTrans[27]引入跳转感知机制将控制流信息直接嵌入到Transformer模型中,更准确地表示二进制函数的语义特征. OrderMatters[28]引入节点顺序信息的提取机制,提出一个综合性的神经网络模型,将语义、结构和节点顺序信息相结合,全面捕捉二进制函数的特征,提高相似性检测的准确性. 由于几乎所有基于语言模型的方法是在函数级别进行预训练和特征提取却

忽略了函数内部的细粒度语义信息,这种粗粒度的特征提取方法导致模型在处理复杂的代码结构和逻辑时,相似性检测性能不佳.

3 方法框架

本文提出的基于跨模态协同表示学习的二进制相似性检测方法(CMRL)框架如图1所示,该方法由汇编指令-编程语言协同表示学习、二进制函数嵌入向量生成和相似性检测3部分组成.各部分的功能设计如下:

(1)汇编指令-编程语言协同表示学习.构建汇编指令-编程语言数据集,并在此基础上使用编程语言作为监督信息,通过跨模态对比学习训练任务,对齐汇编指令编码器APECL-Asm和编程语言编码器UniXcoder生成的语义特征表示.通过优化NT-Xent损失,最大化汇编指令与编程语言正样本对之间的互信息,从而增强APECL-Asm编码器对汇编指令的特征表示能力.

(2)二进制函数嵌入向量生成.提取二进制函数的控制流图,生成邻接矩阵表示控制流图的结构信息,使用APECL-Asm编码器提取函数基本块的语义信息.通过语义结构感知网络从包含语义信息和结构信息的特征融合图上学习生成二进制函数特征嵌入向量.

(3)相似性检测.通过三元组损失函数优化语义结构感知网络,由语义结构感知网络最小化相似二进制代码在语义空间中的余弦距离、最大化不相似二进制代码在语义空间中的余弦距离,提升其生成的函数嵌入向量质量.通过计算函数嵌入向量之间的余弦距离实现对二进制代码的相似性检测.

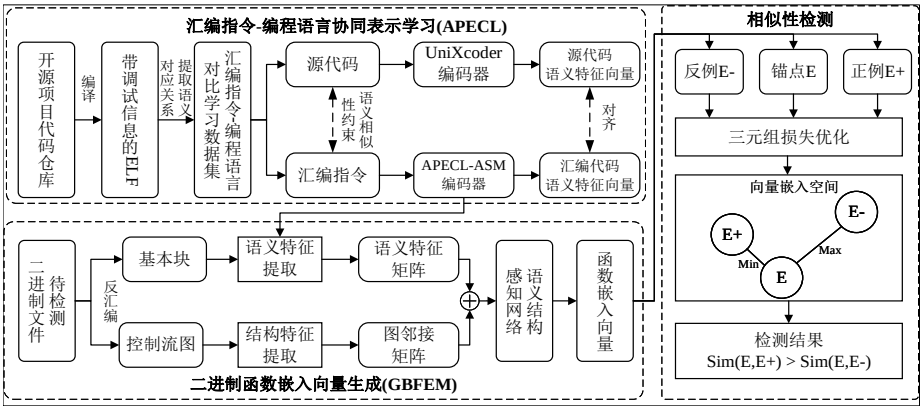


图1 二进制相似性检测方法框架

CMRL的构建流程如下:

首先,从带调试信息的二进制文件中提取汇编指令及其对应的源代码片段,构建跨模态对比学习数据集.在汇编指令-编程语言协同表示学习(APECL)过程中,采用自监督学习和跨模态对比学习,将汇编指令和编程语言特征对齐,增强APECL-Asm编码器对汇编指令的特征表示能力.

然后,在二进制函数嵌入向量生成(GBFEM)阶段,使用控制流图和由APECL-Asm生成的汇编指令特征向量构建特征融合图,并通过图神经网络生成包含语义与结构信息的二进制函数嵌入表示.

最后,在相似性检测阶段,通过三元组损失函数优化二进制函数嵌入向量间的距离,进一步提升模型的相似性检测精度和鲁棒性.

4 汇编指令-编程语言协同表示学习

对于基于汇编指令表示学习的BCSD方法,二进制代码表示学习的效果直接影响相似性检测的性能,Wang等人[9]的研究已经证明多模态对比学习是一种非常有潜力的二进制代码表示学习方法,但训练多模态模型所需的大规模高质量数据集十分欠缺.此外,现有方法在函数级别进行预训练和特征提取,忽略了函数内部的细粒度语义信息,这种粗粒度的特征提取方法导致模型在获取细微语义差异时表现不佳.为解决上述问题,本文提出汇编指令-编程语言协同表示学习方法,该方法由三个部分组成:(1)高质量对比学习数据集的构建;(2)面向汇编指令特点的汇编指令与编程语言编码器设计;(3)基于对比学习数据集的汇编指令语义表示学习.

4.1 汇编指令-编程语言对比学习数据集构建

CMRL的实现依赖于高质量的跨模态对比学习数据集,该数据集包含语义一致的汇编指令和编程语言代码片段.该数据集的设计思路及其构建过程如下:

在进行逆向分析时,通常使用GNU调试器(GNU Debugger,GDB)对ELF文件进行调试.如果ELF文件中包含调试信息,GDB将加载并展示ELF文件的源代码及相应的汇编指令(如图2所示).在本文研究中注意到:在调试过程中,一行源代码的单步执行通常对应着

多条汇编指令的执行,表明源代码与汇编指令在语义上存在对应关系,这种对应关系有助于理解汇编指令的执行过程.

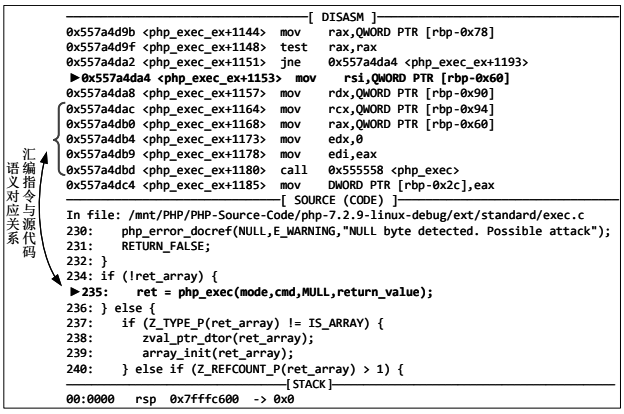


图2　GDB源码调试

在本文的研究中,受到GDB源代码调试过程的启发,通过利用编程语言的高层次语义信息增强编码器对汇编指令的低层次语义表示学习能力,提升汇编指令编码器的语义表示学习效果.

使用源代码作为汇编指令表示学习监督信息的前提条件是需要大量语义一致的编程语言和汇编指令作为训练数据. 为此,在本文研究中从BinKit数据集[29]中挑选4个被广泛使用且维护良好的开源项目:OpenSSL、Coreutils、FFmpeg、Curl,在Ubuntu20.04平台上使用GCC编译器将项目源代码编译为带有调试信息的二进制ELF文件;根据调试信息,使用反汇编器Objdump提取汇编指令和对应的源代码,同时去除语义重要性低、对后续进行语义表示学习造成困难的汇编指令[26].

经过上述处理后,生成包含66 159个二进制函数和1 297 996对汇编指令-C语言代码对照样本的数据集,数据集中部分代码对如图3所示. 图3(a)中的代码实现调用字符串复制函数并保存结果到内存的操作,展示结构体成员赋值的实现. 图3(b)中的代码实现了调用字符串比较函数并检查结果,如果不相等则跳转到指定位置,展示了条件判断的实现. 图3(c)中的代码展示格式化输出字符串的实现. 图3(d)中的代码展示简单for循环结构的实现.

该数据集包含高层次编程语言与低层次汇编指令之间的语义对应关系,展示了汇编指令的具体功能实现. 将此数据集用于跨模态协同表示学习训练,能够增强汇编指令编码器对汇编指令细粒度局部逻辑的理解,从而更有效学习并获取二进制函数的全局语义信息.

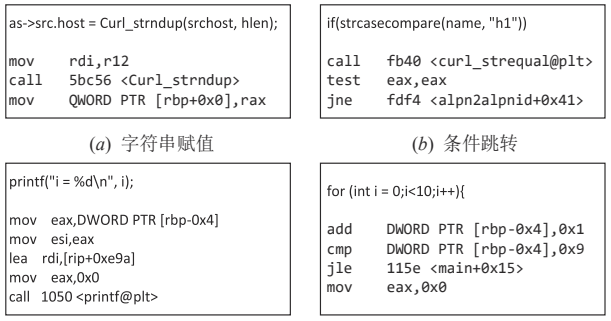


图3　对比学习数据集示例

4.2　汇编指令与编程语言编码器设计

进行汇编指令语义表示学习之前,需设计汇编指令编码器和编程语言编码器,将数据集中的汇编代码和源代码编码为神经网络模型可以处理的数值向量形式.

BERT作为一种自然语言编码器已在NLP领域中广泛应用,在本文研究中选用BERT作为汇编指令编码器的基础架构,构建汇编指令编码器APECL-Asm. APECL-Asm编码器结构如图4所示,该编码器具有8个注意力层、8个注意力头、512个隐藏单元、约1 600万个参数.

源代码的轻微变动经过编译后会造成汇编指令中寄存器和内存地址的大幅改变,给模型获取源代码、汇编指令之间的语义一致性造成困难. 为减小这些差异对汇编指令表示学习带来的影响,在输入APECL-Asm编码器前,需要对汇编指令进行数据清洗处理,本文的解决方法:将直接寻址的内存地址操作数全部表示为<MEM>;将字符串统一规范化为<STR>;将大于4位十六进制的常数替换为<ADDR>,小于等于4位的十六进制常数保留.

APECL-Asm编码器由分词器(Tokenizer)、嵌入层、编码层、输出层四部分组成,各部分设计如下:

(1)分词器设计. BERT模型原有的分词器是针对自然语言特点设计,不能处理汇编指令. 为使汇编指令编码器理解指令的内部结构,学习汇编指令深层次的语义信息,在本文研究中,APECL-Asm的分词器将每条指令视为一个句子,并将其按照操作码和操作数分解为更基本标记(Tokens). 例如,给定一条指令“mov eax,[ebp-0x8]”,将其分为“mov”“eax”“[”“ebp”“-”“0x8”“]”.

(2)嵌入层设计. 经过分词器处理的汇编指令,由APECL-Asm的嵌入层编码为词向量、位置向量和指令向量三个部分. 词向量是对汇编指令分词后的每个标记生成的嵌入表示,指令lea rdi, <STR>会被分成lea, rdi, <STR>三个部分,每个部分都有对应的词向量;位

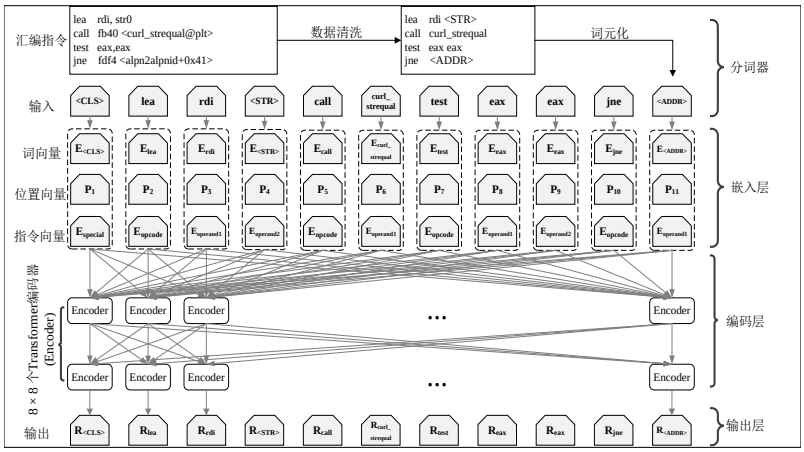


图4 汇编指令编码器结构图

置向量用于表示每个标记在输入序列中的位置，APECL-Asm通过位置向量了解输入序列中各个标记的相对位置，从而获取到序列信息；指令向量用于区分汇编指令的操作码和操作数，标识每个标记的类型，在lea rdi, <STR> 指令中，lea的指令向量编码为$\boldsymbol{E}_{\text{opcode}}$，rdi的编码为$\boldsymbol{E}_{\text{operand1}}$，<STR>的编码为$\boldsymbol{E}_{\text{operand2}}$.

(3)编码层设计. 汇编指令经过分词器和嵌入层的处理，已转换为神经网络模型可以处理的数值向量形式，由编码层学习其语义表示. APECL-Asm的编码层由8个Transformer编码器堆叠而成，每一层有8个自注意力头(Self-Attention Heads).

(4)输出层设计. 汇编指令经过编码层处理后，由APECL-Asm的输出层获取最后一层的隐藏状态，并对其进行平均池化，输出汇编指令的语义表示嵌入向量. 此外，输出层可在每个Token的表示上添加全连接层，以满足后续不同预训练任务的需求.

本文设计的APECL-Asm编码器可在汇编指令序列粒度上学习汇编指令的语义表示，生成语义特征向量. 接下来，为生成源代码的语义特征向量，本文选择现有的研究方法作为编程语言编码器.

UniXcoder[30]是为编程语言设计的跨模态预训练模型，通过对齐代码注释和编程语言之间的嵌入表示，增强模型对代码片段语义特征学习，可有效支持源代码理解与生成任务. 因此，本文使用UniXcode作为编程语言编码器，通过UniXcode处理数据集中的C语言源代码，生成其语义特征向量.

4.3 汇编指令语义表示学习

本文设计的APECL-Asm编码器通过自监督学习和跨模态对比学习实现汇编指令的语义表示学习. 为提高训练效率，本文设计一种双阶段训练方法. 其中：在第一阶段，APECL-Asm在大规模汇编指令上进行自监督学习；在第二阶段，APECL-Asm在编程语言作为监督信息的条件下进行对比学习. 第一阶段的预训练为APECL-Asm初始化一组良好的权重，有利于第二阶段的训练加速收敛，使APECL-Asm更高效学习汇编指令的语义表示.

(1)第一阶段训练

在第一阶段训练过程中，汇编指令编码器APECL-Asm在掩码语言建模(Masked Language Model，MLM)训练任务和中间指令操作码预测(Middle Opcode Prediction，MOP)训练任务上进行联合训练. MLM是一个经典的训练任务，在该任务中，APECL-Asm需要正确预测输入中被掩码的标记，在此过程中学习每个输入标记的嵌入表示，进而理解汇编指令内部的结构和语义信息. MOP是本文设计的训练任务，在该任务中，APECL-Asm需要正确预测中间指令的操作码，进而理解汇编指令序列之间的关系.

第一阶段两个训练任务的设计如下：

(1)MLM训练任务. 设定汇编指令$I=\{t_1,t_2,t_3,\cdots,t_n\}$由一系列标记组成. 对于输入的汇编指令$I$随机选择其中15%的标记进行替换，将掩码处理后的指令表示为I_{masked}. MLM的目标是优化APECL-Asm以正确预测出被掩码的标记. 对于被掩码标记t_i的集合M，损失函数定义如下：

$$\mathcal{L}_{\text{MLM}}=-\sum_{t_i\in M}\log P\left(t_i|I_{\text{masked}}\right) \tag{1}$$

其中，$P\left(t_i|I_{\text{masked}}\right)$是模型在给定掩码处理后的指令$I_{\text{masked}}$下，正确预测出被掩码的标记$t_i$的概率，通过优化上述损失函数，APECL-Asm能够准确预测被掩码的标记，从而更完整地理解汇编指令的内部结构.

(2)MOP训练任务. 从汇编指令中提取连续的三条指令序列(I_1,I_2,I_3)，将I_2的操作码用掩码标记[MASK]替换，设备掩码处理后的中间指令为$I_2{}^{\text{masked}}$. MOP的目标是优化APECL-Asm以正确预测出中间指令I_2的操作码. 对于每个指令序列(I_1,I_2,I_3)损失函数定义如下：

$$\mathcal{L}_{\text{MOP}}=-\log P\left(\text{Opcode}(I_2)\,|\,I_1,I_2^{\text{masked}},I_3\right) \tag{2}$$

其中,$P\left(\text{Opcode}(I_2)\mid I_1, I_2^{\text{masked}}, I_3\right)$是模型在给定前导指令$I_1$、被掩码的中间指令$I_2^{\text{masked}}$和后继指令$I_3$的情况下,正确预测中间指令$I_2$操作码的概率. 通过优化上述损失函数,APECL-Asm能够学习到不同汇编指令之间的潜在关系,例如数据依赖、控制流变化等,从而能够更准确地理解程序的执行流程和逻辑结构.

第一阶段训练的损失函数是上述两个损失函数的和:

$$\mathcal{L}_{\text{Stage1}}=\mathcal{L}_{\text{MLM}}+\mathcal{L}_{\text{MOP}} \tag{3}$$

第一阶段作为预训练过程能够使APECL-Asm学习到汇编指令浅层的特征和模式,减少第二阶段的训练时间和训练数据规模.

在第二阶段训练过程中,APECL-Asm在编程语言监督的对比学习任务上进行训练. 在对比学习训练任务中,APECL-Asm需要将其生成的汇编指令特征表示与UniXcoder生成的编程语言特征表示在语义空间中对齐,在此过程中学习汇编指令的深层次特征,进而增强APECL-Asm对汇编指令的表示能力.

第二阶段对比学习训练任务的具体设计如下:

选择数据集中的一段汇编指令通过APECL-Asm编码器得到向量表示作为锚点,与锚点语义一致的C语言代码作为正例,与锚点语义不一致的C语言代码和汇编指令作为反例,将正例和反例通过相对应的编码器获得向量表示,以正例的向量表示作为监督信息,通过对比学习任务训练APECL-Asm编码器,将锚点与正例的向量表示在语义特征空间中对齐,编程语言监督对比学习训练过程如图5所示.

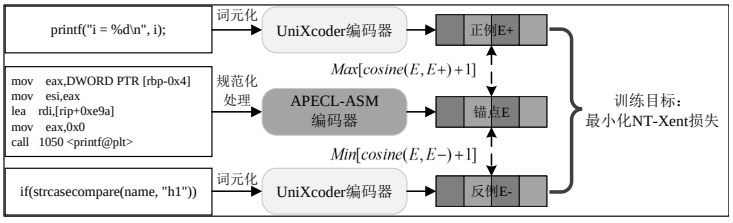


图5　编程语言监督对比学习训练

对比学习训练目标是使得语义一致(锚点与正例)的代码对在嵌入空间中相互接近,而语义不一致(锚点与反例)的代码对远离. 为实现对比学习训练目标,本文设计了归一化温标交叉熵损失函数(NT-Xent Loss),对于输入批量大小为N的汇编指令和C语言代码对,NT-Xent定义为

$$\mathcal{L}_{\text{Stage2}}=-\log\frac{\exp\left(\text{sim}(\boldsymbol{x}_i,\boldsymbol{x}_j)/\tau\right)}{\sum_{k=1}^{2N}1_{[k\neq i]}\exp\left(\text{sim}(\boldsymbol{x}_i,\boldsymbol{x}_k)/\tau\right)} \tag{4}$$

其中,$\boldsymbol{x}_i$和$\boldsymbol{x}_j$分别表示锚点与正例特征向量,$\boldsymbol{x}_k$表示反例的特征向量. $\text{sim}(\boldsymbol{x}_i,\boldsymbol{x}_j)$是$\boldsymbol{x}_i$和$\boldsymbol{x}_j$之间的相似度,使用余弦相似度计算. τ是温度参数,用于控制分布的平滑程度. 在第二阶段对比学习训练过程中,冻结UniXcoder的权重参数以确保源代码特征向量作为监督信息的有效性,通过优化NT-Xent损失更新汇编指令编码器APECL-Asm的参数,增强汇编指令嵌入表示和编程语言嵌入表示之间的互信息,对齐二者在语义特征空间中的表示,从而增强APECL-Asm对汇编指令深层次语义特征的表示学习能力.

此外,较大的批次大小(Batch Size)为每个锚点提供了更多的负样本,有助于提升对比学习的训练效果. 负样本数量越多,锚点与所有反例保持较大距离的可能性就越大,模型越能接近对比学习的实际目标. 因此,将训练过程中的批次大小设置为1 024,这不仅提升了APECL-Asm的表示学习效果,还进一步提高了第二阶段的训练效率.

经过双阶段训练之后,汇编指令编码器APECL-Asm能够为汇编指令生成准确的语义嵌入表示,并将其生成的嵌入向量用于后续处理.

5　二进制函数嵌入向量生成与相似性检测

5.1　二进制函数嵌入向量生成

二进制函数嵌入向量的生成依赖于APECL-Asm编码器生成的汇编指令语义嵌入表示,将该语义表示与二进制代码的控制流信息相结合,从而生成融合语义与结构特征的高质量二进制函数嵌入向量. 在CMRL中,二进制函数嵌入向量生成方法的设计理念和构建过程如下:

二进制函数中的汇编指令并不按照单一的线性顺序执行,而是根据代码逻辑分块跳跃执行,现有方法多数忽略二进制函数实际执行的信息. 为解决上述问题,本文提出一种基于图神经网络的二进制函数嵌入向量生成方法(GBFEM),该方法通过语义结构感知网络对APECL-Asm提取的语义信息和函数实际执行信息进行融合并生成二进制函数嵌入向量,生成过程如图6所示.

二进制函数内存在复杂的控制结构(如循环、条件分支等),二进制函数的控制流图(Control Flow Graph,

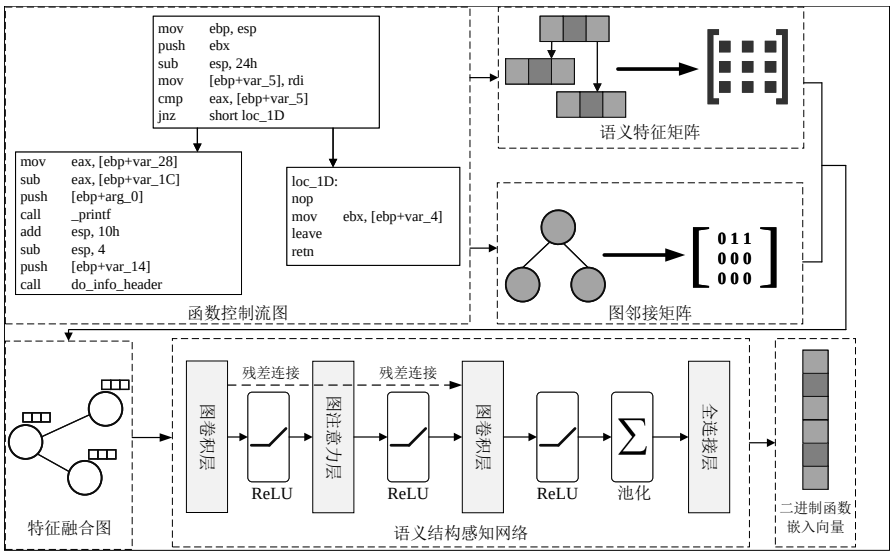


图6 基于图神经网络的二进制函数嵌入向量的生成过程

CFG)可直观地表示出这些结构. 在CFG中,每个节点代表一个二进制函数的基本块,基本块是由一系列汇编指令组成的没有分支和跳转的直线代码段;每条边表示基本块的执行路径和控制依赖关系. 因此,在GBFEM中,采用控制流图表示二进制函数,经过语义-结构特征提取、特征融合图构建、语义-结构特征融合三个过程,生成二进制函数嵌入向量. GBFEM的具体过程设计如下:

(1)语义-结构特征提取. 通过APECL-Asm对CFG中的基本块进行语义特征提取,得到二进制函数的语义信息. 通过生成CFG的图邻接矩阵进行结构特征提取,得到二进制函数的实际执行信息. 对于控制流图$G=(V,E)$,V表示CFG中所有基本块的集合,E表示所有控制流的集合. 每个节点$v\in V$的初始特征向量$\boldsymbol{h}_v$由APECL-Asm编码器生成,CFG所有节点特征向量构成节点特征矩阵$\boldsymbol{X}$. 控制流图的邻接矩阵$\boldsymbol{A}$可以表示为

$$\boldsymbol{A}_{ij}=\begin{cases}1, & \exists e_{(i,j)}\in E\\ 0, & \forall e_{(i,j)}\notin E\end{cases} \tag{5}$$

其中,$\boldsymbol{A}_{ij}$表示控制流图的邻接矩阵$\boldsymbol{A}$的元素,表示节点i和节点j之间的连接关系. $e_{(i,j)}$表示在控制流图中从节点i到节点j的边(控制流).

(2)特征融合图构建. 由过程(1)提取到的函数控制流图的邻接矩阵$\boldsymbol{A}$和特征矩阵$\boldsymbol{X}$,构建出的语义-结构特征融合图(Semantic-Structural feature fusion Graph, SSG)可表示为$G=(\boldsymbol{X},\boldsymbol{A})$.

(3)语义-结构特征融合. 通过语义结构感知网络融合和表达SSG中的语义和结构特征,生成高质量的二进制函数特征向量. 首先,对函数的特征融合图使用图卷积网络进行特征提取,图卷积网络每层的参数更新的计算过程为

$$\boldsymbol{H}^{(l+1)}=\sigma(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}) \tag{6}$$

其中,$\hat{\boldsymbol{A}}$是归一化的邻接矩阵,$\boldsymbol{H}^{(l)}$是第l层的节点特征矩阵,特别的:$\boldsymbol{H}^{(0)}=\boldsymbol{X}$,$\boldsymbol{W}^{(l)}$是第$l$层的权重矩阵,$\sigma$是激活函数. 其次,在图注意力层计算节点之间的注意力系数,并进行语义特征与结构特征的融合,注意力系数的计算过程和特征融合的计算过程为

$$e_{ij}=\mathrm{LeakyReLU}\left(\boldsymbol{a}^{\mathrm{T}}\left[\boldsymbol{W}\boldsymbol{h}_i\|\,\boldsymbol{W}\boldsymbol{h}_j\right]\right) \tag{7}$$

$$\boldsymbol{h}_{i'}=\sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\boldsymbol{W}\boldsymbol{h}_j\right) \tag{8}$$

其中,$\boldsymbol{a}$是注意力向量,$\|$表示连接操作,e_{ij}是注意力系数. 特征融合操作中的a_{ij}为注意力权重,由注意力系数通过SoftMax归一化得到. 然后,使用带残差连接的图卷积层进一步提取特征,对图卷积网络每层的参数进行更新的计算过程为

$$\boldsymbol{H}^{(l+1)}=\sigma(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)})+\boldsymbol{H}^{(l)} \tag{9}$$

最后,通过全局平均池化获得图的整体表示,并通过全连接层输出二进制函数的嵌入表示为

$$\boldsymbol{Z}=\boldsymbol{W}_{\mathrm{out}}\frac{1}{|V|}\sum_{v_i\in V}\boldsymbol{h}_i+\boldsymbol{b}_{\mathrm{out}} \tag{10}$$

其中,通过全连接层得到的向量$\boldsymbol{Z}$即为融合语义和结构特征的二进制函数的嵌入表示.

在GBFEM中,基本块的语义信息和控制流图的结构信息作为语义结构感知网络的输入,与将整个二进制函数直接作为输入相比,基本块具有更小的粒度,这种细粒度的表示可以更细致地体现每个指令序列的局部语义和行为特征,使语义结构感知网络在学习过程中能够更准确地理解和区分二进制函数代码不同部分的语义和功能,从而生成更准确的二进制函数嵌入

向量.

5.2 二进制函数相似性检测

在生成二进制函数嵌入向量后,CMRL通过相似性检测部分对输入的函数嵌入向量进行相似性评估,并通过三元组损失函数优化相似性检测性能.

为提升本文方法对二进制相似性检测的有效性和抗干扰性,对语义结构感知网络进行进一步优化. 为此,选用余弦相似度作为函数相似度的度量指标,目的是对同源二进制函数嵌入向量之间的相似度进行最大化,同时对不同源函数嵌入向量之间的相似度进行最小化. 对语义结构感知网络的优化过程如下:

对于一个函数F,其相似函数$G+$是与F相同源代码编译而来的函数,而函数$G-$则是与F无关的任意函数,包含函数F、$G+$和$G-$的三元组被定义为D. 优化过程中使用的三元组损失函数为

$$\mathcal{L}(F,G+,G-)=\max\left(0,\mathrm{sim}(F,G-)-\mathrm{sim}(F,G+)+\alpha\right) \tag{11}$$

其中,$\mathrm{sim}(\boldsymbol{x},\boldsymbol{y})$表示向量$\boldsymbol{x}$和$\boldsymbol{y}$之间的余弦相似度,$\alpha$是一个超参数,用于控制正负样本之间的距离间隔. 通过优化三元组损失函数,语义结构感知网络能够更准确地生成二进制函数的嵌入向量,提高相似性检测的性能.

之后,使用APECL-Asm和语义结构感知网络进行相似性检测,具体步骤如下:

(1)由APECL-Asm编码器对输入的二进制函数进行特征提取,生成对应的基本块语义向量. 构建控制流图,并计算其邻接矩阵$\boldsymbol{A}$以及节点特征矩阵$\boldsymbol{X}$.

(2)由节点特征矩阵$\boldsymbol{X}$和邻接矩阵$\boldsymbol{A}$构建特征融合图,并输入语义结构感知网络中. 该网络通过图卷积和注意力机制进行特征融合,生成包含语义和结构信息的函数嵌入向量$\boldsymbol{Z}$.

(3)使用余弦相似度计算函数嵌入向量之间的相似度. 对于给定的函数嵌入向量$\boldsymbol{Z}$和$\boldsymbol{Z}'$,其相似度定义为

$$\mathrm{cosine\_similarity}(\boldsymbol{Z},\boldsymbol{Z}')=\frac{\boldsymbol{Z}\cdot\boldsymbol{Z}'}{\|\boldsymbol{Z}\|\times\|\boldsymbol{Z}'\|} \tag{12}$$

其中,$\boldsymbol{Z}\cdot\boldsymbol{Z}'$表示向量的点积,$\|\boldsymbol{Z}\|$和$\|\boldsymbol{Z}'\|$表示向量的范数. 对于两个相似的二进制函数,其嵌入向量之间的余弦距离较小,反之对于两个不相似的二进制函数,其嵌入向量之间的余弦距离则较大. 通过比较两个二进制函数嵌入向量之间的余弦距离,即可实现对二进制函数的相似性检测.

6 实验与结果分析

为了验证本文提出方法的有效性,从以下两个方面进行评估试验:

(1)本文方法在二进制相似性检测中的效果和性能优势;

(2)与同样使用多模态对比学习策略的CLAP[9]方法相比,本文方法在代码混淆场景下二进制相似性检测中的效果和性能优势.

在实验中,使用的计算机配置:Intel(R) Xeon(R) Gold 6248R处理器,256 GB内存,4×NVIDIA Tesla A100 GPU. 模型训练和实验测试环境为Ubuntu20.04,使用IDA Pro 7.5的Python API从二进制文件中提取反汇编指令和控制流图,使用Python3.8和Pytorch2.0实现实验代码.

本文训练所使用的数据集,以及神经网络结构和网络算法的实现代码已在GitHub平台公开,网络地址为:https://github.com/CMRL-paper/CMRL-code.

6.1 二进制相似性检测性能评估

相似性检测实验旨在评估BCSD方法在大规模二进制函数中的相似性检测能力. 在实验中,选用BinaryCorp-3M Test[27]作为二进制相似性检测性能评估实验的数据集. 该数据集包含从364个开源项目的1 908个二进制文件中提取的444 574个函数,能全面评估本文方法在二进制相似性检测中的表现.

在实验中,对于指定函数F,从BinaryCorp-3M Test生成函数池G,G中只有一个函数为F的正例,其余全部为反例. 将函数F作为查询条件,在函数池G中搜索与之最相似的函数. 记录函数F的正例在搜索结果中的排名,使用平均倒数排名(Mean Reciprocal Rank,MRR)和Recall@K(如式(13)和式(14)所示)两个指标评估检测性能.

$$\mathrm{MRR}=\frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\mathrm{Rank}_i} \tag{13}$$

$$\mathrm{Recall@}K=\frac{1}{|Q|}\sum_{i=1}^{|Q|}\mathrm{Hit@}K_i \tag{14}$$

其中,$\mathrm{Hit@}K_i$是一个指示函数,对于第i个查询,如果正确答案在前K个结果中,则$\mathrm{Hit@}K_i=1$,否则$\mathrm{Hit@}K_i=0$.

(1)实验1:相似性检测性能对比实验

在本实验中,选取7个基于汇编指令语义特征表示学习的方法作为对比对象. 其中的6个为已有方法,分别为Trex[22]、OrderMatters[28]、BinAIV[31]、SAFE[23]、Asm2Vec[4]和PalmTree[6]. 另外,为验证本文所提出的跨模态协同表示学习有效性,将APECL-Stage1作为对比方法. APECL-Stage1和CMRL同样由APECL-Asm和GBFEM组成,但是在汇编指令语义表示学习阶段,前者只经过在MLM和MOP任务上进行第一阶段的预训练优化,未经过在编程语言监督的对比学习任务上进行第二阶段的训练优化.

为全面评估上述8种方法在二进制相似性检测中的表现,在本实验中设置3种级别的相似性检测任务,分别为(O0,O3)、(O1,O3)和(O2,O3). 其中,O0、O1、O2、O3代表不同的编译优化设置,以测试方法在不同

编译优化条件下的相似性检测性能. 分别在两种函数池规模(32和1 000)上进行二进制相似性检测性能评估实验,记录每种方法在不同场景下的MRR和Recall@1指标,实验结果如表1和表2所示.

表1 不同方法在二进制相似性检测性能评估实验上的结果对比(函数池规模=32)

方法名	MRR				Recall@1			
	O0,O3	O1,O3	O2,O3	Average	O0,O3	O1,O3	O2,O3	Average
BinAIV	0.776	0.857	0.879	0.837	0.729	0.817	0.853	0.799
Trex	0.594	0.831	0.862	0.762	0.497	0.793	0.845	0.712
OrderMatters	0.574	0.826	0.874	0.758	0.466	0.785	0.852	0.701
PalmTree	0.525	0.808	0.888	0.740	0.413	0.763	0.854	0.676
SAFE	0.446	0.684	0.835	0.655	0.294	0.602	0.808	0.568
Asm2Vec	0.387	0.597	0.798	0.594	0.253	0.489	0.751	0.498
APECL-Stage1	0.479	0.788	0.871	0.713	0.351	0.738	0.852	0.647
CMRL	0.816	0.896	0.911	0.874	0.753	0.843	0.875	0.823

表2 不同方法在二进制相似性检测性能评估实验上的结果对比(函数池规模=1 000)

方法名	MRR				Recall@1			
	O0,O3	O1,O3	O2,O3	Average	O0,O3	O1,O3	O2,O3	Average
BinAIV	0.622	0.718	0.747	0.696	0.535	0.632	0.687	0.616
Trex	0.454	0.706	0.749	0.636	0.357	0.618	0.694	0.556
OrderMatters	0.386	0.717	0.748	0.617	0.287	0.620	0.695	0.534
PalmTree	0.318	0.721	0.738	0.583	0.212	0.618	0.688	0.503
SAFE	0.252	0.517	0.689	0.486	0.143	0.423	0.637	0.401
Asm2Vec	0.262	0.374	0.657	0.431	0.174	0.286	0.607	0.354
APECL-Stage1	0.255	0.651	0.719	0.542	0.157	0.565	0.667	0.463
CMRL	0.691	0.796	0.807	0.763	0.592	0.703	0.754	0.682

由表1和表2可知,与现有基于汇编指令语义特征表示学习的检测方法相比,CMRL在所有场景下的MRR和Recall@1指标均有明显提升,其中MRR和Recall@1的最大提升幅度分别为32.8%和33.2%.

与未经过对比学习任务训练优化的APECL-Stage1相比,CMRL在Recall@1指标上平均提升幅度为20%. CMRL通过编程语言监督的对比学习训练及优化,能够更有效地提取汇编指令的深层次语义信息,进而能够生成更准确的二进制函数语义特征向量,有效提升二进制相似性检测性能. 而APECL-Stage1经过第一阶段的预训练优化,仅学习到汇编指令浅层的特征和模式,在提取二进制代码的语义特征时效果不佳.

相比之下,APECL-Stage1和PalmTree之间的性能差距相对较小. 由于PalmTree在大规模汇编指令上进行无监督训练时采用了定义-调用预测(Def-Use Prediction,DUP)任务,使得PalmTree能够捕捉到一定的控制流和数据依赖信息,进而在相似性检测任务上的表现优于APECL-Stage1. 然而,由于PalmTree未使用二进制函数完整的控制流信息,仅依靠在汇编指令序列上进行无监督训练来学习二进制代码语义表示,导致其生成的二进制函数嵌入向量质量不佳,相似性检测性能与CMRL相比仍存在较大差距.

BinAIV通过综合函数嵌入、函数名称和子函数嵌入等多种特征来表示二进制代码,但在二进制函数嵌入向量生成过程中,BinAIV未能充分考虑函数的控制流信息以及实际执行信息. 而CMRL通过图神经网络融合了二进制函数的语义和控制流信息,使得CMRL能够更全面地捕捉函数的细粒度语义特征和执行依赖. 因此与BinAIV相比,CMRL在相似性检测中表现更为优异.

OrderMatters在词元化过程中将每个基本块视为“句子”,其中的指令视为“单词”. 在自然语言中,句子中的单词之间有明确的语法和语义关联,而汇编指令之间的关联取决于控制流和数据流. OrderMatters单纯将指令作为独立的“单词”,无法捕捉指令间复杂的关联性,导致语义表达能力不足. 在本文研究中,CMRL的分词器将每条指令视为一个句子,并将其按照操作码和操作数分解为更基本的标记,使得编码器能够理解指令的内部结构,从而学习到汇编指令深层次的语义信息. 因此,CMRL的相似性检测性能显著高于OrderMatters.

通过进一步对比可发现,Trex在相似性检测性能上与CMRL接近,Trex通过微指令模拟执行策略,使其在模型训练中纳入了完整的控制流信息,从而提升了

其在二进制代码语义特征提取方面的表现. 这一结果表明,控制流信息对于二进制代码表示学习至关重要. CMRL在使用完整控制流信息的基础上,构建了语义-结构特征融合图(SSG)这一全新的二进制函数表示方法,进而通过图神经网络获得了更加优质的二进制函数嵌入向量,因此CMRL在Recall@1和MRR指标上的表现优于Trex.

上述实验结果表明,通过编程语言监督的对比学习训练及优化,以及图神经网络对二进制代码语义特征和执行信息的融合,CMRL能够更准确地提取二进制函数的语义信息,有效提升二进制相似性检测性能.

(2)实验2:模拟真实场景下的相似性检测性能评估实验

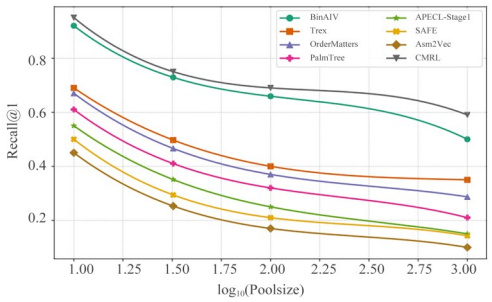
为评估实验1中8种方法在不同规模函数池和不同编译优化条件下的泛化能力,在本实验中设置4种函数池规模:10、32、100和1 000,同时设置3种编译优化级别:(O0,O3)、(O1,O3)和(O2,O3). 通过上述设置模拟真实世界的相似性检测场景并在此场景下进行多次相似性检测性能评估实验,实验结果如图7所示.

由图7可知,CMRL在所有函数池规模和编译优化条件下始终表现最佳,Recall@1的曲线高于其他方法,尤其在函数池规模较大时(PoolSize=1 000),其性能平均下降幅度为14.1%,低于其他方法,表现出良好的泛化能力.

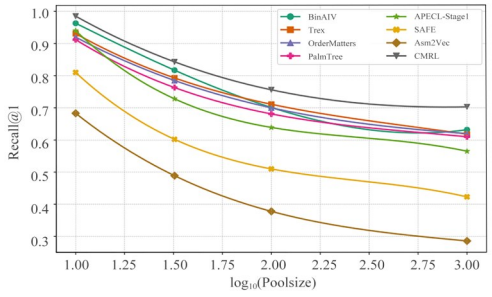
在较大规模函数池(PoolSize=1 000)情况下,BinAIV的性能相对稳定. 说明其多层次特征融合策略在复杂场景下能够较好地发挥作用. 由于对控制流特征的深度建模,Trex在所有场景下性能表现稳定,但整体略逊于CMRL. 在中小规模函数池(PoolSize=10, 32, 100)中,OrderMatters的表现优于SAFE,但在函数池规模达到1 000时,OrderMatters的性能下降幅度较大. 表明OrderMatters对代码控制流顺序的建模虽然有效,但在处理复杂优化条件和大规模场景时仍存在不足.

在小规模函数池(PoolSize=10, 32)中,SAFE和Asm2Vec表现尚可,当函数池规模从10增加到1 000时,Asm2Vec和SAFE的Recall@1指标迅速下降. 此现象表明,在大规模函数池中,SAFE和Asm2Vec生成的嵌入表示难以体现不同二进制函数间的差异,导致二者的相似性检测性能下降. 两种方法的共同问题在于训练数据集规模较小,导致其对大规模场景的特征学习不够充分,无法有效区分大量相似的嵌入特征,检测模型的泛化能力不足.

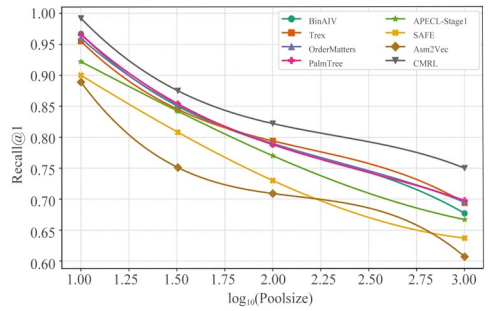
上述实验结果表明,CMRL具有较强的泛化能力,故在模拟真实场景的大规模函数池下仍能保持较高的检测性能.



(*a*) (O0, O3)优化级别检测结果



(*b*) (O1, O3)优化级别检测结果



(*c*) (O2, O3)优化级别检测结果

图7　不同方法在模拟真实场景下的相似性检测结果

6.2　混淆场景下的相似性检测性能评估

为了分析CMRL和CLAP的语义表示学习能力和相似性检测性能,开展混淆场景下的相似性检测实验、模型参数规模和推理效率对比实验.

为比较CMRL和CLAP对复杂二进制代码语义的学习能力,开展代码混淆场景下的相似性检测实验. 在实验中,选取Coreutils作为实验数据集,使用Obfuscator-LLVM的指令替换功能编译生成混淆后的二进制文件. 使用CMRL和CLAP分别在混淆前后的二进制文件上进行相似性检测实验,记录两种方法在混淆前后的Recall@1指标,实验结果如表3所示.

由表3可知,在无混淆场景下的二进制代码相似性检测中,虽然CMRL的Recall@1指标低于CLAP,但是CMRL在混淆场景下的性能衰减为18%,明显小于CLAP的29%,说明在面对代码混淆时,CMRL具有更强

表3 混淆场景下的二进制代码相似性检测实验结果

方法名	混淆设置	O0,O3	O1,O3	O2,O3	Average	性能衰减
CMRL	无混淆	0.75	0.84	0.87	0.82	**0.18**
	指令替换	0.58	0.65	0.69	0.64	
CLAP	无混淆	0.90	0.95	0.98	0.94	0.29
	指令替换	0.59	0.67	0.68	0.64	

的抗干扰能力.

CMRL和CLAP的参数规模和推理效率对比实验结果如表4所示. 由表4可见,CLAP的训练集是包含了1.95亿对汇编指令和自然语言的数据集,在训练资源消耗和训练数据量方面,CMRL远远小于CLAP.

表4 参数规模和推理效率对比

方法名	推理时间/s	训练数据/M	参数数量/M
CLAP	0.012 90	195.0	110.25
CMRL	0.001 95	1.3	16.04

由表3和表4可知,CMRL的参数规模仅为CLAP的15%,在训练所消耗的资源远小于CLAP的条件下就能达到CLAP模型80%的检测性能,CMRL在混淆场景下的二进制相似性检测性能更具优势. 其原因分析如下:

(1)自然语言解释具有多样性和模糊性,同一个代码片段可以用多种不同的方式进行描述,导致CLAP模型在训练过程中难以获取到准确的语义信息,CLAP模型需要大量的训练数据来覆盖各种不同的描述方式. 但是,源代码本身是严格规范的,其可以提供更加一致和准确的语义监督信号,从而减少CMRL模型学习过程中的不确定性. 所以,CMRL能够在更小的训练数据上使用更少的模型参数规模,就能达到与CLAP方法相近的二进制代码相似性检测性能.

(2)CLAP在训练过程中使用自然语言解释对整个二进制函数进行整体描述,这种粗粒度的训练策略忽略了函数内部的局部细节和具体指令之间的差异,限制了CLAP模型对函数内部语义信息的获取能力. 而源代码监督方法可以在汇编指令序列的粒度上进行语义表示学习,能够充分利用细粒度的语义信息,显著提升CMRL模型对代码细节的理解和表示能力.

因此,CMRL在面对编译器优化和代码混淆等挑战时具有更强的抗干扰能力.

7 结束语

针对现有的二进制相似性检测方法忽略程序实际执行信息和局部语义信息,导致语义特征提取效果不佳、模型训练效率低下的问题,本文提出一种基于跨模态协同表示学习的二进制代码相似性检测方法CMRL. 从带有调试信息的二进制ELF文件中提取语义一致的汇编指令序列和编程语言代码片段,构建一个高质量的对比学习数据集. 针对汇编指令特点并基于BERT设计一个汇编指令编码器APECL-Asm,使用编程语言作为监督信息设计跨模态对比学习预训练任务,通过优化温标交叉熵损失函数在汇编指令序列上进行语义表示学习. 设计一种基于图神经网络的二进制函数语义嵌入向量生成方法,从包含二进制函数的语义信息和程序实际执行信息的特征融合图上学习生成高质量的二进制函数特征嵌入向量. 通过计算函数嵌入向量之间的余弦距离对二进制代码进行相似性检测. 在BinaryCorp-3M Test数据集上的实验结果表明,本文方法在相似性检测性能方面优于现有方法. 在代码混淆场景下的二进制相似性检测中,与经典CLAP方法相比,本文方法在模型参数规模更小、训练资源消耗更低的条件下具有更强的抗干扰能力.

在未来研究中,为了进一步提升在不同场景下的二进制相似性检测性能,提高模型的辨别能力和泛化性能,将继续优化模型的结构和训练方法,设计更具针对性的损失函数;为了提高模型在对比学习训练任务中的稳定性,将探索更加智能的正反例选择策略和采样方法.

参考文献

[1] Synopsys. Open source security and risk analysis report[EB/OL]. (2022-10-01)[2024-8-21]. https://www.synop-sys.com/content/dam/synopsys/sigassets/reports/rep-ossra-2022.pdf.

[2] NIST. CVE-2024-3094 vulnerability details[EB/OL]. (2024-06-21) [2024-08-21]. https://nvd. nist. gov/vuln/detail/CVE-2024-3094.

[3] 于颖超, 甘水滔, 邱俊洋, 等. 二进制代码相似度分析及在嵌入式设备固件漏洞搜索中的应用[J]. 软件学报, 2022, 33(11): 4137-4172.
YU Y C, GAN S T, QIU J Y, et al. Binary code similarity analysis and its applications on embedded device firmware vulnerability search[J]. Journal of Software, 2022, 33(11): 4137-4172. (in Chinese)

[4] DING S H H, FUNG B C M, CHARLAND P. Asm2Vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization[C]//2019 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE, 2019: 472-489.

[5] ZUO F, LI X P, YOUNG P, et al. Neural machine translation inspired binary code similarity comparison beyond function pairs[C]//Proceedings of the 26th Network and Distributed System Security Symposium (NDSS). San Diego: Internet Society, 2019: 1-13.

[6] LI X, QU Y, YIN H, et al. PalmTree: Learning an assem-

bly language model for instruction embedding[C]//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2021: 3236-3251.

[7] LUO Z H, WANG P F, WANG B S, et al. VulHawk: Cross-architecture vulnerability detection with entropy-based binary code search[C]//Proceedings of the Network and Distributed System Security Symposium (NDSS). San Diego, CA: Internet Society, 2023: 14-25.

[8] RADFORD A, KIM J W, HALLACY C, et al. Learning transferable visual models from natural language supervision[C]//Proceedings of the 38th International Conference on Machine Learning. New York: ACM, 2021: 8748-8763.

[9] WANG H, GAO Z Y, ZHANG C, et al. CLAP: Learning transferable binary code representations with natural language supervision[C]//Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2024: 503-515.

[10] BAKER B S, MANBER U, MUTH R. Compressing differences of executable code[C]//ACMSIGPLAN Workshop on Compiler Support for System Software (WCSS). Princeton: Citeseer, 1999: 1-10.

[11] ESCHWEILER S, YAKDAN K, GERHARDS-PADILLA E. DiscovRE: Efficient cross-architecture identification of bugs in binary code[C]//Proceedings of the Network and Distributed System Security Symposium. San Diego: Internet Society, 2016: 137-172.

[12] ZYNAMICS. BinDiff home[EB/OL]. (2023-08-15)[2024-08-21]. https://www.zynamics.com/bindiff.html.

[13] FENG Q, ZHOU R D, XU C C, et al. Scalable graph-based bug search for firmware images[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2016: 480-491.

[14] PEWNY J, GARMANY B, GAWLIK R, et al. Cross-architecture bug search in binary executables[C]//Proceedings of the IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2015: 709-724.

[15] LUO L N, MING J, WU D H, et al. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection[J]. IEEE Transactions on Software Engineering, 2017, 43(12): 1157-1177.

[16] MING J, XU D P, JIANG Y F, et al. BinSim: Trace-based semantic binary diffing via system call sliced segment equivalence checking[C]//Proceedings of the 26th USENIX Conference on Security Symposium. Vancouver: USENIX Association, 2017: 253-270.

[17] GAO J, YANG X, FU Y, et al. VulSeeker: A semantic learning based vulnerability seeker for cross-platform binary[C]//2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE, 2018: 896-899.

[18] LIN H, ZHAO D D, RAN L J, et al. CVSSA: Cross-architecture vulnerability search in firmware based on support vector machine and attributed control flow graph[C]//Proceedings of the 2017 International Conference on Dependable Systems and Their Applications (DSA). Piscataway: IEEE, 2017: 35-41.

[19] XU X J, LIU C, FENG Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2017: 363-376.

[20] DUAN Y, LI X, WANG J H, et al. Deepbindiff: Learning program-wide code representations for binary diffing. 2020[C]//Proceedings of the Network and Distributed System Security Symposium. New York: ACM, 2020: 23-26.

[21] YANG J, FU C, LIU X Y, et al. Codee: A tensor embedding scheme for binary code search[J]. IEEE Transactions on Software Engineering, 2021, 48(7): 2224-2244.

[22] PEI K X, XUAN Z, YANG J F, et al. Trex: Learning execution semantics from micro-traces for binary similarity[EB/OL]. (2020-10-01)[2024-08-21]. https://arxiv.org/abs/2012.08680v3.

[23] MASSARELLI L, DI LUNA G A, PETRONI F, et al. SAFE: Self-attentive function embeddings for binary similarity[M]//Detection of Intrusions and Malware, and Vulnerability Assessment. Cham: Springer International Publishing, 2019: 309-329.

[24] WANG J L, ZHANG C, CHEN L F, et al. Improving ML-based binary function similarity detection by assessing and deprioritizing control flow graph features[C]//Proceedings of the 33rd USENIX Conference on Security Symposium. New York: ACM, 2024: 4265-4282.

[25] ALLAMANIS M, BARR E T, DEVANBU P, et al. A survey of machine learning for big code and naturalness[J]. ACM Computing Surveys (CSUR), 2018, 51(4): 1-37.

[26] XU X Z, FENG S W, YE Y P, et al. Improving binary code similarity transformer models by semantics-driven instruction deemphasis[C]//Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2023: 1106-1118.

[27] WANG H, QU W J, KATZ G, et al. Jtrans: Jump-aware

transformer for binary code similarity detection[C]//Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2022: 1-13.

[28] YU Z P, CAO R, TANG Q Y, et al. Order matters: Semantic-aware neural networks for binary code similarity detection[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2020, 34(1): 1145-1152.

[29] KIM D, KIM E, CHA S K, et al. Revisiting binary code similarity analysis using interpretable feature engineering and lessons learned[J]. IEEE Transactions on Software Engineering, 2023, 49(4): 1661-1682.

[30] GUO D Y, LU S, DUAN N, et al. UniXcoder: Unified cross-modal pre-training for code representation[C]//Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Stroudsburg: USAACL, 2022: 7212-7225.

[31] GU Y M, SHU H, KANG F. BinAIV: Semantic-enhanced vulnerability detection for Linux x86 binaries[J]. Computers & Security, 2023, 135: 103508.

作者简介

杨宏宇 男，1969 年 12 月出生，吉林长春人. 博士，中国民航大学教授、博士生导师. 主要研究方向为网络与系统安全、漏洞分析与评估、云计算与大数据安全.
E-mail: yhyxlx@hotmail.com

胡 泽 男，1989 年 7 月出生，山西临汾人. 博士，中国民航大学讲师. 主要研究方向为自然语言处理、人工智能、信息安全.
E-mail: zhu@cauc.edu.cn

王云龙 男，1998 年 12 月出生，河北沧州人. 中国民航大学硕士研究生. 主要研究方向为网络信息安全、软件供应链安全、数据安全.
E-mail: luckyfuture0177@163.com

成 翔 男，1988 年 9 月出生，新疆乌鲁木齐人. 博士，扬州大学实验师. 主要研究方向为网络与系统安全、网络安全态势感知、联邦学习、边缘计算.
E-mail: huozhai9527@126.com