计算机科学
COMPUTER SCIENCE

基于Jump-SBERT的二进制代码相似性检测技术研究

严尹彤，于璐，王泰彦，李宇薇，潘祖烈

引用本文

严尹彤，于璐，王泰彦，李宇薇，潘祖烈．基于Jump-SBERT的二进制代码相似性检测技术研究[J]．计算机科学，2024，51(5)：355-362.

YAN Yintong, YU Lu, WANG Taiyan, LI Yuwei, PAN Zulie. Study on Binary Code Similarity Detection Based on Jump-SBERT [J]. Computer Science, 2024, 51(5): 355-362.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230400011

基于 Jump-SBERT 的二进制代码相似性检测技术研究

严尹彤　于　璐　王泰彦　李宇薇　潘祖烈
国防科技大学电子对抗学院　合肥 230037
网络空间安全态势感知与评估安徽省重点实验室　合肥 230037
(yanyintong.edu@nudt.edu.cn)

摘　要　二进制代码相似性检测技术在不同的安全领域中有着重要的作用。针对现有的二进制代码相似性检测方法面临计算开销大且精度低、二进制函数语义信息识别不全面和评估数据集单一等问题,提出了一种基于 Jump-SBERT 的二进制代码相似性检测技术。Jump-SBERT 有两个主要创新点,一是利用孪生网络构建 SBERT 网络结构,该网络结构能够在降低模型的计算开销的同时保持计算精度不变;二是引入了跳转识别机制,使 Jump-SBERT 可以学习到二进制函数的图结构信息,从而更加全面地捕获二进制函数的语义信息。实验结果表明,Jump-SBERT 在小函数池(32 个函数)中的识别准确率可达 96.3%,在大函数池(10000 个函数)中的识别准确率可达 85.1%,比最先进(State-of-the-Art,SOTA)的方法高出 36.13%,且 Jump-SBERT 在大规模二进制代码相似性检测中的表现更加稳定。消融实验表明,两个主要创新点对 Jump-SBERT 均有积极作用,其中,跳转识别机制的贡献最高可达 9.11%。

关键词:二进制代码;相似性检测;语义信息;SBERT 网络结构;跳转识别机制
中图分类号　TP312

Study on Binary Code Similarity Detection Based on Jump-SBERT

YAN Yintong,YU Lu,WANG Taiyan,LI Yuwei and PAN Zulie
College of Electronic Engineering,National University of Defense Technology,Hefei 230037,China
Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation,Hefei 230037,China

Abstract　Binary code similarity detection technology plays an important role in different security fields. Aiming at the problems of the existing binary code similarity detection methods,such as high computational cost and low accuracy,incomplete semantic information recognition of binary function and single evaluation data set,a binary code similarity detection technique based on Jump-SBERT is proposed. Jump-SBERT has two main innovations. One is to use twin networks to build SBERT network structure,which can reduce the calculation cost of the model while keeping the calculation accuracy unchanged. The other is to introduce jump recognition mechanism,which enables Jump-SBERT to learn the graph structure information of binary functions. Thus,the semantic information of binary function can be captured more comprehensively. Experimental results show that the recognition accuracy of Jump-SBERT can reach 96.3% in the small function pool(32 functions) and 85.1% in the large function pool(10000 functions),which is 36.13% higher than state-of-the-art(SOTA) methods. Jump-SBERT is more stable in large-scale binary code similarity detection. Ablation experiments show that both of the two main innovation points have positive effects on Jump-SBERT,and the contribution of jump recognition mechanism is up to 9.11%.

Keywords　Binary code,Similarity detection,Semantic information,SBERT network structure,Jump recognition mechanism

1　引言

二进制代码相似性检测是将一对函数的二进制表示作为输入,并产生一个数值作为输出来捕捉该对函数之间“相似性”的技术,该技术在不同的安全研究领域中有着至关重要的作用。例如,逆向工程师经常处理静态链接剥离的二进制文件,二进制代码相似性检测技术可以用于将未知函数与先前生成的数据库中的函数相匹配,从而节省大量的逆向工程时间[1-2]。此外,二进制代码相似性检测技术在修补第三方库中的漏洞中也有重要的应用。给定一个含有漏洞的函数,二进制代码相似性检测技术有助于在一个或多个二进制文件中查找相同或相似的函数,从而更快地识别和修补相关漏洞[3-5]。

随着机器学习技术的快速发展与应用,现有最先进(SOTA)的二进制代码相似性检测技术都是基于机器学习开发

到稿日期:2023-04-03　返修日期:2023-07-28
基金项目:国家自然科学基金青年科学基金(62202484)
This work was supported by the Young Scientists Fund of the National Natural Science Foundation of China(62202484).
通信作者:潘祖烈(panzulie17@nudt.edu.cn)

的。一般而言，这些检测技术将目标二进制函数表征为向量，并计算函数在向量空间中的相似度。二进制代码相似性检测技术按表征形式可以分为基于代码嵌入[6-8]和基于图嵌入[9-11]两种，基于代码嵌入的检测技术将每个函数看作一份文档，将每一条汇编指令看作一个单词，进而构建神经网络模型，基于上下文关系感知完成相似性检测等任务；基于图嵌入的检测技术对二进制函数生成更加富有语义信息的图结构表示，如控制流图(Control Flow Graph，CFG)、数据流图(Data Flow Graph，DFG)和抽象语法树(Abstract Syntax Tree，AST)等，进而利用深度神经网络进行图结构表征与相似性检测。尽管这些方法在二进制代码相似性检测上的表现有所提升，但仍存在一些局限性且有较大的改进空间。

首先，现有基于代码嵌入的二进制代码相似性检测技术大多基于 BERT 建立，该模型需要把两个二进制函数同时输入网络中，这在检测样本较大的情况下会导致巨大的计算开销。例如，从 10000 个二进制函数中找出最相似的函数对，需要 5000 万次计算，这种模型结构使得 BERT 不适用于语义相似性检测。目前已有方法尝试将二进制函数转化为向量，通过向量之间的相似性来表征函数对之间的相似性。这种方式可以降低计算开销，但计算精度完全依赖于表征向量，本质上是通过牺牲部分计算精度的方式来降低计算开销。

其次，现有二进制代码相似性技术不能完全捕获二进制函数的语义信息。基于代码嵌入的二进制代码相似性检测技术主要考虑指令的顺序和它们之间的上下文关系，较少考虑程序实际的执行信息，如控制流信息和数据流信息等。基于图嵌入的二进制代码相似性检测技术考虑了控制流和数据流信息，但会遗漏基本块中指令的语义信息和顺序，且手动提取的特征不能完全捕捉二进制代码的语义信息。

最后，现有最先进(SOTA)的二进制代码相似性检测技术的评估数据集不够多样化。不同的检测技术通常针对不同的检测目标，每一种技术解决方案都建立了自己的数据集，并在该数据集上进行训练和测试，使用不同的评估指标(ROC 曲线面积、MRR10 或 Recall@5)表示结果。而现实世界中的二进制代码相似性检测技术应用范围更广，需要处理的数据集更加多样，现有技术在小型数据集中的测试结果与实际应用时的测试结果差距较大。

为此，本文提出 Jump-SBERT 来解决以上问题，该模型基于 SBERT 网络结构[12]开发。SBERT 网络结构在 BERT 的基础上利用孪生神经网络生成具有二进制语义信息的嵌入向量，进而利用嵌入向量进行二进制函数之间的相似性比较。该网络结构在进行相似性检测时可以极大地降低计算开销，同时使计算精度保持不变。在 SBERT 网络结构的基础上，引入跳转识别训练机制，使得 Jump-SBERT 可以对二进制函数中的跳转信息进行学习，获得函数中的控制流和数据流等图结构信息，从而捕获更加全面的二进制函数语义信息。

本文在现实场景下对构建完成的 Jump-SBERT 进行评估，评估结果表明 Jump-SBERT 显著优于现有最先进(SOTA)的二进制代码相似性检测技术，如 jTrans[13]。实验以 Recall@1 为评价指标，在检测场景较为简单的情况下(32 个函数)，Jump-SBERT 对二进制函数的识别准确率达到了 96.3%；在检测场景比较复杂的情况下(10000 个函数)，Jump-SBERT 对二进制函数的识别准确率达到了 85.1%，比现有最先进(SOTA)的技术方案高出 36.13%。此外，本文还进行了消融实验，以评价跳转识别机制对 Jump-SBERT 的贡献，并将 Jump-SBERT 与 BERT 进行了对比。本文的主要贡献如下：

(1)提出了 Jump-SBERT，该模型的 SBERT 网络结构相比 BERT 增加了池化操作，可以降低二进制函数匹配所需的计算开销，同时保持计算精度不变。

(2)在 SBERT 的基础上引入了跳转识别机制，使得 Jump-SBERT 在快速且精准地进行二进制代码相似性检测的同时，能够更加全面地捕获二进制函数的语义信息。

(3)在更加多样化的数据集上进行了大量的实验，实验结果显示 Jump-SBERT 优于现有最先进(SOTA)的二进制代码相似性检测方案，其应用场景更加广泛。

2 相关工作

基于学习的二进制代码相似性检测技术主要受自然语言处理(NLP)的启发，将二进制函数和汇编指令类比为 NLP 中的句子或单词，并通过实值向量对这些信息进行表征。一些方法[7,14]基于 word2vec 模型进行二进制代码的表征，将每条汇编指令看作一句话，将操作数和操作码看作词单元，之后构建神经网络模型进行向量表征和相似性计算。第二类方法[15-16]基于 seq2seq 模型开发，它们将来自不同指令集架构的二进制函数映射到相同的嵌入空间，从而进行跨架构的二进制代码相似性检测。目前最先进(SOTA)的二进制代码相似性检测方法大多建立在 BERT[17]基础之上，例如 OrderMatters[18]，Trex[19]和 PalmTree[20]。BERT 是基于 Transformer 设计的 NLP 中最先进的预训练模型。

基于图嵌入的二进制代码相似性检测技术适合捕获二进制函数的控制流图(CFG)、数据流图(DFG)和抽象语法树(AST)等图结构信息，并通过结合图神经网络实现较高的检测准确率。由于二进制函数的图结构信息在不同指令集架构上的变化不大，因此基于图嵌入的检测技术有利于进行跨架构的二进制代码相似性检测[21-22]，比较有代表性的方法包括 Genius[9]，Gemini[10]，VulSeeker[11]，Asteria[23]，CodeCMR[24]和 GMN[25]。其中，CodeCMR 是一个端到端的跨模态搜索模型，用于实现二进制代码与程序源代码之间的匹配，其结合消息传递与 Set2Set 机制的 GGNN(Gated Graph Sequence Neural Networks)[26]作为函数的图结构表征模型，最终实现了较高的检测准确率；GMN 是近年来提出的一种基于图神经网络的检测模型，经过大规模实验评估[27]发现其检测效果最为突出。

总体而言，基于学习的二进制代码相似性检测技术在已知漏洞发现[28-32]、恶意软件检测[33-35]和补丁分析[36-38]等领域取得了较好的应用成果。然而，现有最先进(SOTA)的技术仍存在较大的局限性，例如，一些方法[7,8,14,32,39-40]没有考虑二进制函数的控制流、数据流等图结构信息，使得模型忽略了二进制函数的部分或全部结构信息；一些方法[9-10,18,41]主要使用图神经网络学习二进制函数的属性控制流图(ACFG)，忽略了基本块中汇编指令之间的关系。

3 方法设计

3.1 整体设计

为弥补现有二进制代码相似性检测技术的不足，本文提出了 Jump-SBERT，该模型基于 SBERT 网络结构设计，并引入了跳转识别机制。全新的设计理念使得 Jump-SBERT 能够有效解决现有二进制代码相似性检测技术遇到的挑战。

首先，Jump-SBERT 保留了 BERT 的基本训练方法，包括下一指令预测（Next Sentence Prediction，NSP）和掩码语言模型（Masked Language Model，MLM）。下一指令预测（NSP）可以使模型理解汇编指令上下文之间的联系；掩码语言模型（MLM）可以使训练的模型对掩码令牌具有一定的预测推理能力，迫使模型对汇编指令的语义进行理解。

其次，Jump-SBERT 基于 SBERT 设计，该模型利用孪生网络生成具有语义信息的嵌入向量，并对输出的结果增加了池化操作，从而生成固定大小的嵌入向量，使得 Jump-SBERT 在进行二进制代码相似性检测时有更低的计算开销和更高的计算精度。此外，在池化操作中，本文使用 MEAN 策略计算二进制函数的嵌入向量，实验表明该策略得到的结果的准确率最高[12,42]。

最后，Jump-SBERT 在 SBERT 网络结构的基础上引入了跳转识别机制，该机制对汇编指令中的跳转指令进行识别，并学习跳转指令与目标指令之间的上下文关系，从而使 Jump-SBERT 学习到二进制函数的图结构信息，则其生成的嵌入向量能更全面地表征二进制函数的语义信息。Jump-SBERT 的整体设计如图 1 所示。

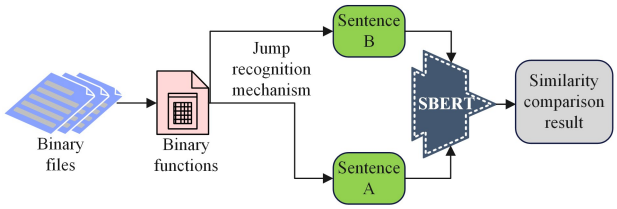

图 1 Jump-SBERT 整体设计

Fig. 1 Overall design of Jump-SBERT

3.2 SBERT 网络结构

BERT 是目前自然语言处理中最先进的模型，本文提出的 Jump-SBERT 仍基于 BERT 建立，并根据二进制函数的实际检测需求对 BERT 进行修改，设计出了 SBERT 网络结构。SBERT 的整体结构如图 2 所示。

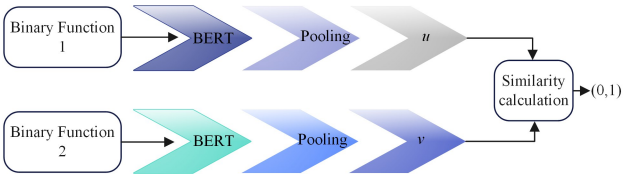

图 2 SBERT 整体结构

Fig. 2 Overall structure of SBERT

SBERT 网络结构利用孪生网络生成具有语义信息的嵌入向量，并通过嵌入向量计算二进制函数之间的相似性。该网络结构相比 BERT 增加了一个池化操作，即通过对 BERT 的输出结果进行池化操作来生成固定大小的嵌入向量，从而降低二进制函数相似性检测时所需的计算开销，而池化操作可以保证嵌入向量的语义信息完整，不影响最终的计算精度。

在池化操作中，SBERT 网络结构使用 MEAN 策略计算二进制函数的嵌入向量；在嵌入向量相似性比较时，本文采取了 4 种策略进行计算，分别是余弦相似度、曼哈顿距离、欧氏距离和点积相似度。其中，余弦相似度（Cosine-similarity）用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小，相比距离度量，余弦相似度更加注重两个向量在方向上的差异，而非距离或长度上。余弦相似度的计算式为：

$$sim(u,v)=\frac{\vec{u}\cdot\vec{v}}{\|u\|\cdot\|v\|} \tag{1}$$

曼哈顿距离（Manhattan-distance）是一种在几何度量空间中标明两个点在标准坐标系上的绝对轴距总和，计算式为：

$$d_{uv}=\sum_{i=1}^{n}|u_i-v_i| \tag{2}$$

欧氏距离（Euclidean-distance）也称欧几里得距离，是最常见的距离度量指标，用来衡量多维空间中两个点之间的绝对距离，计算公式为：

$$d_{uv}=\sqrt{\sum_{i=1}^{n}(u_i-v_i)^2} \tag{3}$$

点积相似度（Dot-product-similarity）和余弦相似度类似，都不是距离度量指标。余弦相似度只关心向量角度差，而点积相似度关注向量大小和向量之间的角度，计算式为：

$$sim(u,v)=|\vec{u}|*|\vec{v}|*\cos\theta \tag{4}$$

最后，通过实验来对比 4 种嵌入向量计算策略得到的结果，进而为 Jump-SBERT 选出最优的嵌入向量计算策略。

3.3 跳转识别机制

汇编语言与自然语言有很大的不同，为使自然语言处理模型能够更好地应用于汇编语言，需要对相关模型进行修改，同时对训练数据和测试数据进行处理。对于传统 NLP 模型修改，本文使用了 SBERT 网络结构；对于训练数据和测试数据处理，本文引入了跳转识别机制。

由于二进制代码中有太多的词汇（如常量、字符串等），无法在词汇表中覆盖，通常会导致 OOV（out-of-vocabulary）问题，因此需要对未标记词进行处理。对于常量数据，使用[data]标签代替；对于字符串数据，使用[str]标签代替；对于跳转地址，使用[addr]标签代替；对于调用的函数名，使用[function]代替；其余未知项使用[unk]标签代替。通过对出现频率较低的词汇进行替代，以解决 NLP 模型在处理汇编语言时遇到的 OOV 问题。

现有基于代码嵌入的二进制代码相似性检测技术无法捕获二进制函数的图结构信息；基于图嵌入的二进制代码相似性检测技术往往忽略了基本块中指令的语义信息和顺序。针对此问题，本文提出了跳转识别机制。跳转指令的源令牌与目标令牌间有着紧密的“上下文”联系，这种联系与相邻指令间的联系是同等重要的。对相邻指令的处理和学习，能使模型获得指令间的语义信息和顺序；而对跳转指令的处理和学习，则能使模型捕获二进制函数的图结构信息。

在实际操作中，首先对获得的二进制函数代码进行标准化处理，进而对跳转指令源令牌和目标令牌进行识别。在对模型进行训练时，每次遇到跳转指令便以一定的概率（如 α，β，γ）跳转至目标令牌学习新的指令，或不跳转继续学习

下一指令，这样模型不仅可以学习到直接跳转内容，也可以学习到间接跳转内容，从而捕获更加全面的二进制函数语义信息。在 Jump-SBERT 的具体实验评估中，α，β，γ 的取值分别为 0.1，0.3 和 0.6。以一段二进制函数代码为例，展示跳转识别的处理机制，Jump-SBERT 输入指令表征如图 3 所示。

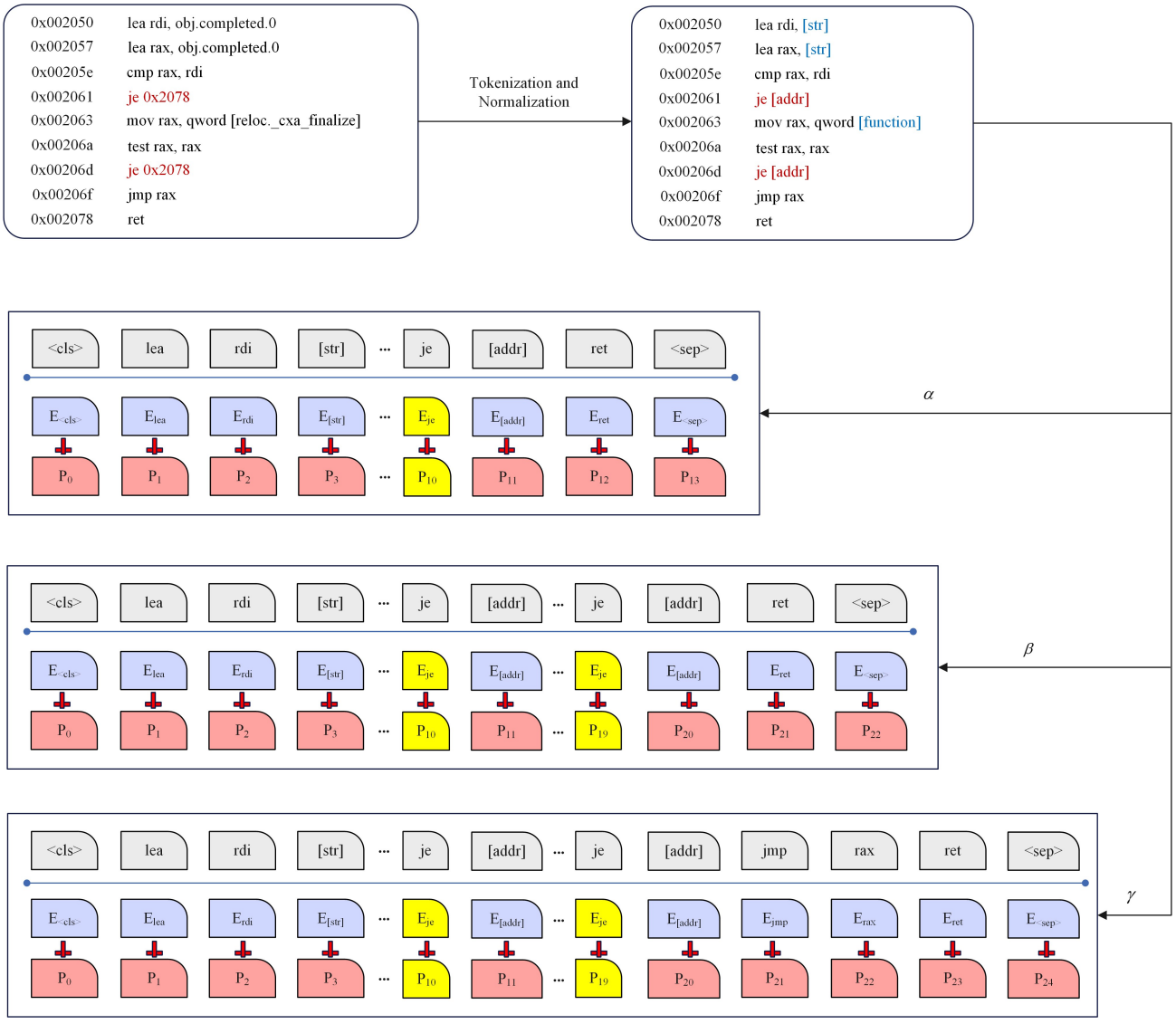

图 3 Jump-SBERT 输入指令表征

Fig. 3 Jump-SBERT input instruction representation

除了跳转识别机制，Jump-SBERT 还有两个无监督学习任务，分别是下一指令预测（NSP）和掩码语言模型（MLM）。下一指令预测旨在使 Jump-SBERT 具有判断汇编指令是否连续的能力；掩码语言模型使 Jump-SBERT 具有一定的指令推理能力；跳转识别机制可以提升 Jump-SBERT 对跳转指令“上下文”的理解。对于掩码语言模型的设计，本文在所有汇编指令中选择 15%的指令进行替换，在需要替换的指令中，随机选择 90%的指令使用[mask]标签替换，剩余 10%的指令使用随机指令替换。

4 实验设置

4.1 实验环境

本次实验的环境为 Linux 64 位下的 Ubuntu 20.04 系统，服务器的主要配置为 Intel Xeon Gold 6230R @ 104x 4GHz 处理器、256GB 内存以及 RTX3090 显卡。

4.2 实验数据

现有二进制代码相似性检测技术往往只能针对某些特定的数据集进行检测，这与现实世界中的实际应用脱节。为此，本文决定在大规模数据集上对 Jump-SBERT 进行训练和测试，以得到更真实的检测结果。通过分析和筛选，本文的数据集选择使用 BinaryCorp[13]，该数据集是目前已知最多样化的数据集，最能代表现实世界中的实际应用，包括 9819 个项目、48130 个二进制文件和 25877011 个函数。具体训练数据和测试数据的数量统计情况如表 1 所列。

表 1 训练与测试数据集统计

Table 1 Training and test data set statistics

Datasets	# Projects	# Binaries	# Functions
BinaryCorp Train	7845	38455	21085338
BinaryCorp Test	1974	9675	4791673

4.3 评价指标

为评价 Jump-SBERT 在二进制代码相似性检测中的表现，需要建立评价指标。首先定义两个函数池 F 和 G：

$$F=\{f_1,f_2,\cdots,f_i,\cdots,f_n\} \tag{5}$$

$$G=\{g_1,g_2,\cdots,g_i,\cdots,g_n\} \tag{6}$$

函数池 F 和函数池 G 中的函数部分相同，首先从函数池 F 中选择函数 f_i，并从函数池 G 中选择与其最为相似的函数 g_i，最后通过 Jump-SBERT 判断两者之间相似性。为表示函数的召回率 $Recall@k$，设评价函数为 E，其具体公式为：

$$E(x)=\begin{cases}0, & x=\text{False}\\ 1, & x=\text{True}\end{cases} \tag{7}$$

则函数召回率的计算式可以表示为：

$$Recall@k=\frac{1}{|F|}\sum_{f_i}^{n}(Rank_{g_i}\leqslant k) \tag{8}$$

5 实验评估

实验评估旨在回答以下问题：

问题 1 Jump-SBERT 在进行二进制代码相似性检测时的准确率如何？

问题 2 Jump-SBERT 与现有最先进(SOTA)的二进制代码相似性检测技术相比主要有哪些优势？

5.1 二进制代码相似性检测

本文在数据集 BinaryCorp 上进行实验评估，并将实验结果与最先进(SOTA)的二进制代码相似性检测技术 jTrans 进行对比。为此，本文设计了两种函数池对 Jump-SBERT 进行评估。第一种函数池包含的二进制函数数量较少，称为小函数池，此时，F 和 G 函数池中均只有 32 个二进制函数；第二种函数池包含的二进制函数数量较多，称为大函数池，此时，F 和 G 函数池中均有 10 000 个二进制函数。其中，小函数池的构造方法为：从数据集 BinaryCorp 中随机选取 8 个二进制文件，并从每个二进制文件中随机选择 4 个二进制函数得到小函数池 F；仍是这 8 个二进制文件，从每个二进制文件中再随机选择 4 个二进制函数得到小函数池 G。大函数池的构造方法为：从数据集 BinaryCorp 中随机选取 1 000 个二进制文件，并从每个二进制文件中随机选择 10 个二进制函数得到大函数池 F；仍是这 1 000 个二进制文件，从每个二进制文件中再随机选择 10 个二进制函数得到大函数池 G。

在模型评价中，小函数池和大函数池中的二进制函数均是从数据集 BinaryCorp 中按一定规则选择的，并利用 Recall@1 指标对 Jump-SBERT 的相似性计算方式进行评估，实验结果如表 2、表 3 所列。

表 2 Jump-SBERT 在小函数池中的评估结果

Table 2 Jump-SBERT evaluation results in small function pool

Jump-SBERT	O0,O3	O1,O3	O2,O3	O0,Os	O1,Os	O2,Os	Average
Cosine-similarity	0.951	**0.974**	0.958	0.956	**0.972**	**0.966**	**0.963**
Manhattan-distance	**0.960**	0.958	0.942	0.908	0.952	0.944	0.944
Euclidean-distance	0.958	0.960	0.930	0.906	0.954	0.949	0.943
Dot-product-similarity	0.945	0.969	0.952	**0.959**	0.969	0.950	0.957
jTrans	0.941	0.970	**0.981**	0.949	0.964	0.964	0.962

表 3 Jump-SBERT 在大函数池中的评估结果

Table 3 Jump-SBERT evaluation results in large function pool

Jump-SBERT	O0,O3	O1,O3	O2,O3	O0,Os	O1,Os	O2,Os	Average
Cosine-similarity	**0.868**	**0.880**	0.826	0.836	**0.860**	0.834	**0.851**
Manhattan-distance	0.861	0.848	**0.839**	**0.849**	0.854	**0.841**	0.849
Euclidean-distance	0.853	0.855	0.844	**0.849**	0.859	0.835	0.849
Dot-product-similarity	0.856	0.872	0.821	0.826	0.847	0.827	0.841
jTrans	0.499	0.668	0.736	0.550	0.648	0.648	0.625

由表 2 可知，Jump-SBERT 在二进制代码相似性检测中，Recall@1 最高可达 96.3%；由表 3 可知，Jump-SBERT 在大函数池中的评估结果最高可达 85.1%，评估结果均优于最先进(SOTA)的二进制代码相似性检测技术 jTrans，其中，Jump-SBERT 在大函数池中的识别准确率比 jTrans 高出 36.13%。而 jTrans 已经被证明优于 Genius[9]，Gemini[10]，Asm2Vec[7]，SAFE[8]，GraphEmb[41] 和 OrderMatters[18] 等模型，表明本文提出的模型在二进制代码相似性检测中的表现优异。此外，在 4 种相似性计算方式(Cosine-similarity，Manhattan-distance，Euclidean-distance 和 Dot-product-similarity)中，通过余弦相似性(Cosine-similarity)计算得到的平均 Recall@1 最高。在下面的讨论中，使用余弦相似性作为 Jump-SBERT 计算相似性的方式。

5.2 Jump-SBERT 的优势

本文提出的 Jump-SBERT 相比现有二进制代码相似性检测技术有两个主要的优势：一是对二进制函数相似性检测的 Recall@1 较高；二是在进行大规模检测时，模型仍能保持较高的检测准确率。

Jump-SBERT 与 jTrans 均能捕获二进制函数的图结构信息，但二者的实现原理并不相同。Jump-SBERT 所用的跳转识别机制是在数据层面对汇编代码进行处理，即跳转识别机制对原始汇编代码进行处理，生成具有二进制函数图结构信息的汇编代码，进而将这些代码输入 SBERT 网络结构；jTrans 所用的跳跃感知方法是在 BERT 模型内对跳转指令进行编码，即 jTrans 将原始汇编代码直接输入 BERT 中，在指令位置编码时对跳转指令进行标记，从而学习二进制函数的图结构信息。jTrans 的跳跃感知方法只能识别二进制函数的直接跳转信息，而 Jump-SBERT 的跳转识别机制不仅可以识别直接跳转信息，还可以识别间接跳转信息。

将 Jump-SBERT 与 jTrans 在不同函数池数量下的表现进行对比，随着函数池 F 和 G 中二进制函数数量的增加，两个模型检测准确率的变化如图 4 所示。其中，设两个函数池中的二进制函数数量之和为 Q，横坐标为 $\log_{10}Q$，纵坐标为模型的检测准确率。

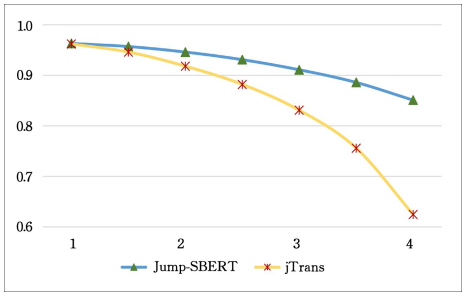

图 4 Jump-SBERT 与 jTrans 在不同函数池数量下的准确率对比

Fig. 4 Comparison of accuracy of Jump-SBERT and jTrans at different number of function pools

由图 4 可知，随着函数池的增大，两个模型的检测准确率均有所下降，而 Jump-SBERT 的检测准确率下降幅度较小。当函数池的数量由 10 增加至 10000 时，jTrans 的检测准确率的降幅达 35.03%，而 Jump-SBERT 的检测准确率的降幅为 11.63%，这表明 Jump-SBERT 在大规模二进制代码相似性检测时仍能保持较优的性能。

6 消融实验

消融实验旨在探究以下问题：

问题 1 跳转识别机制对 Jump-SBERT 有怎样的贡献？

问题 2 Jump-SBERT 相比 BERT 在性能上有怎样的变化？

6.1 跳转识别机制对 Jump-SBERT 的影响

为了探究跳转识别机制对 Jump-SBERT 的贡献，本文仍使用数据集 BinaryCorp，并在小函数池和大函数池中对 SBERT 网络结构的表现进行测试。鉴于 Jump-SBERT 使用余弦相似性进行相似性计算得到的结果最好，为保证公平，在对 SBERT 网络结构的测试中，仍在相似性计算中使用余弦相似性进行度量。将 SBERT 与 Jump-SBERT 的测试结果进行比较，对比结果如图 5、图 6 所示。

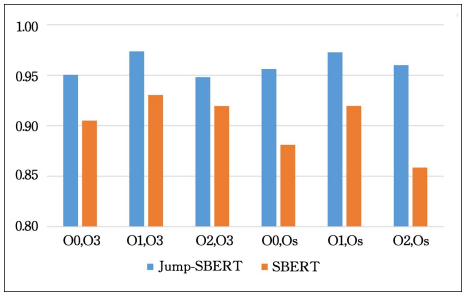

图 5 Jump-SBERT 与 SBERT 在小函数池中的识别准确率对比

Fig. 5 Comparison of recognition accuracy of Jump-SBERT and SBERT in small function pool

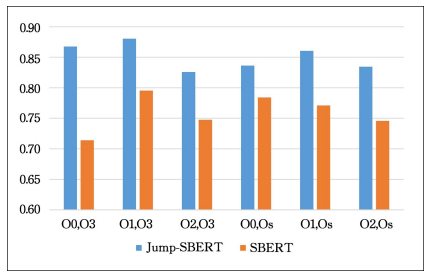

图 6 Jump-SBERT 与 SBERT 在大函数池中的识别准确率对比

Fig. 6 Comparison of recognition accuracy of Jump-SBERT and SBERT in large function pool

显然，不管在小函数池还是在大函数池中，跳转识别机制的引入对 Jump-SBERT 的识别准确率提升均有积极作用。其中，跳转识别机制使 Jump-SBERT 在小函数池中识别准确率平均有 7.07%的提升，在大函数池中平均有 9.11%的提升。

6.2 Jump-SBERT，SBERT 与 BERT 的性能对比

为探究 Jump-SBERT，SBERT 与 BERT 在性能上有怎样的差异，进行了此次对比实验。SBERT 相比 BERT 修改了网络结构，Jump-SBERT 相比 SBERT 引入了跳转识别机制，这些改变对模型的性能是否有提升，如果有提升，那么提升的效果如何，回答这些问题需要进行实验探究。为保证测试结果合理，实验仍使用 BinaryCorp 数据集，在大函数池中进行实验测试，相似性计算方式均使用余弦相似性。Jump-SBERT，SBERT 与 BERT 的识别准确率对比结果如图 7 所示。

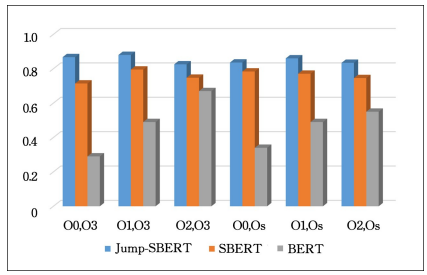

图 7 Jump-SBERT，SBERT 与 BERT 识别准确率对比

Fig. 7 Comparison of recognition accuracy of Jump-SBERT, SBERT and BERT

由图 7 可知，Jump-SBERT 和 SBERT 相比 BERT 在性能上有明显的提升，且 Jump-SBERT 和 SBERT 的表现更加稳定，在不同测试环境下的识别准确率波动不大，而 BERT 在不同测试环境下的识别准确率波动较大。实验结果显示，Jump-SBERT 相比 BERT 在二进制代码相似性检测上的平均识别准确率提升了 37.91%，SBERT 相比 BERT 在二进制代码相似性检测上的平均识别准确率提升了 28.79%，表明在 BERT 基础上设计的 SBERT 网络结构和引入的跳转识别机制对 Jump-SBERT 均有积极作用，且对模型识别准确率的提升显著。

同时，为探究 Jump-SBERT，SBERT 与 BERT 的二进制代码相似性检测效率，对这 3 个模型在大函数池中的检测时间进行统计，检测时间单位为 s，不同优化选项下的检测用时如表 4 所列。

表 4 Jump-SBERT，SBERT 与 BERT 在大函数池中的检测时间对比

Table 4 Comparison of detection time of Jump-SBERT，SBERT and BERT in large function pool

	O0，O3	O1，O3	O2，O3	O0，Os	O1，Os	O2，Os	Average
Jump-SBERT	523	509	431	591	494	548	516
SBERT	**505**	**482**	**403**	**588**	**455**	**529**	**493.7**
BERT	775	754	634	905	714	822	767.3

由表 4 可知，Jump-SBERT，SBERT 和 BERT 在大函数池下的平均检测用时分别为 516 s，493.7 s 和 767.3 s，表明 SBERT 网络结构有效地降低了模型的计算开销，对二进制代码相似性检测效率的提升有重要贡献。

结束语 本文提出了 Jump-SBERT，该模型在 BERT 的基础上设计了 SBERT 网络结构，并引入了跳转识别机制。其中，SBERT 网络结构使模型在降低计算开销的同时仍能保持计算精度不变，跳转识别机制则是让模型学习到了二进制函数的图结构信息，二者相结合使 Jump-SBERT 可以捕获更全面的二进制函数语义信息，反映到实验结果上便是识别准确率的大幅提升。在小函数池（32 个二进制函数）中，Jump-SBERT 的识别准确率达到 96.3%；在大函数池（10 000 个二进制函数）中，Jump-SBERT 的识别准确率达到 85.1%，比 jTrans 高出 36.13%。

通过实验观察到现有最先进（SOTA）的二进制代码相似性检测技术存在较大的缺陷，随着函数池的不断扩大，现有二进制代码相似性检测技术的识别准确率下降较快。实验表明，当函数池中二进制函数的数量由 10 增加至 10 000 时，jTrans 识别准确率的降幅达 35.03%，而 Jump-SBERT 识别准确率的降幅仅为 11.63%。本文提出的 Jump-SBERT 在进行大规模的二进制代码相似性检测时结果更加稳定，更适合在现实世界中的复杂场景下使用。

然而，本文提出的 Jump-SBERT 主要在 X86 架构下训练和测试，虽然这种方法也可以应用于其他架构（如 ARM，MIPS 和 PowerPC 等）下的二进制代码相似性检测，但本质上并不属于跨架构二进制代码相似性检测技术，在需要进行跨架构检测二进制代码相似性的场景下劣势显著。在下一步的工作中，将尝试对 Jump-SBERT 进行改进，使其在进行跨架构二进制代码相似性检测时仍能保持较高的识别准确率。

参 考 文 献

[1] MIYANI D，HUANG Z，LIE D. Binpro：A tool for binary source code provenance[J]. arXiv：1711.00830，2017.

[2] SHAHKAR A. On matching binary to source code [D]. Montreal：Concordia University，2016.

[3] DAVID Y，PARTUSH N，YAHAV E. Firmup：Precise static detection of common vulnerabilities in firmware[J]. ACM SIGPLAN Notices，2018，53(2)：392-404.

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

[5] HUANG H，YOUSSEF A M，DEBBABI M. Binsequence：Fast，accurate and scalable binary code reuse detection[C]// Proceedings of the 2017 ACM on Asia Conferenceon Computer and Communications Security. 2017：155-166.

[6] SHALEV N，PARTUSH N. Binary similarity detection using machine learning[C]// Proceedings of the 13th Workshop on Programming Languages and Analysis for Security. 2018：42-47.

[7] 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）. IEEE，2019：472-489.

[8] MASSARELLI L，DI LUNA G A，PETRONI F，et al. Safe：Self-attentive function embeddings for binary similarity[C]// 16th International Conference（DIMVA 2019）. Springer International Publishing，2019：309-329.

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

[10] XU X，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. 2017：363-376.

[11] XIU H，YAN X，WANG X，et al. Hierarchical graph matching network for graph similarity computation [J]. arXiv：2006.16551，2020.

[12] REIMERS N，GUREVYCH I. Sentence-bert：Sentence embeddings using siamese bert-networks [J]. arXiv：1908.10084，2019.

[13] WANG H，QU W，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. 2022：1-13.

[14] RRDMOND K，LUO L，ZENG Q. A cross-architecture instruction embedding model for natural language processing-inspired binary code analysis[J]. arXiv：1812.09652，2018.

[15] ZUO F，LI X，YOUNG P，et al. Neural machine translation inspired binary code similarity comparison beyond function pairs [J]. arXiv：1808.04706，2018.

[16] ZHANG X，SUN W，PANG J，et al. Similarity metric method for binary basic blocks of cross-instruction set architecture [C]// Proceedings of 2020 Workshop on Binary Analysis Research. 2020.

[17] DEVLIN J，CHANG M W，LEE K，et al. Bert：Pre-training of deep bidirectional transformers for language understanding[J]. arXiv：1810.04805，2018.

[18] YU Z，CAO R，TANG Q，et al. Order matters：Semantic-aware neural networks for binary code similarity detection[C]// Pro-

ceedings of the AAAI Conference on Artificial Intelligence. 2020:1145-1152.
[19] PEI K, XUAN Z, YANG J, et al. Trex: Learning execution semantics from micro-traces for binary similarity[J]. arXiv:2012.08680, 2020.
[20] LI X, QU Y, YIN H. Palmtree: Learning an assembly language model for instruction embedding[C]// Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 2021:3236-3251.
[21] HAQ I U, CABALLERO J. A survey of binary code similarity [J]. ACM Computing Surveys(CSUR), 2021, 54(3):1-38.
[22] PAN Z, WANG T, YU L, et al. Position Distribution Matters: A Graph-Based Binary Function Similarity Analysis Method [J]. Electronics, 2022, 11(15):24-46.
[23] YANG S, CHENG L, ZENG Y, et al. Asteria: Deep learning-based AST-encoding for cross-platform binary code similarity detection[C]// 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2021:224-236.
[24] YU Z, ZHENG W, WANG J, et al. Codecmr: Cross-modal retrieval for function-level binary source code matching[J]. Advances in Neural Information Processing Systems, 2020, 33: 3872-3883.
[25] LI Y, GU C, DULLIEN T, et al. Graph matching networks for learning the similarity of graph structured objects[C]// International Conference on Machine Learning. PMLR, 2019: 3835-3845.
[26] LI Y, TARLOW D, BROCKSCHMIDT M, et al. Gated graph sequence neural networks[J]. arXiv:1511.05493, 2015.
[27] MARCELLI A, GRAZIANO M, UGARTE-PEDRERO X, et al. How machine learning is solving the binary function similarity problem[C]// 31st USENIX Security Symposium (USENIX Security 22). 2022:2099-2116.
[28] DAVID Y, PARTUSH N, YAHAV E. Similarity of binaries through re-optimization [C] // Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2017:79-94.
[29] FENG Q, WANG M, ZHANG M, et al. Extracting conditional formulas for cross-platform bug search[C]// Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. 2017:346-359.
[30] LIN J, WANG D, CHANG R, et al. EnBinDiff: Identifying Data-only Patches for Binaries[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 20(1):343-359.
[31] HEMEL A, KALLEBERG K T, VERMAAS R, et al. Finding Software License Violations Through Binary Code Clone Detection—A Retrospective[J]. ACM SIGSOFT Software Engineering Notes, 2021, 46(3):24-25.
[32] LIU B, HUO W, ZHANG C, et al. αdiff: cross-version binary code similarity detection with dnn[C]// Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018:667-678.
[33] CESARE S, XIANG Y, ZHOU W. Control flow-based malware variant detection[J]. IEEE Transactions on Dependable and Secure Computing, 2013, 11(4):307-317.
[34] DAREM A, ABAWAJY J, MAKKAR A, et al. Visualization and deep-learning-based malware variant detection using OpCode-level features[J]. Future Generation Computer Systems, 2021, 125:314-323.
[35] LUO L, MING J, WU D, 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.
[36] KARGEN U, SHAHMEHRRI N. Towards robust instruction-level trace alignment of binary code[C]// 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2017:342-352.
[37] PENG J, LI F, LIU B, et al. 1dvul: Discovering 1-day vulnerabilities through binary patches[C]// 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2019:605-616.
[38] XU Y, XU Z, CHEN B, et al. Patch based vulnerability matching for binary programs[C]// Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020:376-387.
[39] DUAN Y, LI X, WANG J, et al. Deepbindiff: Learning program-wide code representations for binary diffing[C]// Network and Distributed System Security Symposium. 2020.
[40] 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.
[41] MASSARELLI L, DI LUNA G A, PETRONI F, et al. Investigating graph embedding neural networks with unsupervised features extraction for binary analysis[C]// Proceedings of the 2nd Workshop on Binary Analysis Research (BAR). 2019:1-11.
[42] THAKUR N, REIMERS N, DAXENBERGER J, et al. Augmented sbert: Data augmentation method for improving bi-encoders for pairwise sentence scoring tasks [J]. arXiv: 2010.08240, 2020.

YAN Yintong, born in 1997, postgraduate. His main research interests include network security and binary code similarity detection.

PAN Zulie, born in 1976, Ph.D, professor. His main research interests include network security, vulnerability discovery and computer science.

(责任编辑:杨雪敏)