Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

二进制代码相似性检测技术

硕士研究生 高玺凯

2024年10月08日

问题回溯

- 总结反思
  - 语速过快，内容过多，重点不够突出
  - 时间安排不合理

- 相关内容
  - 2024.02.04 高玺凯《基于深度学习的二进制函数相似性分析》
  - 2022.11.27 沈宇辉《二进制函数相似性分析》
  - 2022.06.26 邢继媛《二进制程序开源成分分析》

内容提要

- 预期收获
- 内涵解析与研究目标
- 研究背景与意义
- 研究历史与现状
- 知识基础
- 算法原理
  - CEBin
  - HermesSim
- 特点总结与未来展望
- 参考文献

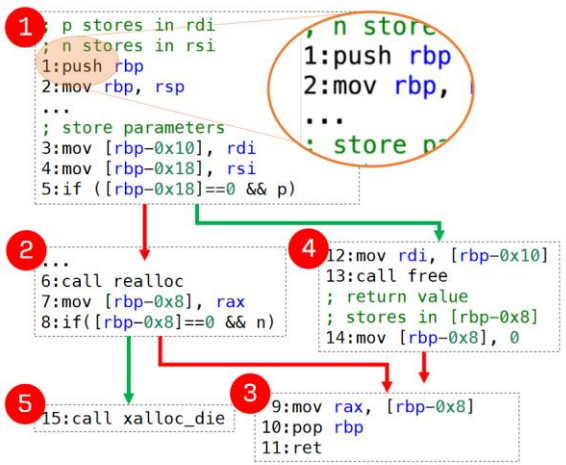
预期收获

- 预期收获
  - 掌握二进制代码相似性检测技术的基本概念
  - 了解二进制代码相似性检测技术的研究背景和意义
  - 理解两种目前最先进的二进制代码相似性检测方法

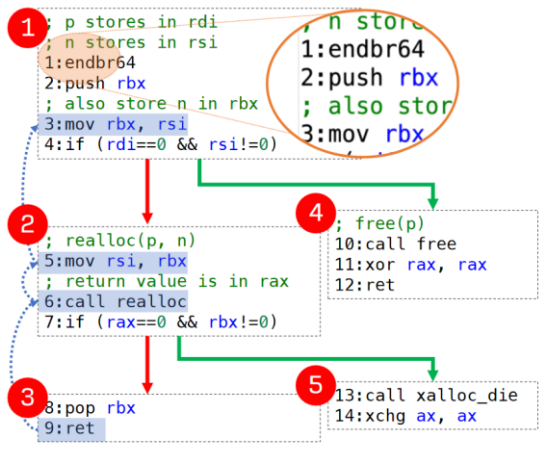
内涵解析与研究目标

- 内涵解析
  - 二进制函数：由高级编程语言编写的源代码函数经过编译生成的机器码函数
  - 二进制代码相似性检测：在没有源代码和符号信息的前提下，给定一个查询函数，从候选函数中识别相似函数
    - 由同一源代码函数编译得到的二进制函数称为相似函数
    - 同一源代码函数在不同编译器、优化选项或目标架构下编译生成的二进制函数可能会有显著差异
- 研究目标
  - 结合深度学习、二进制程序分析等理论
  - 开发能够在跨编译器、跨优化选项、跨架构场景下有效执行相似性分析的方法

```
1   void xalloc_die(){
2       ...
3       exit(-1);
4   }
5   void*xrealloc(void*p,
6                 size_t n){
7     if (!n && p){
8       free (p);
9       return NULL;
10    }
11    p = realloc (p, n);
12    if (!p && n)
          xalloc_die ();
13    return p;
14  }
```



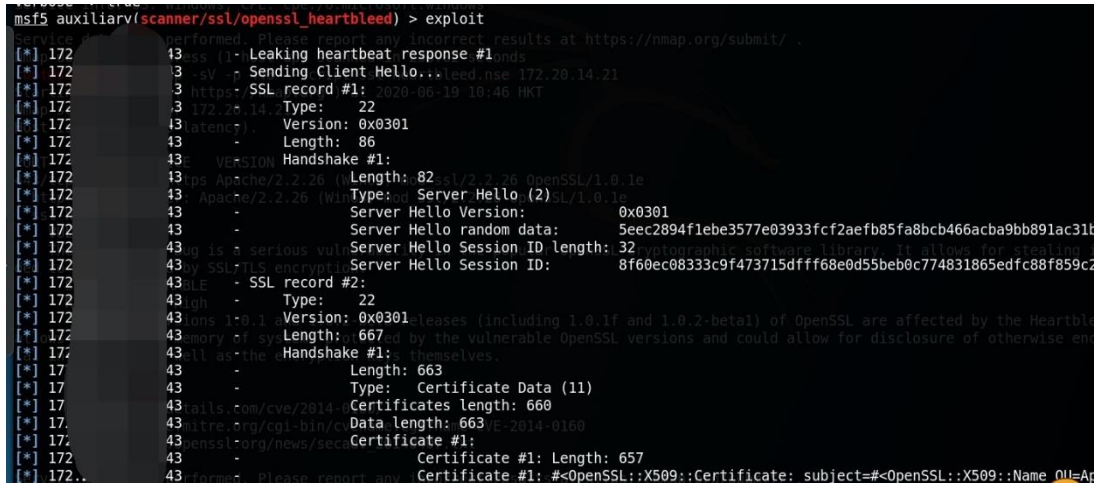
(c) CFG (Clang -O0)



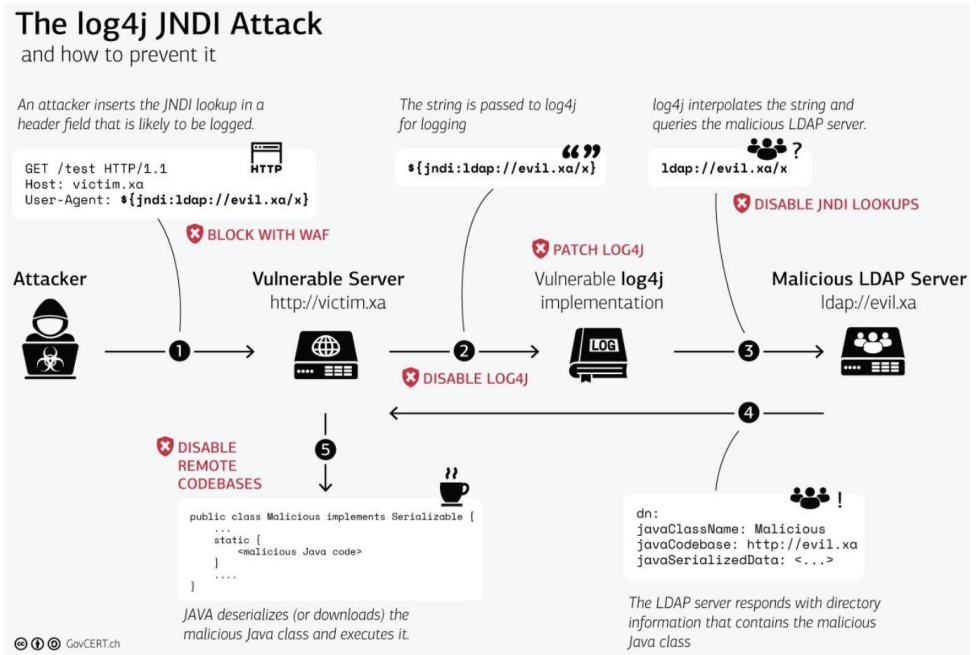
(b) CFG (GCC -O3)

研究背景

- 研究背景
  - 开发人员越来越广泛地复用开源项目和第三方库中的代码
    - 在一定程度上提升了软件开发效率、降低了开发成本
    - 带来了潜在的安全隐患，使得已知漏洞在大量计算机和物联网固件设备中迅速传播
  - 由于知识产权保护等原因，厂商通常只提供软件被剥离符号信息的二进制文件版本，并且其源代码通常也难以获取



Heartbleed漏洞 CVE-2014-0160 OpenSSL



Log4Shell漏洞 CVE-2021-44228 Apache Log4j

研究意义

- 研究意义
  - 二进制代码相似性检测在1-Day漏洞检测、代码克隆检测、恶意软件检测、软件剽窃检测和自动软件修复等多个应用领域中具有广泛的应用
  - 研究二进制代码相似性分析技术能够在无源码条件下自动化分析二进制程序中的漏洞，以有效监控和分析漏洞代码的传播情况
  - 现有二进制代码相似性检测方法无法在复杂的实际场景下应用，亟需开发更高效且准确的方法以适应实际需求！
    - HUAWEI于2023年7月17日在黄大年茶思屋网站发布的技术难题和技术诉求

- **基于二进制相似度匹配的二进制漏洞扫描：**预训练二进制大语言模型[4]，具备漏洞补丁的识别能力，能够**部分识别漏洞修复补丁**是否打入并提升漏洞扫描准确率(约为80%)
  - 该技术严重受到**编译选项优化等级**影响，在部分恶劣条件下，准确率下降到不足10%

技术诉求

二进制漏洞扫描算法：在Linux或Windows的二进制数据集上准确识别二进制文件中的漏洞，漏洞识别准确率90%以上。

研究历史与现状

2017 Xu等人提出Gemini方法。首次引入深度学习技术，基于基本块统计特征生成属性控制流图，利用s2v孪生网络将其嵌入向量空间，并通过余弦距离衡量函数相似性

2019 Li等人提出图匹配网络(GMN)。以两个CFG作为输入，进行联合嵌入和推理，有效捕获图之间的复杂相互作用，最终输出两个CFG的相似度

2019 Massarelli等人提出SAFE方法。使用i2v模型将指令映射为嵌入向量，然后利用自注意力网络将指令嵌入聚合为函数嵌入

2020 Yu等人提出OrderMatters方法。使用BERT生成指令嵌入；使用消息传递图神经网络生成属性控制流图嵌入；使用ResNet网络提取基本块顺序信息

2022 Pei等人提出Trex方法。利用分层Transformer模型在预训练阶段自动从微轨迹（即在特定约束下的动态执行轨迹）中学习指令序列的执行语义

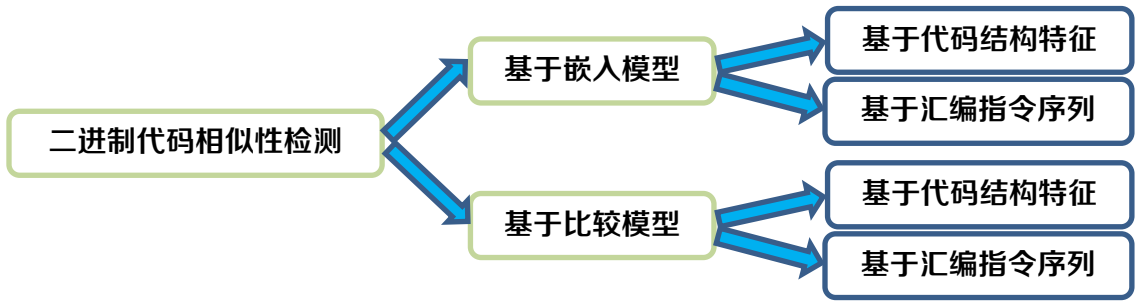
2022 Wang等人提出jTrans方法。在预训练阶段引入跳转目标预测任务，将控制流信息嵌入到Transformer模型中，有效地学习二进制函数的结构和语义特征

2023 Yang等人提出Asteria-Pro方法。以二进制代码反编译得到的抽象语法树作为特征表示；使用Tree-LSTM生成AST的嵌入向量表示；此外，结合预过滤和重排序模块，有效提升了方法性能

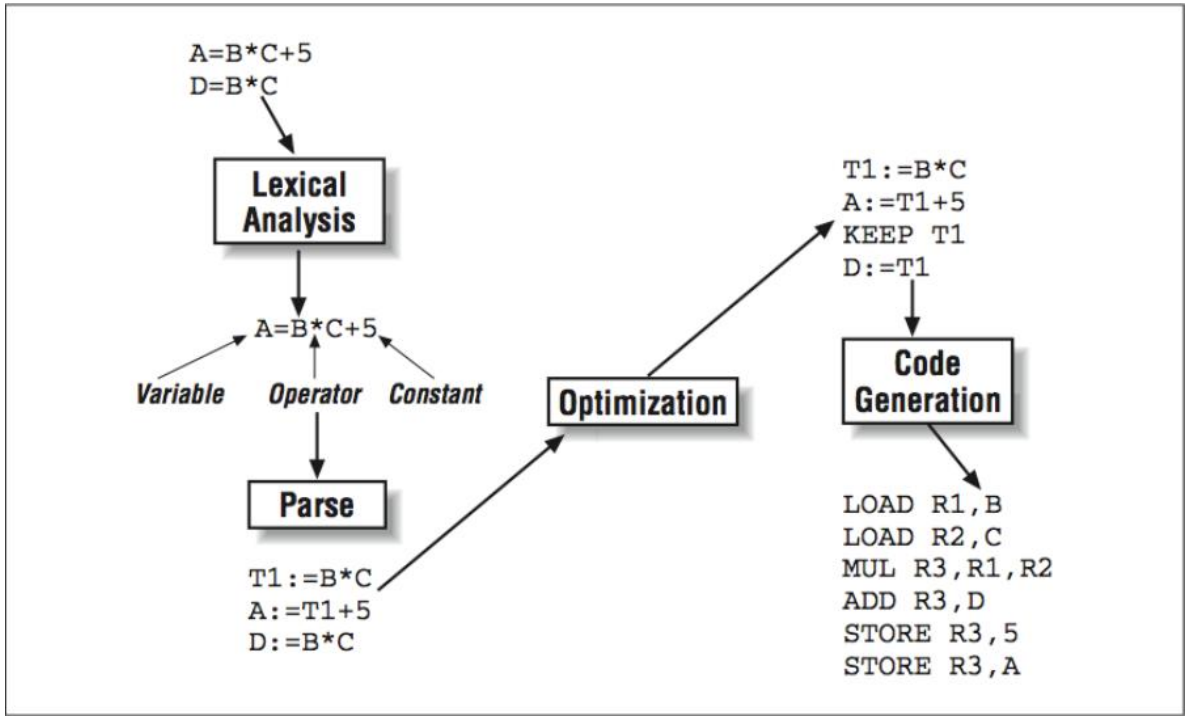
2023 Yang等人提出DiEmph方法。通过识别指令分类重要性和语义重要性，降低不表示程序关键语义的指令的重要性，有效防止模型学习由编译器引入的指令分布偏差

2024 He等人提出HermesSim方法

2024 Wang等人提出CEBin方法



- 编译器类型
  - 不同编译器（如GCC/Clang/MSVC/ICC）具有各自的特性和固有约定
- 编译器优化选项（影响编译时间、目标文件大小和执行效率等）
  - O0：不启用优化
  - O1：启用基本优化（去除调试信息、函数内联、死代码消除等）
  - O2：启用更多优化（循环展开等）
  - O3：启用几乎所有的优化技术
  - Os：优化代码的空间占用
- 目标架构
  - 不同架构（如x64/x86/ARM/MIPS）有其特定的指令集、优化策略和执行模式



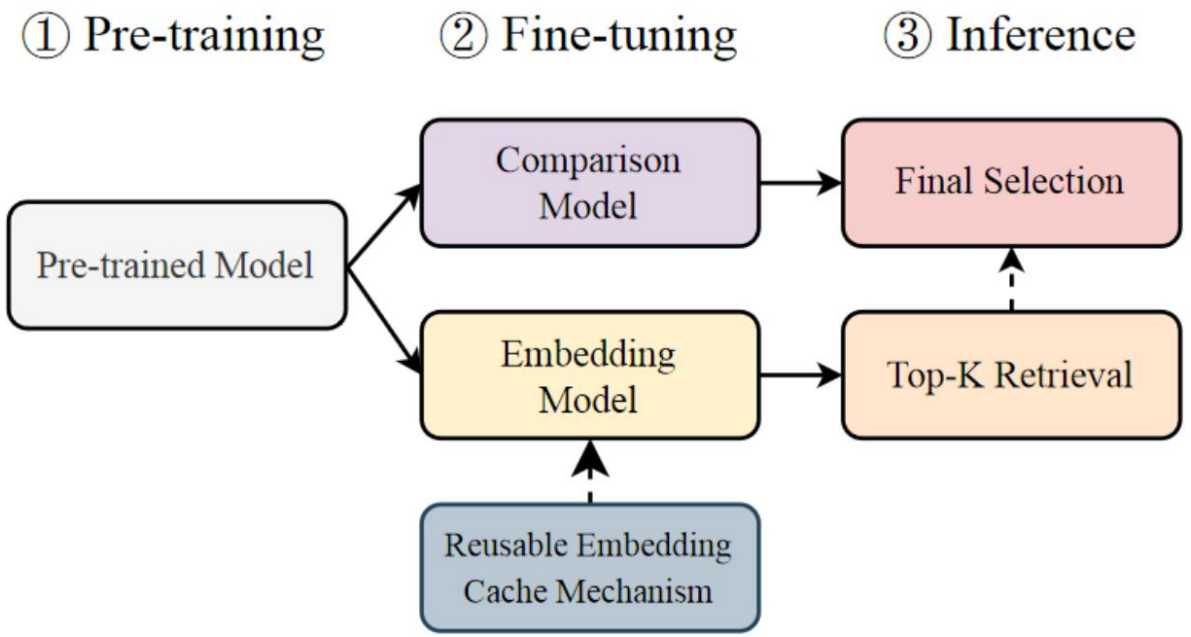
CEBin: A Cost-Effective Framework for Large-Scale Binary Code Similarity Detection

CEBin TIPO

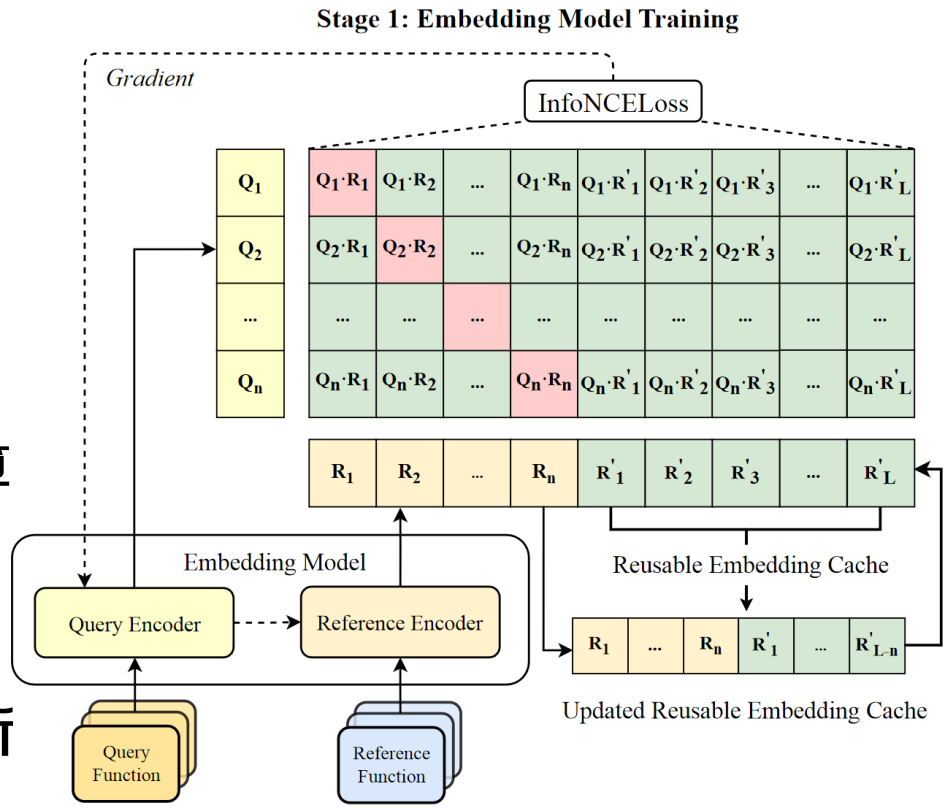
T	目标	兼顾准确性和效率，实现高效、准确的大规模二进制代码相似性检测
I	输入	查询二进制函数*1，待比较二进制函数*N
P	处理	1. 模型预训练：训练一个能够生成二进制代码嵌入表示的语言模型 2. 模型微调：分别微调一个嵌入模型和一个比较模型 3. 模型推理：先用嵌入模型进行Top-K检索，再用比较模型计算最终相似度
O	输出	基于相似度排名的待比较函数列表*1

P	问题	1.现有方法在准确性和效率之间的平衡较差 2.现有方法的训练目标与实际应用场景不匹配
C	条件	数据集中的二进制程序可被正常反汇编且未经过代码混淆技术处理
D	难点	1.如何在大规模检测中提升模型准确性的同时保持效率 2.如何在模型微调过程中不增加计算成本的情况下引入大量负样本
L	水平	ISSTA 2024 CCF A

- 算法原理图
  - 预训练阶段
    - 目标：训练一个能够生成二进制代码嵌入表示的语言模型
    - 使用jTrans作为基础模型，并且采用相同的预训练任务
  - 模型微调阶段
    - 嵌入模型训练（可重用嵌入缓存机制）
    - 比较模型训练
  - 模型推理阶段
    - 嵌入模型推理(Top-K Retrieval)
    - 比较模型推理(Final Selection)



- 嵌入模型训练
  - 问题：现有方法的训练目标与更具挑战性的现实世界场景不匹配
  - 挑战：直接采样大量负样本需要消耗大量的计算资源
  - 解决方法：提出可重用嵌入缓存机制，重用之前编码过的嵌入，降低计算开销
  - 可重用嵌入缓存机制(RECM)
    - 将嵌入模型拆分为查询编码器和参考编码器
    - 模型训练过程，输入n个查询函数$\{Q_i|i = 1, 2, \dots, n\}$以及与其相似的$n$个参考函数$\{R_i|i = 1, 2, \dots, n\}$
    - 对输入进行编码，并从嵌入缓存中检索$R'_{1:L}$，计算$Q_{1:n}$与$Concate(R_{1:n}, R'_{1:L})$的点积
    - 只有(Q_i, R_i)为正样本，其他均为负样本
    - 模型更新后，嵌入缓存会根据新编码$R_{1:n}$同步更新



- 嵌入模型训练
  - 模型参数更新
    - 查询编码器参数θ_q
      - 通过反向传播和梯度下降进行更新，使用InfoNCE损失函数

正样本对之间的相似度

$$\mathcal{L}_E = -\log \frac{\exp(f(Q_i, R_i))}{\sum_{j=1}^{N} \exp(f(Q_i, R_j))}$$

    - 参考编码器参数θ_r
      - 在编码期间被冻结，然后使用基于动量的方法进行更新

$$\theta_r \leftarrow m\theta_r + (1-m)\theta_q$$

动量系数

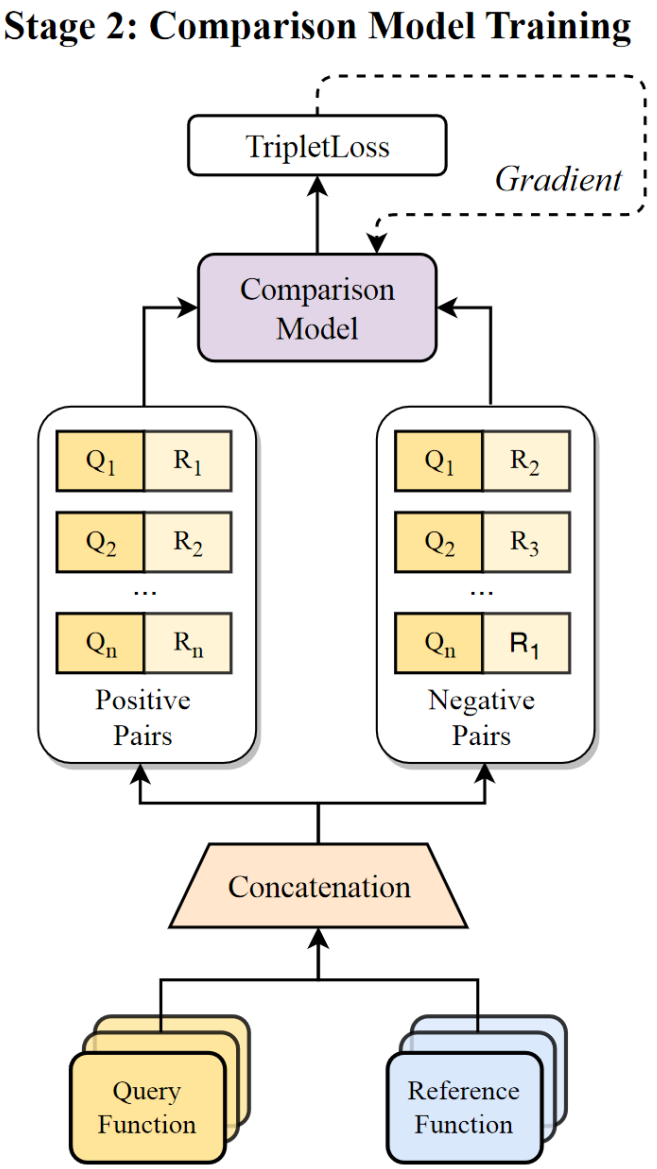
  - 通过集成RECM，在几乎不增加训练成本的情况下引入了大量负样本，提升了模型的判别能力

- 比较模型训练
  - 将模型输入改变为：一对二进制函数
  - 将模型输出改变为：给定函数对的相似度得分
  - 在模型训练过程中，输入n个查询函数$\{Q_i|i=1,2,\dots,n\}$以及与其相似的n个参考函数$\{R_i|i=1,2,\dots,n\}$
    - 正样本对由Q_i和R_i组成
    - 负样本对由Q_i和R_{i+1}组成
  - 训练使用的损失函数改变为三元组损失，以使模型能够更好地区分相似和不相似的函数对

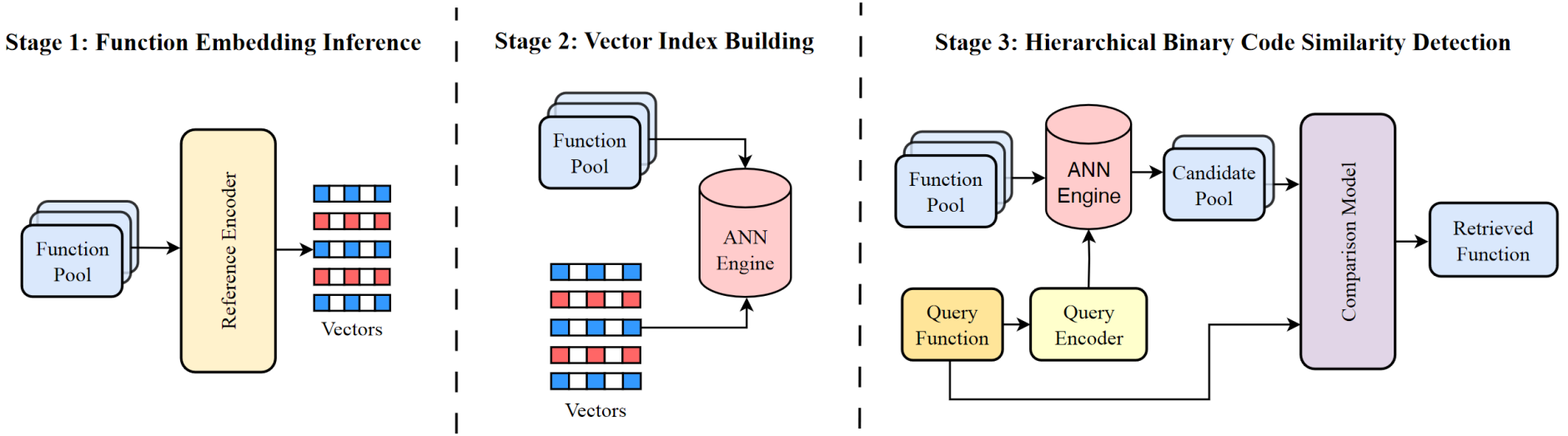
$$L_c = \max(0, \alpha - (D(Q_i, R_i) - D(Q_i, R_{i+1})))$$

正负样本对相似度的边距

Q_i和R_i的相似度得分



- 第1阶段：函数嵌入推理
  - 使用嵌入模型的参考编码器将所有待比较函数编码为嵌入向量
- 第2阶段：向量索引构建
  - 使用近似最近邻(ANN)算法为每个嵌入向量及其对应的函数构建一个向量索引
- 第3阶段：分层二进制代码相似性检测
  - 使用嵌入模型的查询编码器编码查询函数，利用构建的向量索引从函数池中检索出与其最相似的前K个函数，将它们与查询函数一起输入比较模型



- 数据集

数据集名称	BinaryCorp(BC)	Cisco Dataset-1	Trex Dataset
样本数量	3570K个二进制函数	8600K个二进制函数	1472K个二进制函数
编译器类型与版本	GCC v11.0.0	GCC v4.8.5/v5.5.0/v7.4.0/v9.2.1 Clang v3.5.2/v5.0.1/v7.0.0/v9.0.0	GCC v7.5.0
优化选项	O0/O1/O2/O3/Os	O0/O1/O2/O3/Os	O0/O1/O2/O3
目标架构	x64	x86/x64/ARM32/ARM64/MIPS32/MIPS64	

- 测试任务
  - XA：函数池中的待比较函数与查询函数的目标架构不同
  - XO：函数池中的待比较函数与查询函数的优化选项不同
  - XC：函数池中的待比较函数与查询函数的编译器类型、编译器版本和优化选项不同

- 对比方法
  - jTrans(2022)
  - Trex(2022)
  - OrderMatters(2020)
  - SAFE(2019)
  - Asm2Vec(2019)
  - GraphEmb(2019)
  - GNN和GMN(2019)
  - Gemini(2017)
  - Genius(2016)

- 评价指标
  - 平均倒数排名(MRR)：$MRR = \frac{1}{N}\sum_{1}^{N}\frac{1}{p_i}$

    N：查询样本总数　　p_i：第i个查询样本对应的相似函数的排名
  - Recall@1

$$Recall@K = \frac{\text{所有查询的前}K\text{个结果中正样本的总数}}{\text{正样本总数}}$$

每个查询函数在待比较函数池中有且仅有1个相似函数（正样本）

- 超参数设置
  - 可重用嵌入缓存大小L=8192
  - 动量系数m=0.99
  - 三元组损失边距α=0.25
  - 模型推理阶段K=50（实验表明嵌入模型的Recall@50接近1.0）

- 评估CEBin在BinaryCorp-3M上的表现（函数池大小为10000）
  - CEBin在不同任务中显著优于所有基线方法
  - CEBin-E代表嵌入模型，比较模型在更具挑战性的任务中带来的性能提升更为明显

	MRR							Recall@1						
Models	**O0,O3**	**O1,O3**	**O2,O3**	**O0,Os**	**O1,Os**	**O2,Os**	**Average**	**O0,O3**	**O1,O3**	**O2,O3**	**O0,Os**	**O1,Os**	**O2,Os**	**Average**
Genius	0.041	0.193	0.596	0.049	0.186	0.224	0.214	0.028	0.153	0.538	0.032	0.146	0.180	0.179
Gemini	0.037	0.161	0.416	0.049	0.133	0.195	0.165	0.024	0.122	0.367	0.030	0.099	0.151	0.132
GNN	0.048	0.197	0.643	0.061	0.187	0.214	0.225	0.036	0.155	0.592	0.041	0.146	0.175	0.191
GraphEmb	0.087	0.217	0.486	0.110	0.195	0.222	0.219	0.050	0.154	0.447	0.063	0.135	0.166	0.169
OrderMatters	0.062	0.319	0.600	0.075	0.260	0.233	0.263	0.040	0.248	0.535	0.040	0.178	0.158	0.200
SAFE	0.127	0.345	0.643	0.147	0.321	0.377	0.320	0.068	0.247	0.575	0.079	0.221	0.283	0.246
Asm2Vec	0.072	0.449	0.669	0.083	0.409	0.510	0.366	0.046	0.367	0.589	0.052	0.332	0.426	0.302
Trex	0.118	0.477	0.731	0.148	0.511	0.513	0.416	0.073	0.388	0.665	0.088	0.422	0.436	0.345
jTrans	0.475	0.663	0.731	0.539	0.665	0.664	0.623	0.376	0.580	0.661	0.443	0.586	0.585	0.571
CEBin-E	0.787	0.874	0.924	0.858	0.909	0.893	0.874	0.710	0.818	0.885	0.795	0.863	**0.842**	0.819
CEBin	**0.850**	**0.886**	**0.953**	**0.903**	**0.927**	**0.895**	**0.902**	**0.776**	**0.826**	**0.920**	**0.839**	**0.874**	0.834	**0.845**

Github

model	O0-O3	O1-O3	O2-O3	O0-Os	O1-Os	O2-Os
jTrans	0.6027	0.7514	0.8631	0.6233	0.7046	0.7870
jTrans(reported in your paper)	0.376	0.580	0.661	0.443	0.586	0.585
CEBin(in your paper)	0.776	0.826	0.920	0.839	0.874	0.845

实验设计　对比实验

- 评估CEBin在Cisco Dataset-1和Trex Dataset上的表现（函数池大小为10000）
  - CEBin在不同任务中显著优于所有基线方法
  - 比较模型带来的性能提升在这两个数据集上相对于BinaryCorp-3M更加明显

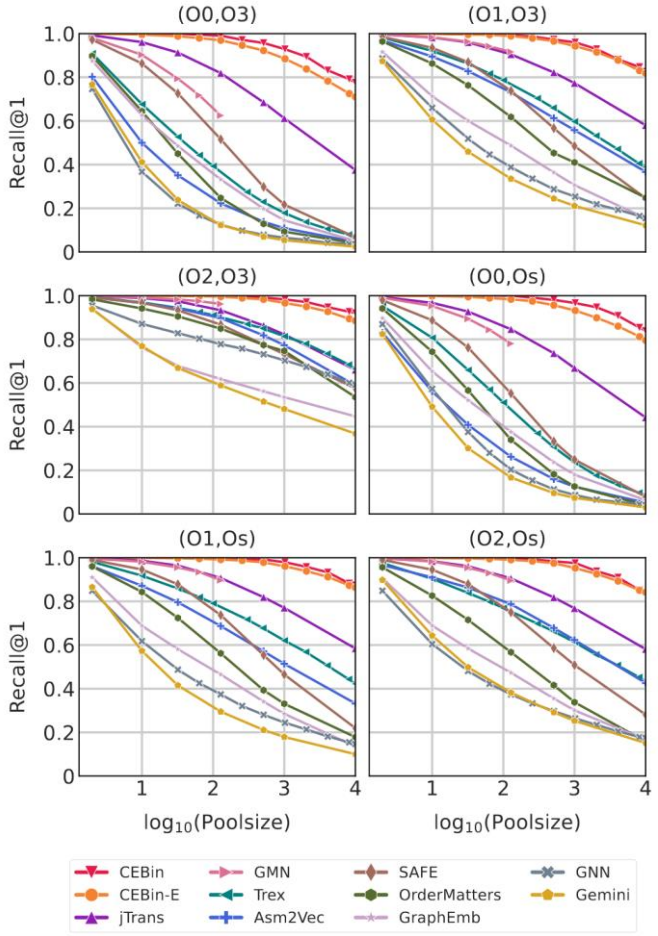
Cisco Dataset-1

	MRR						Recall@1					
Models	XA	XC	XO	XA+XO	XC+XO	XA+XC+XO	XA	XC	XO	XA+XO	XC+XO	XA+XC+XO
GNN	0.205	0.158	0.104	0.119	0.189	0.093	0.129	0.104	0.080	0.084	0.165	0.063
Trex	0.085	0.401	0.410	0.145	0.313	0.124	0.052	0.341	0.360	0.113	0.268	0.096
CEBin-E	0.760	0.907	0.859	0.817	0.866	0.766	0.692	0.871	0.816	0.766	0.823	0.706
CEBin	**0.977**	**0.992**	**0.973**	**0.978**	**0.984**	**0.961**	**0.968**	**0.988**	**0.963**	**0.969**	**0.977**	**0.946**

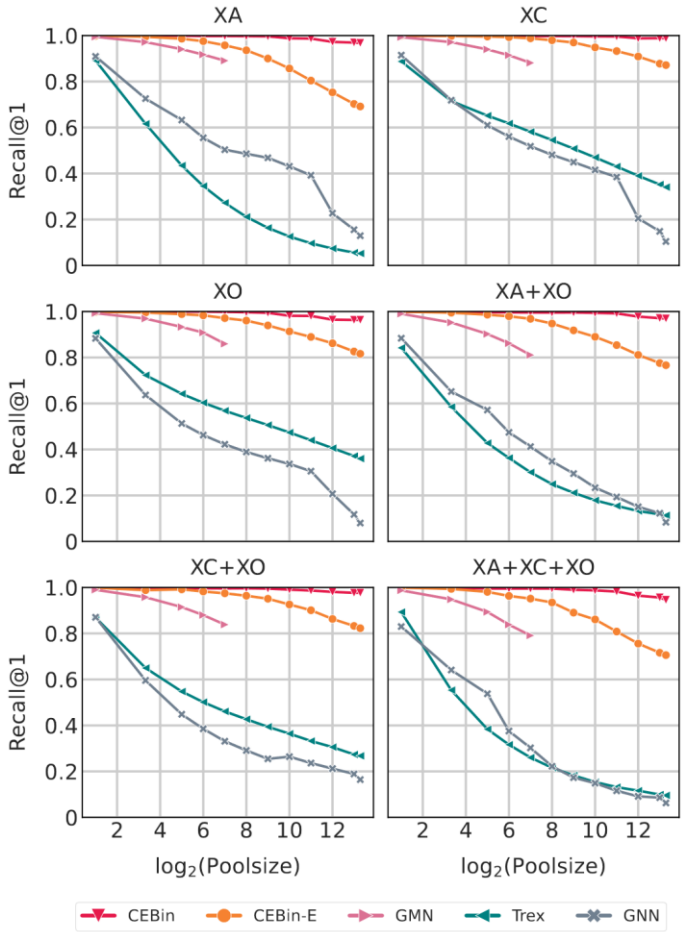
Trex Dataset

	MRR			Recall@1		
Models	XA	XO	XA+XO	XA	XO	XA+XO
Trex	0.142	0.218	0.175	0.065	0.123	0.107
GNN	0.163	0.148	0.151	0.145	0.102	0.109
CEBin-E	0.612	0.646	0.576	0.509	0.553	0.474
CEBin	**0.911**	**0.933**	**0.911**	**0.882**	**0.906**	**0.870**

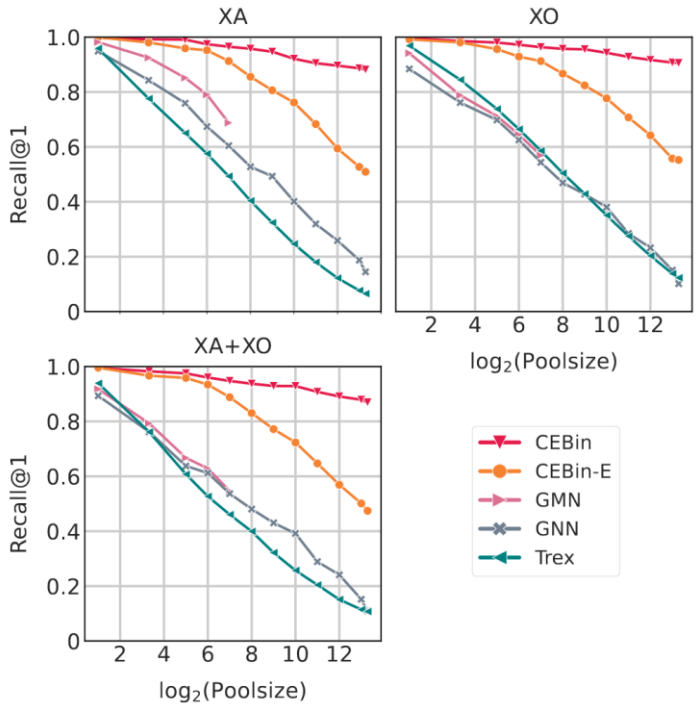
- 评估CEBin在不同函数池大小下的表现
  - 随着池大小的增大，CEBin的性能具有更高的稳定性，表现出更好的实用性



BinaryCorp-3M

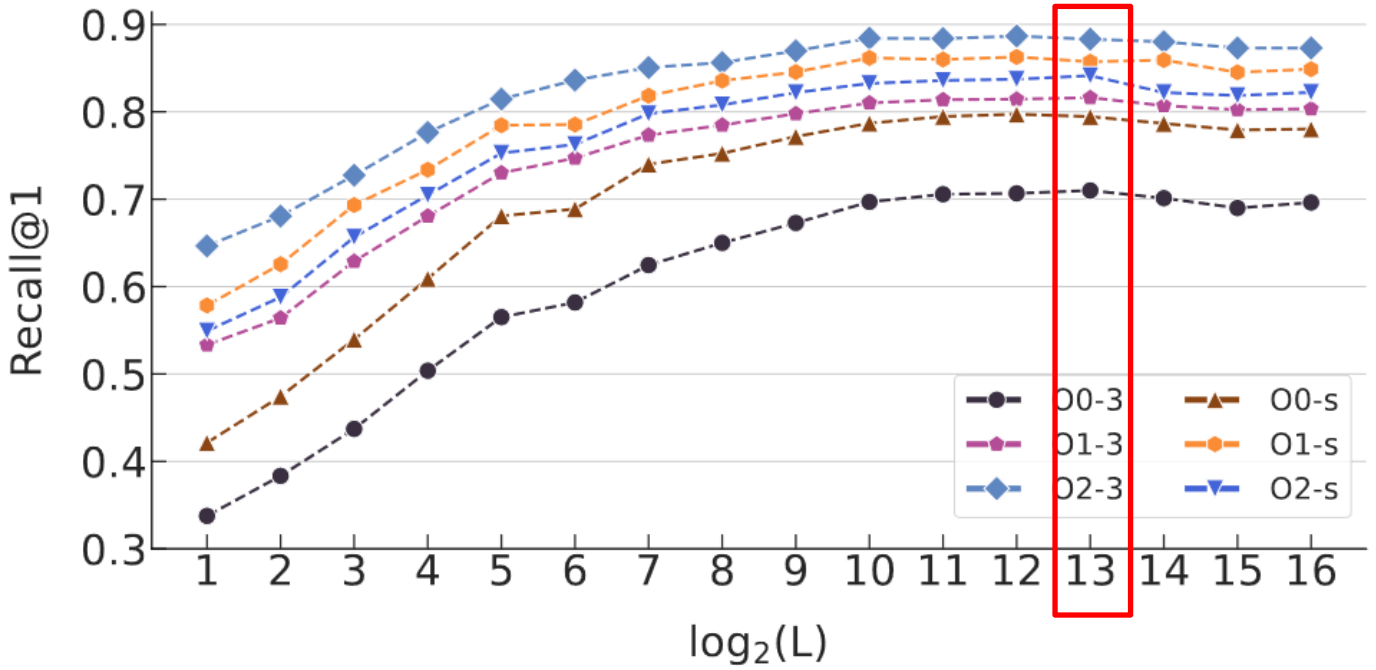


Cisco Dataset-1



Trex Dataset

- 研究训练过程中负样本数量的影响
  - 数据集：BinaryCorp-3M
  - 函数池大小：10000
  - 只改变训练过程中使用的RECM的缓存大小L，范围从2到65536
  - 实验结果表明，增加负样本的数量显著提升了嵌入模型的性能



实验设计　运行时间效率实验

- 评估CEBin在不同池大小下的推理成本（以秒为单位）
  - 实验设置
    - 从获取目标函数的嵌入向量开始，到将其与函数池中的每一个函数进行相似度比较，最终获得相似度结果的整体计算成本

	Poolsize			
Model	100	10,000	1,000,000	4,000,000
GNN	0.004	0.004	0.012	0.037
Trex	0.018	0.018	0.050	0.182
CEBin-E	0.807	0.814	0.887	1.034
CEBin	2.717	2.772	2.898	3.105

  - 实验结论
    - CEBin的推理成本相对高于基线方法
    - 随着池大小的增加，CEBin的推理成本并没有迅速上升

算法总结 CEBin

- 算法贡献
  - 融合基于嵌入和基于比较的方法：使用嵌入模型快速缩小候选相似函数的范围，利用比较模型进行精确的相似度计算
    - 显著提升了模型准确性，同时有效的控制了计算开销
  - 提出可重用嵌入缓存机制：重复使用已编码的负样本嵌入
    - 在不显著增加计算成本的前提下引入大量负样本，大幅提升了模型性能

- 算法不足
  - 反汇编工具(如IDA Pro等)生成的汇编代码质量直接影响方法性能
  - 模型仅编码指令上下文语义和控制流信息，忽略了对指令数据流信息和其他方面的分析

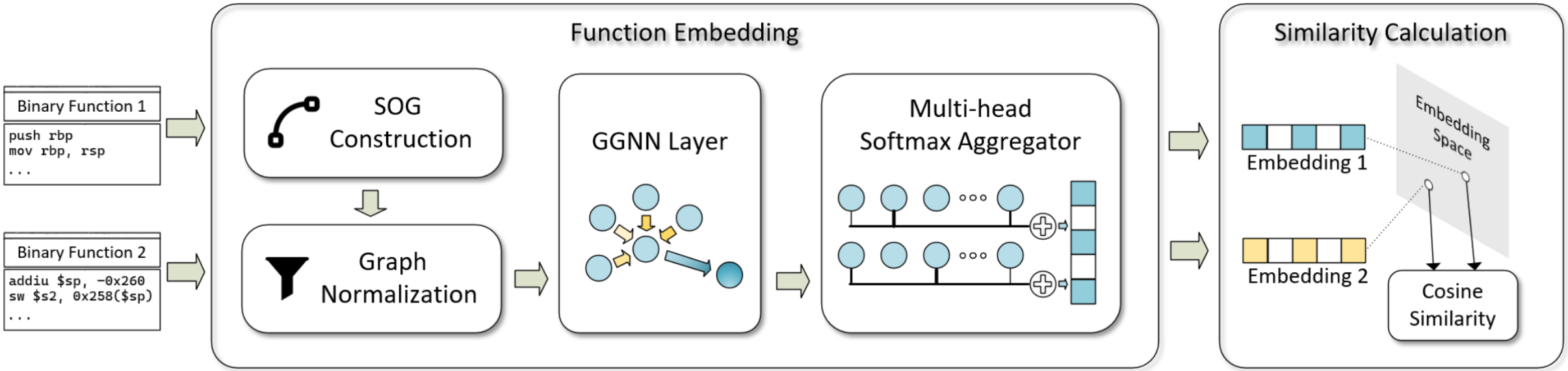
Code is not Natural Language: Unlock the Power of Semantics-Oriented Graph Representation for Binary Code Similarity Detection

HermesSim TIPO

T	目标	提取能够揭示指令间关系、指令内部结构、隐式调用约定，有效排除语义无关元素的二进制函数特征表示
I	输入	查询二进制函数*1，待比较二进制函数*N
P	处理	1. 语义导向图构建：构建输入二进制函数的语义导向图(SOG) 2. 图规范化处理：将每个节点和边转换为可学习的嵌入向量 3. 节点嵌入生成：使用GGNN聚合每个节点的邻居结构，生成节点嵌入 4. 图嵌入生成：使用多头Softmax聚合器将节点嵌入聚合为图嵌入 5. 相似度计算：计算嵌入向量的余弦相似度，作为函数的相似性度量
O	输出	基于相似度排名的待比较函数列表*1

P	问题	现有基于汇编指令序列的方法忽略了指令间关系，且未有效排除语义无关元素；现有基于代码结构特征的方法忽略了指令内部结构和隐式约定
C	条件	数据集中的二进制程序可被正常反汇编且未经过代码混淆技术处理
D	难点	如何尽可能全面地提取二进制函数的特征表示
L	水平	USENIX Security 2024 CCF A

- 算法原理图
  - 语义导向图构建：构建输入二进制函数的语义导向图(SOG)
  - 图规范化处理：将每个节点和边转换为可学习的嵌入向量
  - 节点嵌入生成：使用GGNN聚合每个节点的邻居结构，生成节点嵌入
  - 图嵌入生成：使用多头Softmax聚合器将节点嵌入聚合为图嵌入
  - 相似度计算：计算嵌入向量的余弦相似度，作为函数的相似性度量

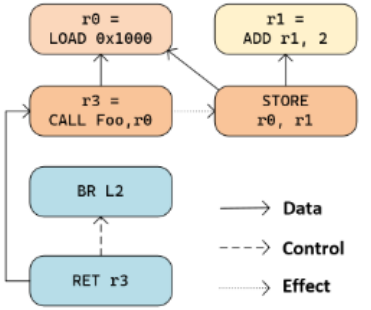


语义导向图构建

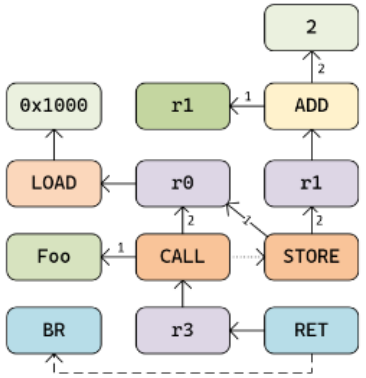
- 原始指令序列→基于指令的语义完备图(ISCG)
  - 以指令作为节点，指令间关系作为边
  - 在数据流图中添加额外的控制流和效果流边
  - 携带与原始指令序列完全相同的语义信息
- ISCG→基于token的语义完备图(TSCG)
  - 将指令拆分为token来揭示指令内部结构
  - 将指令内部结构解释为另一种数据流关系
  - 在数据流边上标注操作数的位置
- TSCG→语义导向图(SOG)
  - 移除在指令之间缓存数据的临时存储节点（例如寄存器或栈帧区域），并直接连接它们的输入和输出
  - 隐式约定：函数调用约定、栈帧等

```
1 L1:
2   r0 = LOAD 0x1000
3   r1 = ADD r1, 2
4   STORE r0, r1
5   r3 = CALL Foo, r0
6   BR L2
7 L2:
8   RET r3
```

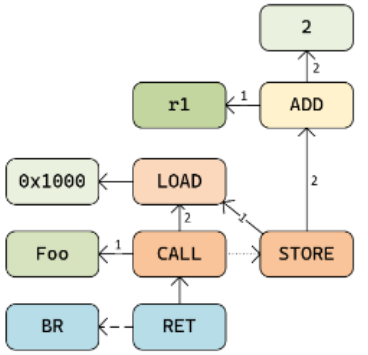
(a) Linear Representation in the toy IR



(b) ISCG: Instruction-based Semantics-Complete Graph



(c) TSCG: Token-based Semantics-Complete Graph



(d) SOG: Semantics-Oriented Graph

图规范化处理和节点嵌入生成

- 图规范化处理
  - 节点：将节点token分为三种类型，即指令token、整数字面量和寄存器token
    - 对于指令token，从训练数据集中出现的59个token中选择最常见的55个加入词汇表
    - 对于整数字面量和寄存器token，仅将那些在训练数据集中经常出现的加入词汇表
  - 边：分别将类型属性和位置属性转换为两个嵌入，将它们相加以形成最终的边嵌入
- 节点嵌入生成
  - 使用双向GGNN层捕获每个节点的邻居结构
    - 节点信息的聚合和更新公式

h_u^l表示第l层节点u的嵌入向量　　$e_{v,u}$表示从节点v到节点u的边的嵌入

$$\mathbf{m\_in}_u^l = \sum_{v \in In(u)} \mathrm{MLP}_{in}\left(\mathbf{h}_u^l \mid \mathbf{h}_v^l \mid \mathbf{e}_{v,u} \right)$$

In(u)表示节点u的入边的邻居集合

$$\mathbf{m\_out}_u^l = \sum_{v \in Out(u)} \mathrm{MLP}_{out}\left(\mathbf{h}_v^l \mid \mathbf{h}_u^l \mid \mathbf{e}_{u,v} \right)$$

$$\mathbf{h}_u^{l+1} = \mathrm{GRU}\left(\mathbf{h}_u^l, \mathbf{m\_in}_u^l + \mathbf{m\_out}_u^l \right)$$

- 节点嵌入生成模块的输出是图中所有节点嵌入向量的集合$\{h_u^r\}$，本模块使用多头Softmax聚合器将所有节点嵌入聚合为图嵌入
  - 引入多个头部来扩展Softmax聚合器
  - 每个头部的输入通过线性层和ReLU层转换为不同的表示空间

Softmax聚合器

可学习参数　节点u的嵌入向量

$$\mathrm{SoftmaxAggr}(\mathcal{H}|\mathbf{t}) = \sum_{\mathbf{h}_u^r \in \mathcal{H}} \frac{\exp(\mathbf{t} \odot \mathbf{h}_u^r)}{\sum_{\mathbf{h}_v^r \in \mathcal{H}} \exp(\mathbf{t} \odot \mathbf{h}_v^r)} \odot \mathbf{h}_u^r$$

所有节点嵌入向量的集合

头部的索引

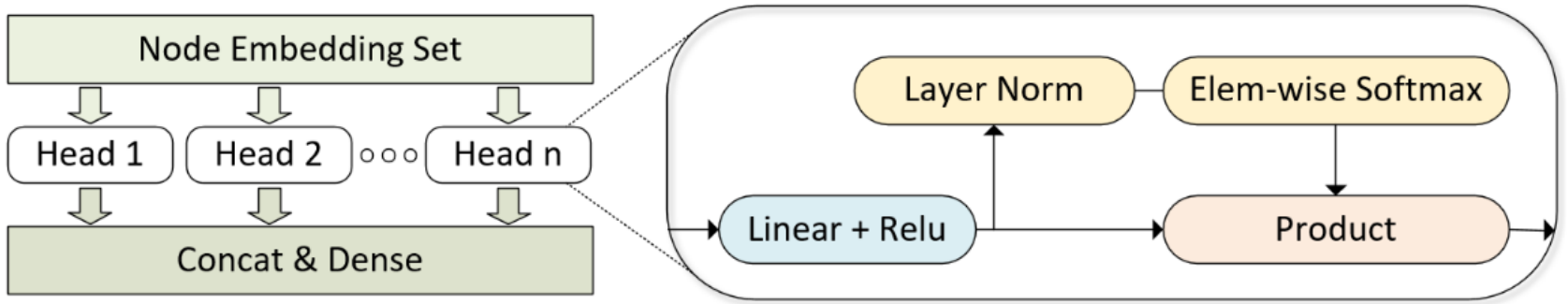
$$\mathbf{x}_u^k = \mathrm{Relu}(\mathrm{Linear}_k(\mathbf{h}_u^r))$$

图中所有节点的集合

$$\mathbf{g}^k = \mathrm{SoftmaxAggr}\left(\left\{\frac{\mathbf{x}_u^k - \mathrm{E}[\mathbf{x}_u^k]}{\sqrt{\mathrm{Var}[\mathbf{x}_u^k] + \varepsilon}} | u \in \mathcal{V}\right\} | \mathbf{t}^k\right)$$

$$\mathbf{g} = \mathrm{Linear}(g^1|g^2|\ldots|g^h)$$

头部的总数



- 数据集：Cisco Dataset-1
  - 样本数量
    - 训练集包含257K个二进制函数
    - 验证集包含13K个二进制函数
    - 测试集包含522K个二进制函数
  - 编译环境
    - 3个目标架构(x86/ARM/MIPS)
    - 2个位数(32/64)
    - 5个优化选项(O0/O1/O2/O3/Os)
    - 8个编译器类型与版本组合(GCC-4.8.5/5.5.0/7.4.0/9.2.1、Clang-3.5.2/5.0.1/7.0.0/9.0.0)
  - 筛选出一个仅包含x64架构函数的子数据集
    - 仅支持单架构的对比方法在x64子数据集上进行训练
    - 其他支持多架构的方法在整个数据集上进行训练

数据集名称	Cisco Dataset-1
样本数量	8600K个二进制函数
编译器类型与版本	GCC v4.8.5/v5.5.0/v7.4.0/v9.2.1 Clang v3.5.2/v5.0.1/v7.0.0/v9.0.0
优化选项	O0/O1/O2/O3/Os
目标架构	x86/x64/ARM32/ARM64/MIPS32/MIPS64

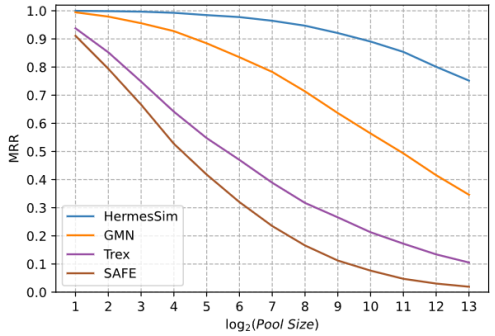
- 测试子任务（对于每个任务，随机选择1000个查询函数进行测试）
  - XA：函数池中的待比较函数与查询函数的目标架构不同
  - XO：函数池中的待比较函数与查询函数的优化选项不同
  - XC：函数池中的待比较函数与查询函数的编译器类型、编译器版本和优化选项不同
  - XM：函数池中的待比较函数与查询函数的目标架构、位数、编译器类型、编译器版本和优化选项均不同
- 对比方法
  - jTrans(2022)、Trex(2022)、SAFE(2019)、Asm2Vec(2019)、GMN(2019)
- 评价指标
  - 平均倒数排名(MRR)：$MRR = \frac{1}{N}\sum_{1}^{N}\frac{1}{p_i}$
  - Recall@1：返回的第1个结果是正样本的数量占总样本数的比例

- 实验结论
  - HermesSim在各项设置中都显著优于所有基线方法
  - 基于NLP的方法(SAFE/Asm2Vec/Trex/jTrans)在XA任务中表现不佳
  - GMN在XA任务中显著优于Trex，而在其他任务中表现相近

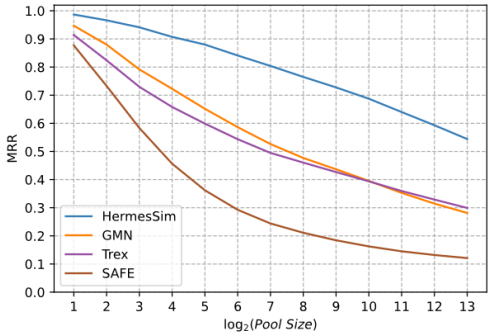
Recall@1/MRR

	XA	XO	XC	XM		x64-XO	x64-XC		N Params
poolsize		100		100	10000	100	100	10000	
SAFE	13.4/26.4	21.1/27.5	20.1/27.6	9.9/18.9	1.4/2.32	18.4/26.2	17.2/24.9	8.1/9.5	8.93M
Asm2Vec	-	24.6/30.1	25.8/31.7	-	-	31.8/37.7	29.0/35.0	13.5/16.6	-
Trex	31.2/42.1	46.8/53.1	45.4/52.5	24.4/34.4	8.6/11.1	51.5/57.7	45.9/53.2	26.2/30.1	61.8M
GMN	72.6/81.7	50.3/58.1	52.3/59.8	44.7/53.7	10.5/15.9	52.4/60.2	48.0/56.2	21.9/26.7	60.5K
jTrans	-	-	-	-	-	66.9/76.0	65.0/73.8	31.4/37.4	87.9M
HermesSim	**95.5/97.5**	**81.0/85.3**	**78.0/83.2**	**74.5/80.2**	**43.8/50.8**	**81.9/86.0**	**75.6/80.7**	**48.1/54.6**	388K

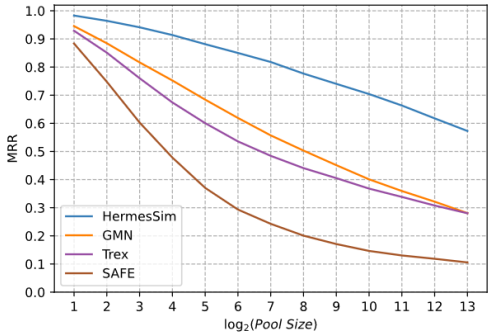
- 实验结论
  - 随着函数池大小从2增加到8192，HermesSim的性能表现出更高的稳定性
  - 在函数池大小较大时，GMN在XO和XC任务中的表现不如Trex
    - GMN仅通过CFG编码控制流信息



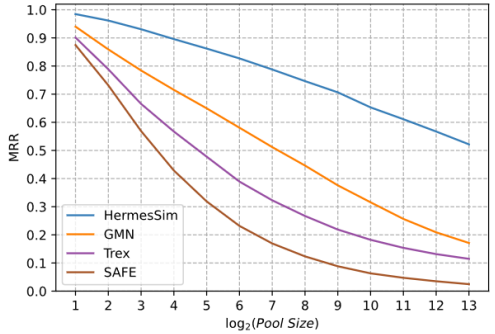
(a) XA



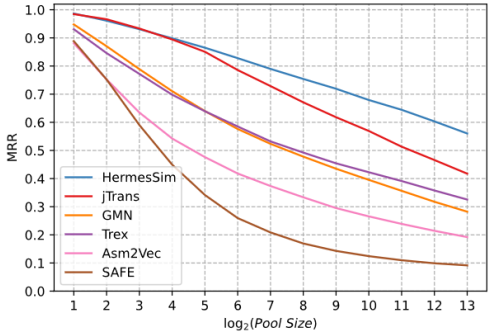
(b) XO



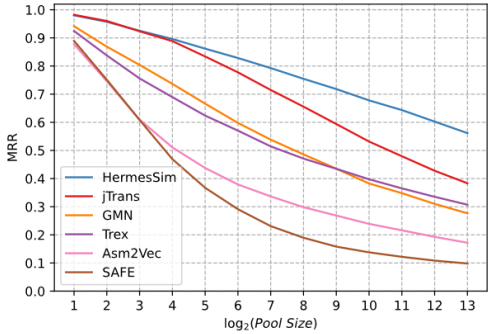
(c) XC



(d) XM



(e) x64-XO



(f) x64-XC

- SOG的前置表示方法
  - P-DFG：以指令为节点，指令之间的def-use关系作为边（数据流边）
  - P-CDFG：在P-DFG的基础上，增加控制流边
  - P-ISCG：在P-CDFG的基础上，增加效果流边
  - P-TSCG：在与P-ISCG的基础上，额外揭示指令内部结构

Recall@1/MRR

		XA	XO	XC	XM		x64-XO	x64-XC	
poolsize			100		100	10000	100	100	10000
MSoft	CFG-OPC200	92.8/95.6	67.7/73.9	67.1/73.2	62.9/69.3	32.0/38.7	69.9/75.7	64.2/70.3	36.8/42.9
	CFG-PalmTree	-	-	-	-	-	70.8/76.4	66.1/72.3	36.3/43.0
	CFG-HBMP	94.3/96.6	69.5/75.8	68.9/74.9	65.3/71.9	33.7/40.6	72.0/77.4	67.2/73.0	39.0/45.5
MSoft	P-DFG	93.4/96.2	76.4/81.6	74.1/79.8	69.7/76.0	37.4/44.7	76.3/81.6	69.9/75.7	42.8/49.3
	P-CDFG	94.6/96.9	77.2/82.5	75.3/80.8	71.1/77.3	38.6/45.9	77.2/82.4	72.4/77.8	43.6/50.5
	P-ISCG	95.1/97.2	78.0/82.9	74.8/80.3	71.3/77.2	40.1/47.2	78.1/82.6	71.9/77.1	44.2/50.9
	P-TSCG	95.5/97.4	79.2/83.6	76.2/81.7	73.2/78.9	41.3/48.8	79.1/83.7	73.5/78.7	46.4/52.9
Set2Set	P-SOG	89.5/93.0	67.2/72.2	64.5/69.9	61.2/66.9	29.8/36.6	69.2/73.7	60.7/66.1	37.3/43.1
Softmax		90.9/94.3	72.9/78.1	71.0/76.6	64.3/71.2	32.6/39.4	74.0/78.7	66.9/72.6	41.2/47.2
Gated		93.5/96.1	75.6/80.3	72.5/77.6	68.3/74.2	38.4/44.9	76.3/80.7	69.0/74.3	43.9/49.9
MSoft	P-SOG	**95.5/97.5**	**81.0/85.3**	**78.0/83.2**	**74.5/80.2**	**43.8/50.8**	**81.9/86.0**	**75.6/80.7**	**48.1/54.6**

- 实验结果
  - 与其他基线表示相比，SOG在所有子任务（除XA）上的性能均有显著提升
    - XA对基于函数结构特征的方法来说是最容易的子任务
  - 使用多头Softmax聚合器的性能显著优于其他基线聚合器

Recall@1/MRR

		XA	XO	XC	XM		x64-XO	x64-XC	
poolsize			100		100	10000	100	100	10000
MSoft	CFG-OPC200	92.8/95.6	67.7/73.9	67.1/73.2	62.9/69.3	32.0/38.7	69.9/75.7	64.2/70.3	36.8/42.9
	CFG-PalmTree	-	-	-	-	-	70.8/76.4	66.1/72.3	36.3/43.0
	CFG-HBMP	94.3/96.6	69.5/75.8	68.9/74.9	65.3/71.9	33.7/40.6	72.0/77.4	67.2/73.0	39.0/45.5
MSoft	P-DFG	93.4/96.2	76.4/81.6	74.1/79.8	69.7/76.0	37.4/44.7	76.3/81.6	69.9/75.7	42.8/49.3
	P-CDFG	94.6/96.9	77.2/82.5	75.3/80.8	71.1/77.3	38.6/45.9	77.2/82.4	72.4/77.8	43.6/50.5
	P-ISCG	95.1/97.2	78.0/82.9	74.8/80.3	71.3/77.2	40.1/47.2	78.1/82.6	71.9/77.1	44.2/50.9
	P-TSCG	95.5/97.4	79.2/83.6	76.2/81.7	73.2/78.9	41.3/48.8	79.1/83.7	73.5/78.7	46.4/52.9
Set2Set	P-SOG	89.5/93.0	67.2/72.2	64.5/69.9	61.2/66.9	29.8/36.6	69.2/73.7	60.7/66.1	37.3/43.1
Softmax		90.9/94.3	72.9/78.1	71.0/76.6	64.3/71.2	32.6/39.4	74.0/78.7	66.9/72.6	41.2/47.2
Gated		93.5/96.1	75.6/80.3	72.5/77.6	68.3/74.2	38.4/44.9	76.3/80.7	69.0/74.3	43.9/49.9
MSoft	P-SOG	**95.5/97.5**	**81.0/85.3**	**78.0/83.2**	**74.5/80.2**	**43.8/50.8**	**81.9/86.0**	**75.6/80.7**	**48.1/54.6**

- 实验结果
  - CFG在跨架构场景中具有优势，而DFG在跨优化和跨编译器任务中表现更优
  - P-DFG、P-CDFG、P-ISCG和P-TSCG的RECALL@1和MRR分数逐步提升，表明揭示控制流关系、效果流关系以及指令内部结构确实有利于提升方法性能
  - SOG优于TSCG，说明在表示中保留与语义无关的元素可能会导致模型性能下降

Recall@1/MRR

		XA	XO	XC	XM		x64-XO	x64-XC	
poolsize			100		100	10000	100	100	10000
MSoft	CFG-OPC200	92.8/95.6	67.7/73.9	67.1/73.2	62.9/69.3	32.0/38.7	69.9/75.7	64.2/70.3	36.8/42.9
	CFG-PalmTree	-	-	-	-	-	70.8/76.4	66.1/72.3	36.3/43.0
	CFG-HBMP	94.3/96.6	69.5/75.8	68.9/74.9	65.3/71.9	33.7/40.6	72.0/77.4	67.2/73.0	39.0/45.5
MSoft	P-DFG	93.4/96.2	76.4/81.6	74.1/79.8	69.7/76.0	37.4/44.7	76.3/81.6	69.9/75.7	42.8/49.3
	P-CDFG	94.6/96.9	77.2/82.5	75.3/80.8	71.1/77.3	38.6/45.9	77.2/82.4	72.4/77.8	43.6/50.5
	P-ISCG	95.1/97.2	78.0/82.9	74.8/80.3	71.3/77.2	40.1/47.2	78.1/82.6	71.9/77.1	44.2/50.9
	P-TSCG	95.5/97.4	79.2/83.6	76.2/81.7	73.2/78.9	41.3/48.8	79.1/83.7	73.5/78.7	46.4/52.9
Set2Set	P-SOG	89.5/93.0	67.2/72.2	64.5/69.9	61.2/66.9	29.8/36.6	69.2/73.7	60.7/66.1	37.3/43.1
Softmax		90.9/94.3	72.9/78.1	71.0/76.6	64.3/71.2	32.6/39.4	74.0/78.7	66.9/72.6	41.2/47.2
Gated		93.5/96.1	75.6/80.3	72.5/77.6	68.3/74.2	38.4/44.9	76.3/80.7	69.0/74.3	43.9/49.9
MSoft	P-SOG	**95.5/97.5**	**81.0/85.3**	**78.0/83.2**	**74.5/80.2**	**43.8/50.8**	**81.9/86.0**	**75.6/80.7**	**48.1/54.6**

- 在测试过程中，HermesSim每处理1000个查询函数的平均时间成本

Training	Lifting	Inferring	Searching
62mins (20 epochs)	3.20s	0.35s	1.50ms

  - 实验环境
    - Intel(R) Xeon(R) Silver 4214 @2.20GHz CPU（单进程）
    - NVIDIA RTX3080 GPU
  - 提升时间是将二进制函数从线性表示提升为SOG的时间
  - 推理时间是将SOG编码为嵌入向量的时间
  - 搜索时间是将查询函数嵌入与其他函数嵌入进行比较的时间

算法总结 HermesSim

- 算法贡献
  - 提出了一种全新的二进制代码表示方法，称为语义导向图(SOG)
    - SOG能够揭示指令间关系、指令内部结构、隐式调用约定，并有效排除语义无关元素
  - 设计了一种新颖的多头Softmax聚合器，用于正确聚合 SOG的多个方面
    - 该模块显著提升了系统的性能

- 算法不足
  - 提取SOG表示的准确性严重受限于程序分析算法的性能
  - 效果流分析仅限于内存效果模型，未考虑I/O效果模型，即与I/O设备通信导致的状态改变，无法充分反映I/O操作对程序执行逻辑的潜在影响

特点总结与未来展望

特点总结与未来展望

- 特点总结
  - CEBin
    - 从数据和模型角度出发，提出可重用嵌入缓存机制，结合嵌入模型和比较模型
      - 在保持较高准确率的同时，显著降低了计算成本
  - HermesSim
    - 从函数特征表示角度出发，提出一种全新的二进制代码表示方法，称为语义导向图
      - 语义导向图揭示了指令间关系、指令内部结构、隐式调用约定，并有效排除了语义无关元素，提升了模型在二进制代码相似性检测中的表现
- 未来发展
  - 研究如何更加充分地利用函数内部丰富的各类信息，进一步挖掘和利用函数内部的各种语义信息、控制流、数据流以及寄存器操作等细节
  - 结合更多的二进制程序分析技术与领域相关知识针对性地提升方法的性能

参考文献

[1] Wang H, Gao Z, Zhang C, et al. CEBin: A Cost-Effective Framework for Large-Scale Binary Code Similarity Detection[C]. Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. 2024: 149-161.

[2] He H, Lin X, Weng Z, et al. Code is not natural language: Unlock the power of semantics-oriented graph representation for binary code similarity detection[C]. 33rd USENIX Security Symposium (USENIX Security 24), PHILADELPHIA, PA. 2024.

[3] 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. New York, NY: ACM, 2022: 1-13.

[4] He K, Fan H, Wu Y, et al. Momentum contrast for unsupervised visual representation learning[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020: 9729-9738.

道德经

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！