CODEE: A TENSOR EMBEDDING SCHEME FOR BINARY CODE SEARCH

Jia Yang[1], Cai Fu[1], Xiao-Yang Liu[2], Heng Yin[3],Pan Zhou[1]

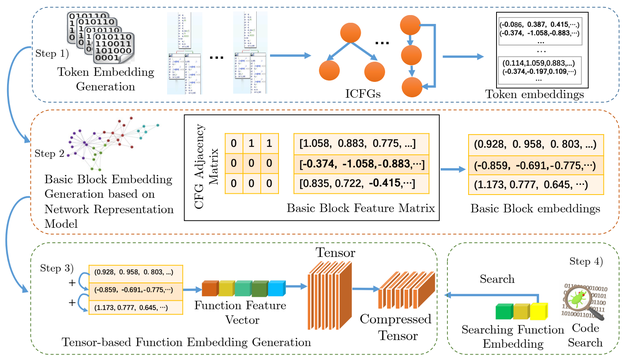
[1]Huazhong University of Science and Technology, Wuhan, China;
[2]Columbia University, New York, USA; [3]University of California, Riverside, USA.

Introduction

Binary code search has recently emerged as a popular topic for solving software security problems, such as finding clone code injection, detecting software plagiarism. Given a target binary function, the binary code search retrieves top-K similar functions in the repository. Searching binary code is particularly challenging due to large variations of compiler tool-chains and options and CPU architectures, as well as thousands of binary codes. In this work, we present an unsupervised tensor embedding scheme, **Codee**, to carry out similar code search efficiently and accurately at the binary function level. Compared with other cross-platform and cross-optimization-level code search schemes, experimental results show that our scheme achieves higher average search accuracy, shorter feature vectors, and faster feature generation performance using four datasets.

Scheme Overview

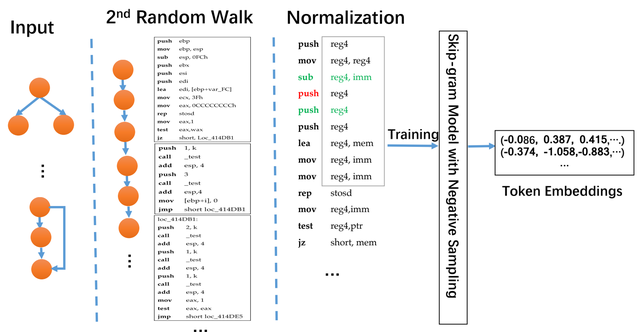
Given a binary function, the code search task is to find similar functions in the repository, and achieve high accuracy and high efficiency. To quickly and accurately search binary codes (e.g., a function), we seek to map each function into a low-dimensional feature vector. Specifically, our scheme follows four steps: **(1) Token embedding generation based on NLP technique**; **(2) Basic block embedding generation based on network representation technique**; **(3) Function embedding based on tensor computation**; **(4) Binary code search using LSH**.



Token Embedding Generation

Given all programs in a repository, we first build an NLP-based learning model to obtain all token embeddings. Specifically, we have the following steps:

1. Generate the token sequences by using the node2vecWalk random walks method.
2. Normalize the serialized codes by defining some specific notations.
3. Train the skip-gram model with negative sampling to obtain the token embedding.



Basic Black Embedding

We combine control-flow graph structural information and semantic information of the tokens to generate high-quality basic block embeddings by loss function:

$$\min_{\mathbf{C}} \; L = \sum_{i=1}^{n} ||\mathbf{S}_i - \mathbf{C}^H \mathbf{C}_i||_2^2 + \lambda(- \sum_{j \in N(i)} a_{ij} \log p(j|i)).$$

Basic Block Embedding

Input: adjacent matrix of CFG: $\mathbf{A}$, basic block feature matrix: $\mathbf{B}$, affinity matrix: $\mathbf{S}$.
Output: Basic block embedding matrix: $\mathbf{C}^T$.
Initiation: $\mathbf{C}^0 \leftarrow$ first d right singular vectors of $\mathbf{B}$.
$\mathbf{Z}^0 = \mathbf{0}$, $\mathbf{H}^0 = \mathbf{C}^0$,
for $t = 0, 1, 2, ..., T-1$:
  for $i = 1, 2, ..., n$:
    Update $\mathbf{C}_i^{t+1}$ using Eq. (1),
  Obtain $\mathbf{C}^{t+1}$, where $\mathbf{C}_i^{t+1}$ is i-th column of $\mathbf{C}^{t+1}$,
  for $i = 1, 2, ..., n$:
    Update $\mathbf{H}_i^{t+1}$ using Eq. (2),
  Obtain $\mathbf{H}^{t+1}$, where $\mathbf{H}_i^{t+1}$ is i-th column of $\mathbf{H}^{t+1}$,
$\mathbf{Z}^{t+1} = \mathbf{Z}^t + (\mathbf{C}^{t+1} - \mathbf{H}^{t+1})$.

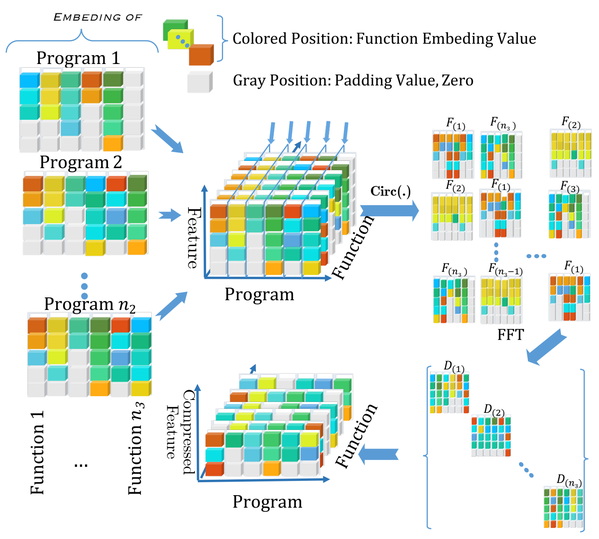
$$\mathbf{C}_i^{t+1} = \frac{2\mathbf{H}^t\mathbf{S}_i + \lambda \sum_{j \in N(i)} a_{ij}\mathbf{H}_j^t + \rho(\mathbf{H}_i^t - \mathbf{Z}_i^t)}{2\mathbf{H}^t(\mathbf{H}^t)^H + \rho\mathbf{I}} - \frac{\lambda \sum_{j \in N(i)} a_{ij} \frac{\sum_{l \in V} \mathbf{H}_l^t \exp(\mathbf{H}_l^{tH}\mathbf{C}_i^t)}{\sum_{l \in V} \exp(\mathbf{H}_l^{tH}\mathbf{C}_i^t)}}{2\mathbf{H}^t(\mathbf{H}^t)^H + \rho\mathbf{I}}. \quad (1)$$

$$\mathbf{H}_i^{t+1} = \frac{2\mathbf{C}^{t+1}\mathbf{S}_i + \lambda \sum_{(i,j) \in E} a_{ij}\mathbf{H}_j^t + \rho(\mathbf{H}_i^t - \mathbf{Z}_i^t)}{2\mathbf{C}^{t+1}(\mathbf{C}^{t+1})^H + \rho\mathbf{I}} - \frac{\lambda \sum_{(i,j) \in E} a_{ij} \frac{\sum_{l \in V} \mathbf{C}_l^{t+1} \exp((\mathbf{C}_l^{t+1})^H\mathbf{H}_i^t)}{\sum_{l \in V} \exp((\mathbf{C}_l^{t+1})^H\mathbf{H}_i^t)}}{2\mathbf{C}^{t+1}(\mathbf{C}^{t+1})^H + \rho\mathbf{I}}. \quad (2)$$

Function Embedding Generation

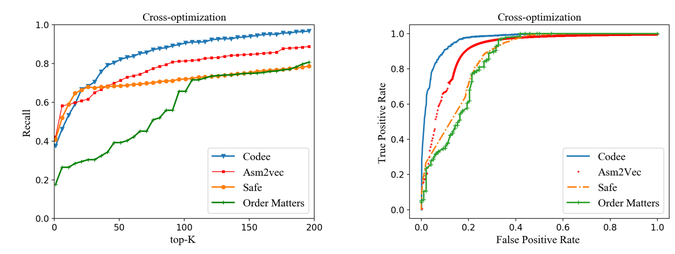
The code tensor can be represented as $\mathcal{F} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, where n_1, n_2, and n_3 denote the length of a function feature, the number of programs, and the number of functions in a program. After building the code tensor, we use tSVD to compress the code tensor to generate the function embeddings [1].

Input: Tensor representation $\mathcal{F} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.
Output: Function embedding tensor $\mathcal{R}$.
$\widehat{\mathcal{F}} =$fft$(\mathcal{F}, [\,], 3)$,
for $i = 1, 2, ..., n_3$:
  $[\widehat{\mathbf{U}}^{(i)}, \widehat{\mathbf{M}}^{(i)}, \widehat{\mathbf{V}}^{(i)}] =SVD(\widehat{\mathcal{F}}(:,:,i))$,
  $\overline{\mathcal{U}}(:,:,i) = \widehat{\mathbf{U}}^{(i)}(:, 1:n_4)$, $\overline{\mathcal{V}}(:,:,i) = \widehat{\mathbf{V}}^{(i)}(:, 1:n_4)$,
  $\overline{\mathcal{M}}(:,:,i) = \widehat{\mathbf{M}}^{(i)}(1:n_4, 1:n_4)$,
$\mathcal{U} =$ifft$(\overline{\mathcal{U}}, [\,], 3)$, $\mathcal{M} =$ifft$(\overline{\mathcal{M}}, [\,], 3)$, $\mathcal{V} =$ifft$(\overline{\mathcal{V}}, [\,], 3)$,
$\overline{\mathcal{F}} = \mathcal{U} * \mathcal{M} * \mathcal{V}^\dagger$,
$\mathcal{R} = \mathcal{U}^\dagger * \mathcal{F}$.



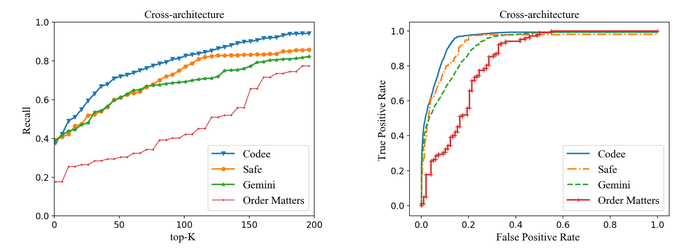
Application 1: Code Search across CPU architectures

Recall	ARM vs. x86-64			ARM vs. MIPS			MIPS vs. x86-64	
Baselines	Codee	Gemini	Safe	Codee	Gemini	Safe	Codee	Gemini
OpenSSL	**0.843**	0.546	0.788	**0.848**	0.549	-	0.718	**0.728**
Coreutils	0.745	0.740	**0.772**	**0.745**	0.720	-	**0.781**	0.746
libgmp	**0.979**	0.847	0.894	**0.975**	0.787	-	**0.987**	0.847
libcurl	**0.836**	0.600	0.784	**0.682**	0.536	-	**0.710**	0.510
Average	**0.851**	0.683	0.810	0.813	0.648	-	**0.799**	0.708
Precision	ARM vs. x86-64			ARM vs. MIPS			MIPS vs. x86-64	
Baselines	Codee	Gemini	Safe	Codee	Gemini	Safe	Codee	Gemini
OpenSSL	**0.937**	0.758	0.885	**0.904**	0.724	-	0.858	**0885**
Coreutils	0.802	0.759	**0.814**	**0.808**	0.784	-	**0.811**	0.798
libgmp	**0.983**	0.895	0.903	**0.989**	0.829	-	**0.977**	0.832
libcurl	**0.845**	0.686	0.798	**0.789**	0.648	-	**0.772**	0.650
Average	0.892	0.775	0.850	0.873	0.746	-	0.855	0.791



Application 1: Code Search across Optimization Levels

Optimization O0 vs. O3					
	Recall				
Baselines	Codee	Asm2Vec	DeepBinDiff	Gemini	Safe
OpenSSL	**0.809**	0.790	0.778	0.774	0.799
Coreutils	0.712	0.670	0.623	0.780	**0.768**
libgmp	**0.979**	0.810	0.720	0.889	0.907
libcurl	**0.798**	0.720	0.714	0.672	0.792
Average	**0.825**	0.748	0.709	0.779	0.817
Optimization O1 vs. O2					
	Recall				
Baselines	Codee	Asm2Vec	DeepBinDiff	Gemini	Safe
OpenSSL	**0.990**	0.971	0.846	0.924	0.961
Coreutils	0.912	**0.975**	0.911	0.901	0.924
libgmp	**0.982**	0.860	0.901	0.916	0.927
libcurl	0.811	0.765	0.721	0.856	**0.877**
Average	**0.924**	0.893	0.845	0.899	0.922



CLANG vs. GCC					
	Recall				
Baselines	Codee	Asm2Vec	DeepBinDiff	Gemini	Safe
OpenSSL	**0.973**	0.895	0.775	0.714	0.846
Coreutils	0.803	0.778	0.842	0.755	**0.847**
libgmp	**0.982**	0.740	0.939	0.704	0.951
libcurl	**0.847**	0.671	0.762	0.722	0.818
Average	**0.901**	0.771	0.830	0.749	0.8655

Conclusions

In this paper, we present a tensor embedding-based scheme, called Codee. Codee consists of an NLP-based token embedding generation method, a network representation-based basic block embedding generation method and an effective tensor-based function embedding generation method. Our evaluation shows that Codee outperforms other current state-of-the-art approaches on the characteristics of similarity detection accuracy, embedding generation time, and overall search time. Our research is one of the first to demonstrate that the tensor based data analysis techniques can have unique strengths in the binary code similarity analysis.

References

[1] Jia Yang, Cai Fu, Xiao-Yang Liu, Heng Yin, and Pan Zhou. Codee: A tensor embedding scheme for binary code search. *IEEE Transactions on Software Engineering*, pages 1–1, 2021. doi: 10.1109/TSE.2021.3056139.

Contact information:
Jia Yang
D201780841@hust.edu.cn