Similarity Metric Method for Binary Basic Blocks of Cross-Instruction Set Architecture

Xiaochuan Zhang[*†], Wenjie Sun[†], Jianmin Pang[†], Fudong Liu[†], Zhen Ma[†]
[*]Artificial Intelligence Research Center, National Innovation Institute of Defense Technology, China
[†]State Key Laboratory of Mathematical Engineering and Advanced Computing, China
Email: zhangxiaochuan@outlook.com, 1399445808@sjtu.edu.cn,
jianmin_pang@hotmail.com, lwfydy@126.com, zhen_ma@163.com

Abstract—**Basic block similarity analysis is a fundamental technique in many machine learning-based binary program analysis methods. The key to basic block similarity analysis is mapping the semantic information of the basic block to a fixed-dimension vector, which is the so-called basic block embedding. However, existing solutions to basic block embedding suffer from two major limitations. 1) The basic block embedding contains limited semantic information; 2) they are only applicable to a single instruction set architecture (ISA).**

To overcome these limitations, we propose a cross-ISA oriented solution for basic block embedding which utilizes an NMT (Neural Machine Translation) model to establish the connection between two ISAs. The proposed embedding model can powerfully map rich semantics of basic blocks from arbitrary ISAs into fixed-dimension vectors. Several measures have been taken to further improve the embedding model. To guide the embedding model to a better state, we creatively use the pre-trained model to generate hard negative samples. To promote the effectiveness of the proposed embedding model, we propose a reasonable assembly instruction normalization method in the data preprocessing phase, which is shown to outperform the previous methods. A similarity metric method is then derived and a million-scale dataset is presented to train and evaluate this method. To the best of our knowledge, this is the first million-scale dataset in this field. We implement a prototype system `MIRROR`. The experimental results show that `MIRROR` significantly outperforms the representative baseline in the respect that the basic block embeddings, i.e., the vectors, are more distinguishable to discriminate between similar basic blocks and dissimilar ones, and as a result, `MIRROR` can obtain obviously more accurate evaluation results. The significance of pre-training, the effectiveness of the proposed negative sampling method, and the instruction normalization method have also been justified in experiments.

I. Introduction

The binary program similarity metric is an important task which is widely used in many applications, like vulnerability detection, malware classification, and authorship analysis. In many previous approaches, the similarity metric between basic blocks is the basis of the binary program similarity metric [1]–[4]. In this paper, we focus on this essential challenge and further explore a similarity metric method for basic blocks of different ISAs.

With the growing success of machine learning, more and more recent researches on program analysis tend to borrow from machine learning methods, especially from natural language processing (NLP) methods. Generally, to solve basic block similarity metric problem by machine learning methods, there are two steps to be done as follows:

1) Mapping basic blocks to fixed-dimension vectors. These vectors are often called basic block embeddings or basic block vectors in previous works.
2) Measuring similarity between the basic block vectors through commonly used vector similarity measurements (Euclidean distance, cosine similarity, etc.), and using these results to represent the similarity between basic blocks.

As there are many mature methods for similarity measurements of vectors (Step 2), the major challenge is therefore how to get a good representation of basic blocks as fixed-dimension vectors (Step 1). Obviously, the more semantic information is involved in basic block embeddings, the more accurate of the similarity metric is.

A natural idea is to utilize neural machine translation (NMT) to automatically extract semantic information since the intermediate result of the NMT model contains rich semantic information of the source text. A typical NMT model is built on a seq2seq (sequence-to-sequence) architecture which contains two components: an encoder and a decoder. The encoder encodes the source text to a representation called context matrix, and then the decoder decodes this context matrix to the target text. In the idealized case (the source text can be accurately translated to the target text), the context matrix contains complete semantic information of the source text as well as the target text.

Motivated by this translation mechanism, in this paper, we build an NMT model which translates basic blocks from x86 to ARM. We convert the context matrices, i.e., the intermediate results of the NMT model, to fixed-dimension vectors and regard them as basic block embeddings. Then the similarity of

Corresponding author is Jianmin Pang.

Workshop on Binary Analysis Research (BAR) 2020
23 February 2020, San Diego, CA, USA
ISBN 1-891562-62-2
https://dx.doi.org/10.14722/bar.2020.23002
www.ndss-symposium.org

basic blocks is expected to be measured through the similarity of basic block embeddings. Note that the encoder of the trained NMT model is an x86-encoder which can only encode the basic blocks on x86. In order to measure the similarity between basic blocks of different ISAs (x86 and ARM), we further train an ARM-encoder which is expected to map the ARM basic blocks to the same vector space as that of the x86 basic block embeddings.

We implement a prototype called **MIRROR**. Extensive experiments have been conducted on **MIRROR** based on a huge dataset, which we collect from five representative open source projects. The experimental results show the superiority of **MIRROR** over the state-of-the-art method. All of the dataset and codes are available on the Internet[1].

The contributions of this paper are summarized as follows:

- A cross-ISA oriented binary basic block embedding method is proposed. This model can map rich semantic information of the basic block of arbitrary ISA to the same embedding vector space.
- A negative sampling method is proposed to generate negative samples for the embedding model training. The effectiveness of the method to improve the performance of the embedding model has been justified in our experiments.
- A novel instruction normalization method is proposed, which is superior to previous methods.
- A large scale dataset is presented which includes over a million x86-ARM basic block pairs, so that the data-driven approach can be well applied.

II. Related Works

A. Basic Block Embedding

To apply machine learning methods to program similarity metric, the crucial step is to convert basic blocks to fixed-dimension vectors, which is the so-called basic block embedding.

To the best of our knowledge, Feng et al. first applied the basic block embedding to the binary similarity metric in 2016, and proposed a bug search engine called Genius [1]. In Genius, each basic block is represented by an eight-dimensional vector, where each dimension corresponds to a manually selected static feature. The eight manually selected features are listed in Table I. This kind of approach, also known as manually feature engineering, has been successfully used and improved in some later works [2], [5]. However, since integers are not continuous, the information involved in integer-representations of basic block embedding is limited. Besides, manually feature engineering needs a lot of domain knowledge of assembly code, which is not friendly for most researchers.

Type	Attribute name
Block-level attributes	String Constants
	Numeric Constants
	No. of Transfer Instructions
	No. of Calls
	No. of Instructions
	No. of Arithmetic Instructions
Inter-block attributes	No. of offspring
	Betweenness

TABLE I: The manually selected basic block features used in Genius [1].

To address the above issues, static word representation based methods are applied to program language processing in recent works [4], [6]–[8]. In these works, tokens in the basic block, like operators (opcodes) and operands, are represented as fixed-dimension vectors. For basic block embedding, Ding et al. [8] utilized doc2vec [9] and Li et al. [4] regarded the summation of token vectors as the basic block embedding. In these works, each element in the basic block embedding is an automatically computed real number, which greatly increases the information capacity in basic block embeddings. Although, these works are successful cases of applying NLP methods to programming language processing, their methods are not suitable to **cross-ISA** basic block embedding. Static word representation methods they utilized are based on the distributional representation hypothesis that words with similar contexts should be semantically similar as well. As tokens in different ISAs are impossible to appear in the same context, they cannot meet the distributional representation hypothesis. Therefore, static word representation methods are not applicable to map basic blocks of different ISAs to the same embedding vector space.

In 2019, Zuo et.al proposed a Long Short-Term Memory (LSMT) based siamese network named INNEREYE-BB for measuring similarity between basic blocks of different ISAs [3]. Their testing data is a collection of x86-ARM basic block pairs, which consists of two types of samples,

- similar sample: an x86 basic block and a semantically equivalent ARM basic block. Both of them are obtained from the same source code;
- dissimilar sample: an x86 basic block and a semantic completely inequivalent ARM basic block, which is filtered through text similarity between the x86 basic block and the ARM-same-sourced x86 basic block.

The authors claimed to have achieved nearly 95% accuracy to discriminate these two types. However, in our opinion, the success mostly owes to the huge difference between similar and dissimilar samples in their testing data, which is not difficult for the neural network to discriminate.

In addition, we find it is not suitable to apply recurrent-based neural networks to assembly codes directly, such as Recurrent Neural Networks (RNN), LSTM [10] and Gated

[1] https://github.com/zhangxiaochuan/MIRROR

Recurrent Unit (GRU) [11]. Recurrent models compute hidden states for every token along the source sequence, and the hidden state h_t of the input token at position t depends on the previous hidden state h_{t-1}. Let's take LSTM as an example to illustrate this process. Take a sequence $S = (s_1, \ldots, s_n)$ as input. The operation on s_t performed by the LSTM layer can be abstracted as:

$$h_t, c_t = F_{LSTM}(s_t, h_{t-1}, c_{t-1}), \tag{1}$$

where F_{LSTM} denotes the operation performed by this LSTM layer, h_t and c_t are the hidden state and the cell state, repectively. In [3], the authors regard the last hidden state h_n as the embedding of the basic block, which depends on the last input token s_n and previous states h_{n-1} and c_{n-1}.

For assembly codes, a basic block usually ends with a jump instruction or function call. That is to say, the last token for a basic block is mostly a basic block label or a function name. According to our statistics shown in Table II, such basic blocks make up nearly 80% of the total in both x86 or ARM. To avoid the out-of-vocabulary (OOV) problem, basic block labels and function names should be normalized into uniform tokens respectively [3]. Thus, although basic blocks are totally different, in most cases, for calculating the final output, the input s_n is fixed. What's more, if the last token is a basic block label or a function name, the penultimate token should be a jump or a call operator. It means for calculating the penultimate hidden state h_{n-1} and the penultimate cell state c_{n-1}, the input s_{n-1} is also fixed in most cases. Due to this inherent characteristic of assembly language, we suspect using LSTM or other recurrent based models will limit the differentiation of basic block embeddings. This conjecture is verified in Subsection V-G.

Token type	x86	ARM
basic block label	77.68%	61.23%
function name	3.71%	12.58%
others	18.60%	26.19%

TABLE II: Statistics of the last token in basic block. The data source is from the public test data in [3]

B. Neural Machine Translation

Machine translation has been a central challenge in the NLP community for a long time. In recent years, the end-to-end neural network based machine translations achieved great improvements compared with the previous approaches. In this subsection, we take a brief overview of NMT models.

Recurrent models have been successfully used for sequence modeling and applied to NMT tasks for a long time. In 2014, Cho et. al. proposed the seq2seq model for the first time in which two GRUs serve as the encoder and the decoder respectively [12]. The authors applied this model to machine translation tasks and showed the superiority of this approach in terms of the translation quality compared with the conventional statistical machine translation systems. To enhance the ability of long-range dependencies modeling for recurrent models, in 2015, Bahdanau et. al. proposed attention mechanism. Equipped with the attention mechanism, the GRU based NMT model improves the translation quality especially for long sentence translation [13].

Although achieved great successes in a board range of NMT tasks, the fundamental constraint of sequential computation limits the computation efficiency of recurrent models. Specifically, recurrent models compute hidden states for every token along the source and the target sequences, and in order to compute the hidden state h_t of the input token at position t, the previous hidden state h_{t-1} must be computed firstly. This inherently sequential modeling property constitutes an obstacle towards parallelization of the process.

In order to solve the above problem, in 2017, Vaswani et. al. proposed a new NMT model named **Transformer**, which is entirely based on self-attention mechanism [14]. Self-attention, a special case of attention mechanism, can model dependencies among tokens in different positions of a single sequence. Due to the highly paralleled computation of self-attention, Transformer has achieved significant improvements in computational efficiency, while also improving translation quality. Following Transformer, recent efforts have continued to push the boundaries of this architecture [15]–[20], making Transformer become the most popular architecture in NMT.

It is obvious that with the improvement of translation quality, the context matrix contains more semantics of the source text. Thus, in order to generate basic block embeddings with rich semantic information, in this work, we choose the current mainstream Transformer architecture to serve as the NMT model.

III. Embedding Solution for Cross ISA Basic Blocks

The premise to measure the similarity between two basic block embeddings is that the two basic block embeddings are in the same vector space. However, it is very difficult to map basic blocks of different ISAs to the same embedding vector space and this problem has not been well solved in previous works. In this section, we propose a novel solution to this challenge, which is able to map basic block on arbitrary ISA into the same embedding vector space.

Remark. *1. Note that in our proposed basic block embedding model, the two embedding modules can be used for basic blocks on* ***any other architectures****. We specify x86 and ARM just for the sake of description. 2. The NMT model used in our approach is not limited to Transformer only, but actually, any NMT models of seq2seq architecture can be used. We use Transformer just because it is one of the best NMT models currently available.*

To clarify the necessity of negative sampling (Subsection III-C), in the following, we introduce an idealized solution and further put forward a practical solution based on this idealized one.

Idealized Solution

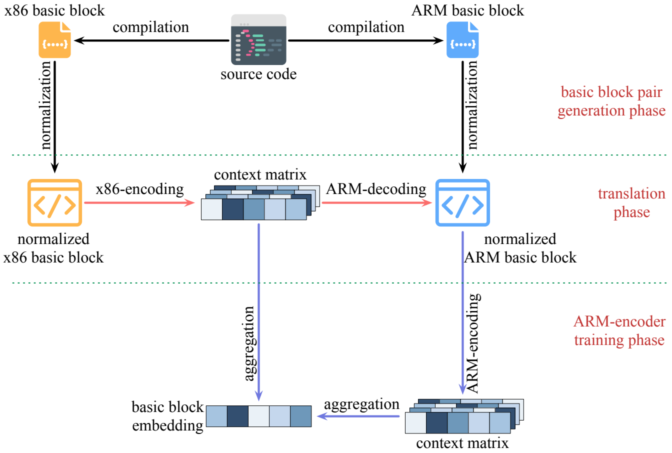


Fig. 1: The idealized solution of the embedding neural network.

To map x86 and ARM basic blocks into the same embedding vector space, we first propose an idealized solution based on the assumption that NMT models are able to **perfectly** translate x86 basic blocks to ARM basic blocks, which means the context matrix contains **complete** semantic of both x86 and ARM basic blocks. This property is perfect for the context matrix to be served as basic block embedding.

We first train an NMT model which translates x86 basic blocks into the corresponding ARM basic blocks. The encoder of the trained NMT model is served as x86-encoder. The context matrix, i.e., the output of x86-encoder, is denoted by $C = \{c_i\} \in \mathbb{R}^{d \times n}$, where n is a dynamic variable denoting the length of x86 basic block. This implies that although d is a fixed number, the size of context matrix $d \times n$ is not invariant. Thus it is not suitable to regard the context matrices as basic block embeddings directly. Therefore, we convert the context matrix $C = \{c_i\}$ to a fixed-dimension vector E by sum aggregation:

$$E = \sum_{i=1}^{n} c_i, \tag{2}$$

which is regarded as the basic block embedding in this work. Then, we fix the x86-encoder and train an ARM-encoder to map ARM basic blocks close to the same vector space of x86 basic block embeddings. To train these neural networks, x86-ARM basic block pairs that share the same semantics are needed. Thus as illustrated in Fig. 1, three phases should be done to implement the proposed idealized solution,

1) the basic block pair generation phase;
2) the translation phase;
3) the ARM-encoder training phase.

The above idealized model is based on the **perfect translation** assumption which means not only the translation is 100% accurate, but also there is not any loss in the translation process. This strict assumption, however, is almost impossible in practice. That is to say, the x86-encoder trained in the translation phase is definitely biased.

Practical Solution

To address this issue, we further fine-tune the x86-encoder during the ARM-encoder training of the idealized solution, and get a practical solution with a slightly different model training process. In specific, the practical model training process can be divided into two phases as illustrated in Fig. 2:

1) the x86-encoder pre-training phase, as the dotted line shown in Fig. 2;
2) the ARM-encoder training & x86-encoder fine-tuning phase, as the solid lines shown in Fig. 2.

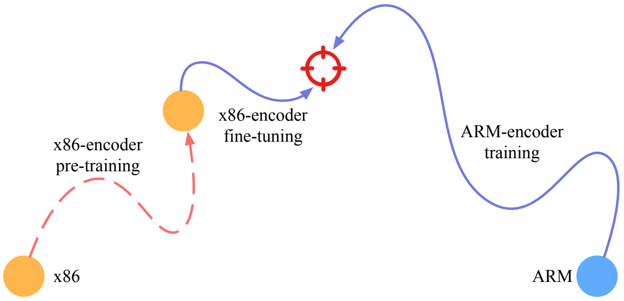


Fig. 2: Illustration of the two training phases of the practical solution.

It is worth noting that this practical solution brings an additional problem. With only positive samples (basic block pairs with equivalent semantic), the training objective is to make the distance of x86-embedding and ARM-embedding as close as possible. Since the x86-encoder and the ARM-encoder are being trained together, both x86 embeddings and ARM embeddings are dynamic. This makes the embedding networks for x86 and ARM tend to map arbitrary inputs to the same vector. As a result, x86 basic block embeddings are always close to ARM basic block embeddings even if these basic blocks are in totally different semantics.

Therefore, we introduce negative samples (semantically inequivalent basic blocks) as a necessary supplement to model training. After introducing negative samples, with the supervision of both positive and negative samples, we are able to train the embedding networks to make the x86-embedding and ARM-embedding close if they are semantically equivalent and apart if not.

In the following, we detail this practical solution.

A. *Basic Block Pair Generation*

To train our proposed basic block embedding neural networks, a huge amount of x86-ARM basic block pairs with the same semantics are needed. Thus we propose a basic block pair generation method for x86-ARM in this subsection.

As assembly codes compiled from the same source code can be regarded as semantically equivalent, our idea is to label basic blocks at the high-level language and compile them to x86 and ARM basic blocks respectively. Then the basic blocks on different ISAs are semantically equivalent if they share the same label. In order to implement this functionality, we integrates a pass into LLVM which labels basic block with a unique ID on LLVM IR. Since LLVM supports multiple backends, we can generate semantically equivalent basic blocks of different ISAs.

However, it is not a good idea to feed neural networks these raw assembly codes directly. This is mainly due to the following two reasons:

- Constants have little effect on program semantics, and the unnormalized constants may lead to OOV problem [3].
- In assembly instructions, registers can be replaced by other registers from the same category without semantics changed in most cases. This property may confuse neural networks especially the NMT model since the NMT model does not know which register should be chosen in a specific register category, and so lead to translation errors.

To avoid these drawbacks, we propose a normalization method for assembly instructions. In this method, constants in assembly codes are divided into five categories, namely, immediately numbers, addresses, variable names, function names, and basic block labels. And each category of constants is normalized by a specific symbolic representation. Similarly, registers are normalized by several categories of symbolic representations according to their usage. For x86 instruction set, registers are divided into 14 categories which contain pointer registers, float registers, four types of general registers, four types of data registers, and four types of address registers. The data registers, address registers and general registers, are divided into four types respectively according to the data length they store, which is 8-bit, 16-bit, 32-bit and 64-bit. For registers in ARM, they are normalized into two categories: general registers and pointer registers.

B. x86-encoder Pre-training

As we mentioned before, the x86-encoder pre-training is integrated in an NMT model training process which translates x86 basic blocks into their corresponding ARM basic blocks. In this subsection, we elaborate this process in detail.

In our solution, a basic block is represented as a sequence, in which each token represents either an operator or an operand of the instructions in the basic block. In this subsection, we denote the x86 basic block (source sequence) and the ARM basic block (target sequence) as $S = (s_1, \dots, s_n)$ and $T = (t_1, \dots, t_m)$, where s_i and t_i are indices of the tokens in the x86 tokens vocabulary V_{x86} and the ARM tokens vocabulary V_{ARM}, respectively.

The training task for this NMT model is similar to the training task for natural language NMT models, which can be described as giving an x86 basic block $S = (s_1, \dots, s_n)$, and the front part of its corresponding ARM basic block $T = (t_1, \dots, t_{k-1})$, the target for the NMT model is to predict the next token t_k of the ARM basic block. The final output of the NMT model is the probability distribution of the next token $y_k \in \mathbb{R}^{|V_{ARM}|}$, where $\sum_i y_{ki} = 1$.

The training objective is based on categorical cross-entropy. In our design, for each basic block pair, the NMT model predicts each token of the ARM basic block from the first one to the last. Thus, the loss function for an x86-ARM basic block pair is expressed as follows:

$$L = -\sum_{k=1}^{m} \sum_{j=1}^{|V_{ARM}|} \hat{y}_{kj} \log\left(y_{kj}\right), \tag{3}$$

where $\hat{y}_k \in \mathbb{R}^{|V_{ARM}|}$ is the one-hot encoding of t_k. We minimize the total loss in the whole training set using the Adam [21] gradient descent algorithm, which computes adaptive learning rates for each parameter.

C. Negative Sampling

As we elaborated in the beginning of the section, negative samples are necessary for training the two embedding networks together.

In our design, the dataset for training the two embedding networks is a collection of triplets. A triplet is composed of an anchor, a positive sample, and a negative sample. The positive sample and the negative sample are basic blocks of the same ISA, while the anchor is the basic block of another ISA. The training task is to discriminate that the positive sample, than the negative one, is semantically closer to the anchor. The anchors and the positive samples are directly sourced from the semantically equivalent basic block pairs generated by the method in Subsection III-A. To generate negative samples, we propose a novel negative sampling method, which is introduced in the following.

The quality of negative samples is related to the quality of the trained model. Inserting hard negative samples into random negative samples is shown to be effective in many computer vision tasks, like person re-identification [22]–[24] and object detection [25]. Inspired by these recent successes, in our solution, we generate hard negative samples as an effective supplement to random samples. We expect this method is better for training the embedding networks than solely random negative sampling which is used in previous program language processing works [2], [6], [8].

Hard negative sampling is to select samples that are hard for the neural network to discriminate from the positive. For example, in person re-identification tasks, a hard negative sample can be a similar but different person to the target person, such as a person with the same T-shirt or the same hairstyle as the target person. Similarly, in our task, a hard negative sample should be a basic block that is similar but not equivalent to the anchor.

Due to the huge amount of dataset and the obscurity of assembly codes, it is almost impossible to manually select these hard negative samples. But how to automate this work? We turn our attention to the pre-trained NMT model. Although the encoder of the pre-trained NMT model is just a semi-finished product for x86 basic block encoding, it is enough to give us a rough conclusion about whether two x86 basic blocks are similar.

Given two x86 basic blocks S_1 and S_2, we first obtain their embeddings $E_1, E_2 \in \mathbb{R}^d$ by utilizing the pre-trained x86-encoder and aggregating the output as shown in Equation 2, where d is the embedding dimension. Then, we can judge the similarity of them by measuring the Euclidean distance of their embeddings:

$$\mathrm{D}(E_1, E_2) = ||E_1 - E_2||^2 = \sqrt{\sum_{i=1}^{d}(e_{1i} - e_{2i})^2}, \tag{4}$$

where $e_{1i} \in E_1$ and $e_{2i} \in E_2$. It is obvious that the smaller of the Euclidean distance is, the more similar of the two basic blocks are.

Note that we have already figured out how to calculate the distance between different x86 basic blocks. Now we clarify how to apply hard negative sampling to generate negative samples.

Recall that to derive a triplet for dataset, we first generate a specific basic block pair consisting of an anchor and a positive sample, where one of two is x86 basic block and the other one is ARM basic block. Then we randomly sample 100 x86 basic blocks and choose the one which is the most similar to the x86 basic block in the basic block pair in terms of Euclidean distance defined in Equation 4. Finally the negative sample is derived according to the following two situations:

- If the anchor is ARM basic block, the negative sample is simply taken as the chosen x86 basic block.
- If the anchor is x86 basic block, the negative sample is taken as the ARM basic block which is semantically equivalent to the chosen x86 basic block.

Although hard negative sampling is proved to be beneficial for model training, only selecting hard negative samples makes the training unstable [22]. Thus, we also apply random negative sampling to building the dataset. The ratio of random negative samples to hard negative samples is empirically set at 2 to 1.

D. Embedding Networks Training

With the negative samples generated in the previous subsection, now we can train the ARM embedding network and fine-tune the pre-trained x86 embedding network together.

For a basic block triplet, in the target embedding vector space, the embedding of the anchor should be closer to that of the positive sample than that of the negative sample. In this concern, we use a margin-based triplet loss function for a basic block triplet which is expressed as follows:

$$L = \max\{\mathrm{D}(E_1, E_2) - \mathrm{D}(E_1, E_3) + \gamma, 0\}, \tag{5}$$

where E_1, E_2 and E_3 are embeddings of the anchor, the positive sample and the negative sample respectively, $\gamma > 0$ is a margin parameter, and the operation D is the Euclidean distance which is shown in Equation 4. Note that E_1, E_2 and E_3 are generated through their corresponding embedding networks. This loss encourages $\mathrm{D}(E_1, E_2)$ to be smaller than $\mathrm{D}(E_1, E_3)$ by at least a margin γ, which is illustrated in Fig. 3.

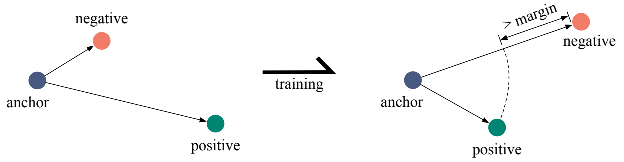


Fig. 3: Illustration of the role of the loss function plays in embedding networks training. In the training process, the loss function encourages the anchor to get close to the positive sample and away from the negative one.

IV. Similarity Metric

The similarity score between two basic blocks should be a real number range from 0 to 1. However, the Euclidean distance of their embeddings is range from 0 to $+\infty$. Thus, we need to map the Euclidean distance to $[0, 1]$.

In this work, the similarity between two basic blocks B_1 and B_2 is defined as:

$$\mathrm{Sim}(B_1, B_2) = \exp(-\frac{\mathrm{D}(E_1, E_2)}{d}), \tag{6}$$

where d denotes the embedding dimension, and E_1, E_2 are embeddings for B_1 and B_2, respectively.

V. Evaluation

In this section, we perform an extensive evaluation on **MIRROR**, which is detailed in the following seven subsections.

In Subsection V-A, we elaborate on the implementation and setup, including the dataset, the evaluation metric, and other details. In Subsection V-B, we utilize a series of experiments to find out a suitable hyperparameter, i.e., the margin γ in Equation 5. In Subsection V-C, we compare the experimental results of **MIRROR** and the baseline method. In Subsection V-D, we compare our proposed assembly instruction normalization method with the other three in terms of the performance in basic block similarity measuring task. In Subsection V-E, we show the superiority of our proposed negative sampling method by comparing with other three methods. In Subsection V-F, we demonstrate the importance of the pre-training phase in **MIRROR**. In Subsection V-G, we further demonstrate the superiority of **MIRROR** through several cases.

A. Implementation and Setup

To evaluate **MIRROR**, we present a large scale dataset named MISA (Multi-ISAs basic block dataset), which is sourced from five well known C/C++ based open source projects in different fields. These projects are listed in Table III.

Project	Version	Description
Binutils	2.30	collection of binary tools
Coreutils	8.29	GNU core utilities
FFmpeg	n3.2.13	collection of multimedia process tools
OpenSSL	1.1.1b	security protocols and cryptographic library
Redis	5.0.5	key-value database

TABLE III: The data source for MISA.

MISA is constructed with seven parts:

- **MISA$_{\textbf{Pair\_All}}$**: consists of 1,122,171 semantically equivalent x86-ARM basic block pairs which are generated from the source code in projects listed in Table III by the method we proposed in Subsection III-A. For each source file, we compile it through four different optimization methods: O0, O1, O2 and O3.
- **MISA$_{\textbf{Pair\_Large}}$**: consists of 500,000 randomly selected basic block pairs from MISA$_{\text{Pair}}$.
- **MISA$_{\textbf{Pair\_Base}}$**: consists of 50,000 randomly selected basic block pairs from MISA$_{\text{Pair}}$.
- **MISA$_{\textbf{Triplet\_Large}}$** : consists of 3,000,000 basic block triplets which are randomly divided into a training set and a test set by a ratio of 80% to 20%. These triplets are sourced from MISA$_{\text{Pair\_Large}}$, and generated by the negative sampling method shown in Subsection III-C. For each basic block pair, we generate four triplets by random negative sampling, and two triplets by hard negative sampling.
- **MISA$_{\textbf{Triplet\_Base}}$**: is the same as MISA$_{\text{Triplet\_Large}}$ except for size which is sourced from MISA$_{\text{Pair\_Base}}$.
- **MISA$_{\textbf{Eval\_Large}}$** : consists of 100,000 semantically equivalent basic block pairs, which is corresponding to the testing set in MISA$_{\text{Triplet\_Large}}$. In other words, the test set of MISA$_{\text{Triplet\_Large}}$ is obtained by negative sampling over MISA$_{\text{Eval\_Large}}$. We use this dataset for evaluating the model trained on MISA$_{\text{Triplet\_Large}}$.
- **MISA$_{\textbf{Eval\_Base}}$** : is the evaluation set for MISA$_{\text{Triplet\_Base}}$.

Our experiments are conducted on a workstation equipped with an Intel Core i9-9980XE CPU, four GeForce RTX 2080Ti GPU cards, 112 GB memory, and 2TB SSD.

In the x86-encoder pre-training phase, the embedding length and the mini-batch size are set to 256 and 30, respectively. We randomly select 300,000 samples from MISA$_{\text{Pair}}$. As illustrated in Fig. 4, after 20 epochs training, the x86-ARM translation model is close to convergence. This pre-trained Transformer is used for hard negative sampling and initializing the x86-encoder of **MIRROR** in the embedding networks training phase.

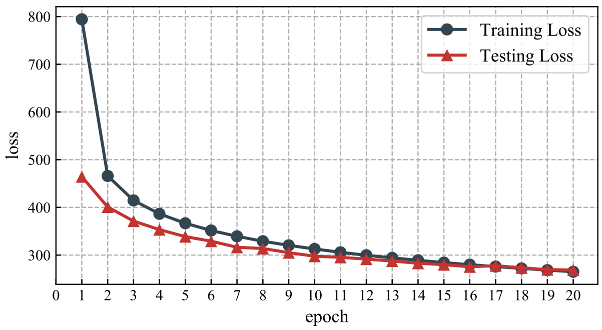


Fig. 4: Learning curves of Transformer in the pre-training phase.

To evaluate **MIRROR**, following previous works for natural language understanding [26], [27], we use precision at N (short for P@N) as an evaluation metric. In specific, given a basic block of A ISA, we select the true response (semantically equivalent basic block) and 99 random selected responses, from B ISA. We measure the similarity between the given basic block and each of the 100 selected responses through the metric method shown in Equation 6 and rank them from the most similar to the most dissimilar. **The P@N score evaluates the proportion of the data whose true response ranks in the top N responses in the whole evaluation data.** That is, the larger the P@N score is, the better the similarity metric method is.

In all of the following experiments, we utilize two P@N tasks to evaluate models: 1) given x86 basic blocks, we find responses in ARM basic blocks; 2) given ARM basic blocks, we find responses in x86 basic blocks. The two tasks are denoted by "x86-ARM" and "ARM-x86" respectively in the following experimental results. **For each task, we measure three indicators, i.e., P@1, P@3, and P@10.**

B. Hyperparameter

The margin γ shown in Equation 5 is the most important hyperparameter in this work. A suitable margin can guide the neural network to train in the right direction. However, a too small margin will simplify training tasks for the model, leading to poor performance in actual evaluation (P@N), while a too large margin will make the training task very difficult, making the model prone to extreme situations and resulting in overfitting.

To find out the best margin for our task, in this subsection, we evaluate different margin values ranging from 60 to 200 with an interval of 20. The models are trained on MISA$_{\text{Triplet\_Base}}$ with 20 epochs and evaluated on MISA$_{\text{Eval\_Base}}$. We list the evaluation results in Table IV. It shows that with margin $\gamma = 140$, **MIRROR** can be trained to the best state. Thus, we set the margin to 140 in all the following experiments.

Margin	**x86-ARM**			**ARM-x86**		
	P@1	**P@3**	**P@10**	**P@1**	**P@3**	**P@10**
60	68.1%	79.6%	87.4%	67.1%	81.8%	90.0%
80	68.5%	80.2%	88.3%	67.2%	81.9%	89.6%
100	68.3%	79.9%	87.8%	**67.9%**	81.7%	89.5%
120	68.6%	80.1%	88.2%	67.3%	81.9%	89.4%
140	69.0%	**83.8%**	**92.9%**	67.0%	**83.0%**	**91.5%**
160	**69.3%**	83.7%	92.2%	67.2%	82.1%	**91.5%**
180	65.8%	79.2%	87.8%	66.6%	81.8%	89.4%
200	65.3%	78.3%	87.5%	66.9%	81.8%	89.2%

TABLE IV: The experimental results on a series of margin ranging from 60 to 200.

C. Comparison with Baseline

In this subsection, we evaluate the P@N scores of our model and compare with a representative baseline INNEREYE-

BB [3]. We train **MIRROR** on $\mathrm{MISA}_{\mathrm{Triplet\_Large}}$ and evaluate on $\mathrm{MISA}_{\mathrm{Eval\_Large}}$. Since the design of INNEREYE-BB is training on basic block pairs which are labeled as positive (similar) or negative (dissimilar), $\mathrm{MISA}_{\mathrm{Triplet\_Large}}$ can not be applied to train INNEREYE-BB directly. Thus, to be fair, we utilize the same positive samples as $\mathrm{MISA}_{\mathrm{Triplet\_Large}}$, e.g., $\mathrm{MISA}_{\mathrm{Pair\_Large}}$, and apply their proposed negative sampling method to construct the training dataset for INNEREYE-BB. Besides, the evaluation set for INNEREYE-BB is also $\mathrm{MISA}_{\mathrm{Eval\_Large}}$. To illustrate the superiority of **MIRROR** over the baseline, we further train **MIRROR** on a much smaller dataset $\mathrm{MISA}_{\mathrm{Triplet\_Base}}$ and still evaluate on $\mathrm{MISA}_{\mathrm{Eval\_Large}}$. All these models are trained for 20 epochs.

We list the experimental results in Table V. A comparison between INNEREYE-BB and **MIRROR** clearly shows that:

1) For dataset $\mathrm{MISA}_{\mathrm{Triplet\_Large}}$, **MIRROR** is at least 14% higher than INNEREYE-BB on each indicator, and even over 40% on P@1 of ARM-x86.
2) For dataset $\mathrm{MISA}_{\mathrm{Triplet\_Base}}$, even only 10% training data is available, the performance of **MIRROR** is surprisingly much better than that of INNEREYE-BB which is trained on a much larger dataset generated from $\mathrm{MISA}_{\mathrm{Pair\_Large}}$.

Model	x86-ARM			ARM-x86		
	P@1	**P@3**	**P@10**	**P@1**	**P@3**	**P@10**
INNEREYE-BB	51.0%	66.6%	77.2%	32.8%	54.8%	79.5%
MIRROR ($\mathrm{MISA}_{\mathrm{Triplet\_Base}}$)	64.0%	77.2%	85.7%	58.7%	73.8%	83.1%
MIRROR ($\mathrm{MISA}_{\mathrm{Triplet\_Large}}$)	**77.4%**	**88.7%**	**94.9%**	**74.2%**	**87.2%**	**94.1%**

TABLE V: The experimental results for our model and INNEREYE-BB. Note that all these results are obtained by evaluating the trained models on $\mathrm{MISA}_{\mathrm{Eval\_Large}}$.

D. Instruction Normalization Methods

In this task, we compare our proposed assembly instruction normalization method with the other three methods. These four methods are detailed in Table VI. In Table VI, Method #1 is proposed by us. Method #2 further abstracts the register at the basis of Method #1. Method #3 removes all operands including registers and keeps only operators. This methods is often used in software theft identifying methods [28], [29]. Method #4 only normalizes constants, and this method is widely used in recent works [3], [4], [8].

Normalized Element		Method			
		#1 (ours)	**#2**	**#3**	**#4**
Constants		✓	✓	○	✓
Registers	Normalized by categories	✓	×	×	×
	Normalized to the same token	×	✓	×	×
	Remove all registers	×	×	✓	×

TABLE VI: Four instruction normalization methods.

For each normalization method, we run the entire training process. We first utilize Transformer to pre-train x86-encoders on the same 300,000 randomly selected basic block pairs from $\mathrm{MISA}_{\mathrm{Pair}}$ with different normalization methods. Then, we use the pre-trained x86-encoders to generate negative samples on $\mathrm{MISA}_{\mathrm{Pair\_Base}}$, through the same method as $\mathrm{MISA}_{\mathrm{Triplet\_Base}}$. Finally, we train embedding networks with the same set of hyperparameters for each normalization method and evaluate these models on $\mathrm{MISA}_{\mathrm{Eval\_Base}}$. That is, all the settings are the same except for normalization methods and the resulting different negative samples.

The experimental results are listed in Table VII. It shows that on all the six indicators, our proposed method (Method #1) is superior to the other three methods. A comparison between Method #2 and Method #4 shows that the more semantic information is involved (Method #4) the better performance of the trained model is. Since Method #2 contains only a little more semantic information than Method #3, the evaluation results of Method #2 are not much better than that of Method #3.

Normalization	x86-ARM			ARM-x86		
	P@1	**P@3**	**P@10**	**P@1**	**P@3**	**P@10**
Method #1 (ours)	**69.0%**	**83.8%**	**92.9%**	**67.0%**	**83.0%**	**91.5%**
Method #2	62.9%	77.1%	86.9%	61.2%	78.8%	88.7%
Method #3	59.6%	77.5%	89.5%	59.7%	77.7%	89.3%
Method #4	65.8%	80.0%	88.7%	66.5%	82.1%	89.6%

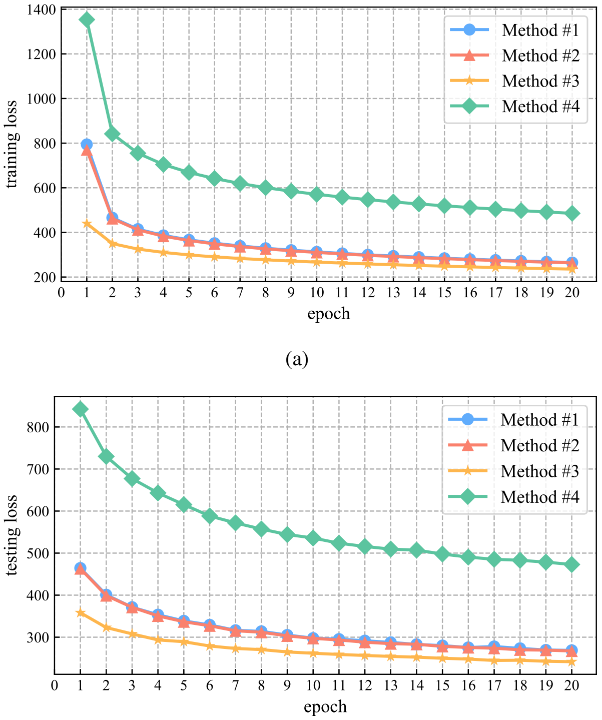
TABLE VII: The experimental results for different basic block normalization methods.

To further explain the differences among these four normalization methods, we draw the learning curves of their pre-training phases in Fig. 5. Combined Fig. 5 with Table VII, it can be clearly seen that: 1) the higher the degree of normalization is, the lower training and testing losses of the Transformer are; 2) but a lower loss in the pre-training phase does not mean the normalization method is better (a higher P@N scores of the model); 3) although Method #4 does not preserve much more semantic information than Method #1 (ours), neural networks especially Transformer can be easily confused by registers of the same category, leading poor performance of the model; 4) the learning curves in the pre-training phases of Method #1 and Method #3 are almost overlapped while the final trained embedding model of Method #1 is much better than that of Method #3. It indicates that register categories carry a lot of semantic information and this information can be captured by the networks easily without any confusion.

E. Negative Sampling Methods

In this task, we are going to demonstrate the superiority of our proposed negative sampling method. We utilize four different negative sampling methods to generate datasets from $\mathrm{MISA}_{\mathrm{Pair\_Base}}$. These four negative sampling methods are:

- **None**: without any negative samples, which is the idealized solution shown in Fig. 1.
- **Random only**: we randomly select six negative samples for each positive basic block pair without any hard negative samples.



(a)

(b)

Fig. 5: Learning curves of Transformer on dataset normalized through four different methods. (a) and (b) demonstrate training and testing losses, respectively.

- **Hard only**: we select six hard negative samples for each positive basic block pair without any randomly selected negative samples.
- **Mixed (ours)**: for each positive sample, two negative samples are selected through hard negative sampling and four negative samples are selected randomly. Actually, this dataset is MISA$_{\text{Triplet\_Base}}$.

For each generated dataset, we train **`MIRROR`** for 20 epochs, and evaluate trained models on MISA$_{\text{Eval\_Base}}$. The evaluation results are shown in Table VIII. It is obviously seen that our proposed negative sampling method is the best of the four. Note that **hard only** is not better than **random only** or **mixed**.

Negative Samples	**x86-ARM**			**ARM-x86**		
	P@1	**P@3**	**P@10**	**P@1**	**P@3**	**P@10**
None	49.6%	56.2%	66.4%	52.5%	62.6%	71.5%
Random only	62.2%	79.2%	89.5%	56.6%	76.1%	87.6%
Hard only	60.0%	74.6%	84.8%	52.7%	70.1%	80.0%
Mixed (ours)	**69.0%**	**83.8%**	**92.9%**	**67.0%**	**83.0%**	**91.5%**

TABLE VIII: The experimental results for different negative sampling methods.

F. Pre-training and Non-pre-training

It seems that with the supervision of positive and negative samples, **`MIRROR`** can be trained directly without x86-encoder pre-training. In this subsection, we are going to demonstrate the importance of the pre-training phase besides being used for hard negative sampling.

A pre-trained model is compared with a non-pre-trained model in this subsection. In order to eliminate the influence of different negative sampling methods, in the subsection, we compare the two models in two cases:

- the pre-trained model and non-pre-trained model are all trained on the same dataset generated by random negative sampling from MISA$_{\text{Pair\_Base}}$.
- the two models are trained on MISA$_{\text{Triplet\_Base}}$.

We utilize MISA$_{\text{Eval\_Base}}$ to evaluate these two models. The experimental results are listed in Table IX.

The P@N scores of our model without pre-training show that **`MIRROR`** can be trained without the pre-training phase. However, the non-pre-trained model lags behind the pre-trained model on all the six indicators with both random and mixed negative samples.

Setting		**x86-ARM**			**ARM-x86**		
Pre-train	**Negative**	**P@1**	**P@3**	**P@10**	**P@1**	**P@3**	**P@10**
False	Random	58.2%	76.3%	88.4%	53.9%	73.8%	85.7%
True	Random	**62.2%**	**79.2%**	**89.5%**	**56.6%**	**76.1%**	**87.6%**
False	Mixed	64.4%	79.4%	89.1%	61.0%	78.7%	87.7%
True	Mixed	**69.0%**	**83.8%**	**92.9%**	**67.0%**	**83.0%**	**91.5%**

TABLE IX: The experimental results for the pre-trained model and non-pre-trained model.

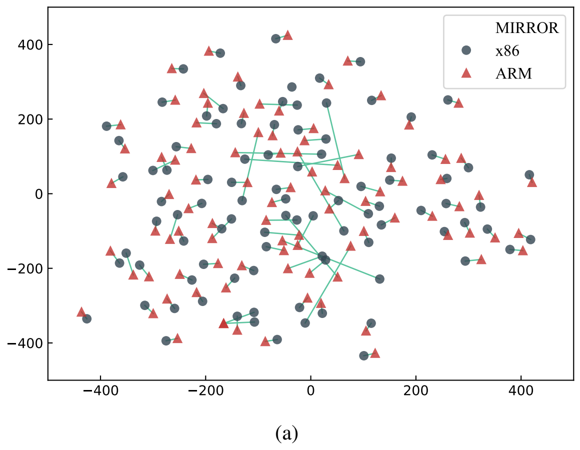
G. Case Study

1) Embedding Vector Space Visualization: To accurately measure the similarity of cross-ISA basic blocks, "good" vector spaces of basic block embeddings should satisfy the following two conditions:

- The embedding spaces of different ISAs are close enough, that is the semantically equivalent basic blocks of different ISAs should be mapped to the positions which are as close as possible, so that the cross-ISA measurement can be well applied.
- The positions of different basic blocks in the embedding space should be discrete enough, so that we can discriminate them from semantically equivalent basic blocks.

To explain why **`MIRROR`** is better than the baseline, INNEREYE-BB [3], we visualize the embedding spaces of these two methods in Fig. 6 and compare them in terms of the above two conditions. We use t-SNE [30] to convert the high dimensional basic block embeddings into 2-dimensional vectors and then visualize them.

Fig. 6(a) and Fig. 6(b) visualize the distance of 100 randomly selected semantically equivalent basic block pairs in



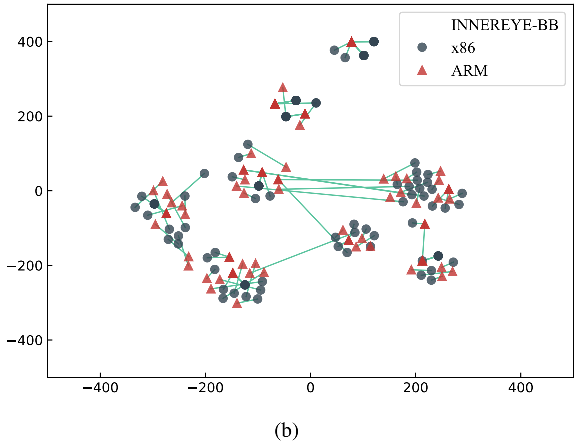


Fig. 6: Embedding vector space visualization. (a) and (b) illustrate the distance of semantically equivalent basic blocks in the embedding space generated by **MIRROR** and INNEREYE-BB, respectively. In (a) and (b), green lines connect two semantically equivalent basic blocks on x86 and ARM.

the embedding vector space of **MIRROR** and INNEREYE-BB, respectively. In these two figures, green lines connect two semantically equivalent basic blocks on x86 and ARM. Obviously, the shorter the green lines are, the closer the vector spaces of x86 and ARM basic block embeddings are. Fig. 6(a) shows a strong ability of **MIRROR** to map semantically equivalent basic blocks of x86 and ARM into approximately the same embedding vector space, and semantically inequivalent basic blocks are discrete enough in the embedding vector space. For INNEREYE-BB shown in Fig. 6(b), it not only gathers different basic blocks into several small spaces, but there is also a large gap between x86 embeddings and ARM embeddings.

To sum up, in terms of the two conditions we mentioned at the beginning of this subsection, **MIRROR** is far better than INNEREYE-BB.

2) Similarity Visualization: Fig. 7 visualizes similarities generated by **MIRROR** among 10 randomly selected semantically equivalent basic block pairs. The figure shows that if the two basic blocks are semantically equivalent, they are probably to obtain a higher similarity score than that of either of them with another basic block. Although the similarity scores of some basic block pairs (such as #1 and #5) are not very high, it is enough to find out the most similar one in all candidates. This ability is of importance for the program similarity metric methods [2], [4].

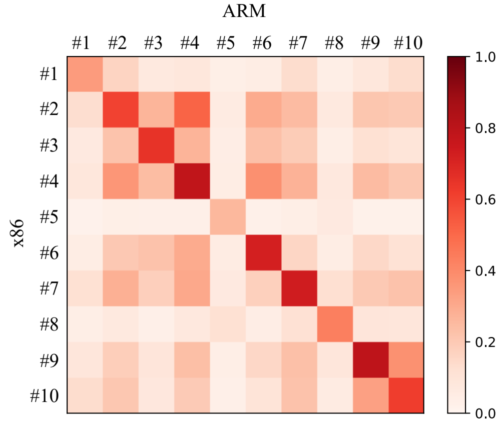


Fig. 7: Similarity Visualization.

One can also find that for semantically inequivalent basic blocks, the similarity scores of some are relatively higher than the others. This is because these basic blocks do have some similar assembly codes like #9 and #10 basic blocks.

3) Translation Case: We utilize a translation case to show what the pre-trained Transformer learned in the pre-training phase. Table X lists the source basic block (x86), the target basic block (ARM) and the translation result. It shows that:

- The pre-trained Transformer has learned the correct syntax of ARM assembly (each instruction consists of an operator and several operands).
- The pre-trained Transformer “knows” the specific usage of different instructions, i.e., it can predict a suitable operand for operators, such as select “BB” as the operand for “bgt” and “b”.
- Most of the semantics are preserved in the translation result, although it can not achieve 100% accuracy. This indicates that the pre-trained x86-encoder, i.e., the encoder of the pre-trained Transformer, needs to be fine-tuned, but is enough to be applied to hard negative sampling.

VI. Limitations

As shown in Fig.7, the similarity scores of some semantically equivalent basic blocks are a little low, although in most cases, this does not affect finding out the most similar one in candidate basic blocks. In our opinion, this issue may be addressed by adding a corresponding penalty term to the loss function shown in Equation 5. Actually, we have tried some penalty terms, but the testing results didn’t become better.

x86 (source)	ARM (target)	Translation Result
cmpl IMM ADDR jns BB	ldr REG_GEN ADDR cmp REG_GEN IMM bge BB	ldr REG_GEN ADDR cmp REG_GEN IMM bgt BB b BB

TABLE X: Translation example of the pre-trained Transformer. Here, "IMM", "BB", "ADDR" and "REG_GEN" denote the normalized immediate number, basic block label, address, and general register, respectively.

Although **MIRROR** solves a fundamental problem in program similarity analysis and achieves quite good results in evaluation, this method has not been tested on program similarity analysis tasks. This is mainly because some previous works on program similarity analysis, such as [3], [4], are not open source yet.

In future work, we plan to focus on these limitations.

VII. Conclusion

Basic block similarity metric is key to program similarity analysis. In this paper, we propose a novel cross-ISA basic block similarity metric method. This is a comprehensive solution including data collection, data pre-processing, model designing and training. The innovations of this solution include an effective basic block embedding model with a novel training mode, a reasonable instruction normalization method, and a creative negative sampling method. Our extensive evaluation shows that the prototype **MIRROR** outperforms the state-of-the-art approach by large margins with respect to the accuracy of basic block similarity metric and quality of basic block embeddings. Furthermore, the experiments justify the effectiveness of pre-training, the instruction normalization method, and the negative sampling method. As a by-product of this paper, the dataset MISA provides a comparable benchmark for relevant researches in this field, including basic block similarity metric methods and negative sampling methods.

Acknowledgment

This work was supported by National Natural Science Foundation of China (Grant Nos. 61802435 and 61802433).

References

[1] Q. Feng, R. Zhou, C. Xu, Y. Cheng, B. Testa, and H. Yin, "Scalable graph-based bug search for firmware images," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, 2016, pp. 480–491. [Online]. Available: https://doi.org/10.1145/2976749.2978370

[2] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection," *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*, pp. 363–376, 2017, arXiv: 1708.06525. [Online]. Available: http://arxiv.org/abs/1708.06525

[3] F. Zuo, X. Li, P. Young, L. Luo, Q. Zeng, and Z. Zhang, "Neural machine translation inspired binary code similarity comparison beyond function pairs," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/neural-machine-translation-inspired-binary-code-similarity-comparison-beyond-function-pairs/

[4] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 2019, pp. 3835–3845. [Online]. Available: http://proceedings.mlr.press/v97/li19d.html

[5] G. Zhao and J. Huang, "Deepsim: deep learning code functional similarity," 10 2018, pp. 141–151.

[6] L. Massarelli, G. A. D. Luna, F. Petroni, R. Baldoni, and L. Querzoni, "SAFE: self-attentive function embeddings for binary similarity," in *Detection of Intrusions and Malware, and Vulnerability Assessment - 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19-20, 2019, Proceedings*, 2019, pp. 309–329. [Online]. Available: https://doi.org/10.1007/978-3-030-22038-9_15

[7] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: learning distributed representations of code," *PACMPL*, vol. 3, no. POPL, pp. 40:1–40:29, 2019. [Online]. Available: https://doi.org/10.1145/3290353

[8] S. H. H. Ding, B. C. M. Fung, and P. Charland, "Asm2vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization," in *the 40th International Symposium on Security and Privacy (S&P19a)*, 2019.

[9] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014, pp. 1188–1196. [Online]. Available: http://proceedings.mlr.press/v32/le14.html

[10] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *arXiv:1412.3555 [cs]*, Dec. 2014, arXiv: 1412.3555. [Online]. Available: http://arxiv.org/abs/1412.3555

[12] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://www.aclweb.org/anthology/D14-1179

[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1409.0473

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[15] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *CoRR*, vol. abs/1901.02860, 2019. [Online]. Available: http://arxiv.org/abs/1901.02860

[16] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser, "Universal transformers," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. [Online]. Available: https://openreview.net/forum?id=HyzdRiR9Y7

[17] Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang, "Star-transformer," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 2019, pp. 1315–1325. [Online]. Available: https://aclweb.org/anthology/papers/N/N19/N19-1133/

[18] J. Hao, X. Wang, B. Yang, L. Wang, J. Zhang, and Z. Tu, "Modeling recurrence for transformer," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 2019, pp. 1198–1207. [Online]. Available: https://aclweb.org/anthology/papers/N/N19/N19-1122/

[19] S. Sukhbaatar, E. Grave, P. Bojanowski, and A. Joulin, “Adaptive Attention Span in Transformers,” *arXiv:1905.07799 [cs, stat]*, May 2019, arXiv: 1905.07799. [Online]. Available: http://arxiv.org/abs/1905.07799

[20] D. So, Q. Le, and C. Liang, “The evolved transformer,” in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 2019, pp. 5877–5886. [Online]. Available: http://proceedings.mlr.press/v97/so19a.html

[21] D. Kinga and J. B. Adam, “Adam: A Method for Stochastic Optimization,” in *International Conference on Learning Representations (ICLR)*, vol. 5, 2015.

[22] A. Hermans, L. Beyer, and B. Leibe, “In defense of the triplet loss for person re-identification,” *CoRR*, vol. abs/1703.07737, 2017. [Online]. Available: http://arxiv.org/abs/1703.07737

[23] Q. Xiao, H. Luo, and C. Zhang, “Margin sample mining loss: A deep learning based method for person re-identification,” *CoRR*, vol. abs/1710.00478, 2017. [Online]. Available: http://arxiv.org/abs/1710.00478

[24] W. Chen, X. Chen, J. Zhang, and K. Huang, “Beyond triplet loss: A deep quadruplet network for person re-identification,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 1320–1329. [Online]. Available: https://doi.org/10.1109/CVPR.2017.145

[25] A. Shrivastava, A. Gupta, and R. B. Girshick, “Training region-based object detectors with online hard example mining,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 761–769. [Online]. Available: https://doi.org/10.1109/CVPR.2016.89

[26] M. Henderson, R. Al-Rfou, B. Strope, Y. Sung, L. Lukács, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil, “Efficient natural language response suggestion for smart reply,” *CoRR*, vol. abs/1705.00652, 2017. [Online]. Available: http://arxiv.org/abs/1705.00652

[27] Y. Yang, S. Yuan, D. Cer, S. Kong, N. Constant, P. Pilar, H. Ge, Y. Sung, B. Strope, and R. Kurzweil, “Learning semantic textual similarity from conversations,” in *Proceedings of The Third Workshop on Representation Learning for NLP, Rep4NLP@ACL 2018, Melbourne, Australia, July 20, 2018*, 2018, pp. 164–174. [Online]. Available: https://aclanthology.info/papers/W18-3022/w18-3022

[28] G. Myles and C. S. Collberg, “K-gram based software birthmarks,” in *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC), Santa Fe, New Mexico, USA, March 13-17, 2005*, 2005, pp. 314–318. [Online]. Available: https://doi.org/10.1145/1066677.1066753

[29] B. Lu, F. Liu, X. Ge, B. Liu, and X. Luo, “A software birthmark based on dynamic opcode n-gram,” in *International Conference on Semantic Computing (ICSC 2007)*. IEEE, 2007, pp. 37–44.

[30] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.