Order Matters: Semantic-Aware Neural Networks for Binary Code Similarity Detection

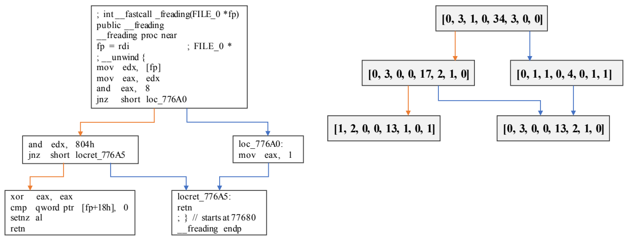
Zeping Yu,[1*] **Rui Cao,**[1*] **Qiyi Tang,**[1] **Sen Nie,**[1] **Junzhou Huang,**[2] **Shi Wu**[1†]
[1]Tencent Security Keen Lab, Shanghai, China
[2]Tencent AI Lab, Shenzhen, China
{zepingyu, ruicho}@foxmail.com, {dodgetang, snie, joehhuang, shiwu}@tencent.com

Abstract

Binary code similarity detection, whose goal is to detect similar binary functions without having access to the source code, is an essential task in computer security. Traditional methods usually use graph matching algorithms, which are slow and inaccurate. Recently, neural network-based approaches have made great achievements. A binary function is first represented as an control-flow graph (CFG) with manually selected block features, and then graph neural network (GNN) is adopted to compute the graph embedding. While these methods are effective and efficient, they could not capture enough semantic information of the binary code. In this paper we propose semantic-aware neural networks to extract the semantic information of the binary code. Specially, we use BERT to pre-train the binary code on one token-level task, one block-level task, and two graph-level tasks. Moreover, we find that the order of the CFG's nodes is important for graph similarity detection, so we adopt convolutional neural network (CNN) on adjacency matrices to extract the order information. We conduct experiments on two tasks with four datasets. The results demonstrate that our method outperforms the state-of-art models.

Introduction

Binary code similarity detection is used for detecting whether two given binary functions are similar. It has many computer security applications, including code clone detection, vulnerability discovery, malware detection, etc. Traditional efforts adopt graph matching algorithm (Liu et al. 2006) to compute the similarity of two functions. However, these graph matching-based methods are very slow, and may be hard to adapt to different applications. Recently, a neural network-based method called Gemini (Xu et al. 2017) has shown great advantages. Figure 1 (left) is an example of a CFG, Gemini first transform it into an attributed CFG with manually selected features (right) in each block. Then Structure2vec (Dai, Dai, and Song 2016) is used to produce the graph embeddings. At last a siamese architecture could be added on the binary functions to compute the similarity score and reduce loss. (Xu et al. 2017) has shown that Gemini outperforms traditional methods in terms of accuracy and speed.



```
; int __fastcall _freading(FILE_0 *fp)
public __freading
__freading proc near
fp = rdi                     ; FILE_0 *
; __unwind {
mov   edx,  [fp]
mov   eax,  edx
and   eax,  8
jnz   short  loc_776A0
```

```
and   edx,  804h
jnz   short  locret_776A5
```

```
loc_776A0:
mov   eax,  1
```

```
xor    eax,  eax
cmp   qword ptr  [fp+18h],  0
setnz  al
retn
```

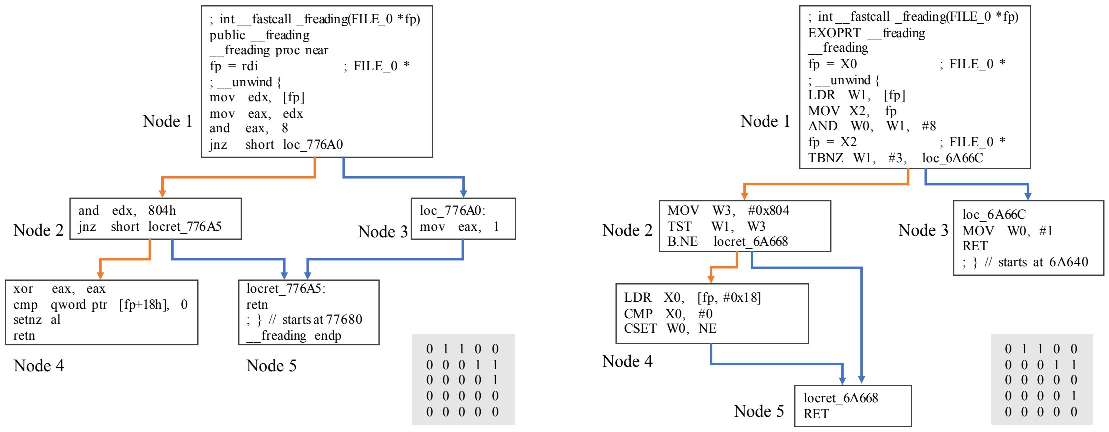
```
locret_776A5:
retn
; } // starts at 77680
__freading  endp
```

Figure 1: An example of a control flow graph (CFG) and its manually selected block features.

Even though neural network-based models have achieved a lot, there are several important things that have not been taken into consideration. Firstly, as shown in Figure 1, each block is represented as a low-dimensional embedding with manually selected features, which will cause the loss of much semantic information. Secondly, the order of the nodes plays an important role in representing binary functions, while previous approaches did not design methods to extract it. To solve these two problems, we propose an overall framework with three components: semantic-aware modeling, structural-aware modeling, and order-aware modeling.

In semantic-aware modeling, we use NLP models to extract the semantic information of the binary code. The tokens in the CFG blocks are regarded as words, and the blocks are regarded as sentences. In previous works, (Massarelli et al. 2019) uses word2vec model to train the token embeddings in the block, and then uses attention mechanism to obtain the block embedding. (Zuo et al. 2018) borrows an idea from neural machine translation (NMT) to learn the semantic relationship between cross-platform binary codes. In this paper, we adopt BERT (Devlin et al. 2018) to pre-train the tokens and blocks. Same as BERT, we mask the tokens to pre-train on the masked language model task (MLM), and extract all the neighboring blocks to pre-train on the adjacency node prediction task (ANP). Instead of learning token embeddings and block embeddings separately, our method could

*Equal contribution
†Corresponding author



```
; int __fastcall _freading(FILE_0 *fp)
public __freading
__freading proc near
fp = rdi                        ; FILE_0 *
; __unwind {
mov   edx,  [fp]
mov   eax,  edx
and   eax,  8
jnz   short  loc_776A0
```

```
and   edx,  804h
jnz   short  locret_776A5
```

```
loc_776A0:
mov   eax,  1
```

```
xor   eax,  eax
cmp   qword ptr  [fp+18h],  0
setnz  al
retn
```

```
locret_776A5:
retn
; } // starts at 77680
__freading  endp
```

$$\begin{array}{ccccc} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}$$

```
; int __fastcall _freading(FILE_0 *fp)
EXOPRT __freading
__freading
fp = X0                         ; FILE_0 *
; __unwind {
LDR   W1,  [fp]
MOV   X2,  fp
AND   W0,  W1,  #8
fp = X2                         ; FILE_0 *
TBNZ  W1,  #3,  loc_6A66C
```

```
MOV   W3,  #0x804
TST   W1,  W3
B.NE  locret_6A668
```

```
loc_6A66C
MOV   W0,  #1
RET
; } // starts at 6A640
```

```
LDR   X0,  [fp, #0x18]
CMP   X0,  #0
CSET  W0,  NE
```

```
locret_6A668
RET
```

$$\begin{array}{ccccc} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 2: Two CFGs and their adjacency matrices of function ”_freading” on different platforms (x86-64 & ARM).

obtain token embeddings and block embeddings at the same time. Additionally, because our final goal is to produce the whole graph representation, we add two graph-level tasks. One is to determine whether two sampled blocks are in the same graph, and we call it block inside graph task (BIG). The other is to distinguish which platform/optimization the block belongs to, which is named graph classification task (GC). We find that the additional tasks could help extract more semantic information and learn block representation better. After pre-training the block embeddings, we fine-tune them on graph-level tasks.

In structural-aware modeling, we use MPNN (Gilmer et al. 2017) with GRU (Cho et al. 2014) update function. (Xu et al. 2018) has proved that graph neural networks could have as discriminate power as Weisfeiler-Lehman test (Weisfeiler and Lehman 1968). We find that using GRU at each step could store more information than only using a tanh function.

In order-aware modeling, we try to design an architecture to extract the node order information of the CFGs. Figure 2 shows a function's two CFGs and the corresponding adjacency matrices on different platforms x86-64 and ARM. The two CFGs have similar node order. For example, in both CFGs node 1 has a connection with node 2 and 3, and node 2 has a connection with node 4 and 5. Their adjacency matrices are very alike. After exploring many cross-platform function pairs, we find that many changes in node order are small. Based on this observation, we propose a simple method to capture the order information, that is, using CNN on the adjacency matrices. We find that only a 3-layer CNN performs well. We further explore other CNN models such as Resnet (He et al. 2016), and discuss what CNN models could learn from adjacency matrices.

Our contributions are as follows:

1) We propose a general framework to learn graph embeddings of CFGs, which could learn semantic information, structural information, and order information.

2) In semantic-aware modeling, we adopt BERT to pre-train token embeddings and block embeddings with masked language model task (MLM) and adjacent node prediction task (ANP). Additionally we add two graph-level tasks to learn block representation better, which are block inside graph task (BIG) and graph classification task (GC).

3) In order-aware modeling, we find that the node order is useful. We adopt CNN models on the adjacency matrices to extract the node order information of CFGs, which makes great achievements. Then we explore what CNN could learn from adjacency matrices.

4) We conduct experiments on two tasks with four datasets, and the results demonstrate that our proposed model achieves much better performance than the previous methods.

Related Work

Graph Neural Networks

Graph neural networks (Scarselli et al. 2008) propose to learn the node representation and graph representation. By adding deep learning components, there are many graph models, such as graph convolutional network (GCN) (Kipf and Welling 2016), GraphSAGE (Hamilton, Ying, and Leskovec 2017), and graph attention network (GAT) (Veličković et al. 2017). GCN uses a convolutional layer to update the node embeddings. GraphSAGE adopts an aggregating function to merge the node and its neighboring nodes. GAT uses attention mechanism to receive more information from the important nodes. MPNN designs an overall framework for graph representation learning, which has a message passing phase and a readout phase. The message passing phase runs several steps to capture the information from the neighboring nodes. The readout phase computes an embedding for the whole graph. Except MPNN, graph networks (GN) (Battaglia et al. 2018) and non-local neural networks (NLNN) (Wang et al. 2018) are also overall frameworks for graph learning.

BERT

BERT is the state-of-art pre-training model in NLP. BERT makes use of Transformer (Vaswani et al. 2017), which is an attention mechanism that learns contextual relations between words in a text. BERT use two strategies to train a

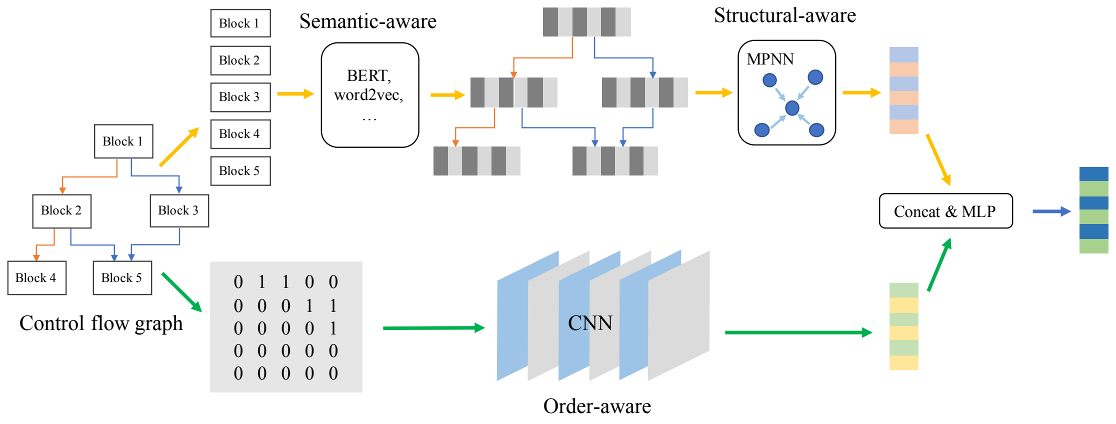


Figure 3: Overall structure of our model. The model has three components: semantic-aware modeling, structural-aware modeling and order-aware modeling.

language model. One is the masked language model task (MLM), which is a self-supervised prediction mask to encourage the model to capture useful knowledge about language. The other is a classification task to make the model distinguish the two sentences in training, which is called next sentence prediction (NSP). BERT can be used for a wide variety of language tasks by only adding a small layer to the core model. On NSP task, [cls] token is usually regarded as the sentence embedding, and a mapping layer could be added for fine-tuning. BERT pre-trained models have achieved very good results on a wide range of downstream tasks, such as cross-lingual language model (Lample and Conneau 2019), question answering (Talmor et al. 2018), and text generation (Song et al. 2019).

Binary Code Similarity Detection

Binary code similarity detection is an important task in computer security. Traditional methods use graph matching algorithms to compute the graph similarity. However, these methods are slow and inefficient. Several researches try to use graph kernel methods (Weisfeiler and Lehman 1968; Borgwardt et al. 2005). Recently (Xu et al. 2017) proposed a GNN-based model called Gemini, which gets better results than the previous methods. But it uses manually selected features to represent the CFG blocks, which may not contain enough semantic information. (Zuo et al. 2018) uses NLP models on this task. They treat a token as a word and treat a block as a sentence, and use LSTM to encode the semantic vector of a sentence. To make sure blocks with the same semantic information have similar representations, they use a siamese network and compute cosine distance for CFG pairs. They consider two basic blocks that have been compiled from the same piece of source code as equivalent. To get the ground truth block pair, they modify the compiler to add the basic block special annotator which annotates a unique ID for each generated assembly block. However, this method has two obvious shortcomings. One is that getting similar block pair is an supervised process which needs expert experience and domain knowledge, and some blocks cannot be uniquely annotated. The other is that different models need to be trained for different platform combinations in practical use.

Our Model

Overall Structure

The input of our model is the binary code functions' CFGs, in which each block is a token sequence with intermediate representation. The overall structure of our model is shown in Figure 3. On the semantic-aware component, the model takes the CFG as input and uses BERT to pre-train the token embeddings and the block embeddings. On the structural-aware component, we use MPNN with GRU update function to compute the graph semantic & structural embedding g_{ss}. On the order-aware component, the model takes the adjacency matrix of the CFG as input, and adopts CNN to compute the graph order embedding g_o. At last we concatenate them and use a MLP layer to compute the graph embedding g_{final}.

$$g_{final} = \text{MLP}([g_{ss}, g_o]) \tag{1}$$

Semantic-aware Modeling

In semantic-aware modeling, we propose a BERT pre-training model with 4 tasks for dealing with CFGs. This model is of several advantages. First, we can extract block embedding from different CFGs which generated from different platforms, different architectures, different compile optimization options based on the same model. Second, we can get both token-level, block-level and graph-level information from pre-training process, because we have one token-level task, one block-level task and two graph-level tasks. Third, the training process is based entirely on the CFG diagram and does not require modification of the compiler or other operations to get similar block pairs.

Our approach is inspired from sentence embedding task in NLP, because the blocks in CFGs are like sentences and the tokens in blocks are like words. This task is to extract

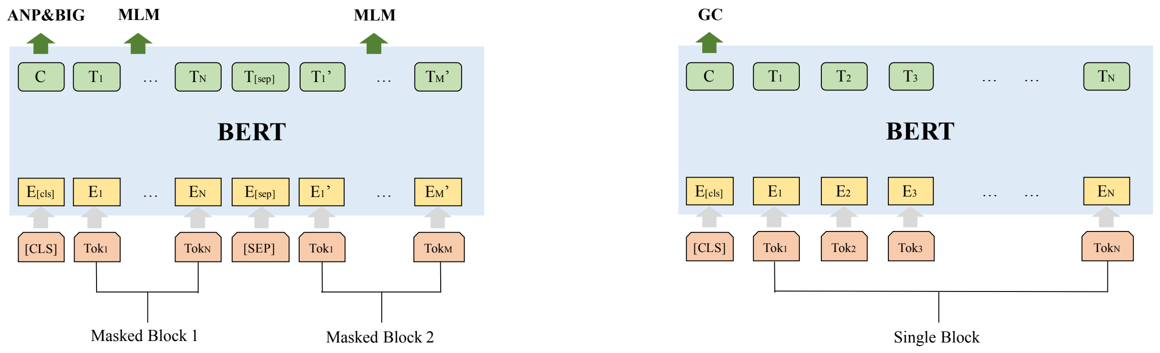


Figure 4: Bert with 4 tasks: MLM, ANP, BIG and GC.

embeddings for a sentence. There are two main methods for this task. One is supervised methods such as text classification training (Joulin et al. 2016). Another is unsupervised methods such as n-gram features and decoder-encoder skip thought (Kiros et al. 2015). We use the improved model based on BERT to extract block embedding on CFGs.

As shown in Figure 4, there are four tasks in our pre-training process. For the token sequences inside the node, we employ Masked language model (MLM) to extract semantic information inside the block. MLM is a token-level task, which masks the tokens on the input layer and predict them on the output layer. Adjacency node prediction task (ANP) is a block-level task. In graphs, the information of the block is not only related to the information of the block itself but also its neighbors, and we want our model to learn this information. In ANP task, we extract all adjacent blocks on a graph and randomly sample several blocks in the same graph to predict whether two blocks are adjacent. The two tasks (MLM & ANP) are similar to the MLM & NSP tasks in the original BERT paper (Devlin et al. 2018).

In order to make better use of the information on the graph, we add two auxiliary supervised tasks: block inside graph task (BIG) and graph classification task (GC). BIG task is similar with ANP. The differences are the ways of sampling different block pairs. BIG task tries to make the model judge whether two nodes exist on the same graph. We randomly sample block pairs in/not in the same graph, and predict them in BIG task. This task helps the model understand the relationship between block and graph, which is helpful for our graph embedding tasks. In our scenario, under different compilation options, the information of the graph and the blocks may be different. In order to make the model to distinguish these differences, we design graph classification task (GC). GC makes the model to classify blocks in different platforms, different architectures, or different optimization options.

Structural-aware Modeling

After obtaining the block embeddings from BERT pre-training, we use MPNN to compute the graph semantic & structural embedding of each CFG. The message passing phase with message function M and update function U runs for T time steps, and then the readout function R computes the whole graph semantic & structural embedding g_{ss}.

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{2}$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{3}$$

$$g_{ss} = R(h_v^T | v \in G) \tag{4}$$

Here G means the whole graph, v means the node, and N(v) means the neighboring nodes of node v. We use MLP on message function M, and adopt GRU on update function U to learn the sequential information of the time iteration. (Xu et al. 2018) has proved that sum function is the best choice for readout function R, so we use sum function, and extract the graph representation of 0th step and Tth step. h_v^0 means the initial block embeddings generated by BERT pre-training, and h_v^t means the block embeddings at step t.

$$m_v^{t+1} = \sum_{w \in N(v)} \text{MLP}(h_w^t) \tag{5}$$

$$h_v^{t+1} = \text{GRU}(h_v^t, m_v^{t+1}) \tag{6}$$

$$g_{ss} = \sum_{v \in G} \text{MLP}(h_v^0, h_v^T) \tag{7}$$

Order-aware Modeling

In this component we aim to extract the node order information of the CFGs. We could think about what information could be learned by CNN models. Figure 5 shows three graphs (without semantic information in blocks) and their adjacency matrices, which could be transferred into each other by adding several small changes. In the three graphs, each graph has a triangle. We could observe that the triangle feature, which is squared in each adjacency matrix, has some common characteristics in adjacency matrices. First consider 5a and 5b, a new node is added onto the triangle, but the node order of the triangle is not broken. Even though the displacement changes, the triangle feature (the 1, 1, 0, 1 square) is not changed in the two adjacency matrices. CNN could capture this information, because CNN has translation invariance when it has seen a lot of training data. Then look at 5c, it seems that the added node 2 breaks the triangle's node order. However, when we remove the adjacency

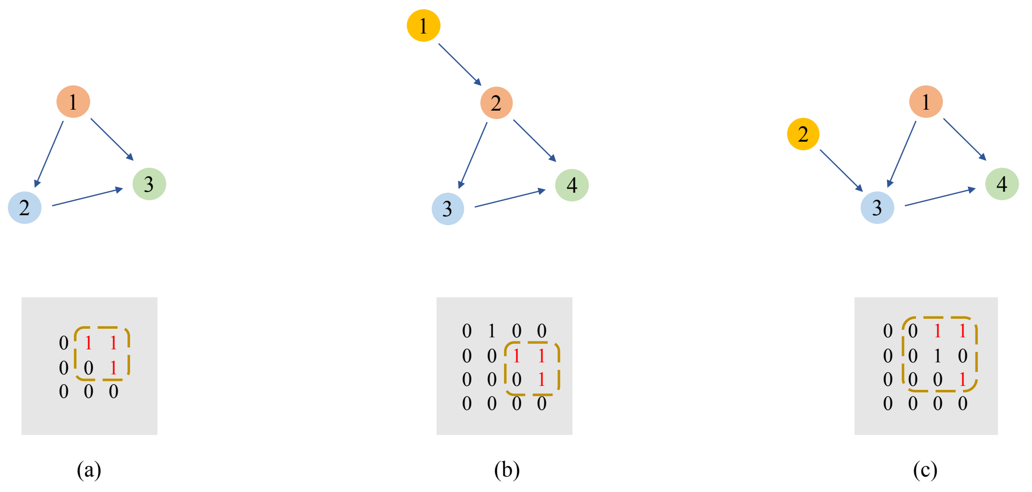


Figure 5: Example graphs and their adjacency matrices. The triangle feature of all the three graphs could be squared in their adjacency matrices.

matrix's second row and second column, the triangle feature square still exists. This is like image scaling. If we regard the triangle feature square as a 2*2 image, it is amplified into a 3*3 image in 5c. CNN could also learn this scale invariance after seeing enough training data.

We have discussed that CNN may learn the small changes of node order because of its translation invariance and scale invariance. In our binary code similarity detection task, the node order usually does not change a lot when the same function is compiled on different platforms. Most node order changes are adding a node, deleting a node, or exchanging several nodes, so CNN is useful on our task. Except the accuracy improvement on learning node order information, CNN has several additional advantages. First, comparing with traditional graph feature extracting algorithms, using CNN directly on adjacency matrices is much faster. Second, CNN could be added on inputs with different sizes, so it could model different-size graphs without pre-processing such as padding and clipping.

$$g_o = \text{Maxpooling}(\text{Resnet}(A)) \tag{8}$$

We use Resnet (He et al. 2016) on the adjacency matrices A in our tasks. Resnet uses a shortcut connection to help the information transmit easily and efficiently. We use a 11-layer Resnet with 3 residual blocks. All the feature maps are 3*3 because we want to learn the small changes of the graphs. Then we use a global max pooling layer to compute the graph order embedding. Note that we do not use any pooling methods unless on the last layer, because the inputs have different sizes. In our experiments we observe that using the power of adjacency matrices as additional inputs could help improve performance. To prevent over-fitting, we do not use them in this paper. (Nguyen et al. 2018) also adopts CNN on graphs, but they try to use CNN to learn semantic features. In our approach we want CNN to learn the node order information, so only adjacency matrices are taken as inputs.

Dataset	Task	Training	Validation	Testing
gcc-O2	1	31,410	3,857	3,884
gcc-O3	1	16,059	4,155	4,077
gcc-x86-64	2	27,761	3,406	3,492
gcc-ARM	2	9,447	4,773	4,933

Table 1: Basic statistics of the datasets.

Experiments

Datasets

We evaluate our model on two tasks. The first task is cross platform binary code detection (Pewny et al. 2015). The same source code is compiled into different CFGs on different platforms. Our goal is to make sure the same source code has higher similarity scores than others. Same as Gemini (Xu et al. 2017), we use a siamese network on our graph embedding model to reduce loss and use cosine distance to compute the graph similarity. We choose x86-64 and ARM as the two platforms, and compile on gcc with O2 and O3 optimization options. The second task is graph classification, in this task we classify the optimization option of the graph embeddings. We use softmax function and choose cross entropy as loss function. We use x86-64 and ARM on gcc with O0 - O3. Note that our method is also useful for detecting binary codes on different compilers (eg: clang & gcc), in this paper we do not choose it as a dataset. The basic statistics of datasets are shown in Table 1.

Compared Methods

Because our model has three components: semantic-aware modeling, structural-aware modeling, and order-aware modeling, we conduct different experiments to find out the effect of each part.

Graph kernel methods We choose the Weisfeiler-Lehman

Model	Task1-O2	Task1-O3
Weisfeiler-Lehman	0.2493 / 0.1810	0.1940 / 0.1565
Gemini	0.6069 / 0.5491	0.5430 / 0.4760
MPNN	0.6096 / 0.5507	0.5501 / 0.4802
word2vec	0.7003 / 0.6534	0.6198 / 0.5555
skip thought	0.6825 / 0.6238	0.5954 / 0.5226
BERT2	0.7591 / 0.7060	0.6507 / 0.5852
BERT4	0.7704 / 0.7233	0.6672 / 0.5989
CNN3 (random)	0.0362 / 0.0020	0.0307 / 0.0015
CNN3	0.4142 / 0.3684	0.3074 / 0.2612
Resnet7	0.4330 / 0.3868	0.3229 / 0.2732
Resnet11	0.4419 / 0.3952	0.3271 / 0.2837
MPNN$_{ws}$	0.3361 / 0.3014	0.2161 / 0.1913
MPNN$_{ws}$+Resnet11	0.4457 / 0.3970	0.3348 / 0.2907
Our model	**0.7922 / 0.7421**	**0.6855 / 0.6114**

Table 2: Performance comparison of task 1 in terms of MRR10 / Rank1.

kernel (Weisfeiler and Lehman 1968) to compute the similarities of the graphs.

Gemini (Xu et al. 2017) uses Structure2vec to compute the graph embedding of CFGs, in which each block is a 8-dimensional manually selected feature.

MPNN To compare our semantic-aware & structural-aware model with Gemini, we use MPNN (Gilmer et al. 2017) with 8-dimensional feature.

Word2vec Word2vec (Mikolov et al. 2013) is a fundamental method to learn word embeddings in NLP. We take the sum of the token embeddings as the block embedding.

Skip thought Skip thought (Kiros et al. 2015) is another method to learn sentence embeddings in NLP. It takes the middle sentence as input and uses sequence-to-sequence model to predict its previous and later sentence.

BERT (2 tasks) We use BERT (Devlin et al. 2018) to pre-train the block embeddings of CFGs. The two pre-training tasks are the MLM task and the ANP task.

BERT (4 tasks) Except the MLM task and the ANP task, we add two graph-level tasks (BIG and GC) to learn the graph-level information.

CNN-based models To figure out the effect of order-aware modeling, we only use CNN-based models on adjacency matrices. We use 3-layer CNN, 7-layer Resnet, and 11-layer Resnet to evaluate whether CNN-based models are useful. To reduce parameters and prevent over-fitting, we do not use larger CNN models.

CNN (random) To see whether CNN could capture the node order information, we random shuffle each CFG and feed the corresponding adjacency matrices into CNN.

MPNN (without semantic) (Xu et al. 2018) has shown that GNN models could learn the structural information of the graphs. We use MPNN without semantic information, each block has the same original input (a random vector).

MPNN (without semantic) + CNN We concatenate the CNN node order embedding and the GNN embedding, to see whether these models could have better results when they are used together.

Model	Task2-x86-64	Task2-ARM
Weisfeiler-Lehman	-	-
Gemini	77.88	79.89
MPNN	79.65	80.62
word2vec	82.24	84.23
skip thought	80.43	83.74
BERT2	82.67	85.19
BERT4	83.74	86.33
CNN3 (random)	66.06	64.57
CNN3	82.11	83.70
Resnet7	82.56	84.13
Resnet11	82.64	84.24
MPNN$_{ws}$	76.29	76.90
MPNN$_{ws}$+Resnet11	82.92	85.05
Our model	**86.14**	**88.41**

Table 3: Performance comparison of task 2 in terms of accuracy (%).

Our model Our model is BERT (4 tasks) + MPNN + 11-layer Resnet, which contains both semantic-aware modeling, structural-aware modeling, and order-aware modeling.

Training We implement the models with Tensorflow. Optimizer is Adam. Dimension for BERT pre-training embedding is 128, while the whole graph embedding is 64. Max sequence length for BERT pre-training is 128, transformer depth is 12 and feed-forward dim is 256. We adopt grid search to find the best hyper-parameters for each model with validation set. Optimal settings for our model are: *learning rate* is 0.0001, *batch size* is 10, *iteration time steps* is 5.

Evaluation metrics On task 1, our goal is to find the binary codes compiled by the same source code on different platforms. This task is similar to the recommender system, thus we use Rank1 and mean reciprocal rank (MRR) as the evaluation metrics. Rank1 means whether the rank of the true pair has the highest score. MRR is used for evaluating ranking tasks, which uses the multiplicative inverse of the rank of the first correct answer. Task 2 is a classification task, so we use accuracy to evaluate our model.

Results

Overall performance Table 2 and Table 3 show the overall performance of different models on two tasks. BERT 2 & 4 is short for BERT with 2 tasks (MLM & ANP) and with 4 tasks (MLM & ANP & BIG & GC), MPNN$_{ws}$ is short for MPNN without semantic. The first block contains previous methods, the second block shows the results of semantic-aware modeling, and the third block shows the effect of order-aware modeling. Comparing with Gemini, our model achieves much better performance on both tasks. MPNN outperforms Gemini on all the datasets, this is because the GRU update function could store more information, so in all the other models we use MPNN. We could observe that

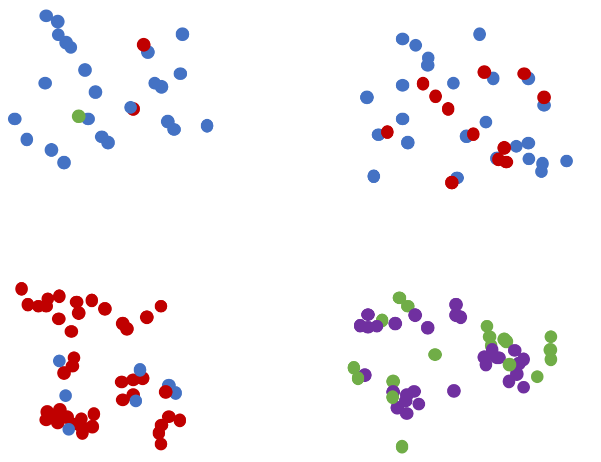
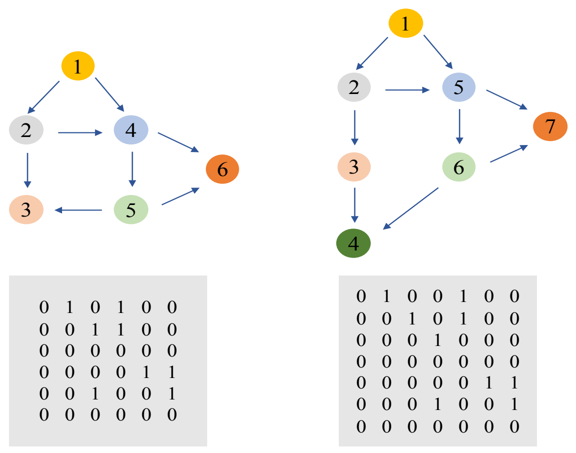


Figure 6: Visualization of 4 CFGs and their block embeddings. Different classes clustered by K-means have different colors.


$$\begin{matrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Figure 7: An example of CFG changes, these two CFGs' cosine similarity score is 0.971.

NLP-based block pre-training features are much better than manual features, and order-aware modeling also has good results on two tasks. On cross platform binary code detection task, semantic information is more useful than order information. Different CFGs may have similar node orders, so only using node order information is not enough. At last, our final model outperforms all the other models. Next we will investigate the effectiveness of each component separately.

Model variants for semantic-aware modeling To verify whether BERT pre-training is necessary and effective, we study several variants. First, NLP-based pre-training block features (word2vec, skip thought, BERT 2 & 4) achieve better performance than manual features, which demonstrates that building complex models for CFG blocks is essential. Comparing with word2vec and skip thought, BERT with MLM and ANP tasks considers not only block-level prediction, but also token-level prediction, and the bidirectional transformer has more ability to extract useful information. The BIG task and GC task are also useful, whose results has 1% - 2% increase. In these two tasks the block embeddings could learn graph-level information, which could help with graph-level tasks. We show the block embeddings in Figure 6. Four CFGs and their block embeddings are set in four directions. We adopt K-means to cluster these block embeddings into four classes, and different clusters have different colors (red, blue, green and purple). We could observe that the blocks in the same graph trend to have the same color, and different graphs have different main colors.

Model variants for order-aware modeling Only using CNN-based models could achieve good results on both two tasks. 11-layer Resnet is a little better than 3-layer CNN and 7-layer Resnet. Comparing with MPNN$_{ws}$, CNN-based models achieve much better performance. When random shuffling the nodes, CNN could learn nothing. This means CNN models could learn the node order of graphs, and adding CNN on adjacency matrices is meaningful for extracting node order information. Moreover, using MPNN and CNN together could have better results, this means our structural-aware component is also useful. To explore what the CNN models learned, we show an example of CFG changes in Figure 7. The two CFGs are compiled by the same source code, whose name is "_ZN12libfwbuilder15RuleElementRGtw-13validateChildEPNS_8FWObjectE". To save space, we show the graphs without semantic codes. The left one is compiled on gcc and x86-64, while the right one is compiled on gcc and ARM. On different platforms, the code is compiled into different CFGs: node 3 in the left graph is split into node 3 and 4 in the right graph. In their adjacency matrices, the node order of nodes "1, 2, 3" and "4, 5, 6" could be captured. By extracting features from the adjacency matrices, our CNN model learns that these two CFGs' cosine similarity score is 0.971, and the detecting rank among the codes on the whole platform is 1. This means CNN could extract the node order information from the adjacency matrices, which matches our assumption.

Conclusion

In this paper, we propose a novel framework for binary code graph learning, which contains semantic-aware component, structural-aware component and order-aware component. We observe that semantic and node order information are both important for representing CFGs. To capture the semantic feature, we propose BERT pre-training for the blocks of CFGs with two original tasks MLM & ANP, and two additional graph-level tasks BIG & GC. Then we use MPNN to extract the structural information. We further propose a CNN-based model to capture the node order information. We have conducted experiments on two tasks with four datasets, and the experiments demonstrate that our proposed model outperforms the state-of-art methods.

References

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Borgwardt, K. M.; Ong, C. S.; Schönauer, S.; Vishwanathan, S.; Smola, A. J.; and Kriegel, H.-P. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21(suppl_1):i47–i56.

Cho, K.; Van Merriënboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Dai, H.; Dai, B.; and Song, L. 2016. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, 2702–2711.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1263–1272. JMLR. org.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 1024–1034.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Joulin, A.; Grave, E.; Bojanowski, P.; and Mikolov, T. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kiros, R.; Zhu, Y.; Salakhutdinov, R. R.; Zemel, R.; Urtasun, R.; Torralba, A.; and Fidler, S. 2015. Skip-thought vectors. In *Advances in neural information processing systems*, 3294–3302.

Lample, G., and Conneau, A. 2019. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.

Liu, C.; Chen, C.; Han, J.; and Yu, P. S. 2006. Gplag: detection of software plagiarism by program dependence graph analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 872–881. ACM.

Massarelli, L.; Di Luna, G. A.; Petroni, F.; Baldoni, R.; and Querzoni, L. 2019. Safe: Self-attentive function embeddings for binary similarity. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 309–329. Springer.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Nguyen, M. H.; Le Nguyen, D.; Nguyen, X. M.; and Quan, T. T. 2018. Auto-detection of sophisticated malware using lazy-binding control flow graph and deep learning. *Computers & Security* 76:128–155.

Pewny, J.; Garmany, B.; Gawlik, R.; Rossow, C.; and Holz, T. 2015. Cross-architecture bug search in binary executables. In *2015 IEEE Symposium on Security and Privacy*, 709–724. IEEE.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1):61–80.

Song, K.; Tan, X.; Qin, T.; Lu, J.; and Liu, T.-Y. 2019. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*.

Talmor, A.; Herzig, J.; Lourie, N.; and Berant, J. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, X.; Girshick, R.; Gupta, A.; and He, K. 2018. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7794–7803.

Weisfeiler, B., and Lehman, A. A. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia* 2(9):12–16.

Xu, X.; Liu, C.; Feng, Q.; Yin, H.; Song, L.; and Song, D. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 363–376. ACM.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Zuo, F.; Li, X.; Zhang, Z.; Young, P.; Luo, L.; and Zeng, Q. 2018. Neural machine translation inspired binary code similarity comparison beyond function pairs. *arXiv preprint arXiv:1808.04706*.