IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 22 August 2023, accepted 12 September 2023, date of publication 18 September 2023,
date of current version 26 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3316215

RESEARCH ARTICLE

Feature Extraction Methods for Binary Code Similarity Detection Using Neural Machine Translation Models

NORIMITSU ITO[1,2], MASAKI HASHIMOTO[2], (Member, IEEE), AND AKIRA OTSUKA[2], (Member, IEEE)

[1]Police Info-Communications Research Center, National Police Academy, Fuchu, Tokyo 183-0003, Japan
[2]Institute of Information Security, Yokohama, Kanagawa 221-0835, Japan

Corresponding author: Masaki Hashimoto (hashimoto@iisec.ac.jp)

ABSTRACT Binary code similarity detection is an effective analysis technique for vulnerability, bug, and plagiarism detection in software for which the source code cannot be obtained. The recent proliferation of IoT devices has also increased the demand for similarity detection across different architectures. However, there are currently not many examples of feature extraction methods using neural machine translation (NMT) models being applied to similarity detection in basic block units across different architectures. In this research, we propose new methods that extract features at a higher speed and detect similarities across different architectures with higher accuracy than existing methods for basic block feature extraction using neural machine translation models. We assume that the intermediate representation of the NMT model, which learned the translation of basic blocks across different architectures, includes the semantics of the instructions in the basic block. Hence we adopted the intermediate representation as the features of the basic blocks. Then, we applied the linear transformation used in bilingual word embedding to match the embedding space of basic blocks across different architectures. This enables the similarity detection in basic block units across different architectures with higher accuracy than the distance learning method used in existing research to match the embedding space. In the evaluation experiment, we compare the Precision at k (P@k) on the same dataset with existing research methods and our method achieved the highest accuracy of 92%. In addition, We also compare the time required for feature extraction using GPUs, and found that it was up to 16 times faster.

INDEX TERMS Binary code similarity detection, machine learning, neural machine translation.

I. INTRODUCTION

In software vulnerability, bug, and plagiarism detection, various methods for comparing source code have been proposed for many years in order to detect similarity of function and behavior [1], [2], [3]. However, the source code of commercial software and pirated software is often not disclosed. In such cases similarity comparisons are performed using only binary code such as one in executable files. Similarity comparisons using only binary code are more difficult than those with source code and human analysis requires more time to obtain results. This can lead to errors in the analysis results. For this reason, many proposals have been made not only for source code similarity detection, but also for binary code similarity detection [4], [5]. Furthermore, recent extensive studies utilize machine learning and natural language processing techniques for binary code similarity detection.

However, the rapid proliferation of IoT devices [6] has led to a wide range of Instruction Set Architectures (ISA) used for binary code. These include ARM, MIPS and RISC-V for embedded systems, in addition to x86 and x86_64 for general personal computers. This means that if source code containing a vulnerability exists, it may be cross-compiled and provided for multiple ISAs. When a vulnerability is found in software for a certain ISA, a binary

The associate editor coordinating the review of this manuscript and approving it for publication was Yassine Maleh.

code similarity detection method across different ISAs helps to detect whether the vulnerability is also present in software for different ISAs. However, there are few applications of similarity detection using natural language features extracted by applying a Neural Machine Translation (NMT) model to this method [7], [8], [9], [10].

Zuo et al. [11] are the first to introduce an NMT model in distance learning for binary code similarity comparisons across different ISAs. Zhang et al. [12] then improved the accuracy by training translations of basic blocks across different ISAs as a pre-training for the NMT model for distance learning. However, the P@k is 77% at best, leaving room for improvement in accuracy, and the NMT model was not lightweight. Various problems would arise when non-lightweight models were used in actual binary code similarity comparison tools. For example, it takes time to analyze software that contains tens of thousands of basic blocks, making it difficult to install in IoT devices and devices with limited resources. Therefore, we decided to develop feature extraction methods using a lightweight model with a target accuracy of over 90%. We also studied another method that does not require a linear transformation as a comparison.

In this study, we propose feature extraction methods for basic blocks using NMT models to improve the accuracy of binary code similarity detection across different ISAs. We applied one training method that requires a linear transformation of features and four training methods that do not require a linear transformation to GRU [13]- or Transformer [14]-based NMT models. We then evaluated a total of 10 feature extraction methods. In these methods, the NMT model learns x86_64-ARM translations as a natural language in which x86_64 and ARM instructions are words and basic blocks are sentences. The intermediate representation is then taken as the features.

In the field of natural language processing, NMT models have allowed translation between different languages with high accuracy [14]. The intermediate representation is considered to contain the semantics of the sentence. Our proposed methods use an intermediate representation of the NMT model trained to translate across different architectures to compare the similarity of basic blocks between different architectures. We attempt to use the semantics of the basic block for similarity comparisons, based on the idea that the intermediate representation contains the semantics of the instruction sequence in the basic block, similar to Zhang et al. [12]. Since the basic block features are not selected by humans, it is not possible to identify which features are extracted in the intermediate representation. However, we believe that hidden features that cannot be grasped by humans can also be extracted. In addition, since insertion of vulnerabilities or bugs, or code plagiarism do not always occur in function units [11], we compare similarities by a smaller unit of basic block. We use cosine similarity as the measure for similarity comparisons. In the intermediate layer of the neural network, layer normalization is performed to improve learning efficiency. Considering that the magnitude of the vector output from the final layer is not very significant, we chose a similarity measure that takes into account the direction of the vector. Although we use cosine similarity, which is relatively computationally expensive among similarity measures, it accounts for only a small percentage of the computational effort of a neural network. Therefore, the computational effort of the cosine similarity has not been a problem.

The contributions of this research are summarized in the following three points.

- We utilized Mikolov et al.'s linear transformation [15] to the similarity detection method of basic blocks across different ISAs. By doing so, similarity was detected with higher accuracy than Zhang et al.'s MIRROR [12] and Zuo et al.'s INNEREYE-BB [11] which perform distance learning.
- We compared the following two types of methods for basic block similarity detection across different ISAs. One is Mikolov et al.'s method that utilizes linear transformation, and the other is a method that incorporates autoencoder training instead of linear transformation. As a result, we showed that the accuracy of the method utilizing linear transformation is higher.
- We compared the time required for feature extraction among our methods, INNEREYE-BB by Zuo et al. and MIRROR by Zhang et al., using a GPU. The results indicate that our methods are able to extract features in the shortest time.

This paper is organized as follows: Section II describes research related to binary code similarity detection. Section III describes our proposed methods. Section IV describes evaluation experiments for comparison with the baselines, and discussions on the results. Section V describes the discussions of the proposed methods, and Section VI summarizes our study.

II. RELATED RESEARCH

A. BINARY CODE SIMILARITY DETECTION BASED ON FEATURES

Research on feature-based binary code similarity detection includes methods that compare similarities per function [16], [17], [18], [19], [20], [21], [22], [23], [24] and methods that compare similarities per basic block [11], [12], [25]. These methods use machine learning to learn instructions, basic blocks, and Control Flow Graph (CFG) embeddings, and measure the similarity of feature vectors obtained with Euclidean distance, Manhattan distance, Jaccard coefficient, cosine similarity, and so on.

Zuo et al. [11], Redmond et al. [25], Ding et al. [19], and Massarelli et al. [20], [21] used word2vec or doc2vec-based methods that learn distributed representations of natural languages for instruction embedding. This is because the semantics of binary code are not sufficiently included in

human-designed features such as those used by Feng et al. [16], Xu et al. [17], and Gao et al. [18]. Other NLP-based methods include Transformer [14], a deep learning model, utilized by Zhang et al. [12], Yu et al. [22], Masubuchi et al. [23], and Wang et al. [24] and its pre-training model, BERT [26], to embed binary code.

B. BINARY CODE SIMILARITY DETECTION USING NMT MODELS

Feature extraction methods, which compare the similarity of basic blocks across different ISAs, incorporate learning using NMT models in order to extract more basic block semantics. Similarity detection of natural language sentences has also been studied using NMT models [7], [8], [9], [10]. The method of using the intermediate representation of the NMT model to compare similarities between sentences has also been applied to binary code.

1) METHOD OF ZUO ET AL.: INNEREYE-BB

Zuo et al. configured the Long-Short Term Memory (LSTM) with a Siamese network to embed features of basic blocks of different ISAs in the same vector space in order to enable similarity comparisons. Features of basic blocks are the final state of LSTM. Their method uses an NMT model, however it does not learn translations across different ISAs. Instead, it learns feature vectors for basic block pairs with similar semantics to be close together, and feature vectors for basic block pairs without similar semantics to be far apart. As a result, this achieved an AUC of 98% for the ROC curve when similar and dissimilar pairs were identified. However, Zhang et al. argue that this result is due to the large difference between similar and dissimilar pairs in the test data [12]. Also, most of the instructions at the end of basic blocks are jump instructions, return instructions, and jump destination symbols. Therefore, if the final state of LSTM is an intermediate representation of the basic block, there is a problem that the differentiation of embedding is limited [12].

2) METHOD OF ZHANG ET AL.: MIRROR

The model used by Zhang et al. is constructed with Transformer, and the output of the encoder is used as the basic block features. In this method, only one of the two NMT models prepared for each ISA is trained in advance to translate basic blocks across different ISAs. In order to make the trained encoder output and the untrained encoder output close to each other, the basic block features of different ISAs are embedded in the same vector space by training with triplet loss using hard negative samples. As a result, MIRROR's P@1 was 77% in the Zhang et al.'s experiment to evaluate Precision at K (P@K) when retrieving similar blocks from 100 basic blocks. Whereas the P@1 of INNEREYE-BB was 51%. INNEREYE-BB tokenizes basic blocks per instruction when they are input into the model. MIRROR, on the other hand, tokenizes per operand or opcode, which are further division of instructions. Therefore, MIRROR has the problem of longer sequence lengths and longer embedding time for the basic block [27].

III. PROPOSED METHOD

This chapter describes one type of model that utilizes the linear transformation of Mikolov et al. [15] used in bilingual word embedding as a similarity detection method for binary code. This is inspired by Seki et al.'s sentence similarity detection method using natural language [9]. In addition, it also describes four other models for which we added auto-encoder training to eliminate the need for linear transformations of the first model. Furthermore, these five types of models include GRU-based and Transformer-based models, which means there are a total of ten combinations. The difference from the methods of Zuo et al. [11] and Zhang et al. [12] is that our methods do not use any distance learning. Instead, our methods are based only on the NMT model and linear transformation, or the NMT model and autoencoder training. The differences between the proposed methods and the baselines are shown in Table 1.

A. NORMALIZATION OF INSTRUCTIONS

Instruction normalization is performed as a preprocessing method for inputting basic blocks into the NMT model. Instructions consist of opcodes representing operations such as ADD (add) and MOV (move), and operands representing operation targets such as registers, numerical constants, string constants, and symbolic constants. Since the operand registers are allocated from unused ones, they may change depending on the compilation environment. Therefore, it is necessary to classify registers by usage and normalize them so that the semantics of the basic blocks and instructions do not change even if the registers used change. Therefore, we adopt the register replacement method by Zhang et al. [12] and normalize registers as follows;

- x86_64 instructions (partial excerpt)
  rax,rbx,rcx,rdx → reg_data_64
  esi,edi → reg_addr_32
  rbp,rsp,ebp,esp → reg_pointer
- ARM instructions
  r0 - r15 → reg_gen
  pc,sp,lr → reg_pointer

Some operand constants do not change at the compile time, however there are countless variations because they are values or strings. To tokenize instructions, the size of vocabulary must be finite, and if constants are treated as they are, the Out Of Vocabulary (OOV) problem will occur. Therefore, the constants must also be normalized, and we adopt the constant replacement A in the method of Zuo et al. [11] to normalize the constants as follows.

- Replace numeric constants with 0 or -0
- Replace string constants with $< STR >$
- Replace function names with FOO
- Replace other symbolic constants with $< TAG >$

TABLE 1. List of differences between the proposed methods and baselines.

Item	Zuo *et al.*'s [11] (INNEREYE-BB)	Zhang *et al.*'s [12] (MIRROR)	GRU/Trm-NMT +LT	GRU/Trm-NMT +SA/SE/SD/AE
Instruction normalization	Const replacement A	Const replacement B Register replacement	Const replacement A Register replacement	Const replacement A Register replacement
Tokenization unit	Instruction	Operand Opcode	Instruction	Instruction
NMT model	Word2Vec+LSTM	Transformer	GRU+Seq2Seq Transformer	GRU+Seq2Seq Transformer
Embedding vector	Final state	Ave. encoder output	Final state Ave. encoder output	Final state Ave. encoder output
Embedding training	SiameseNetwork	NMT TripletNetwork	NMT Linear transformation	NMT Autoencoder
Similarity index	Manhattan distance	Euclidean distance	Cosine similarity	Cosine similarity
Evaluation index	ROC curve AUC	ROC curve AUC Precision at k	Precision at k	Precision at k

By doing this, instructions with different character strings but similar semantics can be normalized as the same instruction.

B. TOKENIZATION OF BASIC BLOCKS

In order to input basic blocks into an NMT model we tokenize basic blocks. A basic block consists of several instructions. An instruction consists of an opcode and operands. Therefore, a basic block can be regarded as sequence data with consecutive opcodes and operands. Tokenization of such sequence data includes the method [11], which combines operands and opcodes together, and the method [12], which separates the operands and opcodes. The former has the advantage that the sequence length does not become long, but the disadvantage is that the size of vocabulary increases. The latter is the opposite, having a trade-off relationship with the former. We adopted a method that combines operands and opcodes to tokenize basic blocks without increasing the sequence length. The reason for this is the use of GRUs in the NMT model and the reduction of the instruction vocabulary by Zuo et al.'s method [11].

C. CONFIGURATION OF NMT MODELS

The NMT model requiring a linear transformation is shown in Fig. 1. It consists of an NMT model of x86_64 to ARM that translates x86_64 basic blocks to ARM basic blocks, and an NMT model of ARM to x86_64 that translates ARM basic blocks to x86_64 basic blocks. The NMT model without requiring a linear transformation is shown in Fig. 2. In addition to the NMT model described above, an autoencoder is configured between the x86_64 encoder and the x86_64 decoder, plus between the ARM encoder and the ARM decoder.

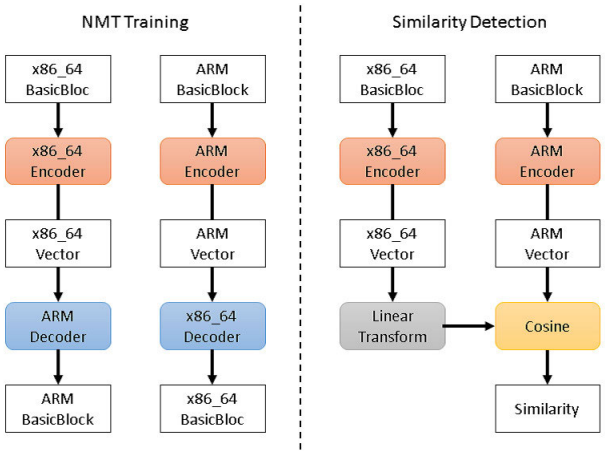

FIGURE 1. NMT models requiring a linear transformation.

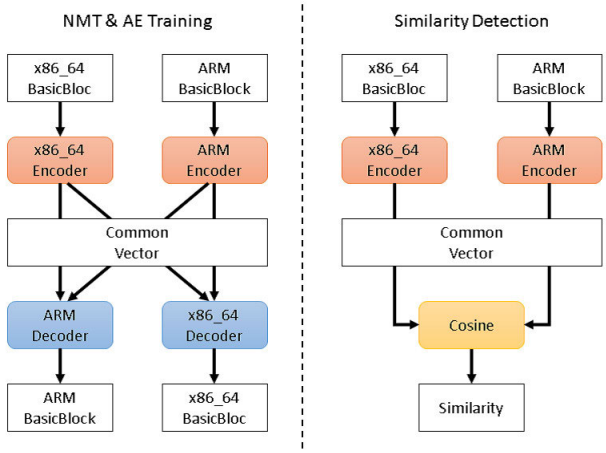

FIGURE 2. NMT models without requiring a linear transformation.

1) GRU-BASED NMT MODEL: GRU-NMT

We construct a GRU-NMT of the sequence to sequence model using the following; a single-layer bidirectional GRU for the encoder of the NMT model requiring a linear transformation, a three-layer bidirectional GRU for the encoder of the NMT model without requiring a linear transformation, and a single-layer uni-directional GRU for the decoder of both models. A Layer Normalization layer is inserted between each layer and the hidden state of the bidirectional GRU is connected. The feature vectors of basic blocks used for

similarity comparison are the averaged time-sequence data of the bi-directional GRUs, concatenated with the final state. The hyperparameters are shown in Table 2.

TABLE 2. Hyperparameters of the GRU models.

Hyperparameter	Value
No. of Embedded Instructions	128
No. of Hidden Dimensions	128+128
No. of Hidden Layers (Encoder/Decoder)	1/1 or 3/1
Dropout Rate	0.0
Normalization Layer	LayerNormalization
Optimisation Algorithm	RMSProp
Learning Rate	0.001
Batch Size	512
No. of Epochs	40

2) TRANSFORMER-BASED MODEL: TRM-NMT

We construct an NMT model Trm-NMT using a single-layer Transformer. The feature vector of basic blocks used for similarity comparison is the average of the encoder outputs. The hyperparameters are shown in Table 3.

TABLE 3. Hyperparameters of the transformer models.

Hyperparameter	Value
No. of Embedded Instructions	128
No. of Hidden Dimensions (MHA/FFN)	256/1024
No. of Hidden Layers (Encoder/Decoder)	1/1
No. of MHA Heads	8
Dropout Rate	0.3
Normalization Layer	LayerNormalization
Optimisation Algorithm	Adam
Learning Rate	0.0001
Batch Size	128
No. of Epochs	40

D. TRAINING METHODS FOR NMT MODELS

1) TRAINING METHOD REQUIRING A LINEAR TRANSFORMATION: +LT

For the NMT models GRU-NMT+LT and Trm-NMT+LT applying a training method that requires a linear transformation, we trained the x86_64 to ARM model and the ARM to x86_64 model for 40 epochs each. We then adopted the model with the parameters that yielded the highest P@1.

Let ϕ_{src} be the encoder parameters of the translation source ISA and ψ_{tgt} be the decoder parameters of the translation target ISA, where src, tgt $\in$ {x86_64, ARM}. Let $\hat{\boldsymbol{y}}_{jk}$ be the j-th instruction of the basic block $\hat{\boldsymbol{y}}e$ output by the decoder of the target ISA and an element of the probability vector satisfying $\sum_j \hat{y}_{jk} = 1$. Also, let y_{jk} be the j-th instruction of basic block $\boldsymbol{y}$ of the target ISA and an element of the One-Hot vector corresponding to the element of the k-th set of instructions of the target ISA. Then, the loss function of this NMT model is defined as follows.

$$L_{NMT} = -\sum_{j=1}\sum_{k=1} y_{jk} \log \hat{y}_{jk} \tag{1}$$

We optimized ϕ_{src} and ψ_{tgt} per batch size unit to minimize the loss function.

2) TRAINING METHOD WITHOUT A LINEAR TRANSFORMATION: +SA/SE/SD/AE

There is a training method that trains the NMT model to handle natural language for autoencoding and back-translation without using bilingual pairs by Artetxe et al. [30]. This improves the accuracy of unsupervised NMT models by embedding the feature vectors of each language in the same space. We adopted only the autoencoder training method of Artetxe et al. for the purpose of embedding the feature vectors of the basic blocks of each ISA in the same space, instead of unsupervised learning.

For the NMT models GRU-NMT+SA/SE/SD/AE and Trm-NMT+SA/SE/SD/AE, we applied a training method that does not require a linear transformation. For these, NMT models that translate from x86_64 to ARM and from ARM to x86_64 are trained for one epoch each. After that, the autoencoder that re-configures from x86_64 to x86_64 and the autoencoder that re-configures from ARM to ARM are trained for one epoch each. These processes are repeated for 40 times. Then we adopted the model with the parameters that yielded the highest P@1.

The loss function used to train these NMT models is the same as in equation (1). In addition, when a basic block with randomly permuted instructions is input to the encoder, the autoencoder is given the task of learning to remove noise so that the output of the decoder will be in the correct sequence.

Let ϕ_{isa} be the encoder parameters of the ISA of the input basic block and ψ_{isa} be the decoder parameters of the same ISA as the input basic block, where isa $\in$ {x86_64, ARM}. Let $\hat{\boldsymbol{x}}_{jk}$ be the j-th instruction of the basic block $\hat{\boldsymbol{x}}$ output by the decoder and an element of the probability vector satisfying $\sum_j \hat{x}_{jk} = 1$. Also, let x_{jk} be the j-th instruction of the basic block $\boldsymbol{x}$ with the correct instruction sequence and an element of the One-Hot vector corresponding to the element of the k-th instruction set. Then, the loss function of this autoencoder is defined as follows.

$$L_{AE} = -\sum_{j=1}\sum_{k=1} x_{jk} \log \hat{x}_{jk} \tag{2}$$

We optimized ϕ_{isa} and ψ_{isa} per batch size unit to minimize the expected value of this loss function.

3) CONSTRAINTS WHEN TRAINING AUTOENCODERS

For training autoencoders of the four types of NMT models that do not require a linear transformation, we restrict them to share the following parameters. This was inspired by the multitask learning model used in the experiments of Luong et al. [28]. A model that shares the encoder and the decoder is also used in Google's Zero-Shot translation method of embedding languages in the same space [29]. In addition, a model that shares only the encoder is also used in Artetxe et al.'s unsupervised learning model [30].

- GRU-NMT+SA/Trm-NMT+SA
  The x86_64 encoder and the ARM encoder share the same parameters, and the ARM decoder and the x86_64 decoder share the same parameters. Hence, the constraints are $\phi_{x86\_64} = \phi_{ARM}$ and $\psi_{x86\_64} = \psi_{ARM}$.
- GRU-NMT+SE/Trm-NMT+SE
  The x86_64 encoder and the ARM encoder share the same parameters. Hence, the constraint is $\phi_{x86\_64} = \phi_{ARM}$. However, for the Transformer model only, we did not let the encoder parameters update when training the autoencoder.
- GRU-NMT+SD/Trm-NMT+SD
  The ARM decoder and the x86_64 decoder share the same parameters. Hence, the constraint is $\psi_{x86\_64} = \psi_{ARM}$. However, for the Transformer model only, we did not let the decoder parameters update when training the autoencoder.
- GRU-NMT+AE/Trm-NMT+AE
  The x86_64 encoder and the ARM encoder do not share parameters, and the ARM decoder and the x86_64 decoder do not share parameters. Hence, there are no constraints. However, for the Transformer model only, we did not let the encoder parameters update when training the autoencoder.

E. SIMILARITY DETECTION METHOD

For NMT models requiring a linear transformation, 10,000 similar pairs of basic blocks taken from the training data are input to the trained encoder. The x86_64 feature matrix V_{x86_64} and the ARM feature matrix V_{ARM} are then obtained. Then, using the Moore-Penrose pseudo-inverse matrix $V^{\dagger}_{x86\_64}$ in Equation (3), the linear transformation matrix W is obtained as in Equations (4) and (5). After applying the linear transformation to the x86_64 encoder output, we calculate the cosine similarity with the ARM encoder output.

$$V^{\dagger}_{x86\_64} = (V^{T}_{x86\_64} V_{x86\_64})^{-1} V^{T}_{x86\_64} \tag{3}$$

$$W = \underset{W}{arg\,min} \, ||V_{x86\_64} W - V_{ARM}||^2_F \tag{4}$$

$$= V^{\dagger}_{x86\_64} V_{ARM} \tag{5}$$

However, the NMT models, which do not require a linear transformation, simply calculate the cosine similarity between the x86_64 encoder output and the ARM encoder output.

IV. EVALUATION EXPERIMENTS

This chapter describes the evaluation processes of the proposed methods and the results of the experiments. We evaluate the accuracy of detecting similarity of a basic block of one ISA among 100 basic blocks of another ISA. We also evaluate whether the model is lightweight or not by the embedding time per basic block. These are to evaluate whether the binary code similarity detection methods have practical capabilities when it is made into a practical tool.

A. EVALUATION INDEX

When k blocks similar in function to a basic block of one ISA are extracted from 100 basic blocks of another ISA in order of cosine similarity, we evaluate P@k by whether the correct answer was included in the k blocks.

Prepare pairs (v_1, u_1) to (v_{100}, u_{100}) of the feature vector v of the x86_64 basic block and the feature vector u of the ARM basic block, which have similar functions. List those similar to vector v_i from u_1 to u_{100} in descending order of cosine similarity. Let $Vk(i)$ be a function that returns 1 if it ranks in the top k ranks, and 0 otherwise. Also, list those similar to vector u_i from v_i to v_{100} in descending order of cosine similarity. Let $Uk(i)$ be a function that returns 1 if it ranks in the top k ranks, and 0 otherwise. The P@k for the feature vectors of 100 pairs of x86_64 basic blocks and ARM basic blocks with similar functionality is defined as in Equation (6).

$$P@k = \sum_{i=1}^{100} (Vk(i) + Uk(i))/200 \tag{6}$$

Note that if multiple 100 pairs of feature vectors are extracted without replacement from the dataset, the P@k obtained each time is averaged.

B. EVALUATION ENVIRONMENT

We used a desktop computer with the following specifications for the evaluation experiments.

- MEM: 32GB
- CPU: Intel(R) Core(TM) i7-8700
- GPU: NVIDIA GeForce GTX 1660

The development environment for NMT models is as follows.

- Python 3.10.5
- TensorFlow GPU 2.10.0
- Keras 2.10.0
- Pytorch 1.12.1+cu116
- CUDA 11.7
- cuDNN 8.5.0

C. DATASET

This subsection describes the dataset used in the evaluation experiments. We used the dataset published by Zuo et al. [31]. Zuo et al.'s dataset contains similar and dissimilar pairs of x86_64 basic blocks and ARM basic blocks compiled with Clang (ver.6.0.0) with optimization option O2. The source code is from OpenSSL(v1.1.1-pre1), coreutils(ver.8.29), findutils(ver.4.6.0), diffutils(ver.3.6), and binutils(ver.2.30).

We used 42,343 similar pairs in Zuo et al.'s dataset as training data, and 100 similar pairs not included in the training data as test data.

Basic blocks with too few instructions have similar semantics and we consider that they are unsuitable for evaluating model's accuracy, hence basic blocks with less than 10 instructions were not included in the test data.

D. BASELINES

We decided to use INNEREYE-BB [11] and MIRROR [12] as baselines, because these methods perform similarity detection across different ISAs in basic block units similar to our proposed methods. We used the pre-trained model published by Zuo et al. [31] in INNEREYE-BB. We trained MIRROR on the dataset used in this study using the source code published by Zhang et al. [32]. Note that the INNEREYE-BB used hyperparameters optimized for our dataset, whereas MIRROR used the hyperparameters set in the source code of Zhang et al. as they are, and is not optimized for our dataset.

E. EVALUATION METHODS

To evaluate the model's accuracy, P@k was measured when the number of instructions in the basic block of all 100 test datasets was 10 or more and 20 or more, respectively, from 100 to 42,343 training datasets.

In the evaluation of model embedding time, we measured the embedding time for 32,000 similar pairs of basic blocks when using the CPU and when using the GPU, and obtained the embedding time per pair. Assuming that the proposed method is used in a low-resource environment, the batch size was set to 32, and batch processing was performed 1,000 times. Since the source code of MIRROR was written using Pytorch, in order to match the experimental conditions, we built a Transformer with the same hyperparameters as MIRROR in Keras and measured the embedding time.

F. EVALUATION RESULTS AND DISCUSSION

1) PRECISION AT K

Table 4 shows the P@k of the proposed approaches and the baseline approaches for test data in which the number of instructions in one basic block (szBB) is 10 or more. Table 5 shows the P@k for test data in which the number of instructions in one basic block (szBB) is 20 or more. Values for which the proposed method exceeded the baselines are highlighted in bold.

In the models using a GRU, GRU-NMT+LT, which requires a linear transformation, GRU-NMT+SE, which shares the encoder parameters, and GRU-NMT+SA, which shares all parameters, resulted in higher P@k than any baselines. In addition, we found that it is effective to share encoder parameters in order to embed feature vectors of basic blocks of the different ISA in the same space and improve the P@k.

In the models using a Transformer, the P@k of all models except Trm-NMT+SD, which does not share decoder parameters, was higher than any of the baselines. In addition, we found that sharing encoder parameters tends to be as effective as GRU-NMT in order to embed feature vectors of basic blocks of the different ISA in the same space and improve the P@k.

Comparing the P@k between GRU-NMT and Trm-NMT, the results of GRU-NMT were relatively high overall on test data in which the number of instructions in one basic block (szBB) is 10 or more. This is presumably because there were a lot of test data with sequence lengths short enough to be handled by a recurrent neural network. Also, the Transformer's ability to handle long sequence lengths was not demonstrated. On the other hand, on test data in which the number of instructions in one basic block (szBB) is 20 or more, compared to test data where szBB is 10 or more, the P@k of GRU-NMT was greatly reduced. Whereas the P@k of Trm-NMT was not as degraded as GRU-NMT. We believe that the characteristics of Transformer were relatively demonstrated because the test data of short sequence length, which GRU-NMT excels at, reduced.

TABLE 4. Comparison of P@k (szBB≥10).

Approach	P@1	P@3	P@10
GRU-NMT+LT	**0.92**	**0.97**	**1.00**
GRU-NMT+SA	**0.89**	**0.98**	**0.99**
GRU-NMT+SE	**0.89**	**0.96**	**1.00**
GRU-NMT+SD	**0.75**	**0.88**	**0.99**
GRU-NMT+AE	**0.79**	**0.94**	**0.98**
Trm-NMT+LT	**0.89**	**0.97**	**1.00**
Trm-NMT+SA	**0.85**	**0.93**	**0.98**
Trm-NMT+SE	**0.88**	**0.98**	**1.00**
Trm-NMT+SD	0.57	0.80	**0.94**
Trm-NMT+AE	**0.86**	**0.96**	**0.99**
MIRROR [12]	0.64	0.85	0.93
INNEREYE-BB [11]	0.54	0.81	0.93

TABLE 5. Comparison of P@k (szBB≥20).

Approach	P@1	P@3	P@10
GRU-NMT+LT	**0.83**	**0.96**	**0.99**
GRU-NMT+SA	**0.77**	**0.93**	**0.99**
GRU-NMT+SE	**0.81**	**0.94**	**1.00**
GRU-NMT+SD	0.64	0.84	0.94
GRU-NMT+AE	**0.71**	0.88	**0.97**
Trm-NMT+LT	**0.84**	**0.95**	**1.00**
Trm-NMT+SA	**0.74**	**0.90**	**0.99**
Trm-NMT+SE	**0.87**	**0.96**	**1.00**
Trm-NMT+SD	0.51	0.77	0.92
Trm-NMT+AE	**0.74**	**0.92**	**1.00**
MIRROR [12]	0.69	0.88	0.94
INNEREYE-BB [11]	0.31	0.57	0.83

2) LEARNING CURVES

Figures 3 to 6 show the values of P@1 when the models of the proposed method are trained between 100 and 40,000 training datasets. Among GRU-based NMT models, GRU-NMT+LT requiring a linear transformation was less accurate than other models when the number of training datasets was small. However, the accuracy tended to improve when the number of training datasets exceeded 1,000. This is probably because about 1,000 pieces of data are required to obtain the linear transformation matrix. On the other hand, among Transformer-based NMT models, Trm-NMT+LT requiring a linear transformation did not show the linear transformation-dependent features that appeared in GRU-NMT+LT when the number of training datasets was small. Trm-NMT+LT showed almost the same learning curve as

Trm-NMT+AE, Trm-NMT+SE and Trm-NMT+SA. This indicates that linear transformations in the Transformer-based NMT model have a limited contribution to accuracy.

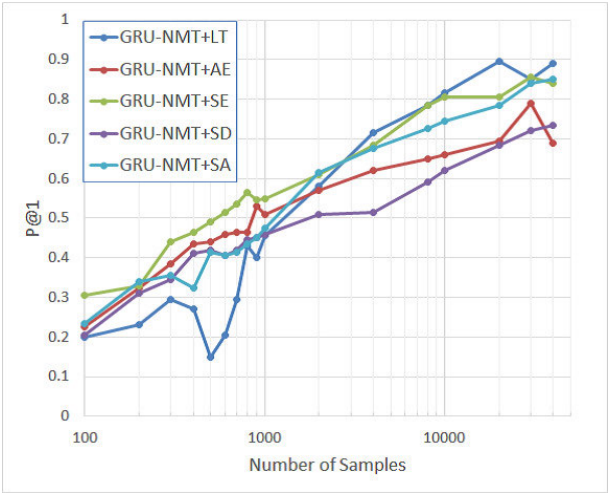

FIGURE 3. Trends in the amount of training data and the P@1 of GRU-NMT.(szBB≥10).

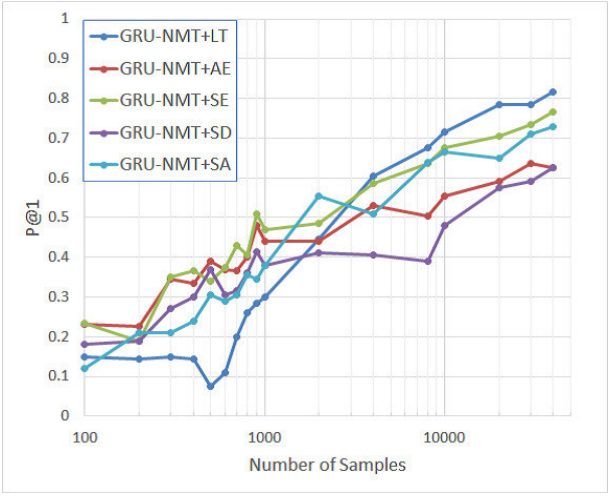

FIGURE 4. Trends in the amount of training data and the P@1 of GRU-NMT (szBB≥20).

3) BASIC BLOCK EMBEDDING TIME

Table 6 shows the embedding time per pair of basic blocks for the proposed methods and the baselines. For both CPU and GPU, GRU-NMT+LT was the fastest and MIRROR was the slowest. GRU-NMT and INNEREYE-BB used the same tokenization method. However, since GRU is a recurrent neural network model that is more lightweight than LSTM, GRU-NMT showed faster embedding time. Both Trm-NMT and MIRROR use a Transformer NMT model. However, due to the use of different tokenization method and different number of layers, the embedding time of Trm-NMT was faster than that of MIRROR.

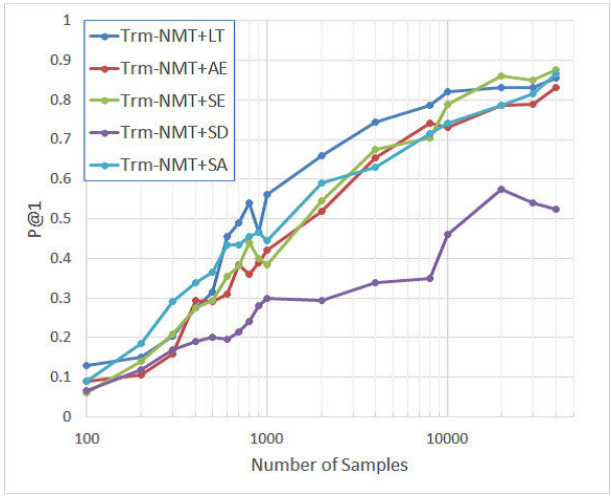

FIGURE 5. Trends in the amount of training data and the P@1 of Trm-NMT.(szBB≥10).

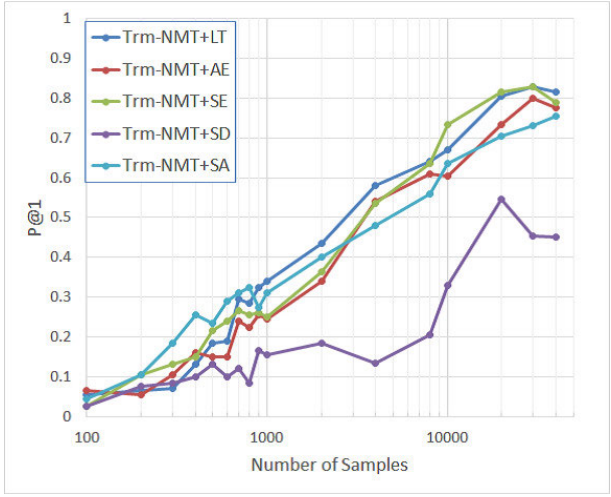

FIGURE 6. Trends in the amount of training data and the P@1 of Trm-NMT.(szBB≥20).

Regarding INNEREYE-BB, the reason why there was no big difference between the embedding time on the CPU and on the GPU is because CuDNN did not support LSTM using ReLU as the activation function, and the CPU was used to compute LSTM.

V. DISCUSSIONS

The number of layers of GRU and Transformer used in the NMT model of our methods is from 1 to 3 layers. We believe that this is lightweight compared to Zhang et al.'s NMT model with the 6-layer Transformer so our methods are more suitable for resource-constrained devices. However, for even less resourced devices, the NMT model should be compressed to make it even lighter. For example, model compression methods such as pruning [33], distillation [34] and quantization [35] may be considered. Pruning is a method of reducing the number of parameters by deleting nodes

TABLE 6. Comparison of the embedding time.

Approach	CPU[ms]	GPU[ms]
GRU-NMT+LT	0.650	0.316
GRU-NMT+SE/SD/SA/AE	1.633	0.601
Trm-NMT+LT	2.158	0.348
Trm-NMT+SE/SD/SA/AE	2.146	0.346
MIRROR [12]	45.50	5.328
INNEREYE-BB [11]	2.132	1.960

with small weights or unimportant nodes. Distillation is a method of making a model with less parameters learn the probability distribution output by a model with many parameters. Quantization is a method of reducing the data size of parameters by keeping weights that are held with 32-bit precision with 8-bit precision. All of these methods have been applied to Transformer [36], [37], [38] and may be applicable to our proposed method as well.

A limitation of the proposed methods is that the models requiring a linear transformation are constructed to compare basic blocks between different ISAs. In other words, they do not support basic block comparisons for the same ISA. Our models, which do not require a linear transformation, share the same embedding space and thus has the potential to compare basic blocks of the same ISA. However, its accuracy has not yet been confirmed.

In addition, in all models of the proposed methods, the datasets used for evaluation use the same compiler and the optimization option is only O2. We have not been able to confirm the accuracy when using datasets mixed with different optimization options, such as those used in the evaluation by Zhang et al. [12]. Learning with triplet loss as used in Zhang et al.'s method [12] may be effective.

VI. CONCLUSION

We proposed feature extraction methods for basic blocks using NMT models. These are intended to improve the accuracy of binary code similarity detection across different ISAs for software vulnerability detection, bug detection, and plagiarism detection. Among the proposed methods, we showed that the NMT models, which requires a linear transformation of features, are able to detect basic block similarity more accurately and faster than previous studies. Our future work will include evaluation with combinations other than x86_64 and ARM, including the same ISA, and evaluation across different compilers and optimizations.

REFERENCES

[1] D. Gitchell and N. Tran, "Sim: A utility for detecting similarity in computer programs," *ACM SIGCSE Bull.*, vol. 31, no. 1, pp. 266–270, 1999.
[2] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-miner: Finding copy-paste and related bugs in large-scale software code," *IEEE Trans. Softw. Eng.*, vol. 32, no. 3, pp. 176–192, Mar. 2006.
[3] A. Ahtiainen, S. Surakka, and M. Rahikainen, "Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises," in *Proc. 6th Baltic Sea Conf. Comput. Educ. Res., Koli Calling*, Feb. 2006, pp. 141–142.
[4] B. S. Baker, U. Manber, and R. Muth, "Compressing differences of executable code," in *Proc. ACMSIGPLAN Workshop Compiler Support Syst. Softw. (WCSS)*, Apr. 1999, pp. 1–10.
[5] Z. Wang, K. Pierce, and S. McFarling, "BMAT—A binary matching tool for stale profile propagation," *J. Instruct.-Level Parallelism*, vol. 2, pp. 1–20, May 2000.
[6] Ministry of Internal Affairs and Communications. *2021 Information and Communications White Paper*. Accessed: Oct. 3, 2021. [Online]. Available: https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r03/html/nd105220.html
[7] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *Proc. AAAI Conf. Artif. Intell.*, Mar. 2016, vol. 30, no. 1, pp. 1–7.
[8] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, "Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 24, no. 4, pp. 694–707, Apr. 2016.
[9] K. Seki, "On cross-lingual text similarity using neural translation models," *J. Inf. Process.*, vol. 27, pp. 315–321, 2019, doi: 10.2197/ipsjjip.27.315.
[10] K. Seki, "Cross-lingual text similarity exploiting neural machine translation models," *J. Inf. Sci.*, vol. 47, no. 3, pp. 404–418, Jun. 2021.
[11] F. Zuo, X. Li, P. Young, L. Luo, Q. Zeng, and Z. Zhang, "Neural machine translation inspired binary code similarity comparison beyond function pairs," 2018, *arXiv:1808.04706*.
[12] X. Zhang, W. Sun, J. Pang, F. Liu, and Z. Ma, "Similarity metric method for binary basic blocks of cross-instruction set architecture," in *Proc. Workshop Binary Anal. Res.*, vol. 10, 2020, pp. 1–12.
[13] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2015, pp. 2067–2075.
[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[15] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," 2013, *arXiv:1309.4168*.
[16] Q. Feng, R. Zhou, C. Xu, Y. Cheng, B. Testa, and H. Yin, "Scalable graph-based bug search for firmware images," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 480–491.
[17] 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," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 363–376.
[18] J. Gao, X. Yang, Y. Fu, Y. Jiang, and J. Sun, "VulSeeker: A semantic learning based vulnerability seeker for cross-platform binary," in *Proc. 33rd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2018, pp. 896–899.
[19] 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 *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 472–489.
[20] L. Massarelli, G. A. Di 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*. Gothenburg, Sweden: Springer, Jun. 2019, pp. 309–329.
[21] L. Massarelli, G. A. Di Luna, F. Petroni, L. Querzoni, and R. Baldoni, "Investigating graph embedding neural networks with unsupervised features extraction for binary analysis," in *Proc. Workshop Binary Anal. Res.*, 2019, pp. 1–11.
[22] Z. Yu, R. Cao, Q. Tang, S. Nie, J. Huang, and S. Wu, "Order matters: Semantic-aware neural networks for binary code similarity detection," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2020, vol. 34, no. 1, pp. 1145–1152.
[23] Y. Masubuchi, M. Hashimoto, and A. Otsuka, "SIBYL: A method for detecting similar binary functions using machine learning," *IEICE Trans. Inf. Syst.*, vol. E105.D, no. 4, pp. 755–765, 2022.
[24] H. Wang, W. Qu, G. Katz, W. Zhu, Z. Gao, H. Qiu, J. Zhuge, and C. Zhang, "jTrans: Jump-aware transformer for binary code similarity detection," in *Proc. 31st ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2022, pp. 1–13.
[25] K. Redmond, L. Luo, and Q. Zeng, "A cross-architecture instruction embedding model for natural language processing-inspired binary code analysis," 2018, *arXiv:1812.09652*.
[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[27] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Comput. Surv.*, vol. 55, no. 6, pp. 1–28, Jul. 2023.

[28] M.-T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser, "Multi-task sequence to sequence learning," 2015, *arXiv:1511.06114*.
[29] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, M. Hughes, and J. Dean, "Google's multilingual neural machine translation system: Enabling zero-shot translation," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 339–351, Oct. 2017.
[30] M. Artetxe, G. Labaka, E. Agirre, and K. Cho, "Unsupervised neural machine translation," 2017, *arXiv:1710.11041*.
[31] *INNEREYE*. Accessed: Oct. 3, 2021. [Online]. Available: https://nmt4binaries.github.io/
[32] *MIRROR*. Accessed: Oct. 3, 2021. [Online]. Available: https://github.com/zhangxiaochuan/MIRROR
[33] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
[34] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.
[35] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, Jan. 2017.
[36] Z. Lin, J. Liu, Z. Yang, N. Hua, and D. Roth, "Pruning redundant mappings in transformer models via spectral-normalized identity prior," in *Proc. Findings Assoc. Comput. Linguistics (EMNLP)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 719–730.
[37] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "TinyBERT: Distilling BERT for natural language understanding," in *Proc. Findings Assoc. Comput. Linguistics (EMNLP)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 4163–4174.
[38] G. Prato, E. Charlaix, and M. Rezagholizadeh, "Fully quantized transformer for machine translation," in *Proc. Findings Assoc. Comput. Linguistics (EMNLP)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 1–14.

NORIMITSU ITO received the bachelor's degree in engineering from the Department of Information Engineering, National Defense Academy, Kanagawa, Japan, in 2015. In 2016, he passed the National Public Service Employment General Service Examination (university graduate level). Since 2017, he has been engaged in information and communication technology at a government agency. Since 2022, he has been a Visiting Researcher with the Hashimoto Laboratory, Institute of Information Security.

MASAKI HASHIMOTO (Member, IEEE) received the Ph.D. degree in informatics from the Institute of Information Security, Kanagawa, Japan, in 2010. From 2010, he was an Assistant Professor, and since 2014, he has been an Associate Professor with the Institute of Information Security. From April 2014 to March 2015, he was a Visiting Researcher with the Information Security Group, Royal Holloway, University of London, U.K. His research includes include intrusion detection/prevention, malware analysis, and OSINT technology. He is a member of the Information Processing Society of Japan, the Institute of Electronics, Information and Communication Engineers, and the Japan Society for Software Science and Technology. He is also engaged as an Editorial Board Member of *IEICE/English Journal D*, Ministry of Economy, Trade and Industry/Electrical Safety System WG Committee, and NETSAP2022 Workshop Organizer.

AKIRA OTSUKA (Member, IEEE) was born in Osaka, in 1966. He received the B.E. and M.E. degrees from Osaka University, in 1989 and 1991, respectively, and the Ph.D. degree from The University of Tokyo, in 2002. Since 2002, he has been a Postdoctoral Fellow and a Cooperative Researcher with The University of Tokyo. From 2003 to 2005, he was a member of Cryptographic Technique Monitoring Subcommittee at CRYPTREC. Since 2005, he has been with the National Institute of Advanced Industrial Science and Technology (AIST). From 2006 to 2010, he was the Leader of Research Security Fundamentals. From 2007 to 2014, he was a Visiting Professor with the Research and Development Initiative, Chuo University. Since 2017, he has been a Professor with the Graduate School of Information Security, Institute of Information Security.