Fusing Multimodal Binary Code Representations for Enhanced Similarity Detection

Taiyan Wang
CEE
NUDT
Hefei, China
wangty@nudt.edu.cn

Ruipeng Wang
CEE
NUDT
Hefei, China
wangruipeng@nudt.edu.cn

Yu Chen
CEE
NUDT
Hefei, China
cy@nudt.edu.cn

Lu Yu
CEE
NUDT
Hefei, China
yulu@nudt.edu.cn

Zulie Pan*
CEE
NUDT
Hefei, China
panzulie17@nudt.edu.cn

Min Zhang*
CEE
NUDT
Hefei, China
zhangmindy@nudt.edu.cn

***Abstract*—As software reuse has become increasingly prevalent in the modern era, binary code similarity detection plays a critical role in program analysis. Numerous machine learning methods have been introduced to this field, with the primary challenge lying in the effective representation of binary code. While existing approaches leverage multimodal features for embedding, they still adopt relatively simple fusion methods to combine different modalities. Therefore, their performance can be limited by inadequate modality interaction during the feature fusion step and an over-reliance on a single modality in the final embedding step.**

To address these issues, we propose *FuseBinRepr*, a method that fuses text modality and graph modality representation techniques using a fusion model architecture and three specialized learning tasks to enhance binary code similarity detection. We design a fusion architecture integrating both text and graph embeddings via cross-attention and self-attention mechanisms. For model training, we developed three tasks: text-graph alignment (TGA), graph masking recovery (GMR), and contrastive learning (CL), to capture and align high-level semantics across modalities. Through evaluation, *FuseBinRepr* demonstrates improvements in Mean Reciprocal Rank (MRR) and Recall compared to state-of-the-art baseline methods , achieving increases of up to 40.8% and 42%, respectively. Ablation studies confirm the robustness of our model design, as well as the effectiveness of pretraining tasks. In the downstream software vulnerability detection task, *FuseBinRepr* achieves the best MRR and Recall performance in ranking CVE binary functions.

***Index Terms*—program analysis, binary code representation, multimodal feature fusion, similarity detection**

I. Introduction

Binary Code Similarity Detection (BCSD) enables researchers to compare and identify code similarities between different binary files or within a single binary, which is particularly useful in binary level program analysis such as detecting plagiarism [1], malware variants [2], [3], vulnerabilities [4]–[7], and corresponding patches [8]–[14]. By leveraging BCSD, researchers gain powerful tools for analyzing compiled programs without access to their underlying source code, therefore enhancing the capabilities of program analysis in both academic and industrial contexts.

Current methods have introduced a variety of machine learning (ML) techniques to this field, enhancing intelligent detection and improving performance. A critical component of these advancements is the embedding phase, which plays a pivotal role in transforming raw data into meaningful feature representations. During this phase, complex data formats like text and graph features from binary code are transformed into numerical vectors that encapsulate the intrinsic characteristics and relationships within the data. The quality of these embeddings greatly impact the overall performance and effectiveness of ML systems, establishing this phase as a cornerstone of modern intelligent detection methodologies.

The features derived from text and graph modalities play a pivotal role for BCSD. High-level text semantics provide resilience against obfuscation techniques [5], [15]–[20], while the topological structure of the code logic graph remains stable and robust across different operating systems (OS) or instruction set architectures (ISA) [7], [21]–[24]. Consequently, methods that rely solely on unimodal features may exhibit limitations in these scenarios, as no single feature set can provide universal robustness. Therefore, the integration of multiple perspectives is essential for achieving comprehensive and reliable binary representation.

While there are approaches that leverage multimodal features [4], [25]–[29], these methods frequently suffer from limitations due to their simplistic fusion strategies. Some methods adopt a multimodal framework for the feature fusion step, but they are limited to simple feature concatenation or processing through shallow multilayer perceptrons (MLPs). As of yet, specialized multimodal training tasks have not been designed in current research. These rudimentary fusion techniques might not adequately leverage the intricate and rich information inherent in multimodal data, which could consequently compromise the model performance. Furthermore,

many multimodal representations integrate text embeddings into graph structures as node attributes, and form an attributed graph for further multimodal representation. This approach has demonstrated effectiveness in existing experiments, while the final graph embedding step can lead to an over-reliance on graph modality features, potentially constraining overall performance.

To address the limitations of existing methods, we propose *FuseBinRepr*, a representation technique that leverages the multimodal paradigm [30], [31] in a more comprehensive manner for representation learning, thereby achieving enhanced similarity detection. Given that current research primarily focuses on embedding text and graph features [32], we have chosen these two modalities as the foundation for building our multimodal framework. Specifically, *FuseBinRepr* integrates advanced techniques to combine textual and graphical information, ensuring that the intrinsic characteristics and relationships within the data are fully captured. We design an innovative multimodal architecture that not only fuses these different types of data but also incorporates specialized training tasks aimed at enabling a deeper understanding of semantics. This approach allows the model to learn richer and more nuanced representations, which in turn leads to improved performance in similarity detection tasks.

The model architecture of *FuseBinRepr* is designed to fuse features in a fundamental yet sophisticated manner, ensuring deep interaction between different modalities at the information level. To achieve this, we employ cross-attention and self-attention mechanisms that enable rich interactions between text and graph features. Cross-attention allows the model to focus on relevant parts of one modality when processing another, while self-attention helps capture intra-modality relationships, enhancing the overall representation. Additionally, we design specialized training tasks to align the vector spaces of the two modality embeddings and further fuse their representations. These tasks include: the *Text-Graph Alignment* task, the *Graph Masking Recovery* task, and the *Contrastive Learning* task. The first two tasks are part of the pretraining phase, aimed at understanding binary code semantics, while the last one is a finetuning task for downstream binary similarity detection.

We implement a prototype of *FuseBinRepr* and compare it with state-of-the-art (SOTA) methods. Our approach outperforms both unimodal and multimodal representation models, including SAFE [16], CLAP [19], GMN [21], sem2vec [27], and others. Additionally, we developed several versions of *FuseBinRepr*, combining different embedding techniques in various ways. Through comprehensive comparisons, we found that both our designed architecture and specialized training tasks contribute to the superior performance of the model. We also evaluated *FuseBinRepr* on the vulnerability detection task and demonstrated that our method achieves the best rankings compared to all baselines.

In summary, the contributions are as follows:

1) We propose *FuseBinRepr*, a multimodal binary code representation technique that integrates different binary code representation methods to create a comprehensive and robust model. The source code for *FuseBinRepr* is publicly available at https://anonymous.4open.science/r/FuseBinRepr-9BEF/README.md.
2) We design a specialized model architecture along with three tailored training tasks aimed at fusing features and aligning the vector spaces of the two modalities. This approach enables a deeper understanding of binary code semantics.
3) We implement a prototype of *FuseBinRepr* and conduct extensive experiments. The results demonstrate that our method significantly outperforms current unimodal and multimodal approaches in various evaluation metrics.

II. Related work

There are many off-the-shelf solutions for handling BCSD using embedding techniques, and these can be distinguished or classified based on the types of features and models they employ in their embedding processes [32].

In this section, we classify current BCSD methods into two categories based on whether they utilize unimodal or multimodal features.

A. Methods utilizing unimodal features

For binary code, multiple perspectives can be used to characterize its features, such as disassembling it into assembly text or using program analysis to construct gWraph features. In this section, we introduce two mainstream types of methods that utilize unimodal features: text embedding-based methods and graph embedding-based methods.

1) Text embedding-based methods: Since binary executables are compiled from source code written in programming languages, they can be reverse-engineered into assembly code after disassembly and sometimes into intermediate language (IL) like LLVM IR after binary lifting, therefore being processed by text-based method.

The first generation of methods for BCSD using text embedding techniques, such as SAFE [16], InnerEye [33], Asm2Vec [15], are based on models like Word2Vec [34], PV-DM [35], RNN, and LSTM. With the increasing popularity of Transformer [36], there has been significant advancement in the field of BCSD, and Bidirectional Encoder Representations from Transformers (BERT) [37] and other hierarchical Transformer models have been used in PalmTree [17], jTrans [5] and BinShot [18].

Large Language Models (LLMs) have revolutionized various fields in recent years, unlocking new potentials in language embedding that many researchers are actively exploring. CLAP [19] aligns assembly language and natural language by leveraging cross-modal data, which is generated by Llama model [38] finetuned on code descriptions produced by ChatGPT [39]. Nova and Nova++ [20] are built upon the code-specific language model StarCoder [40]. These models are trained on a large corpus of binary data using contrastive learning tasks focused on assembly code functionality and optimization.

2) Graph embedding-based methods: Besides text semantics, control and data flow in assembly can be characterized using topological graphs, and Wang et al. propose δCFG [41] to validate the role of Control Flow Graph (CFG) features in ML-based BFSD.

Starting from Gemini [22] using structure2vec [42] to generate vector embeddings for the Attributed Control Flow Graph (ACFG) of binary code, VulSeeker [23], Graph Matching Network (GMN) [21], XBA [24], HermesSim [7] and following researches leverage graph embedding methods including customized deep neural networks and Graph Convolutional Network (GCN) [43].

B. Methods utilizing multimodal features

Given the diversity of binary analysis scenarios, different feature modalities exhibit varying levels of robustness in different contexts. As a result, current methods begin to integrate both text and graph features for representation. We have summarized several approaches to utilizing these multimodal features effectively.

1) Nesting based methods: This primary method involves integrating text features into graph features as node attributes, transforming the graph into an attributed graph with richer semantics. The attributed graph is then embedded into a vector representation, capturing the multimodal features.

GraphEmb [25], BMM [26], sem2vec [27] , IotSim [44], and GraphMoCo [45] employ text embedding techniques such as i2v , StrandCNN, Asm2vecPlus, BERT, RoBERTa [46] , and DeBERTa [47], [48] to embed assembly instructions and tracelet symbolic constraints. These embeddings are then used as nodes to construct a topological graph. Subsequently, graph embedding methods like structure2vec and GCN are applied to derive a representation of the entire binary code.

This approach connects the two embedding steps in series and has achieved promising results in recent studies. However, the order of the embeddings influences the relative weight of contributions from the two modalities. Because the binary code representation concludes with graph embedding, graph semantics carry more weight than text semantics in the final representation.

2) Concatenation based methods: The embeddings from two modalities can be concatenated to form a new high-dimensional vector for further processing. However, current methods rely solely on simple multilayer perceptrons (MLPs) without additional design considerations.

Partially inspired by the multimodality paradigm, studies such as OrderMatters [28] and VulHawk [4] have begun to integrate various types of features, including text embeddings, graph embeddings, and other numerical features from basic blocks, string constants, and import functions. These studies concatenate different types of features and embedding vectors generated by embedding models. After concatenation, they employ customized deep neural networks, such as MLPs, to train models for representation and similarity detection.

This approach is promising for its potential performance. While the multimodality paradigm can be limited in representation performance when using simple embedding model architectures, as shallow model layers may not capture deep semantics in binary code. Additionally, since the representation serves not only for similarity detection but also for various downstream tasks, the training tasks should be diverse to enhance overall representation performance.

III. MOTIVATION

As the two most commonly used feature types, text features extracted from assembly code and graph features derived from CFGs characterize binary code from distinct perspectives, thereby offering different yet complementary information. These varying feature modalities show different degrees of robustness when detecting similarities between binaries compiled under different settings. To illustrate this, we analyze several functions from the *sha512sum* utility in the Coreutils suite.

Take cross-architecture scenario as an example. Fig. 1(a) shows the function *clone_quoting_options()* of the *sha512sum* compiled for 32-bit ARM with -O0 optimization, and Fig. 1(b) shows the same function compiled for 32-bit MIPS with -O0 optimization. Although (a) and (b) come from different ISAs, they share identical CFG structures. This demonstrates the superior robustness of graph features in such cases, making graph embedding methods significantly more efficient than text-based approaches.

However graph-based methods are not universally reliable, as the case illustrated by Fig. 1(c) shows the function *get_quoting_style()* compiled for 32-bit MIPS with -O0 optimization. Despite having identical CFG structure to Fig. 1(b) and sharing MIPS assembly syntax, these functions exhibit totally different semantics. In this scenario, graph embedding methods lead to false positives in similarity analysis, therefore the optimal solution might be to rely on text embedding methods to accurately understand the semantics and identify differences in functionality.

Consequently, binary code representation methods should incorporate both text features and graph features, while current fusion methods in use are insufficient. As discussed in Section II-B1, most current approaches are nesting-based, meaning that the final embedding step is performed within a single modality. For instance, consider Fig. 1. Over-reliance on text features can lead to false negatives when comparing (a) and (b), while over-reliance on graph features can introduce false positives when comparing (b) and (c). Another approach, described in Section II-B2, highlights the issue of insufficient fusion of multimodal features, which lacks consideration for the abstract interactions between different modalities. When addressing more complex problems, such as those shown in Fig. 4 in Appendix A where the same source code results in significantly different assembly and graph structures, the turbulence caused by inadequate integration of both feature modalities can pose challenges.

(a) Assembly and CFG for clone_quoting_options() in coreutils-6.7_arm_32_O0_sha512sum

(b) Assembly and CFG for clone_quoting_options() in coreutils-6.7_mips_32_O0_sha512sum

(c) Assembly and CFG for get_quoting_style() in coreutils-6.7_mips_32_O0_sha512sum

Fig. 1. Cross-architecture example

IV. Design

To address the limitations identified in Section II and cases in Section III, we propose *FuseBinRepr*, a multimodal representation method that integrates text and graph embeddings to achieve a combined representation and enhance binary code similarity detection. Our design encompasses both the model architecture and the learning tasks for training.

The model architecture of *FuseBinRepr* is designed to fuse features from different modalities at the information level, while the training tasks are aimed at aligning and integrating semantics across these modalities.

A. Model architecture

To achieve a comprehensive representation of binary code by fusing multimodal features, we first need to select the feature modalities and their embedding methods respectively. Based on the review in Section II and the survey [32] researchers conducted, current methods primarily utilize text or graph data, with embedding techniques tailored to each type of feature. Therefore, we have chosen two feature modalities-text and graph-along with their corresponding embedding methods.

The *FuseBinRepr* fuses embeddings from different feature modalities at the information level, and its architecture is shown in Fig. 2. *FuseBinRepr* consists of a text embedding model and a graph embedding model, which are existing methods for embedding features from different modalities. Additionally, it includes a *Feature Fusion Model* based on an attention mechanism, which is a key component of the design and comprises two main parts: *Cross Attention Layers* and *Transformer Encoder Layers*.

The primary role of the *Cross Attention Layers* is to facilitate information complementation at the feature level between embedding vectors of two modalities, serving as an initial feature fusion mechanism. For representation vectors of two different modalities $V_1 \in \mathbb{R}^{n \times d_1}$ and $V_2 \in \mathbb{R}^{n \times d_2}$, the cross-attention mechanism treats one representation vector V_1 as the *Query*, while the other representation vector V_2 provides the *Key* and *Value*. The sequence of query vectors V_1 enhances its representation by utilizing information obtained from the key-value vector sequence V_2, resulting in the enhanced vector representation as shown in Equation 1. It is important to note that although this module establishes connections between representation vectors of two different modalities using cross-attention, during the vector calculations, the representation vectors of the two modalities are still processed as distinct vector sequences.

$$CrossAttention(V_1, V_2) = Softmax(\frac{(W_Q V_1)(W_K V_2)^T}{\sqrt{d_2}})W_V V_2 \quad (1)$$

The role of the *Transformer Encoder Layers* is to further fuse features from two modalities using the self-attention mechanism, while stacking neural network parameters to facilitate subsequent training of the model on modality fusion tasks for improved performance. This component consists of multiple layers of Transformer encoders that simultaneously receive representation vectors V_1 and V_2 from the two modalities, which are then concatenated to form a unified model input $V \in \mathbb{R}^{n \times d_k}$. Following this, multi-head self-attention computations are performed on V, where the self-attention

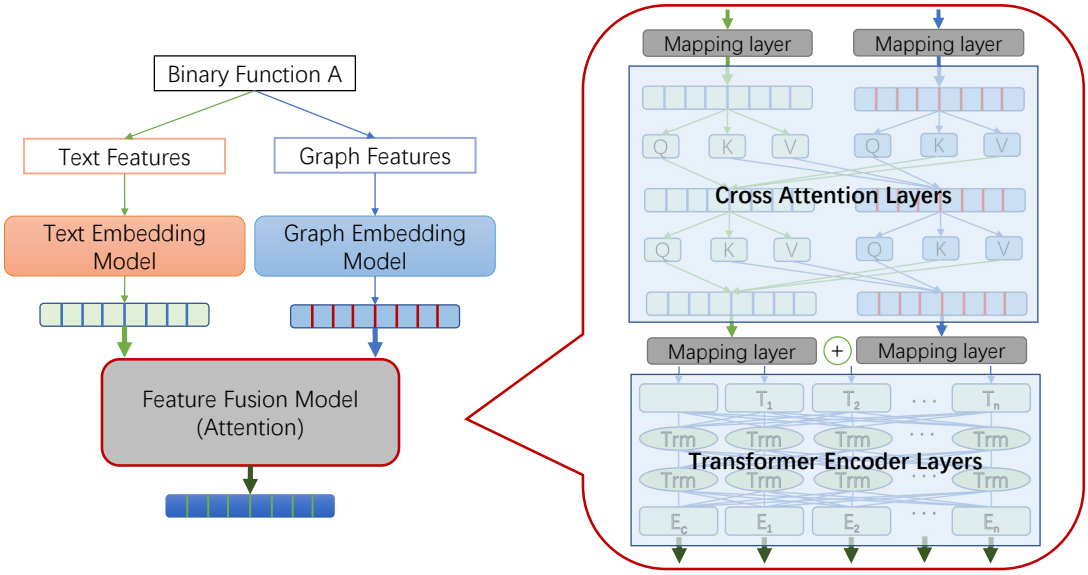


Fig. 2. Architecture of *FuseBinRepr*

calculation is shown in Equation 2, aimed at achieving further fusion of features from different modalities.

$$SelfAttention(V) = Softmax(\frac{(W_Q V)(W_K V)^T}{\sqrt{d_k}})W_V V \quad (2)$$

The model architecture introduced above is designed to fuse multimodal features in a fundamental manner and provides a sufficient number of parameters for training, enabling a better comprehension of binary code semantics. For detailed comparisons of the specific methods chosen, refer to the ablation experiments conducted in Section V-C.

B. Fusing tasks

To enhance the understanding of binary code semantics from two different modalities, we design three training tasks for *FuseBinRepr*: the *Text-Graph Alignment* task, the *Graph Masking Recovery* task, and the *Contrastive Learning* task. Three tasks are illustrated in Fig. 3.

The training process can be divided into two steps:

1) **Pretraining**: In the first step, we pretrain the model to understand binary code semantics using the *Text-Graph Alignment* and *Graph Masking Recovery* tasks. This step aims to align multimodal features effectively. Since they both help capture semantics and occupy different output tokens due to their belonging to different types of tasks (classification and regression), they can be conducted simultaneously within the same round of input and output.
2) **Finetuning**: In the second step, we finetune the model for the downstream task of binary code similarity detection using the *Contrastive Learning* task.

1) Text-Graph Alignment task: The *Text-Graph Alignment* (TGA) task involves determining whether the text features and graph features originate from the same binary code. It is a classification task designed to align the outputs of the text representation model and the graph representation model using the *Feature Fusion Model* mentioned in Section IV-A. Since the corresponding representations of text and graph features for the same binary code should share some semantic similarities at a certain level, this alignment helps capture these commonalities, thereby providing an initial understanding of the relationship between these two modalities.

However, the essence of this task is to align the representation spaces of different modality models, which poses a training challenge. Therefore, we conduct the *Text-Graph Alignment* task alongside the *Graph Masking Recovery* task, as these tasks are related and can help reduce training difficulty and prevent overfitting. For this classification task, we use the first token p of the output vector from the *Feature Fusion Model* as the classification result. We then train the model using the Binary Cross-Entropy Loss (BCE) function as Equation 3, based on the true label y indicating whether the features match.

$$Loss_{BCE} = -y\log(p) - (1-y)\log(1-p) \quad (3)$$

2) Graph Masking Recovery task: The *Graph Masking Recovery* (GMR) task involves partially masking the graph feature embedding vectors as input and then recovering the masked positions using the remaining information. This regression task is designed based on the fact that the topological graph structure of binary code can be generated by analyzing its assembly text. Given that textual data often contains richer information compared to graph data, the purpose of this task is to leverage the information content of textual features to map and complete the topological graph features. This process

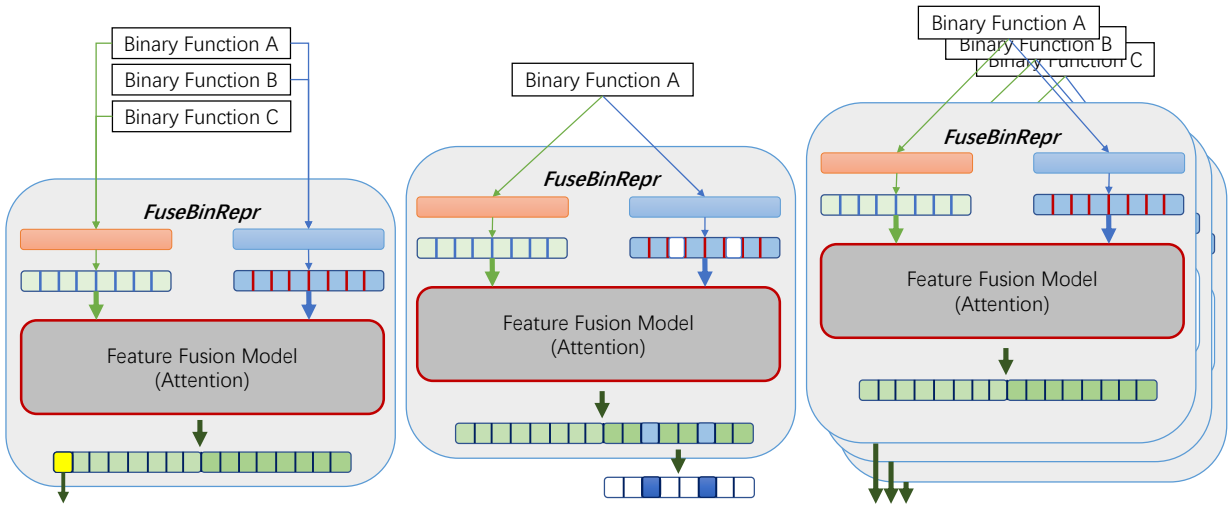


Fig. 3. Three training tasks

facilitates deeper fusion and association between the two modalities.

For this regression task, before the data is fed into the *Feature Fusion Model*, we mask 10% tokens of the representation vectors from the graph embedding. The original masked tokens in the embedding are denoted as y. We then observe the corresponding recoverd tokens f in the output of the *Feature Fusion Model* and train the model using the Huber loss function Equation 5 , which is calculated as the summation of Equation 4.

$$Loss_{\delta}(y,f) = \begin{cases} \frac{1}{2}(y-f)^2, & \text{if } |y-f| \le \delta \\ \delta(|y-f| - \frac{1}{2}\delta) & \text{if } |y-f| > \delta \end{cases} \quad (4)$$

$$Loss_{Huber} = \sum_{i=1}^{N} Loss_{\delta}(y_i, f_i) \quad (5)$$

3) Contrastive Learning task: The *Contrastive Learning* (CL) task for binary similarity detection entails generating representation vectors for two segments of binary code based on their textual and topological graph structure features, and then calculating the distance between these representation vectors. If the binary codes are from the same source (i.e., similar), the vector distance should be 0; otherwise, it should be 1. The purpose of this task is to directly apply the binary code representations to similarity detection by bringing similar code representations closer together in vector space and pushing dissimilar ones apart.

To generate representation vectors for different binary codes, we use the cosine similarity formula Equation 6 to calculate the degree of similarity between the binary code representation vectors. When two segments of binary code are similar, the similarity score should be close to 1; otherwise, it should be close to 0. We train the model using the Contrastive Loss function asgiven by Equation 7.

$$Dist_{cos}(A,B) = \frac{AB}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}, margin = 1 \quad (6)$$

$$Loss_{contrastive}(Y, X_1, X_2) = \frac{1}{2}(1-Y)D_{cos}(X_1, X_2)^2 + \frac{1}{2}Y\{max(0, margin - Dist_{cos}(X_1, X_2))\}^2 \quad (7)$$

V. Evaluation

In this section we investigate the following research questions:

- **RQ1 (Performance)**: How does *FuseBinRepr* perform in terms of similarity detection?
- **RQ2 (Ablation Study)**: How do the selected embedding methods, and pretraining tasks contribute to the overall performance?
- **RQ3 (Vulnerability Detection)**: Is *FuseBinRepr* able to detect vulnerable targets?

A. Experimental setup

Environment. All experiments were conducted on a dedicated server equipped with an Intel Xeon Gold 6230R CPU (104 cores @ 4 GHz), two NVIDIA RTX 3090 GPUs, and 256 GB of memory. The codes were run on Ubuntu 22.04 (Jammy Jellyfish) with NVIDIA GPU driver version 550.

Implementation. We implemented *FuseBinRepr* using Python 3.10 and PyTorch 2.4.0. The model integrates two pretrained binary code representation models for text and graph modalities. The *Feature Fusion Model* consists of three cross-attention layers followed by six transformer encoder

layers with self-attention. The dimensions of input embeddings are derived from the selected text and graph models. Both cross- and self-attention layers have hidden dimensions of 512 and use 8 attention heads uniformly. The output dimension matches the input to meet GMR task requirements.

Datasets. We used distinct binary datasets for training and evaluation. For training, we selected binaries from Trex [49] (18,548 binaries), which offers multiple optimization settings and a diverse collection of software, including ImageMagick, gmp, sqlite, zlib, putty, and tool suites like binutils, coreutils, diffutils, and findutils. To validate that our method effectively learns semantics and generalizes, we used BinKit [50] (243,128 binaries) for evaluation. BinKit also includes hundreds of software binaries, such as coreutils, but with even more diverse optimization options. We used the same comparison pairs and train/test splits for all baselines, without employing any cross-validation.

Training. The training process for the entire model involves freezing the text embedding model and graph embedding model while training only the *Feature Fusion Model* using three training tasks. We pretrain the model using TGA and GMR for three epochs and then finetune it with contrastive learning for an additional epoch to adapt to the BCSD task.

Baselines. We compare *FuseBinRepr* with four unimodal baselines and three multimodal baselines. The unimodal baselines include:

- **Asm2Vec [15]**: This method employs the PV-DM model to learn semantic representations from the text features of binary assembly code.
- **SAFE [16]**: This baseline employs the skip-gram variant of the Word2Vec model to generate instruction representations and uses an RNN with an attention mechanism to produce a fixed-length vectorization of binary functions.
- **GMN [21]**: As introduced in Section II-A2, this graph embedding-based method achieved nearly top performance in evaluations conducted by Cisco Talos [51].
- **CLAP [19]**: This baseline leverages LLMs to generate data and trains alignment between assembly and natural language for binary representation.

The multimodal baselines consist of:

- **PalmTree [17]+GMN**: PalmTree is an instruction embedding model, and it can be combined with graph embedding methods to evaluate on the BCSD task, utilizing the nesting based method described in Section II-B1. We choose GMN for good performance, and train PalmTree on both ARM and MIPS assembly code to enhance its cross-architecture capabilities.
- **CLAP+GMN**: Following the nesting based strategy, we utilize CLAP and GMN to embed text and graph modality features respectively, and perform feature fusion.
- **sem2vec [27](GMN)**: The original sem2vec model combines a finetuned RoBERTa [46] with either GGNN [52] or GTN [53], adopting nesting based method. We replace these components with GMN for better performance.

Metrics. We use mean reciprocal rank (MRR) and recall at K (Recall@K) as evaluation metrics for the BCSD ranking task, following the approach in [5], [6], [19], [51]. Given a set of candidate functions F, we search our binary code database to identify similar functions and rank the results based on similarity scores. The MRR calculates the average of the reciprocals of the ranks for all relevant items, while Recall@K measures the proportion of relevant items that appear within the top K ranked positions. The MRR and Recall@K are calculated as Equation 8 and Equation 9:

$$MRR = \frac{1}{|F|}\sum_{i=1}^{|F|}\frac{1}{rank_i} \tag{8}$$

$$Recall@K = \frac{1}{|F|}\sum_{i=1}^{|F|}\begin{cases}1, & \text{if } rank_i \leq k\\ 0, & \text{if } rank_i > k\end{cases} \tag{9}$$

B. Performance

When setting the pool size (the number of functions in the binary code database to search) to 1000, with thousands of binary target functions to rank, we evaluated performance across various scenarios by calculating the MRR and Recall@K (for K = 1, 5, and 10). The higher the MRR and Recall@K values, the better their performance is. The scenarios included:

- **XA (cross-architecture)**: Functions from different ISAs, including x86, ARM, and MIPS (Excluding rare architectures such as RISC-V, PowerPC, etc.) , but using the same compilers and optimization levels,
- **XC (cross-compiler)**: Functions compiled with different compilers, including GCC and Clang, but using the same ISA and optimization levels,
- **XO (cross-optimization)**: Functions compiled with different optimization options, including -O1, -O2, -O3, and -Os, but using the same ISA and compiler,
- **XA+XC**: Functions from different ISAs compiled with different compilers, but with the same optimization levels,
- **XA+XO**: Functions from different ISAs compiled with different optimization options, but with the same compiler,
- **XO+XC**: Functions compiled with different optimization options using different compilers, but with the same ISA.

We have implemented four different versions of *FuseBinRepr* by combining two text embedding methods (CLAP and SAFE) with two graph embedding approaches (GMN and sem2vec(GMN)): *FuseBinRepr*(CLAP+GMN), *FuseBinRepr*(SAFE+GMN), *FuseBinRepr*(CLAP+sem2vec), and *FuseBinRepr*(SAFE+sem2vec). This evaluation allowed us to compare our method against SOTA works.

The experimental results in Table I demonstrate that, all versions of *FuseBinRepr* demonstrate performance improvements in certain aspects compared to both their constituent unimodal approaches and corresponding nesting-based multimodal methods. Both *FuseBinRepr*(CLAP+GMN) and *FuseBinRepr*(SAFE+GMN) show significant improvement over CLAP, SAFE, and GMN, and nesting based CLAP+GMN ,

TABLE I
COMPARISON OF MRR AND RECALL@1 FOR DIFFERENT METHODS IN BINARY SIMILARITY DETECTION

Method	MRR							Recall@1						
	XA	XC	XO	XA+XC	XA+XO	XO+XC	**ALL**	XA	XC	XO	XA+XC	XA+XO	XO+XC	**ALL**
Asm2Vec	0.060	0.118	0.111	0.042	0.075	0.107	0.103	0.047	0.096	0.094	0.020	0.059	0.084	0.086
CLAP	0.057	0.178	0.143	0.041	0.068	0.152	0.152	0.053	0.170	0.136	0.034	0.065	0.144	0.149
SAFE	**0.124**	0.229	0.174	**0.141**	**0.134**	0.195	0.192	**0.110**	0.217	0.160	0.129	**0.124**	0.180	0.178
GMN	0.093	**0.266**	**0.181**	0.137	0.105	**0.240**	**0.198**	0.097	**0.271**	**0.186**	**0.146**	0.110	**0.248**	**0.200**
PalmTree+GMN	0.035	0.079	0.056	0.035	0.034	0.062	0.067	0.033	0.069	0.050	0.025	0.030	0.056	0.059
CLAP+GMN	0.063	0.124	0.097	0.067	0.065	0.108	0.105	0.061	0.113	0.092	0.061	0.062	0.095	0.105
sem2vec(GMN)	**0.771**	**0.721**	**0.749**	**1.000**	**0.771**	**0.848**	**0.627**	**0.938**	**0.679**	**0.700**	**1.000**	**0.938**	**0.821**	**0.600**
FuseBinRepr(CLAP+GMN)	0.531	0.690	0.553	0.583	0.531	0.723	0.513	0.438	0.714	0.550	0.750	0.438	0.750	0.525
FuseBinRepr(SAFE+GMN)	0.615	0.653	0.542	0.708	**0.615**	0.708	0.487	0.563	0.643	0.525	0.750	0.563	0.714	0.475
FuseBinRepr(CLAP+sem2vec)	0.542	0.811	0.623	0.750	0.542	0.823	0.598	**0.688**	0.857	0.625	**1.000**	**0.688**	0.857	**0.625**
FuseBinRepr(SAFE+sem2vec)	**0.615**	**0.894**	**0.700**	**0.833**	**0.615**	**0.925**	**0.644**	0.563	**0.893**	**0.675**	0.750	0.563	**0.929**	**0.625**

achieving 40.8% improvements in MRR and 42% improvements in Recall@1. This suggests *FuseBinRepr* not only inherits the advantages of the underlying embeddings, but also capture higher-level semantic dimensions that previous methods fail to address.

Among those methods, *FuseBinRepr*(SAFE+sem2vec) achieves the best MRR and Recall@1 performance across all scenarios (XA+XC+XO) , achieving a 1.7% increase in MRR and 2.5% increase in Recall@1. This superior performance can be attributed to the high-quality embeddings and comprehensive representation views provided by sem2vec and SAFE.

However, in certain scenarios such as XA, XA+XC, and XA+XO, the combination of sem2vec(GMN) outperforms *FuseBinRepr*(SAFE+sem2vec) in terms of both MRR and Recall@1. This suggests that SAFE may have limitations in detecting similarities across different architectures, which could impact its performance in these specific contexts. The overall MRR of *FuseBinRepr*(CLAP+sem2vec) is lower than sem2vec(GMN), suggesting that CLAP may occasionally introduce embedding bias that negatively impacts performance. Nevertheless, *FuseBinRepr* still demonstrates significantly greater robustness against interference from underlying embeddings compared to nesting-based methods, as evidenced by CLAP+GMN performing worse than unimodal CLAP and GMN in these scenarios.

Generally, higher-quality unimodal embedding models contribute to better fusion performance. Regarding multimodal fusion techniques, our proposed methods (such as *FuseBinRepr*(CLAP+GMN)) demonstrate superior performance compared to existing nesting-based multimodal fusion methods (like CLAP+GMN). This improvement highlights the effectiveness of our approach in integrating diverse modalities.

The results in Table VII in Appendix B illustrate the Recall@5 and Recall@10 performance of different methods, confirming that our proposed method *FuseBinRepr* outperforms other SOTA methods.

Similar to the studies by [5], [6], [19], we conducted three sets of experiments with different pool sizes in the binary function searching. The pool size refers to the number of candidate functions, where a larger pool size increases the difficulty of similarity detection. We configured the pool size to 32, 100, and 1000. The corresponding results for MRR and Recall are presented in Table II. As the pool size increases, the difficulty of binary searching also increases, leading to a significant decrease in the performance of unimodal methods. In contrast, the decline in performance of *FuseBinRepr* is relatively minor, with some versions maintaining stability across various pool sizes.

C. Ablation study

We conducted ablation studies to evaluate the contribution of each component of *FuseBinRepr*, including the model architecture, the embedding model selection, and training tasks.

For the model architecture, we assessed the effectiveness of combining cross-attention and self-attention mechanisms. Regarding the embedding models, we compared the performance of different combinations of embedding techniques, with particular attention to how performance changes when the left-right order of the embeddings is altered. Finally, for the training tasks, we evaluated various versions with different combinations of training tasks.

1) Model architecture: We designed our multimodal fusion model by integrating cross-attention and self-attention mechanisms sequentially. Since the subsequent training tasks rely on the *Transformer Encoder Layers*, it is not feasible to strip the self-attention component. We compared the performance of a model without cross-attention (*FuseBinRepr* - cross) to our original model that includes both cross-attention and self-attention.

The results in Table III show that our original model outperformed the version without cross-attention, especially when the representation vectors generated by the two embedding models are complementary, such as SAFE/CLAP and GMN. When using higher-performing embedding models, performance still improves since the cross-attention layers truly help to facilitate information fusion at the feature level.

2) Embedding methods: In Section V-B, we introduced several unimodal and multimodal baselines, and our method *FuseBinRepr* builds upon these by selecting two of them to compose our fusion model. Here, we compare the performance of different embedding combinations and examine the effects of changing their orders.

TABLE II
PERFORMANCE OF DIFFERENT METHODS WITH POOL SIZES OF 32, 100, AND 1000

Method	Poolsize = 32				Poolsize = 100				Poolsize = 1000			
	MRR	Recall@1	Recall@5	Recall@10	MRR	Recall@1	Recall@5	Recall@10	MRR	Recall@1	Recall@5	Recall@10
Asm2Vec	0.269	0.207	0.370	0.542	0.190	0.159	0.217	0.288	0.103	0.086	0.125	0.142
CLAP	0.308	0.254	0.402	0.593	0.233	0.204	0.256	0.318	0.152	0.149	0.171	0.186
SAFE	0.357	0.306	0.458	0.566	0.267	**0.242**	0.304	**0.355**	0.192	0.178	0.207	**0.219**
GMN	**0.369**	**0.324**	**0.499**	**0.669**	**0.268**	0.226	**0.336**	**0.429**	**0.198**	**0.200**	**0.209**	**0.219**
CLAP+GMN	0.261	0.216	0.336	0.454	0.159	0.127	0.216	0.304	0.067	0.059	0.079	0.099
PalmTree+GMN	0.273	0.221	0.398	0.520	0.188	0.158	0.233	0.284	0.105	0.105	0.123	0.151
sem2vec(GMN)	**0.689**	**0.675**	**0.750**	**0.850**	**0.627**	**0.600**	**0.700**	**0.750**	**0.627**	**0.600**	**0.700**	**0.750**
FuseBinRepr(CLAP+GMN)	0.535	0.550	0.600	0.650	0.513	0.525	0.575	0.600	0.513	0.525	0.575	0.600
FuseBinRepr(SAFE+GMN)	0.519	0.500	0.550	0.625	0.487	0.475	0.550	0.575	0.487	0.475	0.550	0.575
FuseBinRepr(CLAP+sem2vec)	0.630	0.625	0.625	0.650	0.598	**0.625**	0.625	0.650	0.598	**0.625**	0.625	0.650
FuseBinRepr(SAFE+sem2vec)	**0.677**	**0.675**	**0.750**	**0.750**	**0.644**	**0.625**	**0.725**	**0.725**	**0.644**	**0.625**	**0.725**	**0.725**

TABLE III
ABLATION STUDY OF DIFFERENT ARCHITECTURE

Method	All (XA+XC+XO)			
	MRR	Recall@1	Recall@5	Recall@10
FuseBinRepr(SAFE+GMN)-cross	0.186	0.188	0.219	0.239
FuseBinRepr(SAFE+GMN)	**0.487**	**0.475**	**0.550**	**0.575**
FuseBinRepr(CLAP+GMN)-cross	0.174	0.161	0.193	0.202
FuseBinRepr(CLAP+GMN)	**0.513**	**0.525**	**0.575**	**0.600**
FuseBinRepr(CLAP+sem2vec)-cross	0.544	0.550	0.575	0.600
FuseBinRepr(CLAP+sem2vec)	**0.598**	**0.625**	**0.625**	**0.650**
FuseBinRepr(SAFE+sem2vec)-cross	0.622	0.625	0.700	0.700
FuseBinRepr(SAFE+sem2vec)	**0.644**	0.625	**0.725**	**0.725**

Our method is sensitive to the order of the embedding models because our subsequent *Graph Masking Recovery* (GMR) task is specifically designed to use the full information from text embeddings (on the left side) to recover masked tokens in graph embeddings (on the right side). Therefore, the position of the embeddings can impact performance, particularly when integrating text and graph data.

TABLE IV
ABLATION STUDY OF DIFFERENT EMBEDDING MODEL COMBINATIONS

Method	All (XA, XC, XO)			
	MRR	Recall@1	Recall@5	Recall@10
FuseBinRepr(GMN+SAFE)	0.189	0.193	0.217	0.233
FuseBinRepr(SAFE+GMN)	**0.487**	**0.475**	**0.550**	**0.575**
FuseBinRepr(GMN+CLAP)	0.172	0.163	0.187	0.200
FuseBinRepr(CLAP+GMN)	**0.513**	**0.525**	**0.575**	**0.600**
FuseBinRepr(sem2vec+CLAP)	0.588	0.600	0.600	**0.650**
FuseBinRepr(CLAP+sem2vec)	**0.598**	**0.625**	**0.625**	**0.650**
FuseBinRepr(sem2vec+SAFE)	**0.646**	**0.650**	**0.725**	**0.725**
FuseBinRepr(SAFE+sem2vec)	0.644	0.625	**0.725**	**0.725**

As shown in Table IV, when using GMN as the graph embedding method, the performance is sensitive to the order of embedding modality order. This aligns with our expectations, as the limited information content makes the order-sensitive GMR task really work. In contrast, using sem2vec(GMN) for graph embedding shows robustness to the order changes, likely because it contains duplicated semantics that GMR would provide.

3) Training tasks: We have designed three training tasks for *FuseBinRepr*: the *Text-Graph Alignment* (TGA) task, the *Graph Masking Recovery* (GMR) task, and the *Contrastive Learning* (CL) task. We selected the *FuseBinRepr*(SAFE+sem2vec) version of the model to evaluate the contribution of each task.

The model is trained using at least one of these three tasks. If a specific task is omitted, we denote the configuration accordingly. For example, if *FuseBinRepr*(SAFE+sem2vec) is trained without the contrastive learning task, we refer to it as *FuseBinRepr*(SAFE+sem2vec)-CL. Similarly, other variants omitting different tasks are named accordingly.

TABLE V
ABLATION STUDY OF DIFFERENT LEARNING TASKS FOR *FuseBinRepr*(SAFE+SEM2VEC)

Method	All (XA, XC, XO)			
	MRR	Recall@1	Recall@5	Recall@10
FuseBinRepr(SAFE+sem2vec)-TGA-GMR	0.563	0.575	0.650	0.650
FuseBinRepr(SAFE+sem2vec)-TGA-CL	0.573	0.600	0.675	0.675
FuseBinRepr(SAFE+sem2vec)-GMR-CL	0.587	0.600	0.675	0.700
FuseBinRepr(SAFE+sem2vec)-GMR	0.595	0.575	0.675	0.700
FuseBinRepr(SAFE+sem2vec)-CL	0.602	0.600	0.700	0.700
FuseBinRepr(SAFE+sem2vec)-TGA	0.613	**0.625**	0.700	**0.725**
FuseBinRepr(SAFE+sem2vec)	**0.644**	**0.625**	**0.725**	**0.725**

We select the *FuseBinRepr*(SAFE+sem2vec) model for our ablation study, with the experimental results shown in Table V. The *FuseBinRepr*(SAFE+sem2vec) model appears to benefit from all three training tasks, as its performance improves with the inclusion of more tasks. When two tasks are conducted, like *FuseBinRepr*(SAFE+sem2vec)-TGA, *FuseBinRepr*(SAFE+sem2vec)-CL, and *FuseBinRepr*(SAFE+sem2vec)-GMR, the models perform generally better than the single-task versions.

D. Vulnerability detection

In addition to binary similarity detection, the *FuseBinRepr* is also evaluated on the binary vulnerability detection downstream task. We select 16 CVE (Common Vulnerabilities and Exposures) vulnerabilities involving different vulnerable functions from coreutils, busybox, and openssl, as detailed in Table X in Appendix C.

We rank the vulnerability targets by searching for them in our candidate binary function pool. The scenario is XA + XC + XO and the size of binary function pool is 1000. The higher the rank result is, the better the ranking ability. If the corresponding vulnerability function is ranked within

TABLE VI
RANKING RESULTS FOR VULNERABILITY TARGET

Method	Ranking results for selected CVEs															
	CVE-2015-4041	CVE-2023-42366	CVE-2023-42365	CVE-2023-42363	CVE-2021-42386	CVE-2021-42385	CVE-2021-42384	CVE-2021-42382	CVE-2021-42381	CVE-2021-42380	CVE-2021-42379	CVE-2021-42378	CVE-2017-3733	CVE-2014-0195	CVE-2015-0286	CVE-2015-1788
Asm2Vec	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	
CLAP	1	4	-	1	-	-	-	1	-	1	1	4	3	2	-	7
SAFE	5	-	-	1	-	2	3	2	3	2	1	5	7	1	3	1
GMN	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	
CLAP+GMN	1	-	-	-	-	2	-	-	-	1	-	-	7	-	-	
PalmTree+GMN	-	-	-	-	-	-	-	5	-	-	-	-	5	-	-	
sem2vec(GMN)	9	1	6	-	1	-	1	1	-	-	1	2	-	1	1	1
FuseBinRepr(CLAP+GMN)	2	2	8	2	2	4	3	2	-	2	2	4	3	3	-	7
FuseBinRepr(SAFE+GMN)	2	4	-	2	2	2	3	2	3	2	2	4	5	2	3	3
FuseBinRepr(CLAP+sem2vec)	2	2	4	2	3	4	3	2	-	2	3	4	2	2	-	7
FuseBinRepr(SAFE+sem2vec)	2	4	-	2	2	2	3	2	3	2	2	4	9	2	3	3

the top 10, its rank is recorded in the ranking results shown in Table VI. If it is not among the top 10, it is denoted as "-". The Top-10 setting is commonly chosen because rankings often contain biases or errors. Researchers typically examine the top 10 results to maximize the likelihood of identifying the actual vulnerability target. A number less than 10 is useful when analysis resources are limited, while a number greater than 10 provides a more comprehensive evaluation, particularly when sufficient computational resources are available during the analysis.

From the results, we can see that for our selected CVE function searches, CLAP and SAFE significantly outperform other unimodal methods, even surpassing the nesting-based multimodal methods in terms of MRR. Specifically, at least 11 functions are ranked within the Top-10 results. While sem2vec has a slightly lower MRR, it achieves a comparable performance with a higher Recall@1 score, often achieving Top-1 rankings.

Compared to the baselines, our methods achieve both higher MRR and Recall results. Additionally, our detection capability covers all selected CVEs, which demonstrates the superiority of our proposed method.

Similar to general similarity detection, false results may occur if a non-vulnerable function shares the same compilation settings with the target vulnerability, or if the functions differ only in a few statements (such as in a patched version). Additionally, if the logic of the target vulnerable function is complex, compiling it to another ISA or with multiple optimizations can significantly alter its representation. In such cases, even semantic embedding techniques may struggle to align the functions effectively. These factors contribute to the occurrence of false positives and false negatives.

VI. CONCLUSIONS

Binary code representation shows significant influence and importance in program analysis. While current methods do not effectively utilize all modalities of features, resulting in less robust representations. In this study, we proposed fusing different binary code representation techniques with a designed fusion model and specialized learning tasks to enhance binary-level similarity detection. We introduced a novel fusion method, *FuseBinRepr*, which is built upon SOTA binary code embedding methods, uses cross-attention and self-attention to construct the fusion model, and employs three learning tasks to align features from different modalities and finetune for the BCSD task. Empirical evaluations show that our approach effectively fuses semantics across feature modalities, achieving good performance in BCSD and vulnerability ranking.

For future work, we plan to enrich the embedding methods by using large models, particularly large graph models, as well as incorporating richer feature formats to further improve representation quality. Exploring alternative views or modalities of binary code beyond traditional text and graph representations presents a promising area for research. For vulnerability ranking, a downstream application of BCSD, finer-grained detection is preferred to achieve better performance.

REFERENCES

[1] L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu, "Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection," *IEEE Trans. Softw. Eng.*, vol. 43, no. 12, p. 1157–1177, Dec. 2017. [Online]. Available: https://doi.org/10.1109/TSE.2017.2655046

[2] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based android malware detection using hamming distance of static binary features," *Future Generation Computer Systems*, vol. 105, pp. 230–247, 2020.

[3] T. Frenklach, D. Cohen, A. Shabtai, and R. Puzis, "Android malware detection via an app similarity graph," *Computers & Security*, vol. 109, p. 102386, 2021.

[4] Z. Luo, P. Wang, B. Wang, Y. Tang, W. Xie, X. Zhou, D. Liu, and K. Lu, "Vulhawk: Cross-architecture vulnerability detection with entropy-based binary code search," in *Proceedings 2023 Network and Distributed System Security Symposium*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257501992

[5] 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 *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2022. New

York, NY, USA: Association for Computing Machinery, 2022, p. 1–13. [Online]. Available: https://doi.org/10.1145/3533767.3534367

[6] H. Wang, Z. Gao, C. Zhang, M. Sun, Y. Zhou, H. Qiu, and X. Xiao, "Cebin: A cost-effective framework for large-scale binary code similarity detection," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 149–161.

[7] H. He, X. Lin, Z. Weng, R. Zhao, S. Gan, L. Chen, Y. Ji, J. Wang, and Z. Xue, "Code is not natural language: unlock the power of semantics-oriented graph representation for binary code similarity detection," in *Proceedings of the 33rd USENIX Conference on Security Symposium*, ser. SEC '24. USA: USENIX Association, 2025.

[8] Z. Jiang, Y. Zhang, and J. e. a. Xu, "Pdiff: Semantic-based patch presence testing for downstream kernels," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20, 2020, pp. 1149–1163.

[9] Z. Lang, S. Yang, and Y. e. a. Cheng, "Pmatch: Semantic-based patch detection for binary programs," in *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2021, pp. 1–10.

[10] Y. Xiao, Z. Xu, and W. e. a. Zhang, "Viva: Binary level vulnerability identification via partial signature," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 213–224.

[11] Y. Xu, Z. Xu, and B. e. a. Chen, "Patch based vulnerability matching for binary programs," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2020, 2020, pp. 376–387.

[12] S. Yang, Z. Xu, and Y. e. a. Xiao, "Towards practical binary code similarity detection: Vulnerability verification via patch semantic analysis," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 6, sep 2023.

[13] J. Lin, D. Wang, and R. e. a. Chang, "Enbindiff: Identifying data-only patches for binaries," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 343–359, 2023.

[14] X. Xu, Q. Zheng, and Z. e. a. Yan, "Patchdiscovery: Patch presence test for identifying binary vulnerabilities based on key basic blocks," *IEEE Transactions on Software Engineering*, vol. 49, no. 12, pp. 5279–5294, 2023.

[15] 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 *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 472–489.

[16] 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*, R. Perdisci, C. Maurice, G. Giacinto, and M. Almgren, Eds. Cham: Springer International Publishing, 2019, pp. 309–329.

[17] X. Li, Y. Qu, and H. Yin, "Palmtree: Learning an assembly language model for instruction embedding," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3236–3251. [Online]. Available: https://doi.org/10.1145/3460120.3484587

[18] S. Ahn, S. Ahn, H. Koo, and Y. Paek, "Practical binary code similarity detection with bert-based transferable similarity learning," in *Proceedings of the 38th Annual Computer Security Applications Conference*, ser. ACSAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 361–374. [Online]. Available: https://doi.org/10.1145/3564625.3567975

[19] H. Wang, Z. Gao, C. Zhang, Z. Sha, M. Sun, Y. Zhou, W. Zhu, W. Sun, H. Qiu, and X. Xiao, "Clap: Learning transferable binary code representations with natural language supervision," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 503–515. [Online]. Available: https://doi.org/10.1145/3650212.3652145

[20] N. Jiang, C. Wang, K. Liu, X. Xu, L. Tan, X. Zhang, and P. Babkin, "Nova: Generative language models for assembly code with hierarchical attention and contrastive learning," 2024. [Online]. Available: https://arxiv.org/abs/2311.13721

[21] 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*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 3835–3845. [Online]. Available: https://proceedings.mlr.press/v97/li19d.html

[22] 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 *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 363–376. [Online]. Available: https://doi.org/10.1145/3133956.3134018

[23] J. Gao, X. Yang, Y. Fu, Y. Jiang, and J. Sun, "Vulseeker: a semantic learning based vulnerability seeker for cross-platform binary," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 896–899. [Online]. Available: https://doi.org/10.1145/3238147.3240480

[24] G. Kim, S. Hong, M. Franz, and D. Song, "Improving cross-platform binary analysis using representation learning via graph alignment," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 151–163.

[25] L. Massarelli, G. A. D. Luna, F. Petroni, L. Querzoni, and R. Baldoni, "Investigating graph embedding neural networks with unsupervised features extraction for binary analysis," *Proceedings 2019 Workshop on Binary Analysis Research*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:160018518

[26] Y. Guo, P. Li, Y. Luo, X. Wang, and Z. Wang, "Exploring gnn based program embedding technologies for binary related tasks," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, ser. ICPC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 366–377.

[27] H. Wang, P. Ma, S. Wang, Q. Tang, S. Nie, and S. Wu, "sem2vec: Semantics-aware assembly tracelet embedding," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, may 2023. [Online]. Available: https://doi.org/10.1145/3569933

[28] 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 *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 01, 2020, pp. 1145–1152.

[29] Y. Duan, X. Li, J. Wang, and H. Yin, "Deepbindiff: Learning program-wide code representations for binary diffing," *Proceedings 2020 Network and Distributed System Security Symposium*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:195063875

[30] P. Xu, X. Zhu, and D. A. Clifton, "Multimodal learning with transformers: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 10, pp. 12 113–12 132, 2023.

[31] S. Raschka, "Understanding multimodal llms," Jan. 2025. [Online]. Available: https://magazine.sebastianraschka.com/p/understanding-multimodal-llms

[32] L. Y. e. a. Taiyan WANG, Qingsong XIE, "A survey of binary code representation technology," *Frontiers of Information Technology & Electronic Engineering*, 2024. in press.

[33] F. Zuo, X. Li, Z. Zhang, P. Young, L. Luo, and Q. Zeng, "Neural machine translation inspired binary code similarity comparison beyond function pairs," *ArXiv*, vol. abs/1808.04706, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52004699

[34] Y. Qiao, W. Zhang, X. Du, and M. Guizani, "Malware classification based on multilayer perception and word2vec for iot security," *ACM Trans. Internet Technol.*, vol. 22, no. 1, sep 2021.

[35] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, pp. II–1188–II–1196.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[37] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," 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)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186.

[38] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez,

A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971
[39] OpenAI, "Chatgpt," Feb. 2024. [Online]. Available: https://chat.openai.com/
[40] R. Li, L. B. Allal, and Y. Z. et al., "Starcoder: may the source be with you!" 2023. [Online]. Available: https://arxiv.org/abs/2305.06161
[41] J. Wang, C. Zhang, L. Chen, Y. Rong, Y. Wu, H. Wang, W. Tan, Q. Li, and Z. Li, "Improving ml-based binary function similarity detection by assessing and deprioritizing control flow graph features," in *Proceedings of the 33rd USENIX Conference on Security Symposium*, ser. SEC '24. USA: USENIX Association, 2025.
[42] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 2702–2711.
[43] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ArXiv*, vol. abs/1609.02907, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:3144218
[44] Z. Luo, P. Wang, W. Xie, X. Zhou, and B. Wang, "Iotsim: internet of things-oriented binary code similarity detection with multiple block relations," *Sensors*, vol. 23, no. 18, p. 7789, 2023.
[45] R. Sun, S. Guo, J. Guo, W. Li, X. Zhang, X. Guo, and Z. Pan, "Graphmoco: A graph momentum contrast model for large-scale binary function representation learning," *Neurocomputing*, vol. 575, p. 127273, 2024.
[46] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *ArXiv*, vol. abs/1907.11692, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:198953378
[47] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=XPZIaotutsD
[48] P. He, J. Gao, and W. Chen, "Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing," 2021.
[49] K. Pei, Z. Xuan, J. Yang, S. Jana, and B. Ray, "Learning approximate execution semantics from traces for binary function similarity," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2776–2790, 2023.
[50] D. Kim, E. Kim, S. K. Cha, S. Son, and Y. Kim, "Revisiting binary code similarity analysis using interpretable feature engineering and lessons learned," *IEEE Transactions on Software Engineering*, pp. 1–23, 2022.
[51] A. Marcelli, M. Graziano, X. Ugarte-Pedrero, Y. Fratantonio, M. Mansouri, and D. Balzarotti, "How machine learning is solving the binary function similarity problem," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2099–2116.
[52] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.
[53] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in neural information processing systems*, vol. 32, 2019.
[54] MITRE, "Cve - search cve list," July. 2025. [Online]. Available: https://cve.mitre.org/cve/search_cve_list.html

APPENDIX

A. Challenge of cross-optimization scenario

As depicted in Fig. 4, we present a comparison between two binaries derived from the same source function, *digest_file()*, of the *sha512sum*. These binaries were compiled with different optimization levels, and their assembly code and CFGs are shown for analysis. Specifically, Fig. 4(a) depicts the *digest_file()* function compiled for the 64-bit x86 ISA with the -O0 optimization level (no optimization), while Fig. 4(b) shows the same function compiled with the -O3 optimization level (maximum optimization).

This is a classical scenario in binary code similarity detection, which aims to address two key challenges: distinguishing different functions (the equivalence problem) and identifying similar functions originating from the same source code (the invariance problem) across different optimization levels. Since higher optimization levels aim to produce more efficient and compact code, both the text features of the assembly and the graph features of the CFG may vary. Among these, the graph structure may be the most significantly influenced feature. However, the underlying semantics of the assembly remain unchanged, as the code retains the same functionality. To address these variations, researchers favor text embedding methods over graph embedding approaches for better semantic understanding through assembly text.

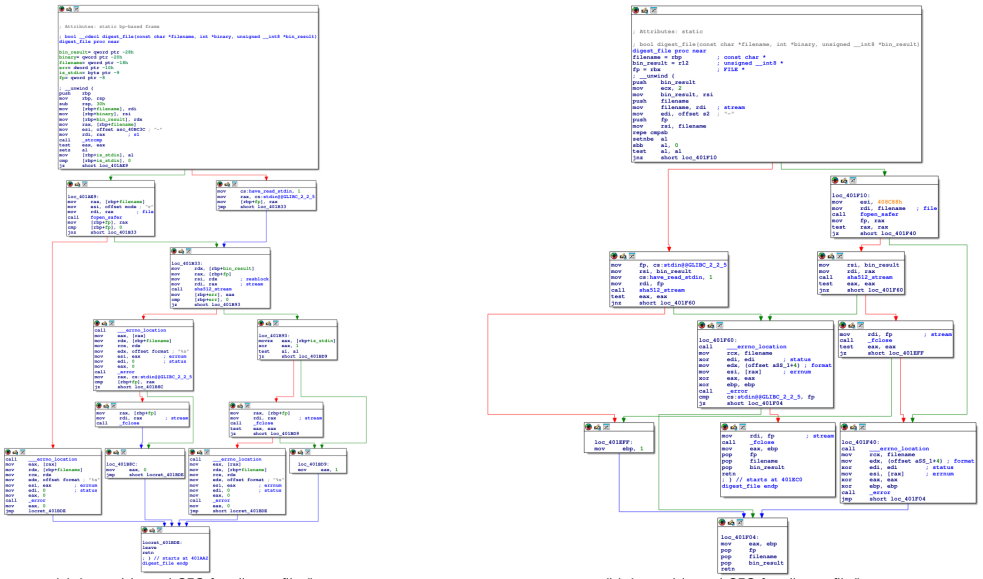
B. Additional evaluation on performance and runtime efficiency

We place Table VII here primarily due to text content limitations and to better organize the content.

Besides evaluating *FuseBinRepr* on the ranking task with different sizes of function pools, we also evaluate it on a threshold-based classification task. This task is based on the unified generation of similar and dissimilar function pairs. Since *FuseBinRepr* and all our baselines primarily generate embedding vectors, they output a confidence score for similarity based on vector distance calculations, rather than a binary flag indicating whether two functions are similar or not. Therefore, we set a threshold for each method. If the confidence score is higher than the threshold, we classify the functions as similar; if the confidence score is lower than the threshold, we classify them as dissimilar.

For each method, we tested thresholds ranging from 0.5 to 0.9. For methods that exhibited high confidence, we also tested higher thresholds of 0.95 and 0.995 to achieve the best performance. The thresholds we determined for each method are as follows: 0.85 for GMN, 0.9 for CLAP, 0.75 for SAFE, 0.5 for Asm2Vec, 0.95 for CLAP+GMN and PalmTree+GMN, 0.995 for sem2vec(GMN), and 0.995 for *FuseBinRepr*.

From the results in Table VIII, we can see that *FuseBinRepr*(SAFE+sem2vec) maintains the best performance in both Accuracy and F1-Score, achieving 10.1% improvements in Accuracy and 11.4% improvements in F1-Score compared to the SOTA baseline. SAFE even outperforms most multimodal methods in terms of accuracy, and CLAP+GMN has a fairly good F1-Score. However, these results do not seem to align with what we observed in MRR and Recall@K. This discrepancy can be attributed to the choice of threshold, which significantly influences performance metrics. These findings suggest that the choice of threshold is closely related to the actual data distribution. Compared to ranking evaluation, classification evaluation is not as perfect for embedding methods, since it introduces some bias due to threshold setting. Despite classifying by threshold, researchers can also add simple classification layers at the end of each embedding model and finetune these layers to convert the models into classification



(a) Assembly and CFG for digest_file ()
in coreutils-6.7_x86_64_O0_sha512sum

(b) Assembly and CFG for digest_file()
in coreutils-6.7_x86_64_O3_sha512sum

Fig. 4. Cross-optimization example

TABLE VII
COMPARISON OF RECALL@5 AND RECALL@10 FOR DIFFERENT METHODS IN BINARY SIMILARITY DETECTION

Method	Recall@5							Recall@10						
	XA	XC	XO	XA+XC	XA+XO	XO+XC	**ALL**	XA	XC	XO	XA+XC	XA+XO	XO+XC	**ALL**
Asm2Vec	0.066	0.152	0.126	0.063	0.085	0.126	0.125	0.084	0.172	0.146	0.079	0.102	0.137	0.142
CLAP	0.060	0.199	0.154	0.039	0.068	0.162	0.171	0.069	0.219	0.171	0.047	0.077	0.186	0.186
SAFE	**0.133**	0.240	0.189	0.146	**0.147**	0.202	0.207	**0.146**	0.263	0.201	**0.170**	**0.159**	0.227	**0.219**
GMN	0.102	**0.288**	**0.194**	**0.156**	0.116	**0.263**	**0.209**	0.111	**0.300**	**0.208**	0.168	0.128	**0.279**	**0.219**
PalmTree+GMN	0.036	0.100	0.064	0.048	0.035	0.071	0.079	0.053	0.119	0.082	0.059	0.050	0.094	0.099
CLAP+GMN	0.073	0.149	0.115	0.073	0.074	0.123	0.123	0.089	0.164	0.133	0.083	0.089	0.138	0.151
sem2vec(GMN)	**1.000**	**0.821**	**0.850**	**1.000**	**1.000**	**0.893**	**0.700**	**1.000**	**0.857**	**0.900**	**1.000**	**1.000**	**0.893**	**0.750**
FuseBinRepr(CLAP+GMN)	**1.000**	0.786	0.650	**1.000**	**1.000**	0.893	0.575	**1.000**	0.821	**0.800**	**1.000**	**1.000**	0.964	0.600
FuseBinRepr(SAFE+GMN)	**1.000**	0.750	0.625	**1.000**	**1.000**	0.857	0.550	**1.000**	0.786	0.775	**1.000**	**1.000**	0.964	0.575
FuseBinRepr(CLAP+sem2vec)	**1.000**	0.857	0.675	**1.000**	**1.000**	0.929	0.625	**1.000**	0.893	0.775	**1.000**	**1.000**	**1.000**	0.650
FuseBinRepr(SAFE+sem2vec)	**1.000**	**0.964**	**0.750**	**1.000**	**1.000**	**0.964**	**0.725**	**1.000**	**0.964**	0.750	**1.000**	**1.000**	0.964	**0.725**

models. This approach makes it suitable for evaluation on classification tasks.

As for runtime efficiency, we can divide the working process of *FuseBinRepr* and the baselines into three parts: the binary preprocessing part, the model loading part, and the embedding part. Since *FuseBinRepr* relies on two foundational models, the embedding before fusion is the same as that of the unimodal baselines. The differences lie in the absence of binary preprocessing and the inclusion of an additional *Feature Fusion Model*. We used all baselines and the *Feature Fusion Model* of *FuseBinRepr* to process 100 functions and generate embeddings. We then calculated the average time spent by each method to form Table IX. From the results, we can see that the *Feature Fusion Model* and unimodal methods consume less time than the multimodal methods. To balance performance and efficiency, researchers can choose *FuseBinRepr*(CLAP+GMN) and *FuseBinRepr*(SAFE+GMN) to save time, while *FuseBinRepr*(SAFE+sem2vec) can be chosen for the best performance.

C. Information of target CVEs

We have attached the detailed information of the 16 selected real vulnerable functions, including their CVE indices, the binaries they belong to, their specific versions, and the names of the vulnerable functions.

Regarding the selection of the 16 vulnerabilities, we first searched the CVE official site [54] for vulnerabilities related to BusyBox, OpenSSL, and Coreutils that exist in our dataset. We filtered out those vulnerabilities that were from different versions than those in the dataset. We also had some compiled additional versions, such as BusyBox v1.36.1, for personal use. Next, we manually checked whether the vulnerable functions existed in the corresponding versions of the binary files using Radare2, an open-source reverse-engineering tool. Only those vulnerabilities with confirmed vulnerable functions were selected as target CVEs. Therefore, other CVE functions can also be selected, and we select these 16 CVEs for the convenience of our dataset.

To use the *FuseBinRepr* for zero-day vulnerability detection,

TABLE VIII
COMPARISON OF ACCURACY AND F1-SCORE FOR DIFFERENT METHODS IN BINARY SIMILARITY DETECTION

Method	Accuracy							F1-Score						
	XA	XC	XO	XA+XC	XA+XO	XO+XC	**ALL**	XA	XC	XO	XA+XC	XA+XO	XO+XC	**ALL**
Asm2Vec	0.542	0.562	0.545	0.548	0.538	0.542	0.554	0.178	0.349	0.253	0.153	0.193	0.235	0.311
CLAP	0.596	0.591	0.588	0.606	0.586	0.608	0.594	0.065	0.161	0.137	0.037	0.073	0.124	0.152
SAFE	**0.641**	**0.749**	**0.681**	**0.712**	**0.661**	**0.742**	**0.674**	0.454	**0.683**	0.576	0.556	0.494	**0.656**	0.575
GMN	0.459	0.518	0.490	0.461	0.470	0.482	0.494	**0.588**	0.637	**0.619**	**0.581**	**0.600**	0.603	**0.621**
PalmTree+GMN	0.477	0.557	0.513	0.501	0.488	0.514	0.514	0.646	**0.716**	0.678	**0.667**	0.656	0.679	0.679
CLAP+GMN	0.500	0.549	0.539	0.499	0.522	0.525	0.528	**0.666**	0.708	**0.700**	0.666	**0.686**	**0.688**	**0.691**
sem2vec(GMN)	**0.564**	**0.637**	**0.590**	**0.590**	**0.566**	**0.606**	**0.598**	0.582	0.666	0.623	0.598	0.593	0.628	0.629
FuseBinRepr(CLAP+GMN)	0.611	0.676	0.633	0.641	0.614	0.652	0.637	0.587	0.675	0.631	0.608	0.601	0.639	0.634
FuseBinRepr(SAFE+GMN)	0.580	0.647	0.603	0.607	0.584	0.616	0.608	0.590	0.670	0.629	0.606	0.603	0.631	0.633
FuseBinRepr(CLAP+sem2vec)	**0.941**	0.469	0.784	0.800	**0.941**	0.718	0.553	**0.970**	0.622	**0.879**	0.889	**0.970**	0.836	0.702
FuseBinRepr(SAFE+sem2vec)	0.882	**0.766**	**0.804**	**1.000**	0.882	**0.795**	**0.776**	0.933	**0.769**	0.875	**1.000**	0.933	**0.862**	**0.805**

TABLE IX
COMPARISON OF RUNTIME EFFICIENCY

Method	Embedding Time (s)			
	binary preprocessing	model loading	embedding	summary
Asm2Vec	0.014582	0.338146	0.081617	**0.434345**
CLAP	0.004867	2.12821	0.054861	2.187938
SAFE	0.006075	1.166326	7.25E-05	1.172473
GMN	0.053979	0.399824	0.0395	0.493303
PalmTree+GMN	199.103850	1.032253	0.121984	200.258087
CLAP+GMN	0.058846	2.528034	0.094361	**2.681241**
sem2vec(GMN)	15.448449	1.572571	0.065654	17.086673
FuseBinRepr (*Feature Fusion Model*)	-	0.398876	0.028351	**0.427227**

TABLE X
INFORMATION OF TARGET CVES

CVE	Binary	Version	Function
CVE-2015-4041	coreutils-sort	8.23	keycompare_mb()
CVE-2023-42366	busybox	1.36.1	next_token()
CVE-2023-42365	busybox	1.36.1	copyvar()
CVE-2023-42363	busybox	1.36.1	xasprintf()
CVE-2021-42386	busybox	1.34.0	nvalloc()
CVE-2021-42385	busybox	1.34.0	evaluate()
CVE-2021-42384	busybox	1.34.0	handle_special()
CVE-2021-42382	busybox	1.34.0	getvar_s()
CVE-2021-42381	busybox	1.34.0	hash_init()
CVE-2021-42380	busybox	1.34.0	clrvar()
CVE-2021-42379	busybox	1.34.0	next_input_file()
CVE-2021-42378	busybox	1.34.0	getvar_i()
CVE-2017-3733	openssl-libssl	1.1.0 before 1.1.0e	ssl3_get_record()
CVE-2014-0195	openssl-libssl	before 0.9.8za, before 1.0.0m, and before 1.0.1h	dtls1_reassemble_fragment()
CVE-2015-0286	openssl-libcrypto	before 0.9.8zf, before 1.0.0r, before 1.0.1m, and before 1.0.2a	ASN1_TYPE_cmp()
CVE-2015-1788	openssl-libcrypto	before 0.9.8s, before 1.0.0e, before 1.0.1n, and before 1.0.2b	BN_GF2m_mod_inv()

researchers should first prepare a large vulnerability database containing vulnerable function samples as our targets. When encountering new binaries, researchers search if any of the vulnerable functions in the database exist within these binaries and then rank the results to identify the top-K similar functions. Finally, researchers will conduct further analysis on the top-K ranked targets to confirm their relevance.