drones

Article

DEGNN: A Deep Learning-Based Method for Unmanned Aerial Vehicle Software Security Analysis

Jiang Du, Qiang Wei, Yisen Wang * and Xingyu Bai

School of Cyber Science and Engineering, Information Engineering University, Zhengzhou 450001, China
* Correspondence: xdeason@126.com

Abstract: With the increasing utilization of drones, the cyber security threats they face have become more prominent. Code reuse in the software development of drone systems has led to vulnerabilities in drones. The binary code similarity analysis method offers a way to analyze drone firmware lacking source code. This paper proposes DEGNN, a novel graph neural network for binary code similarity analysis. It uses call-enhanced control graphs and attention mechanisms to generate dual embeddings of functions and predict similarity based on graph structures and node features. DEGNN is effective in cross-architecture tasks. Experimental results show that in the cross-architecture binary function search, DEGNN's mean reciprocal rank and recall@1 surpass the state of the art by 12% and 28.6%, respectively. In the cross-architecture real-world vulnerability search, specifically targeting drone systems, it has a 33.3% performance improvement over the SOTA model, indicating its great potential in enhancing drone cyber security.

Keywords: unmanned aerial vehicle; cyber security; binary code similarity analysis; graph neural networks

check for updates

Academic Editors: Liuguo Yin and Shu Fu

Received: 1 January 2025
Revised: 28 January 2025
Accepted: 31 January 2025
Published: 2 February 2025

Citation: Du, J.; Wei, Q.; Wang, Y.; Bai, X. DEGNN: A Deep Learning-Based Method for Unmanned Aerial Vehicle Software Security Analysis. *Drones* **2025**, *9*, 110. https://doi.org/10.3390/drones9020110

1. Introduction

With the rapid advancement of technology, unmanned aerial vehicles (UAVs) have become an integral part of modern life. The application spectrum of UAVs has expanded to encompass a wide range of fields, including agricultural monitoring, aerial photography, security surveillance, and power line inspection. UAV technology has demonstrated immense potential and value, significantly improving operational efficiency and substantially reducing costs. It has also provided humanity with unprecedented perspectives and vast amounts of data. Moreover, the progress in UAV technology has spurred the vigorous development of related fields such as UAV battery technology, flight control technology, and simulation technology [1,2]. However, this progress has also introduced new cybersecurity challenges [3]. As UAV systems become more autonomous and intelligent, they face a growing array of security issues, including data breaches, denial of service, and even physical security threats [4].

In the current landscape of UAV network security, the vulnerability of UAV systems to attacks by hackers exploiting software loopholes has emerged as an especially pressing issue [5], making the security analysis of UAV systems critical. Take, for example, the MAVLink protocol, which is widely utilized for command control and telemetry information in UAVs. Due to the deficiencies in its encryption and authentication mechanisms, the security of the UAV system is severely compromised. Currently, ad hoc UAV networks are also faced with a plethora of security threats such as wormhole attacks and rushing attacks. These attacks can disrupt the normal operation of the network and undermine the integrity of data transmission, thus plunging the UAV system into an even more perilous situation. Given that UAVs typically adopt multi-instruction set architectures and that security analysts generally have difficulty accessing the source code of UAV systems, the significance of cross-architecture binary code similarity analysis (BCSA) technology is self-evident. This technology empowers security experts to accurately identify and conduct in-depth analysis of potential security vulnerabilities within the UAV system even in the absence of the source code. Subsequently, they can take proactive measures to address and remediate these vulnerabilities in advance, effectively enhancing the overall security level of the UAV system.

BCSA technology, serving as a pivotal instrument for safeguarding software quality and security, is of essential significance in various critical domains such as malware detection, vulnerability exploration, and copyright compliance verification, thereby establishing a firm foundation for the construction of a stable and dependable software ecosystem. In recent years, the burgeoning growth of deep learning technology has breathed new life into the BCSA domain, with the deep learning-based BCSA technology emerging as a prominent research focus.

Nevertheless, it is undeniable that despite the attainment of certain interim achievements by the deep learning-based BCSA technology, it remains confronted with numerous formidable challenges. From the vantage point of technical methodologies, the extant research efforts can generally be categorized into three principal directions: firstly, the approach based on semantic features, which accomplishes similarity analysis by meticulously excavating the semantic particulars within the code; secondly, the methodology based on functional structural features, which emphasizes the utilization of the functional and structural idiosyncrasies of the code for analytical purposes; and thirdly, the hybrid method integrating both semantic and structural features.

Semantic feature-based methods use natural language processing (NLP) techniques to extract semantic features from instruction sequences, aiming to encapsulate the complete semantics of functions. However, these approaches often overlook structural information, leading to potential information loss. This limitation is particularly evident when identifying structural changes caused by cross-compiler optimizations.

Structural feature-based methods rely primarily on structural features derived from functions, such as Control Flow Graph (CFG) or Abstract Syntax Tree (AST), and employ graph neural networks (GNNs) to vectorize the function structures. However, these methods exhibit relatively poor robustness in cross-architecture BCSA tasks. They struggle to effectively adapt to structural changes in binary functions caused by cross-architecture scenarios.

Methods that combine semantic and structural features integrate NLP techniques with CFG-based processing to enhance the representation of basic blocks. However, most of these approaches require the training of assembly language models, which incurs significant costs. This is primarily due to the lack of publicly available language models in this domain, unlike those for natural languages or source code languages. When researchers attempt to train such language models independently, they face challenges related to computational resource requirements, as well as the adequacy and fairness of the training corpora.

To address the limitations of existing methodologies, this paper presents an innovative BCSA approach leveraging GNNs, namely, the Dual-Embedding Graph Neural Network (DEGNN) model. Specifically, DEGNN incorporates function call information into the CFG structure to obtain the Call-Enhanced Control Graphs (CECGs). This design is inspired by the conclusion in BINKIT [6] and BinFinder [7] that call behaviors exhibit excellent robustness in cross-architecture BCSA scenarios. Subsequently, DEGNN achieves the dual-embedding representation of functions via attention mechanisms, which allows the model to take advantage of both graph structural features and node association characteristics for predicting and analyzing code similarity. Then, through a graph comparison network that combines both function features and node features in dual dimensions, the similarity between two CECGs is calculated to predict the similarity of binary functions. The experimental results validate that our proposed method demonstrates excellent capabilities not only in effectively representing and comparing binary functions within the same-architecture scenarios, but also in exhibiting robust performance in cross-architecture settings. Furthermore, it functions effectively in real-world vulnerability search scenarios, thereby furnishing a more efficient and versatile solution for firmware security analysis.

The main contributions of this paper include the following:

1. We introduce a dual-embedding scheme for functions based on CECGs and GNNs. This scheme effectively circumvents the need for language model pre-training inherent in NLP representation methods and enhances the representation of calling information during node feature initialization.
2. We propose a binary function similarity analysis method utilizing a neural tensor network (NTN) [8] and node statistical matching. This method transforms the problem of binary function similarity analysis into one of similarity score predictions, enabling analyses and similarity predictions of binary functions at both the node and graph levels.
3. We have developed and conducted extensive experiments on DEGNN. The experimental results showcase DEGNN's superiority over state-of-the-art (SOTA) methods. The code for DEGNN is publicly available at https://github.com/kidding1412/DEGNN, accessed on 1 January 2025.

2. Background

2.1. BCSA Technology

The BCSA technique is utilized to identify similarities between binary codes. This approach finds its applications in various tasks such as vulnerability detection, malware classification, and code plagiarism identification [9,10]. For example, when a zero-day vulnerability is discovered within a prevalent fundamental open-source component, it becomes essential to identify all affected downstream software as efficiently and accurately as possible. Most of these downstream software components are binary files without accessible source code, such as the MAVLink protocol specific to UAV systems, the ad hoc network, and the OpenSSL-based UAV ground control station. Among them, OpenSSL has experienced the infamous "Heartbleed" vulnerability [11], which triggered a massive security crisis on a global scale. Numerous software systems that rely on OpenSSL for encrypted communication suffered severe security threats and a large amount of sensitive information was at risk of being stolen.

2.2. Cross-Architecture Challenges

The compilation of source code into binary code is inherently architecture-dependent, yielding disparate binary representations that, despite embodying equivalent functionality, exhibit substantial differences in functional structure and assembly representation due to the variances in instruction sets across architectures. This divergence poses significant challenges for binary function similarity analysis, as the analysis must discern analogous functionalities amid the heterogeneity of binary expressions.

2.3. Problem Definition

The central thesis of this investigation delves into the realm of BCSA, with the primary objective being the assessment of resemblance between pairs of binary function codes.

The underlying assumption guiding this study posits that various incarnations of binary functions, derived from an identical source code but manifested through disparate compilation trajectories—encompassing divergent hardware architectures, compiler technologies, or levels of optimization—are to be perceived as congruent. This paradigm of similarity delineates the foundational benchmark for the scrutiny and evaluation of binary function search methodologies. To clarify the research question, this paper outlines specific functions and definitions of similarity that support the discussion framework.

2.3.1. Definition 1: Function

Consider S as a piece of source code, and let C denote the collection of all possible compilation pathway configurations. Each element $c \in C$ specifies a unique compilation path, encompassing different hardware architectures, compilers, and levels of optimization, among other factors. For each compilation configuration c, a compilation process, denoted as $\text{Compile}(S, c)$, exists. This process transforms the source code S into a binary function F_c, in accordance with the formula

$$\forall c \in C, \exists F_c : F_c = \text{Compile}(S, c) \tag{1}$$

2.3.2. Definition 2: Similarity

For any source code S and two configurations $c_1, c_2 \in C$, if the functions $F_{c_1} = \text{Compile}(S, c_1)$ and $F_{c_2} = \text{Compile}(S, c_2)$ are compiled from the same source code S, then $F_{c_1} \sim F_{c_2}$. In other words, functions generated from the same source code under different configurations are considered similar.

3. Related Work

3.1. Semantic Feature-Based Methods

InnerEYE [12] transforms assembly instructions into embeddings and uses long-short-term memory (LSTM) [13] networks to represent the instructions of each basic block as block embeddings, which are then stored in a locality-sensitive hashing database for efficient online searches. SAFE [14] uses Word2Vec [15] for instruction embedding and constructs semantic vector representations using LSTM and bidirectional recurrent neural network (Bi-RNN) [16] models to measure code similarity. BinDeep [17] extracts instruction sequences from binary functions, vectorizes instruction features with an embedding model, and subsequently applies a recurrent neural network (RNN) [18] deep learning model to identify and compare specific function types. FASER [19] processes binaries with radare2 [20], converting them into ESIL intermediate representation, concatenating all ESIL instructions into a long string as the function's string representation, and employs the LongFormer [21] model to generate function embedding representations.

3.2. Structural Feature-Based Methods

Representative works such as Gemini [22] and VulSeeker [23] utilize the Structure2Vec [24] method to extract structural embeddings from CFGs, which are then applied for similarity evaluation within Siamese networks. Further advancements, such as FuncNet [25] and GMN [26], integrate attention mechanisms, interface call data, and basic block attributes to enrich structural representation and enhance the precision of cross-architecture similarity. Asteria-Pro [27] encodes the AST using Tree-LSTM [28] and computes the similarity between the two encodings with a Siamese architecture.

3.3. Methods Combining Semantic and Structural Features

Asm2Vec [29] linearizes CFG structures into instruction sequences, employing Word2Vec and PV-DM [30] to extract both word and function embeddings. GENN [31] and DeepBinDiff [32] combine instruction embeddings with CFG node features, utilizing Structure2Vec and multi-hop matching algorithms for fine-grained graph embeddings. OrderMatters [33] and Codee [34] extend this by employing BERT [35] and message-passing neural networks (MPNNs) [36], incorporating CFG node and order information, which improves the completeness and precision of similarity computation by blending structural and semantic features. jTrans [9] incorporate BERT models, and enriching the representation with structural information such as CFG node sequences or jump target positions enhances the model's ability to capture code structure. In addition, many works such as CEBIN [37] and Clap [38] are developed based on jTrans.

4. DEGNN Approach

To address the issues presented in Section 1, this paper introduces a novel BCSA solution based on GNNs, named DEGNN. This approach leverages GNNs' capabilities along with CECGs to enhance the understanding of binary code structure and semantics, thereby enabling more effective adaptation to the impacts of compiler optimizations and architectural differences. Furthermore, DEGNN circumvents the prerequisite of language model pre-training inherent in conventional NLP-based methods, thereby reducing computational demands. It also employs a two-tiered feature representation of nodes and graphs, which elevates the model's accuracy and recall in binary function similarity analysis. Additionally, the model enriches node attribute representation and utilizes a method based on NTN and node statistical comparison for similarity prediction, effectively boosting cross-architecture code analysis capabilities.

4.1. Overview

To address the challenging BCSA problem efficiently, our research has explored the DEGNN model. This innovative model melds the pivotal technologies of function vectorization with comparative function search analysis. We delve into the nuanced implementation of these integral stages in the subsequent discussion.

The process by which the function dual-embedding extraction stage operates is illustrated in Figure 1. Initially, DEGNN employs call information to transform the CFG into a CECG. This transformation not only preserves the structural integrity of the original CFG but also enhances the calling information. Subsequently, node information is refined using graph convolutional networks (GCNs) [39], keeping the architecture of the graph intact and thereby achieving superior node embedding representations. Following this, DEGNN utilizes an attention mechanism to amalgamate node embeddings with the graph's architectural data, culminating in a comprehensive vector representation of the function graph, or in other words, graph embedding. In the final step, both node embeddings and graph embeddings are employed as the dual-embedding representations for binary functions, which are then cataloged in the function feature library. This repository aids in the execution of future function searches and comparisons.

Figure 2 presents the workflow associated with the phase of predicting function similarity. Initially, the process of comparing two functions commences with the extraction of graph embedding and node embedding data from the function feature repository. Subsequently, this embedded information is channeled into two distinct alignment processes. On one front, the NTN is harnessed to dissect graph-level data, enabling the delineation of graph relationship attributes. Simultaneously, on the other front, node-level data are scrutinized using a bespoke node comparison technique, facilitating the illumination of node correlation characteristics. Ultimately, similarity scores are predicted through the processing of a fully connected network (FCN).

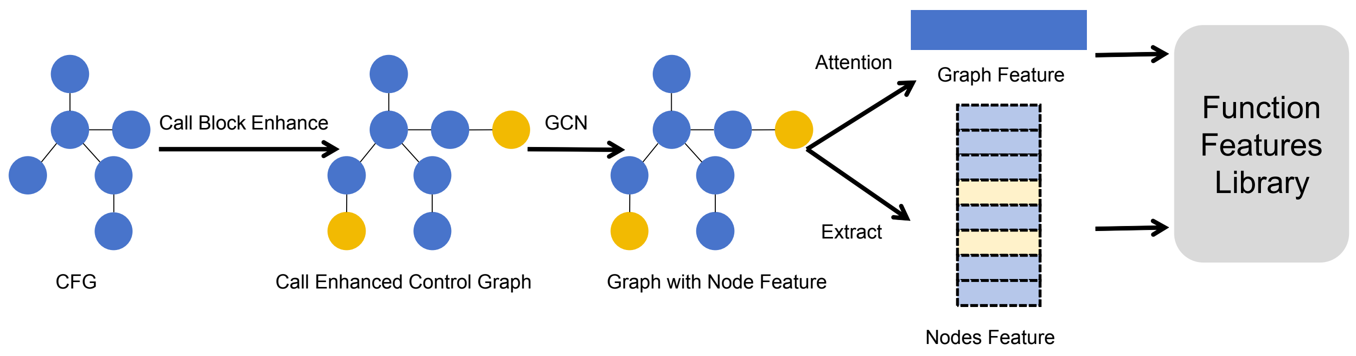

Figure 1. Binary function dual-embedding feature extraction.

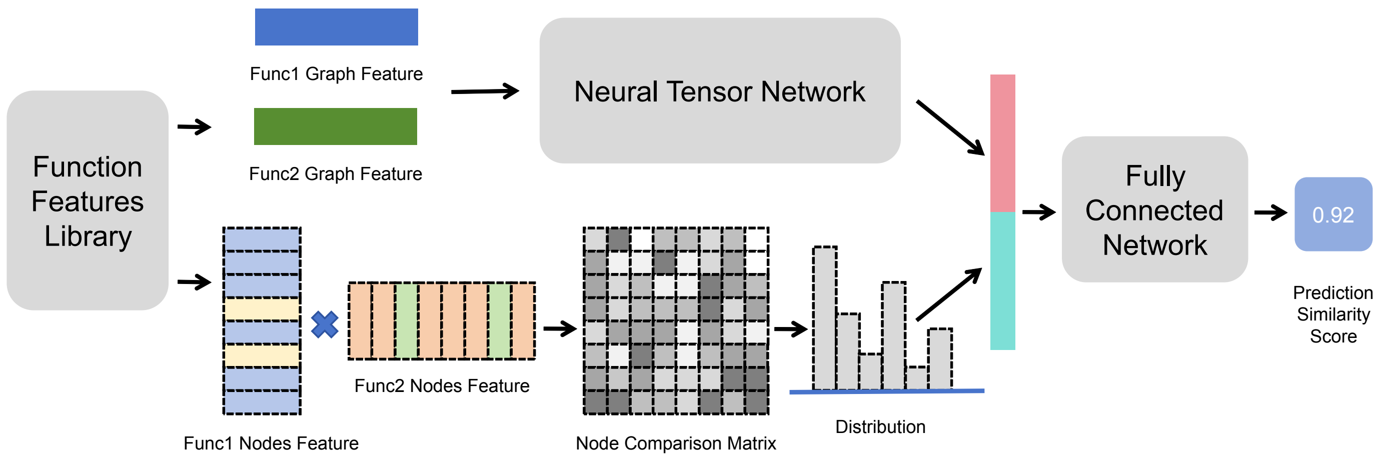

Figure 2. Binary function similarity prediction network.

4.2. *Function Dual-Embedding Feature Extraction*

The extraction of dual embeddings encompasses three pivotal steps: the initial construction of the CECG, the derivation of node attributes via a GCN, and the acquisition of graph characteristics using an attention mechanism.

4.2.1. Constructing CECGs

BINKIT [6] and BinFinder [7] have demonstrated the significant roles of function structures and call information in BCSA tasks, particularly when distinguishing across different architectures and optimization levels. Building upon these insights, this paper introduces the CECG, a novel approach that integrates both function structure and call information.

A CECG is an advanced graph structure that amalgamates function call details with the conventional CFG to substantially improve the representation of function calls. The construction of a CECG involves two critical processes: the establishment of the graph's structure and the extraction of node features.

The main objective of structure construction is to facilitate the establishment of an edge index for the CECG. In this context, the article categorizes basic blocks into two distinct types: the traditional instruction basic block within a conventional CFG (e.g., the blue nodes in the CFG in Figure 1); and the call basic block, defined by information pertaining to function calls (e.g., the yellow nodes in the CFG in Figure 1). When a function call occurs within an instruction basic block, program execution is diverted to the corresponding call basic block, created expressly for this purpose. The function call is then processed, and upon completion, control returns to the original flow. It is critical to note that the construction of the call basic block is non-recursive; such blocks are derived solely from instruction basic blocks exhibiting calling behavior, without generating new call basic blocks for subsequent calls within them.

The feature construction phase is dedicated to formulating node features within the CECG. Following the completion of structure construction, each basic block is represented vectorially based on manually extracted features. To minimize potential biases associated with exclusive reliance on manual feature extraction, this work incorporates a pre-semantic feature system akin to that utilized in the BINKIT framework for constructing binary function features. This system serves as the foundation for refining the feature set associated with call basic blocks. This research adopts pre-semantic features that are universally recognized in the historical literature and eschews more subjective methods, such as the categorization of instruction semantics. Consequently, the selection of features is comprehensive and objective, diminishing the reliance on subjective expertise and thereby reducing bias to the greatest extent feasible.

This research involves the vectorization of two distinct types of basic blocks: instruction and call. These blocks have been transformed into 32-dimensional vectors to facilitate in-depth analysis. For the instruction basic blocks, Table 1 presents the specific characteristics selected for examination in this study. The defining features of the call basic blocks chosen for this research are detailed in Table 2.

Table 1. Instruction basic block characteristics.

Position	Content
0–2	Node ID
3–7	Node length
8–10	Constant number
11–13	Number of calls
14–16	Number of strings
17–19	Out-degree
20–22	In-degree
23	Compiler
24–25	Architecture
26	Bit
27–28	Optimization
29	Ret flag
30	First block flag
31	Instruction block flag

Table 2. Call basic block characteristics.

Position	Content
0–2	Node ID
3–17	Call hash
17–18	Call count in the function
19–20	Called count in the function
21–22	Called count
23–24	Call count in the function
25–26	Call count
27–28	External call count in the function
29–30	Total external call count
31	Call block flag

4.2.2. Node Feature Extraction

Studies have shown that GCNs possess representation invariance and inductive properties [39], and can calculate node embeddings according to their operations for any unseen graphs. Therefore, we choose a GCN to perform the node embedding in the first step.

Central to GCNs' approach to graph data processing is the principle that nodes enhance their own feature representation by assimilating information from their immediate neighbors. This methodology of neighbor-centric information amalgamation empowers the model to discern local structural motifs within the graph, enabling the effective acquisition of node embedding representations.

$$\mathrm{conv}(u_n) = f_1\left(\sum_{m\in\mathcal{N}(n)} \frac{1}{\sqrt{d_n d_m}} u_m W_1^{(l)} + b_1^{(l)}\right) \tag{2}$$

In the context of (2), the term $\mathrm{conv}(u_n)$ epitomizes the transformed feature vector of node n after the application of graph convolution. This procedure orchestrates a weighted summation of feature vectors u_m corresponding to every node m within the vicinity $\mathcal{N}(n)$ of n, thereby updating u_n. The weighting coefficient $1/\sqrt{d_n d_m}$, considering the degrees d_n and d_m of the nodes n and m, respectively, aims to neutralize the undue influence exerted by highly connected nodes. The matrix $W_1^{(l)}$ serves as the adaptable parameter for this layer, tasked with transforming the dimensionality of the node feature vector, while the vector $b_1^{(l)}$ acts as a bias term, enhancing the model's expressive power. The function f_1, often represented by nonlinear activation functions like ReLU, is invoked to augment the model's capacity for nonlinear differentiation.

4.2.3. Function Feature Extraction

By integrating the graph attention mechanism (GAT) [40], a global vector representation of the function graph is generated. The GAT calculates the importance weight of each node, resulting in a comprehensive global representation of the function. The specific calculation formula is as follows:

$$g = \sum_i \alpha_i h_i \tag{3}$$

where α_i represents the attention-based weight, and h_i denotes the node embedding. Through this method, the node features are combined with the global graph embedding, forming a dual-embedding representation of the function, which is then stored in the feature library.

Using this approach, we can characterize the function from both structural and nodal dimensions, enhancing the expressive power of the features and, to some extent, mitigating the information loss inherent in graph-to-embedding transformations.

This comprehensive representation seamlessly integrates individual node information with the broader context, culminating in a dual-embedding representation of each function that is cataloged within the function database.

4.3. Similarity Prediction Network

Essentially, this network consists of three main parts: the NTN, which forms the basic framework; the node comparison network, which enables detailed analysis; and the fully connected prediction network, which is in charge of producing accurate results.

4.3.1. Neural Tensor Network

In the realm of graph-structured data analysis, understanding and delineating the nuances of interactions between two graph embeddings is a pivotal endeavor. Conventional approaches, relying on basic vector computations such as measuring cosine similarity or calculating an inner product between embeddings, are useful in the formative stages of relationship modeling. However, they struggle to identify intricate distinctions within complex graph formations. This insufficiency arises from their inability to fully leverage the detailed structural and attributive nuances embedded in the graph representations, resulting in a relational model that lacks depth and flexibility.

To address these challenges, this section explores the effective deployment of NTN, a neural network architecture designed to manage complex data relationships. Equipped with a sophisticated weight tensor, NTN can decipher not only linear correlations but also the elusive nonlinear and higher-order interactions inherent to the graph embeddings. This capability enhances NTN's effectiveness in unraveling the multifaceted nature of graph data, which is rich in convoluted structural details and sophisticated patterns of node connectivity. To be precise, this paper employs the NTN model to calculate the relationship between two graph embeddings, h_i and h_j:

$$g(h_i, h_j) = f_2\left(h_i^T W_2^{[1:K]} h_j + V\begin{bmatrix} h_i \\ h_j \end{bmatrix} + b_2\right) \tag{4}$$

Within the framework of the model, $W_2^{[1:K]} \in \mathbb{R}^{D\times D\times K}$ represents a defined weight tensor, intricately crafted to capture the complex, high-order interactions among embeddings. Likewise, V, shaped as a matrix with dimensions $V \in \mathbb{R}^{2D\times K}$, aims to delineate the direct impacts exerted by the twin embeddings. Accompanying these are b_2, a bias vector within the realm of $\mathbb{R}^K$, and $f_2(\cdot)$, an activation function, both integral components designed to enhance the model's ability to express nonlinear relationships. Here, K serves as an adjustable hyperparameter, representing the spectrum of interaction types meticulously considered within the model's architecture.

4.3.2. Node Comparison

Graph embedding techniques encapsulate the structural and feature information within a graph into a fixed-dimensional vector, a process that necessitates the integration of all nodes and edges to generate a feature vector representative of the entire graph. Given the inherent challenges of compressing complex, multi-dimensional graph data into a singular vector format, it is almost inevitable that some degree of information loss will occur during this transformation.

In response to the identified challenges, this study also conducted a detailed analysis of node relationships between two graphs during the execution of graph embedding computations. Consider two functions G_1 and G_2, where G_1 contains N_1 nodes and G_2 contains N_2 nodes. Each node is characterized by a D-dimensional feature vector, leading to the creation of two node vector matrices: the node vector matrix U_1 for G_1 is an $N_1 \times D$-dimensional matrix, and the node vector matrix U_2 for G_2 is an $N_2 \times D$-dimensional matrix. Here, D signifies the spatial dimension of the vector. By computing $S = \sigma(U_1 \cdot U_2^T)$, the node interaction relationship matrix S is derived, where σ denotes the sigmoid function used for normalizing the relationship score. Consequently, the dimensions of S are $N_1 \times N_2$. Recognizing that the node counts of the two graphs may differ in practical scenarios, we set $N = \max(N_1, N_2)$ and readjust S to an $N \times N$ matrix, populating the vacancies with zeros to maintain consistency.

Next, the study proceeds to perform histogram feature extraction on the matrix S. Specifically, each element within S is categorized according to its value range and segmented into M bins. Consequently, the elements in S that fall into the range $\left[\frac{(n-1)}{M}, \frac{n}{M}\right)$ are allocated to the nth bin of the histogram, where $n \in \{1, 2, \ldots, M\}$. Following this allocation, the proportion of each value in the histogram, relative to the total number of elements, $N \times N$, is calculated. This step is vital as it reflects the distribution of element values within specified ranges. These proportional values are subsequently aggregated into a vector, which is tailored to accurately represent the correlation between the nodes of the functions.

4.3.3. Fully Connected Network

An FCN is employed to integrate features sourced from the NTN and node comparison strategies for the purpose of predicting the similarity of binary functions. This architecture facilitates the mapping from a multi-dimensional feature space to similarity scores.

The FCN within DEGNN comprises two layers. The network's input is a concatenated vector, integrating both function graph relational features and node association characteristics. Initially, this concatenated vector is processed by the first fully connected layer, which employs the ReLU activation function. The use of ReLU facilitates nonlinear mapping, substantially improving the model's representational capabilities. Following this initial processing, the vector advances to the second fully connected layer. This layer's output dimensionality is purposefully set to one, aiming to distill the complex input feature vector into a singular similarity score.

To ensure that the resultant similarity score resides within a logical bound, the model incorporates the sigmoid activation function after the second layer. The sigmoid function is adept at mapping any real-valued number to the (0,1) interval, rendering it exceptional for the normalization of similarity scores. Moreover, by moderating the effects of outlier values, the sigmoid activation function contributes to the model's overall robustness. The operations executed by the fully connected layers can be formally encapsulated by the following mathematical representation.

The output y of the fully connected layer is given by

$$y = \sigma(W_4 \cdot \mathrm{ReLU}(W_3 x + b_3) + b_4) \tag{5}$$

In the first layer of the fully connected network, the key parameters are the weight matrix, denoted as W_3, the bias vector, b_3, and the activation function, which in this case is the rectified linear unit (ReLU). For the second layer of the fully connected network, the parameters include the weight matrix W_4, the bias vector b_4, and the activation function, which is the sigmoid function, represented as σ.

DEGNN optimizes the network parameters by utilizing the mean squared error (MSE) loss function, which serves as the objective function for optimization. The MSE loss function is widely employed in regression problems and serves to quantify the discrepancy between the predicted similarity scores generated by the model and the actual labels. The mathematical expression for the MSE loss function is given by

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6}$$

In (6), the term n represents the number of samples in the dataset. The variable y_i corresponds to the true label value for the ith sample, and $\hat{y}_i$ denotes the similarity score as predicted by the model for the same sample. This predicted similarity score is the resultant output of the fully connected neural network after processing the ith sample.

5. Experimentation and Evaluation

The experimental evaluation is structured to address the following research questions:

RQ1: How does DEGNN compare with other state-of-the-art baseline models in the BCSA task when using the same architecture? (Section 5.5)

RQ2: What is the performance of DEGNN relative to baselines in cross-architectural BCSA tasks? (Section 5.6)

RQ3: To what extent do the CECG and NTN components influence the performance outcomes of the model? (Section 5.7)

RQ4: How effectively does DEGNN perform in scenarios involving the search for known vulnerabilities? (Section 5.8)

RQ5: How does the inference efficiency of DEGNN compare with baseline models in terms of time consumption during the reasoning process? (Section 5.9)

To evaluate the performance of the DEGNN framework, experiments were conducted in a controlled computing environment consisting of a single Linux server, specifically running Ubuntu 20.04.6 LTS. The server was outfitted with an Intel(R) Xeon(R) W-2245 processor with a clock speed of 3.9 GHz, an NVIDIA RTX A4000 graphics card with 16 GB of VRAM, and a total of 128 GB of system RAM.

5.1. Dataset

The study leverages BINKIT, a comprehensive dataset specifically curated for addressing binary code similarity challenges, to conduct experimental validation. BINKIT encompasses a collection of 243,128 binaries and 36,256,322 functions, compiled from 51 different packages using 1352 unique combinations of compiler configurations, optimization settings, and target architectures. This dataset not only includes the principal components found in existing benchmarks, but also incorporates a diverse array of eight different combinations of processor architectures and versions. It takes into consideration the effects of various compiler options, such as position-independent executables (PIEs), link-time optimization (LTO), and code obfuscation.

We have established five datasets on the foundation of BINKIT for various BCSA tasks.

Dataset 1: Cross-Architecture Comparison Dataset. This dataset is composed of 50,000 non-repeating triplets (f, g^+, g^-), where function f is randomly selected from all architectures in BINKIT, g^+ is a function similar to f, and g^- is a function dissimilar to f. The data are divided into training, validation, and testing sets in an 8:1:1 ratio.

Dataset 2: Same-Architecture Comparison Dataset. Consisting of 50,000 triplets (f, g^+, g^-) randomly selected from the x86 architecture in BINKIT, this dataset is also segmented into training, validation, and testing sets in an 8:1:1 ratio.

Dataset 3: Cross-Architecture Function Search Dataset. This dataset is created by randomly selecting 10,000 dissimilar functions from BINKIT to form a function pool F. The function pool G is composed of similar functions to 100 randomly chosen functions from F, and is used for cross-architecture binary function search tasks.

Dataset 4: Same-Architecture Function Search Dataset. Similar in construction to Dataset 3, this dataset is formed by randomly selecting 10,000 dissimilar functions to create function pool F, with function pool G comprising similar functions to 100 randomly chosen functions from F. It is utilized for testing same-architecture binary function search tasks. Based on different optimization combinations, we have constructed multiple versions of Dataset 3 and Dataset 4, which are used to evaluate the model's search capabilities when dealing with binary functions of different optimization levels.

Dataset 5: Real-World Vulnerability Search Dataset. Constructed from the OpenSSL 1.0.1 project, which contains 21 known vulnerabilities and is widely used in components integral to drone ground control stations, this dataset adheres to the BINKIT methodology for cross-architecture datasets. It includes a set of 999 nonvulnerable functions as the secure function group and a group of 20 vulnerable functions randomly selected from the known vulnerable cross-architecture binary functions.

5.2. Baseline

In this study, we deliberately selected one representative baseline method from each of the three BCSA technology routes introduced in Section 2: Methods Based on Semantic Features, Methods Based on Functional Structural Features, and Methods Combining

Semantic and Structural Features. These baselines were chosen from top-tier academic conferences and are accompanied by the most comprehensive official code implementations, ensuring a high standard and minimal human interference during replication. There follow brief descriptions of each baseline method:

SAFE [14]. A method based on semantic features, SAFE employs an RNN architecture with attention mechanisms to generate a representation of the analyzed function, receiving assembly instructions as input. We implemented this baseline based on its official PyTorch code and used default parameter settings throughout our evaluation.

Asteria-Pro [27]. Representing methods based on functional structural features, Asteria-Pro deeply leverages structural information during 1-to-N searching, rapid pre-filtering, and encoding of the function's AST. We implemented Asteria-Pro based on its official code and adhered to the default parameter settings.

jTrans [9]. A method combining semantic and structural features, jTrans adapts the unique positional vector concept from BERT-based NLP methodologies to interpret jump information within functions. This allows sequence-oriented natural language models to process and understand structural jumps in code functions. We implemented jTrans based on the official BERT source code and utilized the pre-trained models from the official repository. It is important to note that jTrans, as mentioned in its paper, only supports same-architecture comparisons; hence, it was not included in the cross-architecture experiments.

To comprehensively investigate the impact of the CECG and NTN components on the model, two additional models were introduced for ablation experiments. Specifically, the DEGNN-noCECG model was constructed by employing a CFG and an identical node feature construction strategy, with the aim of precisely validating the effect of the CECG on the model's performance. Meanwhile, the DEGNN-noNTN model focused on the performance manifestation of the model when using a common fully connected network with a shape of $2n \times n$, which was designed to verify the influence of the NTN network on the model. Here, n represents the vector dimension extracted by the graph attention network and 2n is the length of the concatenated two-function vectors.

5.3. Parameter Settings and Model Training

In constructing the DEGNN architecture, this study opted to configure the GCN with three layers, a decision supported by widespread findings in GCN research that models with two to three layers often achieve a good balance between leveraging graph structural information and avoiding overfitting, thus attaining high classification accuracy. As the model depth increases, so does the number of parameters, which can lead to overfitting and degrade the model's performance on the test set. Consequently, our initial feature vector dimensions were set to 32, and considering that higher dimensions would increase computational overhead, testing on the validation set validated that 16 dimensions represent a balanced choice between performance and efficiency. Regarding the training process, the study employed the Adam optimization algorithm to adjust network parameters. The learning rate was set at a constant 0.001. After 200 epochs of training and evaluation of model performance in the validation set, the dropout rate was established at 0.2. The model's validation set performance is depicted in Figure 3.

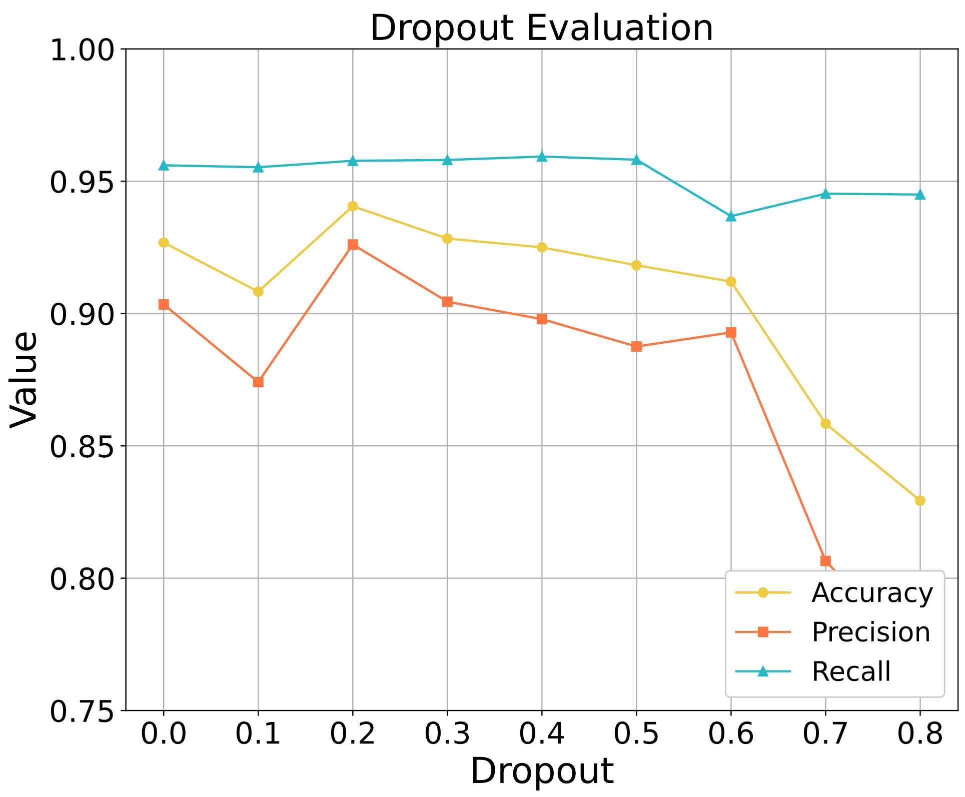

Figure 3. Model performance on the validation set.

5.4. Evaluation Metrics

In this study, we selected AUC, MRR, and recall@1 as our evaluation metrics. The AUC (area under the curve) is a commonly used performance metric in the deep learning domain for binary classification tasks, assessing the model's performance after training on the test set. MRR (mean reciprocal rank) and recall@1 are standard metrics in search tasks, specifically tailored for evaluating the scenario of a single best match in binary function search tasks, which is the primary application scenario of our work, involving one-to-many binary function searches. The authors of the jTrans study also used the MRR and recall@1 metrics in their experiments to evaluate the performance of models in binary function search tasks.

The calculation formula for the AUC (area under the curve) is as follows:

$$\text{AUC} = \frac{1}{|P| \cdot |N|} \sum_{p_i \in P} \sum_{n_j \in N} I(f(p_i) > f(n_j)) \tag{7}$$

where *I(x)* is an indicator function defined as

$$I(x) = \begin{cases} 0, & \text{if } x = \text{False} \\ 1, & \text{if } x = \text{True} \end{cases} \tag{8}$$

In (7), $P = \{p_1, p_2, \ldots, p_m\}$ represents the set of positive samples, $N = \{n_1, n_2, \ldots, n_n\}$ represents the set of negative samples, and $f(x)$ denotes the score assigned by the model to sample x. The AUC measures the probability that a randomly chosen positive sample has a higher score than a randomly chosen negative sample.

The calculation formula for recall@1 is as follows:

$$\text{Recall@1} = \frac{1}{|F|} \sum_{f_i \in F} I(\text{Rank}_{f_i^{gt}} \leq 1) \tag{9}$$

The calculation formula for MRR is

$$\text{MRR} = \frac{1}{|F|} \sum_{f_i \in F} \frac{1}{\text{Rank}_{f_i^{gt}}} \tag{10}$$

In the above formulas, $F = \{f_1, f_2, \ldots, f_i, \ldots, f_n\}$ represents the function pool, $G = \{f_1^{gt}, f_2^{gt}, \ldots, f_i^{gt}, \ldots, f_n^{gt}\}$ represents the corresponding ground truth function pool, and $\text{Rank}_{f_i^{gt}}$ denotes the ranking of the function f_i in the ground truth function pool G corresponding to the true function f_i^{gt}.

5.5. *Same-Architecture BCSA Tasks*

We conducted two experiments within the same architectural context to evaluate the performance of DEGNN and the baseline models on dataset 2.

5.5.1. Experiment 1: One-to-One Binary Function Similarity Matching

In this experiment, jTrans was specifically trained on the training set of Dataset 2, while the other models, having been trained on Dataset 1, were directly tested on the test set of Dataset 2. This task involved determining the similarity between binary functions. The performance metric used was the AUC of the ROC curve. Figure 4 illustrates the ROC curves and the corresponding AUC scores for DEGNN, Asteria-Pro, jTrans, and SAFE on the test set of Dataset 2. DEGNN excelled, with an AUC score of 0.9833, outperforming the second-place competitor by 1.1%.

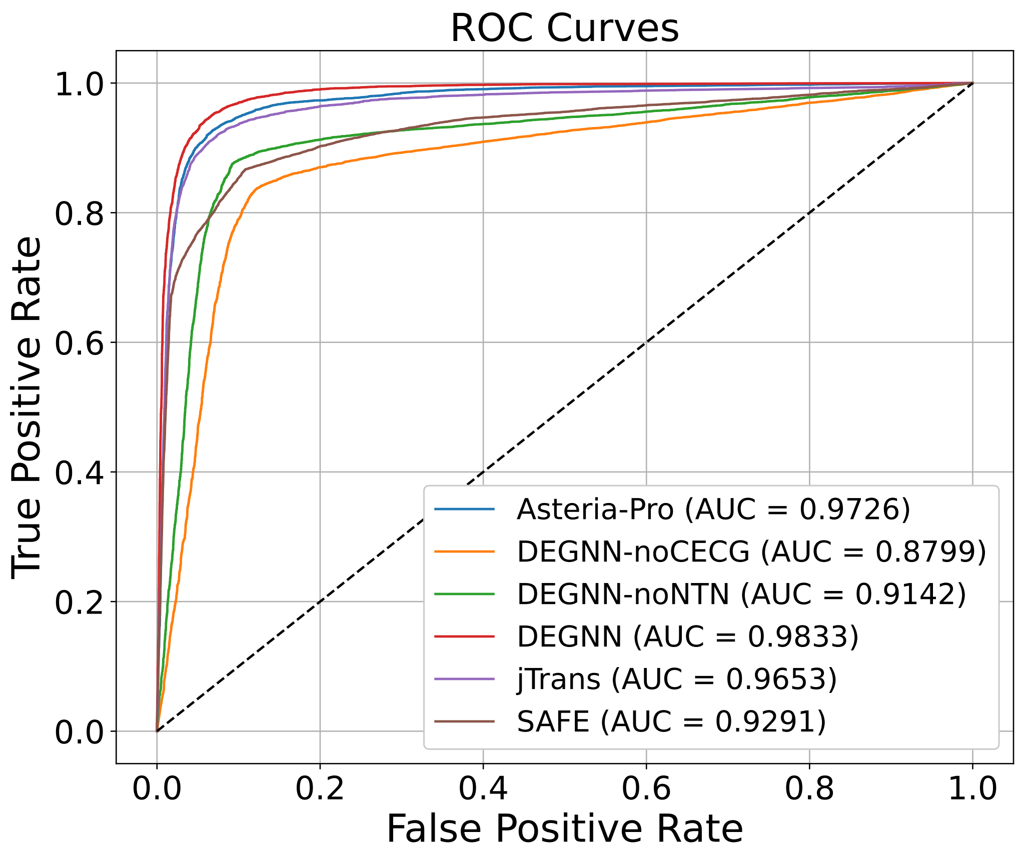

Figure 4. ROC curves and AUC scores on the same-architecture comparison dataset.

5.5.2. Experiment 2: Same-Architecture Binary Function Search

For the second experiment, we evaluated the models' performance on same-architecture binary function search tasks using Dataset 2. For each function in the function pool G, we conducted 10 searches for it within the function pool F. Then, for each of these individual searches, we calculated the MRR and recall@1 and finally averaged these values across all searches. Table 3 presents the results for MRR and recall@1 for DEGNN, Asteria-Pro, jTrans, and SAFE. DEGNN surpassed all baseline models in both the MRR and recall@1 metrics across all optimization levels, achieving the highest average MRR and recall@1 scores of 0.660 and 0.594, respectively.

In the experiments under the same-architecture scenario, the SAFE model, which relies on the attention mechanism, exhibited the poorest performance among the compared baselines. This was primarily due to its deficiency in capturing function structure information. In contrast, jTrans, Asteria-Pro, and our proposed DEGNN model achieved better performance in function representation. jTrans incorporated jump information, Asteria-Pro utilized the AST, and DEGNN integrated the CECG to enrich the function structure information. Notably, DEGNN, which combines call information with the CFG, attained an AUC score of 0.9833, an average MRR of 0.639, and an average recall@1 score of 0.576. These results evidently demonstrate the effectiveness of the CECG design and the excellent performance of the dual-dimension graph comparison network in DEGNN.

Table 3. Same-architectural binary function search.

	MRR							**Recall@1**						
Model	**00, O3**	**01, O3**	**02, O3**	**00, Os**	**01, Os**	**02, Os**	**Avg.**	**00, O3**	**01, O3**	**02, O3**	**00, Os**	**01, Os**	**02, Os**	**Avg.**
SAFE	0.122	0.252	0.617	0.155	0.361	0.385	0.315	0.091	0.263	0.609	0.102	0.283	0.326	0.279
jTrans	0.481	0.682	0.749	0.552	0.683	0.697	0.641	0.392	0.603	0.687	0.458	0.627	0.688	0.576
Asteria-Pro	0.463	0.650	0.711	0.527	0.662	0.681	0.616	0.301	0.554	0.627	0.417	0.591	0.675	0.528
DEGNN-noCECG	0.302	0.463	0.652	0.331	0.519	0.541	0.468	0.224	0.481	0.528	0.283	0.426	0.509	0.409
DEGNN-noTNT	0.429	0.587	0.672	0.481	0.605	0.614	0.565	0.287	0.560	0.608	0.323	0.419	0.653	0.475
DEGNN	0.498	0.701	0.772	0.581	0.696	0.713	0.660	0.405	0.631	0.702	0.492	0.641	0.695	0.594

5.6. Cross-Architectural BCSA Tasks

To thoroughly evaluate the performance of DEGNN in cross-architectural BCSA tasks, we conducted two distinct experiments using Dataset 1 and Dataset 3.

5.6.1. Experiment 1: One-to-One Binary Function Similarity Matching

In the first experiment, we trained DEGNN, Asteria-Pro, and SAFE using the training set of Dataset 1 and evaluated the models using the corresponding test set. This task involved classifying whether two binary functions were similar or not, treating it as a binary classification problem. The performance of the models was measured using the AUC of the ROC curve. Figure 5 displays the ROC curves and the corresponding AUC scores for DEGNN, Asteria-Pro, and SAFE on the test set of Dataset 1. DEGNN demonstrated superior performance, with an AUC score of 0.9808, outperforming the second-place competitor by approximately 1.79%.

5.6.2. Experiment 2: Cross-Architectural Binary Function Search

In the second experiment, we assessed the performance of the models in cross-architectural binary function search tasks using Dataset 3. Each function in the function pool G was searched against the function pool F, and the MRR and recall@1 were calculated and averaged across all searches. Table 4 presents the MRR and recall@1 results for DEGNN, Asteria-Pro, and SAFE. DEGNN outperformed all baseline models in both MRR and recall@1 metrics across all optimization levels. DEGNN achieved the highest average MRR and recall@1 scores of 0.639 and 0.576, respectively, outperforming the second-place competitor by 12.0% in terms of MRR and by 28.6% in terms of recall@1.

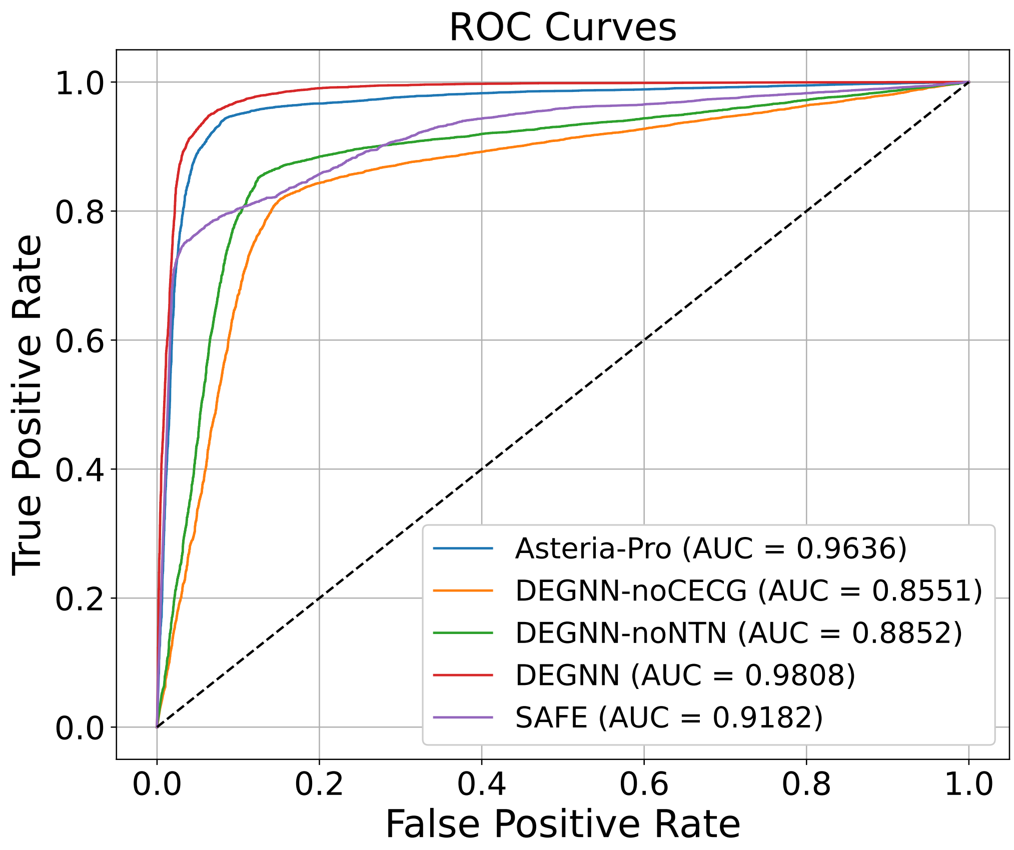

Figure 5. ROC curves and AUC scores on the cross-architecture comparison dataset.

The experimental results demonstrate that DEGNN exhibits strong robustness in cross-architecture and cross-optimization scenarios. In contrast, the SAFE model, lacking function structure information, shows a poor discriminative ability when dealing with binary functions of different architectures. Meanwhile, Asteria-Pro's performance is slightly inferior to DEGNN in the comprehensive scenario of cross-optimization and cross-architecture. This also validates the conclusion of BINKIT and BinFinder that call behavior information demonstrates strong robustness in cross-architecture scenarios. The design of the CECG in DEGNN effectively combines call information and function structure information.

Table 4. Cross-architectural binary function search.

	MRR							Recall@1						
Model	**00, O3**	**01, O3**	**02, O3**	**00, Os**	**01, Os**	**02, Os**	**Avg.**	**00, O3**	**01, O3**	**02, O3**	**00, Os**	**01, Os**	**02, Os**	**Avg.**
SAFE	0.101	0.217	0.519	0.127	0.314	0.326	0.267	0.078	0.229	0.528	0.088	0.244	0.288	0.243
Asteria-Pro	0.431	0.574	0.668	0.489	0.617	0.644	0.571	0.256	0.471	0.533	0.354	0.502	0.574	0.448
DEGNN-noCECG	0.277	0.429	0.613	0.302	0.487	0.522	0.438	0.190	0.409	0.449	0.240	0.362	0.433	0.347
DEGNN-noTNT	0.403	0.552	0.656	0.462	0.561	0.579	0.536	0.244	0.482	0.517	0.275	0.356	0.555	0.405
DEGNN	0.471	0.689	0.752	0.567	0.663	0.691	0.639	0.392	0.607	0.683	0.477	0.609	0.687	0.576

5.7. Ablation Experiment

To thoroughly investigate the specific impact of the CECG and NTN components within DEGNN on model performance, we designed and performed ablation studies. By creating two variant models, namely, DEGNN without CECG (DEGNN-noCECG) and DEGNN without NTN (DEGNN-noNTN), our objective was to isolate each component for the purpose of evaluating its contribution to the overall performance.

The results of the ablation experiments demonstrate that DEGNN, when equipped with both CECG and NTN components, outperforms its variants in cross-architectural and same-architectural BCSA tasks. Specifically, DEGNN achieved superior MRR and recall@1 metrics in cross-architectural tasks compared to DEGNN-noCECG and DEGNN-noNTN, attesting to the significant role of CECG in annotating call information and NTN in extracting relational features. This finding underscores the indispensable role of these two components in enhancing DEGNN's performance and validates the rationality and effectiveness of our methodology.

5.8. Vulnerability Search

Vulnerability detection is crucial in ensuring the security of software systems. In this section, we present the details of our real-world vulnerability search experiment, designed to evaluate the performance of different models in identifying vulnerabilities across architectures.

For each experiment run, we used the recall@1 metric and Dataset 5. Specifically, we iteratively traversed the set of vulnerable functions. In each iteration, a single vulnerable function was selected and combined with the dataset of nonvulnerable functions to form a new dataset containing 1000 functions. Subsequently, the models were tasked with searching for the vulnerable function traversed within this dataset. The recall@1 value was calculated based on whether the model successfully identified the vulnerable function. This process was repeated until all 20 vulnerable functions were traversed. Finally, the average recall@1 value was calculated to comprehensively assess the performance of the models.

As shown in Table 5, significant differences were observed between the models in the experiment. The DEGNN achieved a recall rate at the top-1 position of 0.800. The Asteria-Pro model had a corresponding recall rate of 0.60, while the SAFE exhibited a relatively lower performance with 0.25. It should be noted that DEGNN outperformed the runner-up, Asteria-Pro, by a remarkable margin of 33.3%. These results clearly demonstrate that in this specific vulnerability search task, DEGNN exhibits outstanding performance. This implies that DEGNN has a powerful ability to capture the characteristics of vulnerable functions and distinguish them from nonvulnerable ones.

Table 5. Recall@1 of real-world vulnerability search.

Model	SAFE	Asteria-Pro	DEGNN
Average Recall@1	0.25	0.60	0.80

5.9. Time Efficiency Analysis

In assessing the time efficiency of the similarity detection methods of functions, we conducted a detailed experiment on the Dataset 1 test set for DEGNN, SAFE, and Asteria-Pro. The experiment mainly focused on the time consumption during the data preparation and similarity calculation phases, with the aim of providing a comprehensive analysis of the efficiency performance of each method at different processing stages and to offer reliable evidence for subsequent analysis. In the experiment, we refer to the data preparation step of extracting features from functions as phase 1, and the step of encoding features and calculating similarity as phase 2.

As shown in Figure 6, during the data preparation phase, SAFE only needed to extract the assembly code, which is relatively simple and took 35.24 s. In contrast, DEGNN and Asteria-Pro required the extraction of CFG structures and call information, a more complex process that took 53.91 s, significantly more than SAFE. This is because the extraction of CFG structures and call information involves in-depth analysis and processing of function relationships, which requires more computational resources and time.

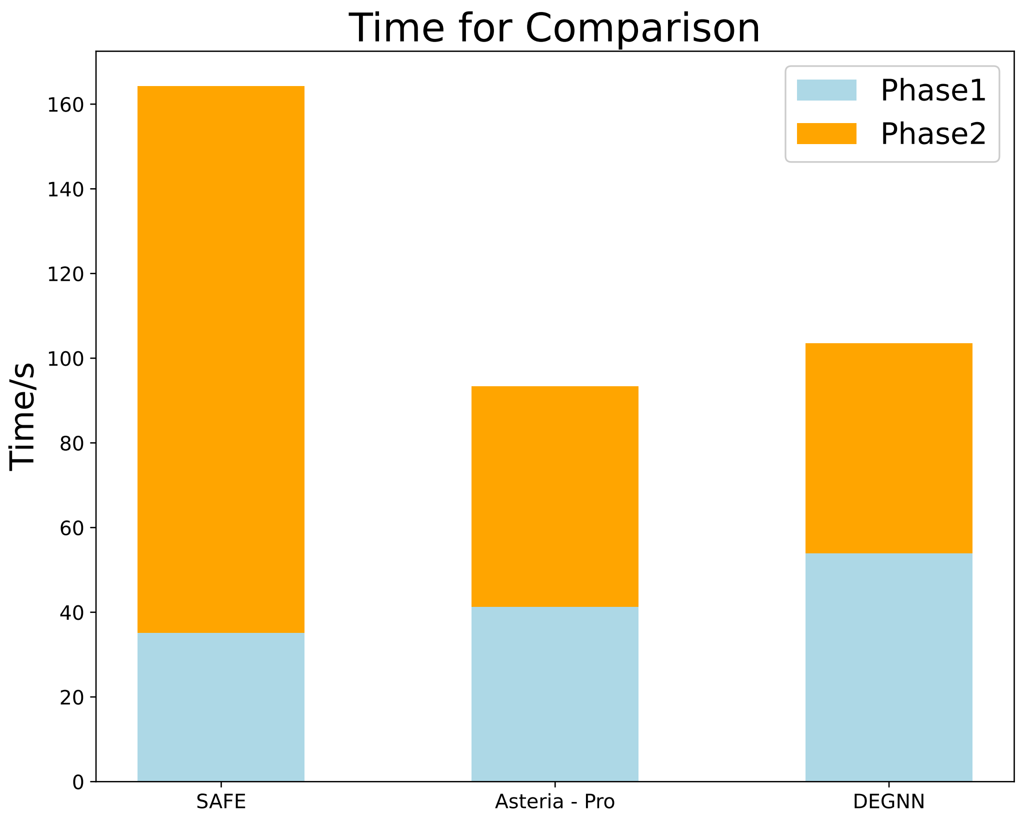

Figure 6. Time efficiency comparison.

In the similarity calculation phase, DEGNN only took 49.6 s. In comparison, SAFE took 129.04 s, and Asteria-Pro took 52.08 s. DEGNN's efficient reasoning is attributed to its concise and efficient feature representation and the parallel computing characteristics of the NTN network, which can quickly process function features and derive similarity results. The 16-dimensional vectors are more efficient in terms of storage and computation, which also means a reduction in the parameters and size of the neural network. The NTN can make full use of hardware resources to process data in parallel, greatly improving reasoning speed.

In terms of total time consumption, Asteria-Pro performed the best, likely due to its significant efficiency advantage in feature extraction based on AST during the data preparation phase. In contrast, DEGNN involves additional computational processes for constructing basic blocks based on features after extracting the CFG. DEGNN had a total time consumption of 103.51 s. Although the data preparation phase took longer, its outstanding performance in the reasoning phase made it somewhat competitive in overall time efficiency.

6. Discussion

In this research, we propose DEGNN, a novel method based on graph neural networks, aimed at facilitating cross-architecture binary code similarity analysis for UAV software systems. Our experimental results demonstrate that DEGNN outperformed the current state-of-the-art models by 12% in MRR and 28.6% in recall@1 for cross-architecture binary function search tasks, highlighting DEGNN's superiority in handling complex and critical binary code analysis and vulnerability detection tasks.

Furthermore, eschewing reliance on pre-trained models was crucial for this research. While mature pre-trained language models based on vast corpora exist in the realms of natural language and source code, no analogous models are present in the binary domain. Although some studies have proposed their own assembly-code pre-trained language models, the scale and generality of these models' corpora are far from matching those in the natural language domain.

At the same time, we observed that compared to the technical route of directly converting binary functions into embeddings and comparing them using simple methods such as cosine similarity, as in jTrans, DEGNN's technical route for the comparison phase results in higher computational costs, representing a limitation of this study.

Ultimately, this study confirmed the importance of both function structural information and function call information in distinguishing BCSA tasks across optimizations and instruction sets, which is vital for the success of BCSA tasks.

7. Conclusions

In our research, we have introduced DEGNN, a groundbreaking GNN model for analyzing the similarity of binary code in a UAV system. The prowess of DEGNN is attributed to its ability to adeptly capture both the structural and semantic nuances of binary code via CECG and a dual-embedding approach. This innovative method not only enhances the precision of code similarity analysis but also pioneers a new path for analyzing binary code across different architectures. In cross-architecture binary function search tasks, DEGNN achieved an average MRR score of 0.639 and an average recall@1 score of 0.576, outperforming all baseline models and demonstrating its strong adaptability in cross-architecture scenarios. In real-world vulnerability search tasks, DEGNN successfully identified multiple real vulnerabilities, achieving an average recall@1 score of 0.8, further validating its practicality and robustness in complex binary code analysis tasks. These concrete results highlight that DEGNN not only excels in cross-architecture scenarios but also effectively supports vulnerability discovery and analysis in real-world security applications. Although DEGNN has achieved remarkable performance in this study, we recognize the ongoing need for further advances. Upcoming research endeavors will focus on enhancing the model's robustness and extending DEGNN's applications to address BCSA challenges complicated by obfuscation and inlining techniques, as well as to broaden its practice within software specific to drone network components and communication protocols.

Author Contributions: Conceptualization, J.D.; methodology, J.D.; project administration, Q.W.; software, J.D. and X.B.; visualization, X.B.; writing—original draft preparation, J.D.; writing—review and editing, Y.W.; investigation, J.D. and X.B.; resources, Y.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Key R & D Program of China under Grant No. 2020YFB2010900, the Program for Innovation Leading Scientists and Technicians of Zhong Yuan under Grant No. 224200510002.

Data Availability Statement: The dataset used in this study was obtained from BINKIT [6]. The code for dataset processing, model training, and evaluation has been made publicly available on GitHub at https://github.com/kidding1412/DEGNN, accessed on 1 January 2025, enabling other researchers to access and reuse it for further studies.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Conti, F.C.; Santoro, C.; Santoro, F.F. Twinflie: A Digital Twin UAV Orchestrator and Simulator. In Proceedings of the 2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Abu Dhabi, United Arab Emirates, 14–17 November 2023; pp. 258–263. [CrossRef]
2. D'Urso, F.; Santoro, C.; Santoro, F.F. Integrating Heterogeneous Tools for Physical Simulation of multi-Unmanned Aerial Vehicles. In Proceedings of the 19th Workshop from Objects to Agents, Palermo, Italy, 28–29 June 2018; pp. 10–15. Available online: https://ceur-ws.org/Vol-2215/paper_2.pdf (accessed on 1 January 2025).
3. Qu, Y.; Dai, H.; Zhuang, Y.; Chen, J.; Dong, C.; Wu, F.; Guo, S. Decentralized Federated Learning for UAV Networks: Architecture, Challenges, and Opportunities. *IEEE Netw.* **2021**, *35*, 156–162. [CrossRef]
4. Wazid, M.; Bera, B.; Das, A.K.; Garg, S.; Niyato, D.; Hossain, M.S. Secure Communication Framework for Blockchain-Based Internet of Drones-Enabled Aerial Computing Deployment. *IEEE Internet Things Mag.* **2021**, *4*, 120–126. [CrossRef]
5. Miao, S.; Pan, Q. Risk Assessment of UAV Cyber Range Based on Bayesian–Nash Equilibrium. *Drones* **2024**, *8*, 556. [CrossRef]
6. Kim, D.; Kim, E.; Cha, S.K.; Son, S.; Kim, Y. Revisiting Binary Code Similarity Analysis Using Interpretable Feature Engineering and Lessons Learned. *IEEE Trans. Softw. Eng.* **2022**, *49*, 1661–1682. [CrossRef]
7. Qasem, A.; Debbabi, M.; Lebel, B.; Kassouf, M. Binary Function Clone Search in the Presence of Code Obfuscation and Optimization over Multi-CPU Architectures. In Proceedings of the ASIA CCS '23: 2023 ACM Asia Conference on Computer and Communications Security, Melbourne, VIC, Australia, 10–14 July 2023; pp. 443–456. [CrossRef]
8. Socher, R.; Chen, D.; Manning, C.D.; Ng, A.Y. Reasoning with Neural Tensor Networks for Knowledge Base Completion. In Proceedings of the Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013, Lake Tahoe, NV, USA, 5–8 December 2013; pp. 926–934.
9. Wang, H.; Qu, W.; Katz, G.; Zhu, W.; Gao, Z.; Qiu, H.; Zhuge, J.; Zhang, C. jTrans: Jump-Aware Transformer for Binary Code Similarity Detection. In Proceedings of the ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Republic of Korea, 18–22 July 2022; pp. 1–13. [CrossRef]
10. Luo, Z.; Wang, P.; Wang, B.; Tang, Y.; Xie, W.; Zhou, X.; Liu, D.; Lu, K. VulHawk: Cross-architecture Vulnerability Detection with Entropy-based Binary Code Search. In Proceedings of the 30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, CA, USA, 27 February–3 March 2023.
11. Synopsys, I. Heartbleed Bug. 2020. Available online: https://heartbleed.com/ (accessed on 1 January 2025).
12. Zuo, F.; Li, X.; Young, P.; Luo, L.; Zeng, Q.; Zhang, Z. Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs. In Proceedings of the 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, CA, USA, 24–27 February 2019.
13. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
14. Massarelli, L.; Luna, G.A.D.; Petroni, F.; Baldoni, R.; Querzoni, L. SAFE: Self-Attentive Function Embeddings for Binary Similarity. In *Detection of Intrusions and Malware, and Vulnerability Assessment, Proceedings of the 16th International Conference, DIMVA 2019, Gothenburg, Sweden, 19–20 June 2019, Proceedings*; Perdisci, R., Maurice, C., Giacinto, G., Almgren, M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11543, pp. 309–329. [CrossRef]
15. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. In Proceedings of the 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, AZ, USA, 2–4 May 2013.
16. Lin, Z.; Feng, M.; dos Santos, C.N.; Yu, M.; Xiang, B.; Zhou, B.; Bengio, Y. A Structured Self-Attentive Sentence Embedding. In Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, 24–26 April 2017.
17. Tian, D.; Jia, X.; Ma, R.; Liu, S.; Liu, W.; Hu, C. BinDeep: A Deep Learning Approach to Binary Code Similarity Detection. *Expert Syst. Appl.* **2021**, *168*, 114348. [CrossRef]
18. Lipton, Z.C.; Berkowitz, J.; Elkan, C. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv* **2015**, arXiv:1506.00019.
19. Collyer, J.; Watson, T.; Phillips, I. FASER: Binary Code Similarity Search through the Use of Intermediate Representations. In Proceedings of the Conference on Applied Machine Learning in Information Security, Arlington, VA, USA, 19–20 October 2023; Volume 3652, pp. 193–202.
20. Radareorg. Radare2: UNIX—Like Reverse Engineering Framework and Command—Line Toolset (Version 5.9.9). GitHub. Available online: https://github.com/radareorg/radare2/releases (accessed on 1 January 2025).
21. Beltagy, I.; Peters, M.E.; Cohan, A. Longformer: The Long-Document Transformer. *arXiv* **2020**, arXiv:2004.05150.
22. Xu, X.; Liu, C.; Feng, Q.; Yin, H.; Song, L.; Song, D. 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, Dallas, TX, USA, 30 October–3 November 2017; pp. 363–376. [CrossRef]

23. Gao, J.; Yang, X.; Fu, Y.; Jiang, Y.; Sun, J. VulSeeker: A Semantic Learning Based Vulnerability Seeker for Cross-Platform Binary. In Proceedings of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, France, 3–7 September 2018; pp. 896–899. [CrossRef]
24. Dai, H.; Dai, B.; Song, L. Discriminative Embeddings of Latent Variable Models for Structured Data. In Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York, NY, USA, 19–24 June 2016; Volume 48, pp. 2702–2711.
25. Luo, M.; Yang, C.; Gong, X.; Yu, L. FuncNet: A Euclidean Embedding Approach for Lightweight Cross-platform Binary Recognition. In *Security and Privacy in Communication Networks, Proceedings of the 15th EAI International Conference, SecureComm 2019, Orlando, FL, USA, 23–25 October 2019* ; Chen, S., Choo, K.K.R., Fu, X., Lou, W., Mohaisen, A., Eds.; Springer: Cham, Switzerland, 2019; pp. 319–337.
26. Li, Y.; Gu, C.; Dullien, T.; Vinyals, O.; Kohli, P. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 3835–3845.
27. Yang, S.; Dong, C.; Xiao, Y.; Cheng, Y.; Shi, Z.; Li, Z.; Sun, L. Asteria-Pro: Enhancing Deep Learning-based Binary Code Similarity Detection by Incorporating Domain Knowledge. *ACM Trans. Softw. Eng. Methodol.* **2024**, *33*, 1:1–1:40. [CrossRef]
28. Tai, K.S.; Socher, R.; Manning, C.D. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, Beijing, China, 26–31 July 2015; Volume 1, pp. 1556–1566. [CrossRef]
29. Ding, S.H.H.; Fung, B.C.M.; Charland, P. Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 472–489. [CrossRef]
30. Le, Q.V.; Mikolov, T. Distributed Representations of Sentences and Documents. In Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21–26 June 2014; Volume 32, pp. 1188–1196.
31. Massarelli, L.; Luna, G.; Petroni, F.; Querzoni, L. Investigating Graph Embedding Neural Networks with Unsupervised Features Extraction for Binary Analysis. In Proceedings of the Workshop on Binary Analysis Research (BAR) 2019, San Diego, CA, USA, 24 February 2019. [CrossRef]
32. Duan, Y.; Li, X.; Wang, J.; Yin, H. DeepBinDiff: Learning Program-Wide Code Representations for Binary Diffing. In Proceedings of the Network and Distributed Systems Security (NDSS) Symposium 2020, San Diego, CA, USA, 23–26 February 2020. [CrossRef]
33. Yu, Z.; Cao, R.; Tang, Q.; Nie, S.; Huang, J.; Wu, S. Order Matters: Semantic-Aware Neural Networks for Binary Code Similarity Detection. In Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, 7–12 February 2020; pp. 1145–1152.
34. Yang, J.; Fu, C.; Liu, X.Y.; Yin, H.; Zhou, P. Codee: A Tensor Embedding Scheme for Binary Code Search. *IEEE Trans. Softw. Eng.* **2022**, *48*, 2224–2244. [CrossRef]
35. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. 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, 2–7 June 2019; Volume 1, pp. 4171–4186. [CrossRef]
36. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural Message Passing for Quantum Chemistry. In Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6–11 August 2017; Volume 70, pp. 1263–1272.
37. Wang, H.; Gao, Z.; Zhang, C.; Sun, M.; Zhou, Y.; Qiu, H.; Xiao, X. CEBin: A Cost-Effective Framework for Large-Scale Binary Code Similarity Detection. In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2024), Vienna, Austria, 16–20 September 2024. [CrossRef]
38. Wang, H.; Gao, Z.; Zhang, C.; Sha, Z.; Sun, M.; Zhou, Y.; Zhu, W.; Sun, W.; Qiu, H.; Xiao, X. CLAP: Learning Transferable Binary Code Representations with Natural Language Supervision. In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2024), Vienna, Austria, 16–20 September 2024. [CrossRef]
39. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv* **2016**, arXiv:1609.02907.
40. Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph Attention Networks. In Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, 30 April–3 May 2018.