

Revisiting Graph Representations for ML-Based Binary Code Similarity Detection: A Systematic Study

TENGTENG YANG*, Sun Yat-sen University, China

YIKUN HU*, Shanghai Jiao Tong University, China

JICAN ZHANG, Sun Yat-sen University, China

LEI XUE†, Sun Yat-sen University, China and Shenzhen Loop Area Institute, China

MING FAN, Xi'an Jiaotong University, China

LIANG ZHANG, Sun Yat-sen University, China

Binary Code Similarity Detection (BCSD) is a foundational capability in software security, underpinning critical applications ranging from vulnerability detection to malware analysis. While recent tools based on Machine Learning (ML) have achieved significant performance improvements, their efficacy is heavily contingent upon the underlying code representation. Through a systematic literature review of ML-based BCSD papers, we find that existing approaches typically leverage linear sequences or adopt graph-based representations, with the latter constituting the majority (77%). Despite this prevalence, there is no consensus on which graph representation yields superior effectiveness. Existing works usually couple graph construction with customized learning backbones and evaluate them on inconsistent benchmarks. This makes isolating the representation's impact difficult. Consequently, *determining which graph topologies most effectively capture robust binary-code semantics under controlled and comparable evaluation settings remains an open problem.*

In this paper, we present a systematic study of graph representations for ML-based BCSD to bridge this gap. Specifically, we implement a modular evaluation framework that decouples graph construction from model training. Using this framework, we systematically evaluate seven representative graph representations, finding that no single representation is universally dominant, that distinct topologies exhibit unique strengths depending on the evaluation scenario, and that their rankings are largely backbone-stable despite varying absolute performance. We further investigate their combination effectiveness in N-day vulnerability detection and employ a tailored post-hoc analysis tool to study model-level structural reliance. The results show that DFG, PDG, and SOG subgraph pairs more often preserve trained models' similarity scores under pruning, while several other representations are more sensitive to structural reduction.

CCS Concepts: • **Security and privacy** → **Software reverse engineering**; • **Computing methodologies** → **Machine learning**.

Additional Key Words and Phrases: Binary Analysis, Binary Code Similarity Detection, Binary Function Representation, Machine Learning, Explainability

*Both authors contributed equally to this research.

†Corresponding author.

Authors' Contact Information: [Tengteng Yang](mailto:yangtt57@mail2.sysu.edu.cn), Sun Yat-sen University, Shenzhen, China, yangtt57@mail2.sysu.edu.cn; [Yikun Hu](mailto:yikunhu@sjtu.edu.cn), Shanghai Jiao Tong University, Shanghai, China, yikunhu@sjtu.edu.cn; [Jican Zhang](mailto:jicanzhang@mail2.sysu.edu.cn), Sun Yat-sen University, Shenzhen, China, jicanzhang@mail2.sysu.edu.cn; [Lei Xue](mailto:xuele3@mail.sysu.edu.cn), Sun Yat-sen University, Shenzhen, China and Shenzhen Loop Area Institute, Shenzhen, China, xuele3@mail.sysu.edu.cn; [Ming Fan](mailto:mingfan@mail.xjtu.edu.cn), Xi'an Jiaotong University, Xi'an, China, mingfan@mail.xjtu.edu.cn; [Liang Zhang](mailto:zhangliang27@mail.sysu.edu.cn), Sun Yat-sen University, Shenzhen, China, zhangliang27@mail.sysu.edu.cn.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2994-970X/2026/10-ARTISSTA167

<https://doi.org/10.1145/3832258>

ACM Reference Format:

Tengteng Yang, Yikun Hu, Jican Zhang, Lei Xue, Ming Fan, and Liang Zhang. 2026. Revisiting Graph Representations for ML-Based Binary Code Similarity Detection: A Systematic Study. *Proc. ACM Softw. Eng.* 3, ISSTA, Article ISSTA167 (October 2026), 25 pages. <https://doi.org/10.1145/3832258>

1 Introduction

Binary Code Similarity Detection (BCSD) stands as a cornerstone technology in software security, underpinning critical applications ranging from vulnerability detection [1, 10, 11, 13, 20, 23, 55, 77, 79, 81–83] and malware identification [32, 60, 70] to software composition analysis [15, 35, 41] and software plagiarism detection [52, 78, 89]. Recently, ML-based BCSD tools [13, 29, 43, 44, 55, 66, 77] have demonstrated superior performance by leveraging various models to analyze different representations of binary functions, outperforming traditional BCSD methods [10, 11, 16, 18, 37]. While the application of advanced ML models has undoubtedly improved detection performance, as pointed out by Domingos [14], *data and features remain the dominant factors driving model efficacy.*

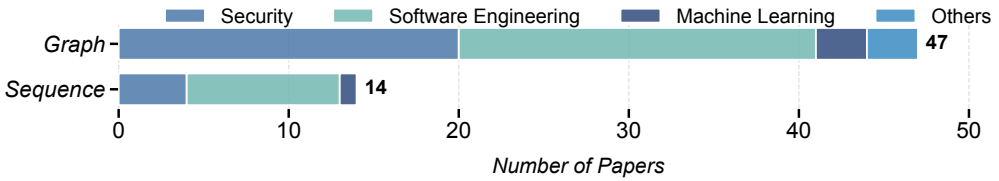


Fig. 1. Distribution of binary function representations (Graph-based vs. Sequence-based) across 61 surveyed ML-based BCSD papers, broken down by research domain.

Motivated by this insight, we conducted a systematic literature review (SLR) of 61 ML-based BCSD papers from leading Security, Software Engineering, and Machine Learning venues. We classify the surveyed works into two groups: *Sequence-based* methods [3, 17, 36, 49, 55, 62, 66, 67, 69], which treat binary functions as linear sequences of bytes, instructions, or pseudo-code and often adapt Natural Language Processing (NLP) techniques [12, 40, 58, 65] for representation learning; and *Graph-based* methods [23, 29, 33, 34, 42, 44, 46, 53, 56, 68, 77, 86], which harness structural dependencies to characterize code semantics.

As illustrated in Figure 1, graph-based representations dominate, accounting for 77% (47/61) of the surveyed works versus 23% for sequence-based ones.¹ This preference is also supported by prior studies [29, 42, 54, 64] reporting superior performance of graph-based approaches. Theoretically, while sequence-based methods flatten execution logic and often lose critical dependencies, graphs naturally align with the non-linear way programs execute by explicitly modeling data flow and control transfer, offering a richer semantic characterization. Given this advantage and their prevalence, we focus on graph-based representations to systematically evaluate the primary representation within ML-based BCSD.

Despite their widespread adoption, achieving an in-depth understanding of graph representations in ML-based BCSD remains challenging due to two primary obstacles:

- **Tight Coupling of Representation and Model.** Existing works [23, 29, 42, 53, 73, 77, 86, 87] usually evaluate graph representations within co-designed pipelines, including node features, learning backbones, and training strategies. For example, VulSeeker [23] and HermesSim [29] differ not only in graph topology, but also in feature encoding and model design. Thus, their end-to-end performance cannot isolate whether observed differences stem from graph topology itself

¹A detailed classification is provided in the supplementary material [84].

or from other pipeline components. This motivates a controlled evaluation of graph topologies under comparable settings.

- **Ambiguity of Model-level Structural Reliance.** The heavy reliance on graph neural networks (GNNs) for graph encoding introduces an interpretability gap [22, 23, 27, 29, 39, 42, 53, 56, 77, 86, 87]. For example, when a Siamese GNN ranks two compiled versions of the same function as highly similar, the score alone does not reveal which graph structures suffice to preserve the model’s learned similarity score. Different representations may induce different preservation behaviors—some subgraph pairs retain high similarity after substantial pruning while others exhibit larger score drops under structural reduction. This gap obscures how trained BCSD models rely on graph structure and leaves future representation design without clear guidance.

To bridge these gaps, we conduct a systematic empirical study of graph representations for ML-based BCSD under a controlled representation-level setting. To address the entanglement issue, we implement a modular framework that standardizes seven representative graph topologies under a shared similarity-learning and retrieval protocol. By decoupling graph construction from model training, this framework reduces pipeline-level confounders and enables comparing how different topologies behave under comparable settings, rather than ranking complete BCSD systems. To study structural reliance in similarity decisions, we design SimGE, a pair-aware post-hoc analysis tool for Siamese GNNs that identifies retained edge subsets preserving the trained model’s original similarity score, providing an operational view of model-level structural reliance across representations. We further evaluate these representations on a large-scale N-day vulnerability benchmark derived from *CVEBinarySheet*², covering 564 CVEs and 3,927 vulnerable instances.

In summary, our contributions are as follows:

- **Study Approach.** We identify seven representative graph topologies through a systematic literature review and propose a standardized framework to unify them. This enables a systematic and disentangled evaluation in the field, revealing how different topologies contribute to code semantics characterization.
- **Interpretability Tool.** We propose SimGE, a pair-aware post-hoc analysis tool for similarity analysis. Using it, we observe that subgraph pairs from DFG, PDG, and SOG more often preserve trained models’ similarity scores under pruning, whereas several other representations exhibit larger score drops after graph reduction.
- **Large-scale Evaluation.** We conduct comprehensive experiments across three datasets, including a real-world vulnerability benchmark. Our results demonstrate that no single representation is universally dominant, as distinct topologies exhibit unique strengths depending on the evaluation scenario.
- **Future Directions.** We identify the distinct characteristics of current graph representations and outline promising future research directions based on these insights. To facilitate future research, we open-source our framework and datasets.³

2 Background and Related Work

2.1 BCSD Problem Definition

Based on application scenarios, BCSD is typically classified into three categories [28]: One-to-One (OO), which computes the similarity score between a function pair; One-to-Many (OM), which retrieves the top- k most similar functions for a query from a function pool; and Many-to-Many (MM), which clusters a function pool by similarity. This paper focuses on One-to-Many BCSD,

²<https://github.com/xmoezz/CveBinarySheet>

³All artifacts and supplementary material are available at https://github.com/yangtt57/empirical_graphs_used_in_ML-BCSD, referred to as [84].

which is central to security applications such as N-day vulnerability detection. We adopt the following definition from prior work [42, 69]:

DEFINITION 1 (ONE-TO-MANY BCSD TASK). *Given a query function f_q and a function pool P , the task retrieves the top- k functions $\{f_1, \dots, f_k \mid f_i \in P\}$ ranked by their similarity scores with respect to f_q , so that functions truly similar to f_q are ranked as high as possible.*

2.2 Graph Representations in ML-Based BCSD

SLR Protocol. To identify graph representations used in ML-based BCSD, we conducted a structured literature review over leading security, software engineering, and machine learning venues, such as S&P, USENIX Security, ISSTA, ICSE, FSE, and ICML. We searched papers published between 2015 and 2025 using BCSD-related keywords, removed duplicates, screened titles, abstracts, and full texts, and further examined the references and citations of the included studies. We retained papers that propose or evaluate ML-based binary similarity methods with identifiable input representations, while excluding non-ML studies, source-code-only studies, and studies whose representations are unclear. This process yielded 61 ML-based BCSD papers, from which we extracted the representation type, graph topology, node granularity, edge semantics, learning backbone, task, and artifact availability. Full details are provided in the supplementary material [84]. Based on this extracted information, we categorize existing graph-based ML-BCSD representations into three groups according to their semantic focus.

Control-Driven Representations. The Control Flow Graph (CFG) models functions as basic blocks connected by jump edges to outline execution paths. To overcome the semantic sparsity of pure structural graphs, subsequent works [19, 42, 54, 56, 77, 87] enrich CFG nodes with handcrafted features or instruction embeddings to capture fine-grained control logic.

Data-Dependency Representations. Data-dependency representations focus on value propagation rather than control transfer. The Data Flow Graph (DFG) [3, 27, 43] models definition-use relations among variables and operations, making it less sensitive to syntactic variations such as instruction reordering. Data Dependency Slicing Graph (DDSG) [30] further uses slicing to build fine-grained dependence views, complementing CFG by exposing data-driven semantics.

Hybrid and Augmented Representations. Certain approaches fuse multiple semantic dimensions or incorporate external knowledge. For multi-view fusion, the Program Dependence Graph (PDG) [24, 90, 91] unifies control and data dependencies into a single structure, the Labeled Semantic Flow Graph (LSFG) [21–23] performs similar fusion at the basic-block level, and the Semantics-Oriented Graph (SOG) [29] aggregates multiple semantic subgraphs to reduce information loss. For semantic augmentation, the Binary Disassembly Graph (BDG) [39] uses compilation-invariant attributes (e.g., string constants), while the Environment Semantic Graph (ESG) [73] embeds environmental information for broader context.

Despite this diversity, it remains unclear *how distinct graph topologies differ in their resilience to compilation variations, including optimization levels, compilers, and architecture shifts*. This gap motivates a systematic evaluation that provides evidence-based guidance for graph selection.

2.3 Graph Representations for Source-Code Analysis

Beyond BCSD, graph representations have also been widely studied for source-code analysis. *Composite graphs* combine multiple code relations into a unified structure for richer semantics: the Code Property Graph (CPG) [80] integrates ASTs, CFGs, and PDGs for vulnerability discovery, and Devign [93] fuses multiple code relations to identify vulnerable functions with GNNs. *Slicing-based* approaches such as DeepWukong [7] extract semantically relevant slices that preserve control- and data-flow information for vulnerability detection. *Neural representation learning* further uses

source-code structures for program analysis: Allamanis et al. [2] represent programs with syntactic and semantic edges trained via GGNNs, GraphCodeBERT [26] incorporates data-flow edges into code pre-training, and MVD [5] explores flow-sensitive graph learning for vulnerability detection.

Our study is complementary in the binary setting. After compilation, source-level structures such as ASTs and source-level token sequences are no longer directly available, and binary graph construction must rely on disassembly, lifting, and IR recovery, which can introduce tool-specific artifacts. Thus, our study examines a different setting: binary function similarity retrieval, where representations must remain effective under compiler, optimization, and architecture shifts.

2.4 Explainability Challenges in Siamese GNNs

The inherent opacity of GNN models makes it difficult to determine which graph structures preserve a trained BCSD model's learned similarity score, and how this behavior varies across representations. However, mainstream graph interpretability techniques are not designed for the pairwise nature of BCSD. Prominent explainers such as GNNExplainer [85], PGExplainer [50, 51], and SubgraphX [88] target single-instance classification, where they isolate salient features within one graph for a specific label. In contrast, ML-based BCSD usually relies on Siamese networks [4] to optimize a metric space where similarity is defined by the distance between two graphs. An effective explainer for BCSD should therefore reason over graph pairs rather than isolated graphs, and assess which structural components help preserve the model's similarity score. Existing explainers do not directly support this pairwise similarity setting, motivating a tailored tool for similarity analysis.

2.5 Divergence from Existing Works

Empirical Studies in BCSD. Prior benchmarking efforts [54, 64] performed large-scale evaluations of ML-based BCSD tools and established a foundation for fair tool-level comparison. However, they assessed *holistic tool efficacy*, treating the representation and learning model as an entangled pipeline. A recent study [71] began to isolate representation factors by investigating features within CFGs, highlighting the importance of CFG-level features. Our work broadens this trajectory beyond CFGs to cover graph representations used in ML-based BCSD.

Comparative Analysis of Graph Representations. Prior method-specific work [29] compares graph types but typically contrasts a standard baseline with a single proposed structure, aiming to validate a new method rather than provide a neutral view of the field. To address this limitation, we establish an extensible framework that standardizes diverse graph representations and evaluates them under shared protocols across multiple GNN backbones. We further analyze model-level structural reliance through interpretability analysis.

3 Study Approach

This section presents the overall design of our empirical study. As shown in Figure 2, the study starts from binary functions, lifts them into multiple graph representations, evaluates the representations through GNN-based similarity learning, and uses SimGE to analyze model-level structural reliance. The right side of the figure maps these components to our three research questions: **RQ1** studies robustness and generalization, **RQ2** studies vulnerability detection and combination strategies, and **RQ3** studies similarity preservation and structural reliance. We define the RQs in §3.1, describe selected representations in §3.2, detail the evaluation framework in §3.3, and provide implementation details in §3.4.

3.1 Research Questions

We investigate three main RQs:

- **RQ1:** How do graph representations compare in terms of compilation robustness and out-of-distribution generalization, and how sensitive are the representation-level trends to GNN backbone choice?
- **RQ2:** How effective are graph representations in large-scale N-day vulnerability detection, and can combination strategies further enhance performance?
- **RQ3:** Which subgraph pairs can preserve a trained BCSD model's learned similarity scores, and how does this preservation behavior differ across graph representations?

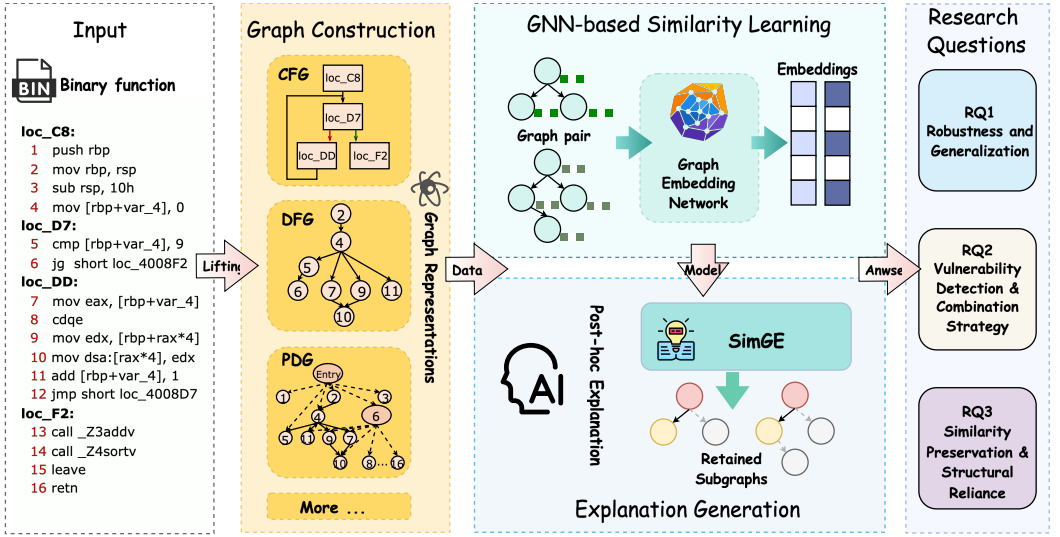


Fig. 2. Overview of our empirical-study approach. The study lifts binary functions into multiple graph representations, evaluates them via GNN-based similarity learning, and uses SimGE to analyze score-preserving subgraph pairs. The right side maps these components to the three research questions.

3.2 Selected Graph Representations

Guided by the SLR in §2.2, we identified 10 graph representations adopted in ML-based BCSD over the past decade, then refined the pool with strict exclusion criteria to focus on intra-function binary semantics. We excluded high-level representations such as ASTs [72, 82, 83] and the Code Property Graph (CPG) [92], as they rely on decompiler-generated pseudocode that shifts the abstraction away from raw binary semantics. We also omitted the EESG [73], which uses functions as nodes to model *inter-procedural* relationships, beyond our intra-function scope. This yielded seven representative graphs (Table 1), categorized by semantic focus into *Control-centric*, *Data-centric*, and *Hybrid*. These representations vary in node granularity (basic blocks to tokens) and edge semantics, enabling us to evaluate how different semantic abstractions influence model performance.

3.3 Evaluation Framework

As shown in the middle of Figure 2, our framework consists of three stages: *Graph Construction*, *Similarity Learning*, and *Explanation Generation*. By decoupling graph construction from similarity

Table 1. Overview of selected graph representations.

Family	Rep.	Representative Studies	#Usage	Node	Edge Semantics		Key Semantics	Code
					CF	DF		
<i>Control</i>	CFG	[19, 42, 54, 56, 77]	31	BB	✓	✗	Execution Flow Sequence	✓
<i>Data</i>	DFG	[3, 27, 43]	4	Inst	✗	✓	Value Dependency	✗
	DDSG	[30]	1	Slice	✗	✓	Computation Slices	✗
<i>Hybrid</i>	PDG	[24, 90, 91]	3	Inst	✓	✓	Control & Data Deps.	✓
	LSFG	[21–23]	3	BB	✓	✓	Inter-block Memory Deps.	✓
	SOG	[29]	1	Tok	✓	✓	Semantic Effect Edges	✓
	BDG	[39]	1	BB	✓	✗	Syscalls & String Constants	✗

Rep.: Representation; **#Usage:** Papers in our SLR using this graph.

Node Types: BB (Basic Block), Inst (Instruction), Slice (Program Slice), Tok (Token).

Edge Types: CF (Control Flow), DF (Data Flow). **Code:** Open Source Availability.

learning, it enables controlled representation-level comparison under a shared protocol rather than end-to-end tool ranking.

3.3.1 Graph Construction. In this stage, each binary function is transformed into the target graph representations that serve as the inputs to our similarity-learning framework. We use reverse engineering tools [31, 61] to recover function-level information, including instruction sequences, basic-block boundaries, control-flow relations, and value-dependence information. These extracted program facts are then normalized and synthesized into the seven graph representations described in §3.2, covering control-flow, data-dependence, and hybrid semantics.

3.3.2 GNN-Based Similarity Learning. We employ a Siamese GNN following the training protocol used in HermesSim [29], with a margin-based loss and distance-weighted negative sampling. Each mini-batch samples N function symbols, with each symbol compiled under two settings to form a positive pair (G_i^1, G_i^2) . To mine informative negatives, we adopt Distance-Weighted Negative Sampling [74]: the probability of selecting a negative G_k for anchor G_i^1 is inversely proportional to their embedding distance:

$$P(G_k | G_i^1) \propto \left(d(\mathbf{h}_{G_i^1}, \mathbf{h}_{G_k}) \right)^{-(D-2)}. \quad (1)$$

The margin-based loss [44] separates equivalent and non-equivalent pairs:

$$\mathcal{L} = \sum_{i=1}^N \left[\max(0, (1-m) - \text{sim}(G_i^1, G_i^2)) + \max(0, \text{sim}(G_i^1, G_{\text{neg}}) - m) \right], \quad (2)$$

where m is the margin, G_{neg} is the sampled negative, $d(\cdot)$ denotes embedding distance, and $\text{sim}(\cdot)$ denotes cosine similarity between graph embeddings.

While the framework is agnostic to the GNN architecture, we instantiate it with multiple backbones, as detailed in §3.4.4.

3.3.3 Explanation Generation. We next identify the structures in an input graph pair that are sufficient to preserve a BCSD model’s similarity score. Given a trained Siamese BCSD model and an input graph pair (G_a, G_p) , we formulate the goal as identifying a subgraph pair $(S_a \subset G_a, S_p \subset G_p)$ that preserves the model’s original similarity score. Mathematically, this can be viewed as

maximizing the mutual information (MI) between the retained structures and the model output Y , where Y denotes the similarity score:

$$\max_{S_a, S_p} MI(Y, (S_a, S_p)) = H(Y) - H(Y | S_a, S_p). \quad (3)$$

Since the model is fixed, maximizing MI corresponds to reducing output uncertainty given the retained subgraph pair. This formulation motivates preserving the original similarity score, while compactness is controlled by the edge-retention ratios and the optional sparsity term below.

Directly optimizing discrete subgraphs is intractable, so we learn continuous masks M_a and M_p as edge-importance scores. During optimization, discrete top- k masks are constructed at multiple retention ratios, and a straight-through estimator (STE) is used to propagate gradients through the discrete selection. Algorithm 1 details the multi-term loss:

$$\mathcal{L} = \mathcal{L}_{\text{sim}} + \alpha \mathcal{L}_{\text{emb}} + \beta \mathcal{L}_{\text{sparse}} + \gamma \mathcal{L}_{\text{ent}}, \quad (4)$$

where $\mathcal{L}_{\text{sim}}$ reproduces the full-graph similarity score, averaged over the retention ratios, as the primary pair-level objective; $\mathcal{L}_{\text{emb}}$ keeps masked embeddings close to the originals for stable optimization; $\mathcal{L}_{\text{sparse}}$ encourages compactness; and $\mathcal{L}_{\text{ent}}$ encourages near-binary mask values.

Algorithm 1 Similarity-Preserving Subgraph Pair Extraction (SimGE)

Input: Trained Siamese GNN $f(\cdot)$, Graph pair (G_a, G_p) , Retention ratios $\mathcal{R}$, Hyperparameters α, β, γ , Learning rate η , Iterations T

Output: Selected subgraph pair S_a, S_p

- 1: **Note:** $H(p) \triangleq -p \log p - (1-p) \log(1-p)$ (element-wise Bernoulli entropy).
 - 2: **Initialization:**
 - 3: Compute original embeddings: $\mathbf{h}_a = f(G_a)$, $\mathbf{h}_p = f(G_p)$
 - 4: Compute target similarity: $s_{\text{orig}} = \text{sim}(\mathbf{h}_a, \mathbf{h}_p)$
 - 5: Initialize soft mask logits M_a, M_p
 - 6: **Optimization Loop:**
 - 7: **for** $t = 1$ to T **do**
 - 8: Compute $Q_i = \sigma(M_i)$ and, for each $r \in \mathcal{R}$, $B_i^{(r)} = \text{TopK}_r(Q_i)$ using STE
 - 9: Compute masked embeddings: $\mathbf{h}_i^{(r)} = f(G_i \odot B_i^{(r)})$, $i \in \{a, p\}$
 - 10: Calculate Loss Components:
 - 11: $L_{\text{sim}} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \|\mathbf{s}_{\text{orig}} - \text{sim}(\mathbf{h}_a^{(r)}, \mathbf{h}_p^{(r)})\|_2^2$
 - 12: $L_{\text{emb}} = \|\mathbf{h}_a - f(G_a \odot Q_a)\|_2^2 + \|\mathbf{h}_p - f(G_p \odot Q_p)\|_2^2$
 - 13: $L_{\text{reg}} = \beta(\|Q_a\|_1 + \|Q_p\|_1) + \gamma(H(Q_a) + H(Q_p))$
 - 14: Total Loss: $\mathcal{L} = L_{\text{sim}} + \alpha L_{\text{emb}} + L_{\text{reg}}$
 - 15: Update Masks via Gradient Descent:
 - 16: $M_a \leftarrow M_a - \eta \nabla_{M_a} \mathcal{L}$
 - 17: $M_p \leftarrow M_p - \eta \nabla_{M_p} \mathcal{L}$
 - 18: **end for**
 - 19: **Finalize:** Retain the top-ranked edges in $\sigma(M_a), \sigma(M_p)$ at the requested ratio to obtain S_a, S_p
 - 20: **return** S_a, S_p
-

3.4 Implementation Details

3.4.1 Baselines and Graph Construction. We use CFG as the primary reference baseline for two reasons. First, CFG is the most widely used graph representation in our SLR, appearing in 31 surveyed ML-based BCSD papers. Second, CFG is the foundational control-flow abstraction behind many richer graph designs [22, 23, 39]. Thus, it serves as a common reference for assessing added graph semantics, not as a presumed strongest baseline.

Since non-CFG representations are often built from different intermediate representations (IRs), comparisons may be affected by implementation choices. We construct three CFG variants to control

IR-level effects: “P-” (Ghidra P-code), “M-” (IDA Microcode), and “A-” (Assembly). To examine node-encoding effects, we fix the topology to A-CFG and instantiate it with five representative encoders: (1) Manual [77]: hand-crafted numeric features; (2) BoW-opc [54]: opcode-frequency Bag-of-Words; (3) W2V+Attn [56]: Word2Vec [59] embeddings with attention aggregation; (4) W2V+RNN [56]: sequential Word2Vec embedding processing via RNN [47]; and (5) HBMP [87]: hierarchical structural embedding with Bi-LSTM [25].

Regarding graph generation, we follow reproducibility-oriented construction procedures. When original artifacts are unavailable or not directly reusable, we re-implement the construction logic from the original papers, such as DDSG due to missing public artifacts and LSFG due to an outdated toolchain. For representations with open-source tools, such as SOG and PDG, we adapt their interfaces to our framework. Detailed specifications are provided in [84].

3.4.2 Featurization and Normalization. All graph representations undergo a unified preprocessing pipeline. We use two node encoding schemes shown effective in prior studies [29, 54]. For block-level graphs, node features are generated with the standard BoW-opc model. For instruction- or token-level graphs, we use ToK-opc [29], which builds fixed-size vocabularies for distinct semantic categories and maps unseen tokens to placeholders to maintain input consistency. Graphs with heterogeneous edges further use learnable edge-type embeddings to capture relational semantics.

3.4.3 Experimental Configuration. We implemented the framework in Python 3.10 with PyG 2.6.1. All similarity models are optimized with Adam ($lr = 10^{-3}$, batch size $N = 80$) using the margin-based loss ($m = 0.1$). Each configuration is trained for at most 50 epochs in 10 independent runs, retaining in each run the checkpoint with the highest MRR on the Dataset-1 validation split under XA+XC+XO. The selected checkpoint is used for all Dataset-1 test tasks as well as Dataset-2 and Dataset-3 without further tuning, with retrieval metrics averaged across runs.

For the SimGE stage, we use Adam with $\alpha=0.1$, $\beta=0.005$, $\gamma=0.1$, $\eta=0.01$, and $T=100$ iterations, selected through preliminary experiments. Experiments ran on a server with an Intel Xeon Platinum 8358 CPU, 1.0 TiB RAM, and four NVIDIA A100 80 GB PCIe GPUs.

3.4.4 Backbone Configuration. We use Structure2Vec [9] and GGNN [45] as the primary backbones, since they are widely adopted in ML-based BCSD [21–23, 27, 44, 54, 56, 77, 87]. To assess backbone sensitivity, we include GIN [76] and SGFormer [75] on Dataset-1, covering a stronger message-passing backbone and a recent simplified graph Transformer backbone with global attentive propagation beyond local message passing. To isolate backbone effects, all backbones share the same graph inputs, node features, Siamese objective, sampling strategy, optimizer, and checkpoint-selection protocol; only architecture-specific hyperparameters are adjusted.

3.4.5 Evaluation Metrics. Following prior BCSD retrieval studies [29, 42, 54, 64, 69, 73], we use Recall@1 and Mean Reciprocal Rank (MRR) for one-to-many retrieval. Recall@1 is the percentage of queries whose ground-truth function is ranked first. MRR averages the reciprocal rank:

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{r_q},$$

where Q is the query set and r_q the ground-truth rank for query q . In RQ1, each query has one ground-truth counterpart, so Recall@1 is top-1 accuracy and MRR reflects how early the correct function appears. In RQ2, since a vulnerability query may have multiple relevant instances, Recall@ K measures the fraction of vulnerable instances retrieved within the top- K . In RQ3, we use a pruning-based similarity-retention metric (defined with the pruning protocols in §4.4).

Table 2. Statistical summary of the experimental datasets.

Dataset	#Train	#Val.	#Test	#Vuln.	# Projs.	Archs.	Comp. Details
Dataset-1	255,723	12,734	519,389	-	7	x86, x64, ARM, MIPS (32/64b)	GCC, Clang (O0–O3, Os)
Dataset-2	-	-	72,215	-	10	x86, x64, ARM-32, MIPS-32	GCC (O0–O3)
Dataset-3	-	-	475,849	3,927	16	x64, ARM-32, MIPS-32	GCC (O0, O3)

#Train, #Val., #Test: function counts in each split. #Vuln.: vulnerable functions.

#Projs.: No. of source projects. Archs.: Architectures. Comp. Details: Compiler/Optimization details.

4 Systematic Evaluation

In this section, we leverage our framework to answer the three research questions through a systematic evaluation. We begin by introducing the datasets (§ 4.1). Following this, we address each research question sequentially by performing the tasks and analyzing the quantitative results (§ 4.2, § 4.3, § 4.4). The section concludes with an examination of the threats to validity (§ 4.5).

4.1 Datasets

We use three complementary datasets for in-distribution evaluation, out-of-distribution generalization, and real-world vulnerability detection (Table 2). For general code similarity, Dataset-1 [54] serves as our in-distribution testbed, covering seven projects compiled across multiple architectures (x86/ARM/MIPS) and optimization levels. Dataset-2 is curated from Trex [62] with strict de-duplication against Dataset-1, supporting out-of-distribution (OOD) generalization evaluation. For realistic security scenarios, Dataset-3 is derived from CVEBinarySheet [6] through symbol validation and feature-vocabulary alignment, comprising 564 CVEs and 3,927 vulnerable instances.

4.2 RQ1: Robustness and Generalization

Experimental Setup. We evaluate detection performance across six scenarios: single-variable shifts in Optimization (XO), Compiler (XC), and Architecture (XA), and their compound variations (XC+XO, XA+XO, XA+XC+XO). All six tasks run on Dataset-1; Dataset-2 is restricted to non-compiler tasks due to its single-compiler nature. We evaluate on 1,000 randomly sampled query functions with a fixed pool size of 10,000, using MRR and Recall@1 as defined in §3.4.5.

4.2.1 In-Distribution Performance Analysis. Table 3 details in-distribution performance on Dataset-1. Under our shared protocol, graph representation dominates detection effectiveness while backbone choice mainly modulates how each representation is exploited. Most configurations perform best on cross-architecture (XA), suggesting that architecture shifts are less disruptive than compiler and optimization changes under the evaluated pipeline.

The combination of GGNN, P-SOG, and ToK-opc achieves the best overall performance, reaching an MRR of 0.489 on the most challenging XA+XC+XO setting—substantially above CFG-based baselines. P-SOG also ranks near the top under Structure2Vec, GIN, and SGFormer, suggesting that its advantage is not limited to the GGNN backbone and is likely driven by its fusion of multi-dimensional semantic relations and pruning of meaningless nodes, which mitigates noise from combined variations. Nevertheless, the drop from 0.815 (XA) to 0.489 (XA+XC+XO) shows that multi-factor variability remains a substantial challenge.

Dependency-oriented representations, particularly P-DFG and P-PDG, are robust to optimization shifts. P-DFG remains near 0.60 MRR on XO under message-passing backbones (lower under SGFormer), suggesting that instruction-level data-dependency chains retain stable semantic signals when optimization restructures local control flow. P-DDSG also benefits from slicing but performs weaker on optimization and compound shifts, likely because its slice-level nodes aggregate multiple operations and its dense topology increases over-smoothing during message passing.

Table 3. Results of the comparative experiments on the Dataset-1 for XA, XO, XC, XA+XO, XC+XO and XA+XC+XO subtasks (*poolsize* = 10, 000). **Bold** indicates the best performance, and gray-shaded cells indicate the second-best. Prefixes “M-”, “P-”, and “A-” denote graphs derived from IDA Pro’s Microcode, Ghidra’s P-code, and raw assembly, respectively.

GNN	Graph	Feat.	MRR						Recall@1					
			XA	XC	XO	XA+XO	XC+XO	XA+XC+XO	XA	XC	XO	XA+XO	XC+XO	XA+XC+XO
Structure2Vec	A-CFG	Manual	0.693	0.449	0.471	0.427	0.369	0.303	0.608	0.386	0.417	0.356	0.307	0.241
		BoW-opc	0.723	0.464	0.488	0.456	0.372	0.319	0.637	0.402	0.442	0.378	0.310	0.265
		W2V+Attn	0.581	0.398	0.401	0.334	0.289	0.214	0.486	0.342	0.355	0.264	0.234	0.166
		W2V+RNN	0.699	0.445	0.457	0.427	0.352	0.295	0.613	0.379	0.405	0.349	0.291	0.238
		HBMP	0.752	0.484	0.529	0.497	0.407	0.364	0.667	0.414	0.477	0.422	0.341	0.305
	M-CFG	BoW-opc	0.691	0.476	0.523	0.449	0.387	0.324	0.603	0.413	0.472	0.374	0.326	0.263
	P-CFG	BoW-opc	0.565	0.395	0.467	0.384	0.344	0.303	0.501	0.337	0.422	0.319	0.293	0.253
	P-DFG	ToK-opc	0.623	0.489	0.594	0.473	0.444	0.389	0.538	0.415	0.535	0.393	0.371	0.321
	P-DDSG	BoW-opc	0.609	0.436	0.508	0.423	0.376	0.339	0.534	0.375	0.454	0.356	0.312	0.285
	M-LSFG	BoW-opc	0.672	0.472	0.529	0.445	0.383	0.317	0.582	0.405	0.480	0.372	0.319	0.256
	P-PDG	ToK-opc	0.641	0.488	0.576	0.483	0.440	0.390	0.560	0.412	0.517	0.406	0.370	0.325
	A-BDG	BoW-opc	0.678	0.569	0.657	0.488	0.498	0.396	0.619	0.508	0.612	0.422	0.437	0.339
	P-SOG	ToK-opc	0.651	0.515	0.615	0.453	0.450	0.363	0.554	0.445	0.562	0.368	0.382	0.294
GCNN	A-CFG	Manual	0.723	0.490	0.532	0.481	0.415	0.360	0.635	0.420	0.472	0.397	0.347	0.293
		BoW-opc	0.755	0.498	0.544	0.505	0.411	0.366	0.671	0.431	0.491	0.428	0.344	0.301
		W2V+Attn	0.654	0.462	0.509	0.433	0.388	0.330	0.561	0.392	0.450	0.352	0.315	0.262
		W2V+RNN	0.731	0.481	0.534	0.485	0.408	0.365	0.644	0.409	0.477	0.399	0.339	0.298
		HBMP	0.764	0.500	0.552	0.523	0.428	0.390	0.684	0.428	0.493	0.444	0.356	0.324
	M-CFG	BoW-opc	0.768	0.499	0.563	0.513	0.424	0.384	0.693	0.432	0.511	0.436	0.359	0.323
	P-CFG	BoW-opc	0.586	0.402	0.481	0.406	0.349	0.323	0.521	0.344	0.430	0.342	0.295	0.275
	P-DFG	ToK-opc	0.670	0.505	0.594	0.508	0.461	0.416	0.588	0.431	0.531	0.429	0.390	0.350
	P-DDSG	BoW-opc	0.678	0.472	0.553	0.479	0.413	0.367	0.607	0.405	0.492	0.406	0.351	0.312
	M-LSFG	BoW-opc	0.767	0.501	0.559	0.515	0.424	0.387	0.689	0.434	0.507	0.437	0.357	0.323
	P-PDG	ToK-opc	0.669	0.496	0.582	0.504	0.448	0.410	0.588	0.418	0.517	0.426	0.378	0.345
	A-BDG	BoW-opc	0.782	0.520	0.578	0.538	0.446	0.397	0.707	0.454	0.524	0.461	0.379	0.332
	P-SOG	ToK-opc	0.815	0.586	0.686	0.620	0.538	0.489	0.737	0.510	0.629	0.535	0.467	0.421
GIN	A-CFG	Manual	0.678	0.448	0.476	0.423	0.362	0.309	0.594	0.387	0.427	0.352	0.301	0.250
		BoW-opc	0.720	0.459	0.489	0.458	0.371	0.325	0.635	0.396	0.441	0.386	0.313	0.271
		W2V+Attn	0.610	0.404	0.418	0.362	0.309	0.249	0.520	0.346	0.373	0.291	0.250	0.195
		W2V+RNN	0.703	0.448	0.469	0.437	0.352	0.306	0.623	0.385	0.418	0.362	0.292	0.249
		HBMP	0.751	0.488	0.530	0.496	0.408	0.364	0.667	0.421	0.478	0.420	0.342	0.305
	M-CFG	BoW-opc	0.725	0.469	0.519	0.477	0.386	0.340	0.643	0.403	0.470	0.401	0.322	0.280
	P-CFG	BoW-opc	0.541	0.387	0.453	0.371	0.336	0.293	0.481	0.333	0.408	0.312	0.286	0.249
	P-DFG	ToK-opc	0.662	0.503	0.599	0.498	0.457	0.409	0.576	0.429	0.536	0.417	0.384	0.344
	P-DDSG	BoW-opc	0.606	0.435	0.503	0.421	0.376	0.333	0.537	0.373	0.443	0.353	0.314	0.279
	M-LSFG	BoW-opc	0.713	0.474	0.525	0.464	0.393	0.338	0.630	0.410	0.477	0.388	0.330	0.280
	P-PDG	ToK-opc	0.662	0.486	0.584	0.493	0.446	0.403	0.580	0.408	0.524	0.414	0.375	0.337
	A-BDG	BoW-opc	0.739	0.507	0.561	0.484	0.429	0.361	0.663	0.449	0.514	0.412	0.372	0.304
	P-SOG	ToK-opc	0.709	0.530	0.638	0.501	0.469	0.407	0.615	0.456	0.584	0.411	0.392	0.337
SGFormer	A-CFG	Manual	0.649	0.436	0.448	0.400	0.333	0.273	0.560	0.377	0.396	0.328	0.273	0.217
		BoW-opc	0.704	0.449	0.475	0.440	0.358	0.305	0.610	0.385	0.429	0.367	0.298	0.249
		W2V+Attn	0.588	0.394	0.398	0.336	0.284	0.224	0.497	0.338	0.356	0.266	0.231	0.176
		W2V+RNN	0.672	0.435	0.446	0.408	0.332	0.280	0.585	0.374	0.398	0.337	0.273	0.226
		HBMP	0.729	0.471	0.508	0.471	0.389	0.344	0.642	0.402	0.457	0.394	0.325	0.281
	M-CFG	BoW-opc	0.683	0.462	0.511	0.442	0.372	0.316	0.598	0.396	0.465	0.370	0.311	0.255
	P-CFG	BoW-opc	0.570	0.391	0.462	0.400	0.338	0.300	0.504	0.333	0.415	0.340	0.290	0.253
	P-DFG	ToK-opc	0.549	0.433	0.524	0.406	0.386	0.326	0.471	0.367	0.469	0.332	0.323	0.269
	P-DDSG	BoW-opc	0.611	0.440	0.514	0.429	0.386	0.335	0.541	0.378	0.454	0.361	0.329	0.279
	M-LSFG	BoW-opc	0.653	0.459	0.511	0.421	0.366	0.303	0.562	0.395	0.466	0.346	0.302	0.245
	P-PDG	ToK-opc	0.602	0.450	0.544	0.444	0.406	0.358	0.519	0.379	0.487	0.369	0.337	0.299
	A-BDG	BoW-opc	0.662	0.538	0.619	0.461	0.464	0.370	0.600	0.478	0.574	0.401	0.409	0.316
	P-SOG	ToK-opc	0.651	0.511	0.605	0.455	0.438	0.362	0.552	0.441	0.552	0.368	0.368	0.291

Stats. Values report means over 10 runs. Adjacent Top-4 MRR ranks are tested by paired Wilcoxon tests with local Holm correction. Top1–Top2 is significant in 20/24 settings, with mean Δ MRR = 0.0345 and mean RBC = 0.868. Full results are in the supplementary material[84].

Abbrev. CFG: Control Flow Graph. DFG: Data Flow Graph. DDSG: Data Dependency Slicing Graph. LSFG: Labeled Semantics Flow Graph. PDG: Program Dependence Graph. BDG: Binary Disassembly Graph. SOG: Semantics-Oriented Graph.

Hybrid representations show a mixed pattern rather than a uniformly positive effect. Although P-SOG achieves the strongest overall performance, other hybrid graphs such as P-PDG, M-LSFG, and A-BDG are not consistently superior to the more focused P-DFG in compound-shift settings. This suggests that semantic fusion is beneficial only when the added relations improve signal clarity rather than introduce topological noise. Control-flow-centric representations can also be sensitive to multi-factor variation, as compiler and optimization changes may reshape control topology despite the usefulness of control-flow signals. Therefore, richer graphs are not automatically more robust; robustness depends on preserving stable semantic signals while avoiding unnecessary structural complexity.

Beyond topology, decompiler IRs also affect robustness as a canonicalization layer. Microcode-based M-CFG outperforms raw-assembly A-CFG on many tasks. For features, simple BoW-opc is surprisingly robust against learned W2V embeddings in cross-compilation settings, likely because opcode-frequency distributions remain stable under compilation variation.

Finding 1.1: SOG is broadly effective, while dependency-oriented graphs resist optimization shifts. Under our unified protocol, most graph configurations perform best on XA. SOG achieves the strongest overall results through fine-grained semantic fusion, while DFG and PDG remain robust on XO, likely due to relatively stable data-dependency signals.

4.2.2 Backbone Sensitivity Analysis. To examine whether the representation-level observations depend on a particular GNN backbone, we measure cross-backbone ranking concordance using Kendall's coefficient of concordance (W) [38]. Kendall's W treats the four backbones as rankers and summarizes their agreement on the 13 graph configurations as a single value in $[0, 1]$, where larger values indicate stronger concordance.

Table 4 reports cross-backbone concordance under both metrics. For MRR, Kendall's W ranges from 0.841 (XA) to 0.945 (XO), with a mean of $\bar{W} = 0.897$; for Recall@1, it ranges from 0.816 (XA+XO) to 0.959 (XO), with $\bar{W} = 0.892$. All values are significant under the χ^2 approximation ($p < 0.001$, $df = 12$). Since all observed values exceed 0.8, the results indicate strong agreement among the four backbones in ranking the 13 graph configurations.

The relatively lower concordance on architecture-shift tasks (MRR $W = 0.841$ on XA; Recall@1 $W = 0.816$ on XA+XO) suggests that some backbones may exploit graph topologies differently when cross-architecture invariance is required. Nevertheless, the overall results show that representation rankings are broadly stable across backbones, and the main trends in Table 3 are unlikely to be artifacts of one particular backbone. For finer-grained diagnostics, we provide the full pairwise Spearman correlation matrix in the supplementary material [84].

Finding 1.2: Rankings are concordant across backbones. Kendall's W yields $\bar{W} = 0.897$ for MRR and $\bar{W} = 0.892$ for Recall@1 across the four backbones (all $p < 0.001$), indicating strong agreement in ranking the 13 graph configurations. Thus, the main representation-level trends are not tied to one GNN architecture, although absolute performance can still vary.

4.2.3 Out-of-Distribution Generalization. To assess generalization capability, we evaluate models trained on Dataset-1 directly against the unseen Dataset-2. Since Dataset-2 is built with a single compiler, this OOD evaluation focuses on non-compiler variation tasks, including cross-architecture (XA), cross-optimization (XO), and their compound setting (XA+XO). Figure 3 visualizes the Recall@1 contrast between the in-distribution Dataset-1 and the out-of-distribution Dataset-2, with full numerical results available in the supplementary material [84].

Table 4. Cross-backbone ranking concordance measured by Kendall's W . For each task and metric, the four backbones rank the same 13 graph configurations. $W = 1$ indicates perfect agreement.

Metric	Statistic	XA	XC	XO	XA+XO	XC+XO	XA+XC+XO
MRR	Kendall's W	0.841	0.891	0.945	0.846	0.944	0.913
	p -value (χ^2)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
Recall@1	Kendall's W	0.860	0.881	0.959	0.816	0.931	0.902
	p -value (χ^2)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001

The transfer behavior is broadly consistent across representations, with no single graph clearly dominating all Dataset-2 axes. From the radar profiles, most representations retain comparable or slightly improved Dataset-2 coverage, suggesting that models trained on Dataset-1 transfer reasonably well under non-compiler shifts. CFG-like variants augmented with additional semantic signals, including A-BDG with external string/syscall anchor nodes and M-LSFG with inter-block memory-dependency edges, remain stable across the three axes, though without a decisive lead. Dependency-oriented graphs, such as P-DFG and P-PDG, retain competitive coverage on the optimization axis (A2: XO), consistent with our earlier observation in §4.2.1 that explicit def-use relations are relatively robust to local control-flow reshaping. By contrast, P-SOG, despite its strong in-distribution performance, does not consistently dominate Dataset-2, suggesting that richer token-level semantic fusion does not automatically translate into stronger OOD transfer. Overall, OOD transfer is not determined by representation richness alone; different graph semantics offer comparable but scenario-dependent robustness.

Finding 1.3: OOD transfer is broadly stable but scenario-dependent. On Dataset-2, most graph representations show comparable performance under XA, XO, and XA+XO, with no single dominant graph. Dependency graphs remain competitive on XO, while richer semantic fusion does not necessarily improve transfer.

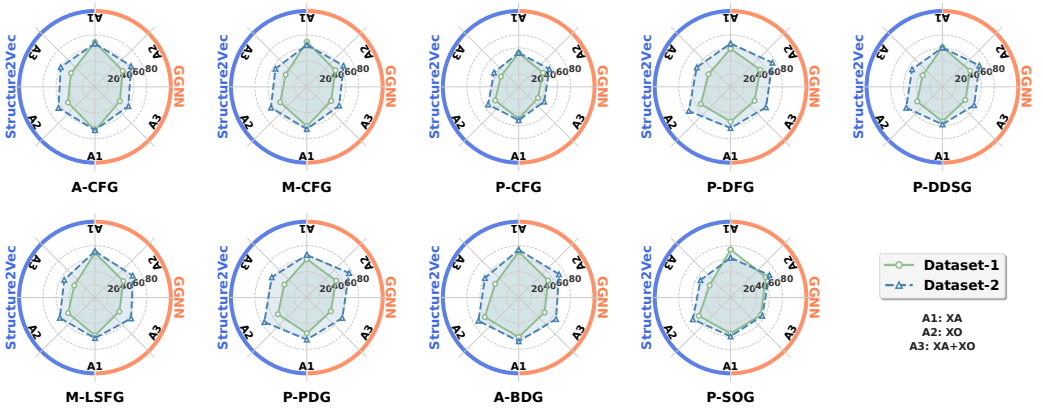


Fig. 3. Recall@1 comparison between Dataset-1 and the unseen Dataset-2 on non-compiler variation tasks. Since Dataset-2 uses a single compiler, compiler-shift tasks are not included. A1–A3 denote XA, XO, and XA+XO, respectively. Each radar chart reports Structure2Vec and GGNN results, with curves for Dataset-1 and Dataset-2.

4.3 RQ2: Vulnerability Detection & Combination Strategy

4.3.1 Vulnerability Detection. To evaluate the practical utility of graph representations in security applications, we simulate a real-world N-day vulnerability search scenario. Unlike the sampled pools used in the general similarity task, we expand the search scope to the entire corpus, comprising $N = 475,849$ functions. For each vulnerability class, we select one vulnerable instance as the query and treat the remaining vulnerable instances as relevant targets. Let t denote the number of relevant targets for the query. We rank all functions in the full corpus and measure Recall@ t as $\frac{m}{t}$, where m is the number of relevant vulnerable instances retrieved within the top- t ranked candidates.

Figure 4 summarizes the full-corpus vulnerability retrieval results across graph representations and backbones. Consistent with 4.2.3, results on the massive Dataset-3 confirm that representations anchored on stable invariants outperform baselines in high-noise environments. While A-CFG maintains a recall of 0.564, it is surpassed by A-BDG, M-CFG, and P-DFG. This superiority stems from distinct structural enhancements: A-BDG introduces external anchor nodes (e.g., const strings and system calls) to stabilize topology; M-CFG lifts raw assembly to Microcode, enriching the graph with explicit semantics; and P-DFG isolates pure definition-use chains, creating a clean logic trace that filters out control-flow noise.

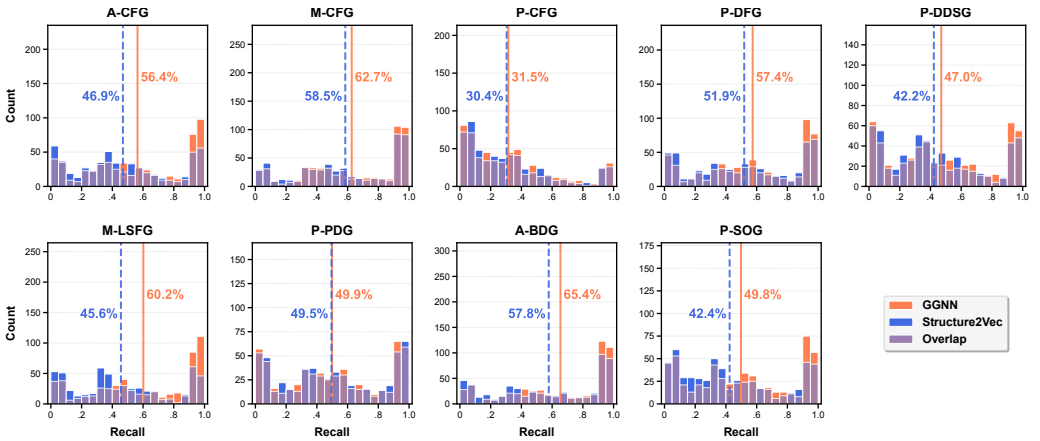


Fig. 4. Recall rates for N-day vulnerability detection across different graph representations. The dashed blue line and solid orange line indicate the mean recall of Structure2Vec and GGNN, respectively.

A notable observation is the performance dip of representations with complex coupling. Both P-SOG and P-PDG fall below the A-CFG baseline. Although these graphs combine richer semantic relations, the combination of multiple subgraphs in SOG, similar to the coupling of control and data dependencies in PDG, may introduce topological noise that weakens the distinctive features characterizing specific vulnerabilities. By comparison, representations focused on a single dimension of semantics, such as the purified logic in P-DFG or the anchor-augmented control flow in A-BDG, provide clearer signals for distinguishing targets from background functions. Furthermore, the unrefined P-CFG performs worst, falling even below A-CFG, as Ghidra’s raw P-code decomposes each assembly instruction into multiple uncanonicalized micro-operations that may inflate the CFG with implementation artifacts and obscure semantic signals.

Finding 2.1: Signal clarity matters more than structural complexity. In the N-day vulnerability detection setting, representations with focused semantic signals generally outperform more heterogeneous graph structures, suggesting that added topology does not necessarily translate into better vulnerability retrieval.

4.3.2 Complementarity Analysis and Ensemble Strategy. The analysis above indicates that models trained on distinct graph representations function as specialized filters, each excelling at capturing specific semantic dimensions while remaining blind to others. This suggests that their failure patterns, specifically the subsets of vulnerabilities missed by each model, might be orthogonal rather than identical. Consequently, we hypothesize that ensembling these diverse models can create a constructive interference effect, where the strengths of one predictor cover the blind spots of another, thereby elevating the upper bound of detection performance.

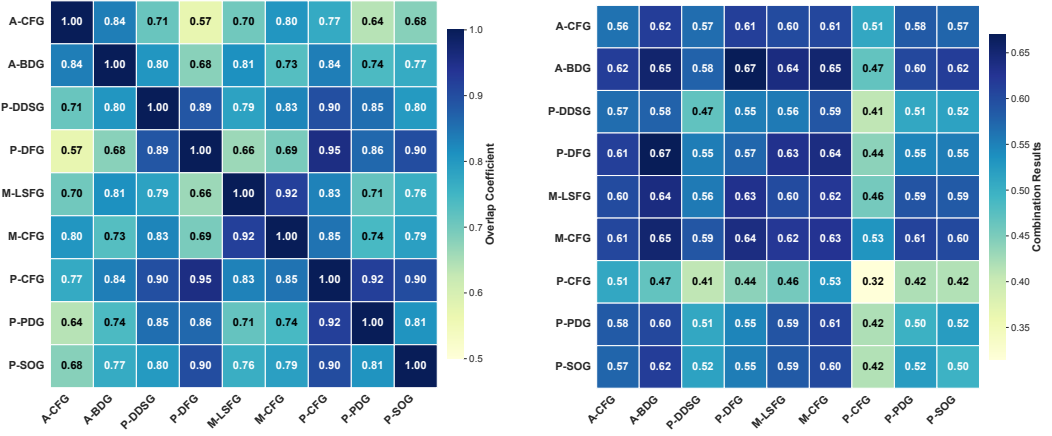


Fig. 5. The overlap coefficients of failed queries and combined retrieval results.

To validate the hypothesis of orthogonal failure patterns, we conduct a consistency analysis on hard-to-detect cases, defined as queries yielding a recall below $1/6$.⁴ We quantify the similarity of blind spots between any two models using the overlap coefficient:

$$\text{overlap}(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)} \quad (5)$$

where sets A and B denote the failed queries for each tool. A lower overlap signifies that the models fail on different vulnerability subsets, implying a higher potential for complementary gains.

The consistency analysis, visualized in Figure 5a, reveals that structural diversity is associated with orthogonal failure patterns. Representations within the same topological family or derived from the same unrefined IR exhibit high redundancy. For instance, P-CFG and P-DFG share an overlap exceeding 0.95, indicating that the noise inherent in raw P-code causes them to fail on nearly identical queries regardless of edge type. In sharp contrast, structurally distinct graphs like A-CFG and P-DFG demonstrate the lowest overlap of approximately 0.57. This suggests that

⁴In Dataset-3, each source function is compiled into 6 distinct binary instances spanning 3 architectures (x86-64, ARM, MIPS) and 2 optimization levels (O0, O3). Therefore, a recall lower than $1/6$ indicates a complete failure to retrieve any of the 6 target instances.

control-flow and data-flow representations possess complementary blind spots, failing on different subsets of vulnerabilities.

To exploit these orthogonal blind spots, we adopt Reciprocal Rank Fusion (RRF), a widely established technique in Information Retrieval for robust rank aggregation [8, 48, 63]. Specifically, we merge the retrieval lists by calculating a unified score $S(d) = \sum \frac{1}{k+r_i(d)}$ with $k = 10$. The results in Figure 5b show that combining low-overlap pairs can yield complementary gains. Merging the structurally orthogonal A-CFG and P-DFG boosts absolute recall to 0.614. More impressively, combining P-DFG with A-BDG yields the peak performance of 0.670, surpassing the best single-model result of 0.654. This combination effectively integrates the internal logic-trace strengths of DFG with the external behavioral anchors of BDG, creating a comprehensive detection filter. Conversely, combining highly correlated representations, such as M-CFG and M-LSFG, results in negligible gains. This demonstrates that maximizing structural diversity, rather than merely stacking high-accuracy models, is the key to effective ensemble strategies in vulnerability detection.

Finding 2.2: Ensemble gains rely on semantic complementarity. Vulnerability retrieval improves when low-overlap representations, such as control-flow and data-dependence graphs, are combined. Thus, effective ensembles rely on complementary failure patterns, not only on strong individual models.

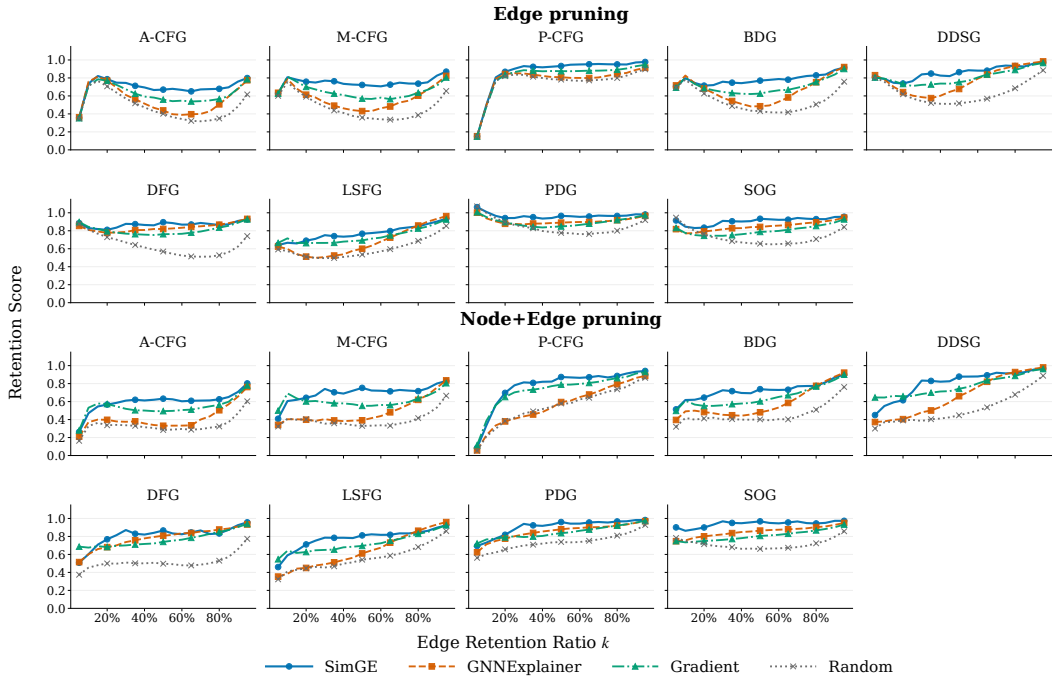


Fig. 6. Similarity retention under edge pruning (top) and node+edge pruning (bottom) across nine graph representations on GGNN. Each panel compares SimGE with three baselines (GNNExplainer, Gradient, Random); higher curves indicate better preservation of the trained Siamese model's original similarity score.

4.4 RQ3: Similarity Preservation and Structural Reliance

Experimental Setup. To answer RQ3, we derive subgraph pairs from each original graph pair and evaluate whether they preserve a trained BCSD model’s learned similarity scores. We consider two pruning protocols: *edge pruning*, which keeps all nodes and node features fixed while retaining only the top-ranked edges selected by each explanation method, and *node+edge pruning*, which further removes nodes disconnected from the retained edges to form compact edge-induced subgraphs. The former isolates the effect of graph connectivity, while the latter provides a stricter test of whether compact subgraph pairs can preserve the model’s learned similarity behavior. We instantiate this analysis on GGNN, the best-performing backbone in Table 3, so that pruning behavior is compared under a shared Siamese BCSD architecture. We further repeat the analysis with Structure2Vec and observe broadly consistent trends, as reported in the supplementary material [84]. We evaluate 300 successful retrieval pairs sampled from the six RQ1 subtasks, where each pair is correctly retrieved by all nine GGNN graph configurations.⁵

We compare SimGE with three baselines. Random assigns random edge scores and averages results over five random seeds. Gradient ranks edges using the single-graph embedding-norm gradient, applied independently to each graph. GNNExplainer [85] learns a single-graph edge mask to preserve the original graph embedding, and we apply it independently to each graph in a pair. For each method, we retain the top- $k\%$ edges with $k \in \{5\%, 10\%, \dots, 95\%\}$. We quantify score preservation using similarity retention:

$$\text{Retention}_{\pi}@k\% = \frac{1}{|P|} \sum_{(g_a, g_b) \in P} \frac{S(\mathcal{T}_{\pi}(g_a, k), \mathcal{T}_{\pi}(g_b, k))}{S(g_a, g_b)}, \quad (6)$$

where $S(\cdot)$ is the trained BCSD model’s cosine similarity score, $\pi \in \{\text{edge}, \text{node+edge}\}$ denotes the pruning protocol, and $\mathcal{T}_{\pi}(g, k)$ denotes the graph obtained after retaining the top- $k\%$ structures under protocol π . A higher retention score indicates that the resulting subgraph pair preserves more of the trained model’s original similarity score under the corresponding pruning protocol.

4.4.1 Effectiveness of SimGE. Figure 6 compares SimGE with Random, Gradient, and GNNExplainer under the two pruning protocols. Across most graph representations, SimGE preserves the original similarity score more effectively than the baselines, especially in the middle pruning range where only a moderate fraction of edges is retained. This trend is consistent under both edge pruning and node+edge pruning, suggesting that the pair-aware objective is better aligned with the similarity-preservation criterion used in this analysis.

To avoid relying on a single retention threshold, we summarize each curve using AUC-Retention,⁶ where higher values indicate better preservation of the original similarity score across pruning levels. Across the nine graph configurations, SimGE achieves the highest mean AUC-Retention under both pruning protocols: 0.840 ± 0.082 under edge pruning, compared with Gradient (0.774 ± 0.086), GNNExplainer (0.749 ± 0.119), and Random (0.651 ± 0.122); and 0.794 ± 0.100 under the stricter node+edge pruning protocol, compared with Gradient (0.731 ± 0.095), GNNExplainer (0.667 ± 0.150), and Random (0.553 ± 0.127). These results indicate that SimGE’s pair-aware objective better preserves learned similarity scores than the considered baselines, supporting its use as the primary model-output-preservation probe for the following representation-level analysis.

⁵We also repeat the pruning analysis on false-positive and false-negative pairs in the supplementary material [84]. The retention curves show broadly similar representation-level shapes, suggesting that the observed pruning patterns are not specific to successful retrieval cases, although we use TP pairs as the main analysis set to focus on comparable learned similarity decisions.

⁶Given retention scores $R(k_i)$ at the evaluated edge-retention ratios k_i , AUC-Retention is computed with the trapezoidal rule: $\sum_i \frac{R(k_i) + R(k_{i+1})}{2} (k_{i+1} - k_i)$.

4.4.2 Representation-Level Structural Reliance. Beyond explainer-level effectiveness, we use SimGE as a model-level probe to compare how different graph representations preserve learned similarity under graph reduction. The retention curves indicate whether a retained edge set or edge-induced subgraph is sufficient to preserve the original GGNN similarity score, rather than identifying unique decision causes or ground-truth program semantics.

Figure 6 compares representation-level pruning behavior under the two complementary protocols. Under edge pruning, DFG, PDG, and SOG maintain high retention with small or moderate edge subsets, suggesting that the trained model’s similarity scores on these representations can often be preserved after substantial edge reduction. In contrast, A-CFG, M-CFG, BDG, DDSG, and LSFG degrade more noticeably as edges are removed, indicating greater sensitivity to graph connectivity. P-CFG behaves as a partial exception among CFG variants: it remains stable in most pruning ranges but drops when the retained edge set becomes very sparse. The stricter node+edge pruning protocol lowers retention overall because it removes both edges and the node context disconnected from the retained structures. Nevertheless, the relative trends remain broadly consistent: DFG, PDG, and SOG are still among the more pruning-tolerant representations, whereas several connectivity-heavy graphs require larger retained subgraphs to preserve comparable similarity scores. This cross-protocol consistency suggests that the observed patterns are not merely artifacts of keeping all nodes fixed during edge pruning. The grouping is also not purely taxonomy-driven; for example, DDSG is data-centric by construction but exhibits a pruning profile closer to the connectivity-sensitive representations, indicating that learned similarity-preserving behavior is not determined solely by the control/data taxonomy.

Finding 3.1: Similarity-preserving subgraph pairs are representation-sensitive.

Pair-aware retention. In our GGNN-based Siamese setting, SimGE achieves higher retention than the baselines across pruning protocols, supporting its use as a pair-aware probe for identifying subgraph pairs that preserve the trained model’s original similarity score.

Representation-specific sensitivity. Using this probe, DFG, PDG, and SOG are more pruning-tolerant, whereas A-CFG, M-CFG, BDG, DDSG, and LSFG exhibit larger score drops under graph reduction. This indicates that different graph representations induce different levels of connectivity sensitivity in the model’s learned similarity behavior.

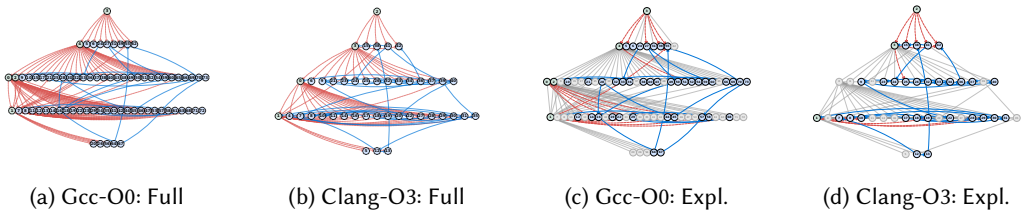


Fig. 7. Case study on P-PDGs for Z3’s `expand_vector` compiled under `x86-gcc-5-00` and `x86-clang-9-03`. The figure compares full graphs with SimGE-selected top-40% subgraphs. Red and blue edges denote control and data dependencies, respectively. After pruning, the learned similarity score decreases from 0.90 to 0.85, showing that this pair’s model similarity score remains largely preserved after retaining only the selected top-40% edges.

4.4.3 Case Study. To complement the retention-curve analysis, we present a qualitative case study on P-PDG, a hybrid representation that shows strong retention behavior in our pruning experiments. We use Z3’s `expand_vector` function [57], compiled under two different settings, `x86-gcc-5-00` and `x86-clang-9-03`. Figure 7 contrasts the full graphs with the subgraphs induced by the top-40%

edges selected by SimGE. After this reduction, the similarity score decreases only moderately from 0.90 to 0.85, suggesting that the learned similarity decision can still be largely preserved by a compact retained subgraph.

The retained subgraphs provide an intuitive example of the pruning behavior observed in RQ3. In the unoptimized binary, the full graph contains many control-dependence edges, which substantially increase graph connectivity. After pruning, many of these edges are removed, while the retained subgraphs include several data-flow-related dependencies that correspond to argument propagation and return-value handling. This pattern is consistent with the quantitative results: for this pair, SimGE finds a compact subgraph pair that preserves most of the model’s learned similarity score. We treat this example as illustrative evidence of model-level similarity preservation, rather than as proof that the retained edges constitute the unique semantic basis of the function.

4.5 Threats to Validity

Internal Validity. The primary threat to internal validity arises from potential biases in our implementation and instrumentation. First, regarding graph construction, we relied on re-implementations for representations without public source code. As described in §3.4, we prioritized official artifacts for CFG⁷, SOG⁸, and PDG⁹ while adhering to algorithmic specifications for others to mitigate implementation errors. We further corroborated the correctness of these re-implementations by confirming that their performance metrics closely align with the results established in prior studies [23, 30, 87]. Second, the reliance on specific reverse engineering tools and their intermediate representations introduces a measurement threat, as the resulting graph topologies may be biased by tool-specific lifting artifacts rather than intrinsic binary semantics. To mitigate this, we adopted the CFG as a transitional perspective by constructing variants corresponding to different IRs.

External Validity. A threat involves dataset bias and generalizability. Although our experiments cover diverse architectures and optimization levels, the datasets are mainly built with GCC and Clang. Thus, the representations may partially reflect compiler-specific heuristics rather than universal binary semantics. Our decoupled setting may differ from end-to-end co-designed BCSD systems, so we interpret the results as representation-level findings under a standardized retrieval protocol. We mitigate this threat through the backbone-sensitivity analysis in §4.2.2. Since GIN and SGFormer are added only on Dataset-1, our four-backbone analysis supports the stability of Dataset-1 rankings, but does not claim exhaustive validation across all datasets or GNN architectures.

5 Discussion

5.1 Finding Summary

Our study leads to three main observations. First, graph representations exhibit scenario-dependent behavior: no single topology dominates across compilation robustness, OOD generalization, and vulnerability detection. Second, representation-level trends are broadly stable across the evaluated GNN backbones, although absolute performance varies with architecture choice. Third, the pruning-based analysis shows representation-specific preservation behavior: subgraph pairs from DFG, PDG, and SOG more often preserve trained models’ similarity scores under graph reduction, whereas several other representations exhibit larger score drops. These findings suggest that graph selection should be treated as a task- and setting-dependent design choice rather than a fixed default.

⁷https://github.com/Cisco-Talos/binary_function_similarity

⁸<https://github.com/NSSL-SJTU/HermesSim>

⁹<https://github.com/yijiuflly/SigmaDiff>

5.2 Future Directions

Invariant-Aware Graph Construction. As shown in §4.3 and §4.4, representations that capture function calls, memory dependencies, and string constants can provide useful signals under compilation variation. Future research could prioritize graph construction that explicitly models such relatively stable program elements, while complementing more variation-sensitive control-flow topology, to improve cross-architecture and cross-compiler robustness.

Task-Adaptive Representation Selection. Our findings in §4.2, §4.3, and §4.4 show that graph representations exhibit different strengths across scenarios, and that models trained on different representations can provide complementary retrieval signals. Future work could therefore explore ensemble learning or mixture-of-experts mechanisms to dynamically fuse diverse representation models according to the target task and compilation setting.

LLM-Synergized Detection Pipeline. To balance scalability with semantic depth, future frameworks could adopt a coarse-to-fine strategy. Lightweight graph models can serve as an efficient first-stage filter for large-scale retrieval, followed by LLM-based analysis to perform fine-grained semantic verification on the top candidates.

5.3 Limitation & Future Work

Our study operates under design constraints. First, we focus on graph-based representations, excluding high-level abstractions such as ASTs or CPGs. Second, benchmarking targets the One-to-Many search scenario, leaving One-to-One diffing and clustering tasks out of scope. Third, SimGE analysis examines model-output preservation under pruning, rather than proving that the selected subgraph pairs are unique semantic explanations of binary functions. Finally, generalizability to proprietary ecosystems such as MSVC and to obfuscated binaries remains unverified.

In future work, we aim to bridge these gaps for a more comprehensive evaluation. We plan to extend our framework to cover more representation modalities and BCSD tasks, especially One-to-One diffing and clustering scenarios. We will also expand the evaluation corpus to include proprietary toolchains and obfuscated binaries, and further validate whether the observed representation-level trends persist under these more challenging environments.

6 Conclusion

In this paper, we present an empirical study of graph representations for ML-based BCSD to address the ambiguity regarding graph selection. By implementing a modular framework that decouples graph construction from model training, we evaluate seven graph topologies identified from our literature review under a shared similarity-learning and retrieval protocol. We systematically analyze their strengths and limitations across diverse scenarios, investigate combination strategies for realistic N-day vulnerability detection, and design SimGE, a pair-aware post-hoc analysis tool that probes model-level structural reliance through similarity retention under pruning. Finally, to facilitate future research and reproducibility, we open-source our source code and datasets.

Data Availability

All our artifacts and additional technical information are available at [84].

Acknowledgments

We thank the anonymous shepherd and reviewers for their insightful comments. This work is supported by the National Natural Science Foundation of China (Nos. 62372490 and W2412110), the CCF-Taobao & Tmall Group TECH Kangaroo Fund (No. 2026009), and the Shanghai Natural Science Foundation (No. 25ZR1402214).

References

- [1] Iftakhar Ahmad and Lannan Luo. 2023. Unsupervised Binary Code Translation with Application to Code Clone Detection and Vulnerability Discovery. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 14581–14592. doi:10.18653/v1/2023.findings-emnlp.971
- [2] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2017. Learning to Represent Programs with Graphs. *CoRR* abs/1711.00740 (2017). arXiv:1711.00740 <http://arxiv.org/abs/1711.00740>
- [3] Fiorella Artuso, Marco Mormando, Giuseppe Antonio Di Luna, and Leonardo Querzoni. 2025. BinBert: Binary Code Understanding With a Fine-Tunable and Execution-Aware Transformer. *IEEE Trans. Dependable Secur. Comput.* 22, 1 (Jan. 2025), 308–326. doi:10.1109/TDSC.2024.3397660
- [4] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "Siamese" time delay neural network. In *Proceedings of the 7th International Conference on Neural Information Processing Systems* (Denver, Colorado) (NIPS'93). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 737–744.
- [5] Sicong Cao, Xiaobing Sun, Lili Bo, Rongxin Wu, Bin Li, and Chuanqi Tao. 2022. MVD: memory-related vulnerability detection based on flow-sensitive graph neural networks. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) (ICSE '22). Association for Computing Machinery, New York, NY, USA, 1456–1468. doi:10.1145/3510003.3510219
- [6] Lingfeng Chen. 2025. CveBinarySheet: A Comprehensive Pre-built Binaries Database for IoT Vulnerability Analysis. arXiv:2501.08840 [cs.CR] <https://arxiv.org/abs/2501.08840>
- [7] Xiao Cheng, Haoyu Wang, Jiayi Hua, Guoai Xu, and Yulei Sui. 2021. DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network. *ACM Trans. Softw. Eng. Methodol.* 30, 3, Article 38 (April 2021), 33 pages. doi:10.1145/3436877
- [8] Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Boston, MA, USA) (SIGIR '09). Association for Computing Machinery, New York, NY, USA, 758–759. doi:10.1145/1571941.1572114
- [9] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*. PMLR, 2702–2711.
- [10] Yaniv David, Nimrod Partush, and Eran Yahav. 2016. Statistical similarity of binaries. *Acm sigplan notices* 51, 6 (2016), 266–280. doi:10.1145/2980983.2908126
- [11] Yaniv David, Nimrod Partush, and Eran Yahav. 2017. Similarity of binaries through re-optimization. In *Proceedings of the 38th ACM SIGPLAN conference on programming language design and implementation*. 79–94. doi:10.1145/3062341.3062387
- [12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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*. Association for Computational Linguistics, 4171–4186. doi:10.18653/v1/N19-1423
- [13] Steven HH Ding, Benjamin CM Fung, and Philippe Charland. 2019. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 472–489. doi:10.1109/SP.2019.00003
- [14] Pedro Domingos. 2012. A few useful things to know about machine learning. *Commun. ACM* 55, 10 (Oct. 2012), 78–87. doi:10.1145/2347736.2347755
- [15] Chaopeng Dong, Siyuan Li, Shouguo Yang, Yang Xiao, Yongpan Wang, Hong Li, Zhi Li, and Limin Sun. 2024. LibvDiff: Library Version Difference Guided OSS Version Identification in Binaries. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)* (2024), 791–802. doi:10.1145/3597503.3623336
- [16] Sebastian Eschweiler, Khaled Yakdan, and Elmar Gerhards-Padilla. 2016. discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code. In *Network and Distributed System Security Symposium*. doi:10.14722/ndss.2016.23185
- [17] Zuo Fei, Li Xiaopeng, Young Patrick, Luo Lannan, Zeng Qiang, and Zhang Zhixin. 2019. Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs. In *26th Annual Network and Distributed System Security Symposium*, NDSS. doi:10.14722/ndss.2019.23492
- [18] Qian Feng, Minghua Wang, Mu Zhang, Rundong Zhou, Andrew Henderson, and Heng Yin. 2017. Extracting conditional formulas for cross-platform bug search. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 346–359. doi:10.1145/3052973.3052995
- [19] Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin. 2016. Scalable graph-based bug search for firmware images. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 480–491. doi:10.1145/2976749.2978370
- [20] Lirong Fu, Peiyu Liu, Wenlong Meng, Kangjie Lu, Shize Zhou, Xuhong Zhang, Wenzhi Chen, and Shouling Ji. 2024. Understanding the AI-powered Binary Code Similarity Detection. *arXiv preprint arXiv:2410.07537* (2024).

- arXiv:2410.07537 [cs.SE] doi:10.48550/arXiv.2410.07537
- [21] Jian Gao, Yu Jiang, Zhe Liu, Xin Yang, Cong Wang, Xun Jiao, Zijiang Yang, and Jianguang Sun. 2021. Semantic Learning and Emulation Based Cross-Platform Binary Vulnerability Seeker. *IEEE Transactions on Software Engineering* 47, 11 (2021), 2575–2589. doi:10.1109/TSE.2019.2956932
 - [22] Jian Gao, Xin Yang, Ying Fu, Yu Jiang, Heyuan Shi, and Jianguang Sun. 2018. VulSeeker-pro: enhanced semantic learning based binary vulnerability seeker with emulation. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) (ESEC/FSE 2018). Association for Computing Machinery, New York, NY, USA, 803–808. doi:10.1145/3236024.3275524
 - [23] Jian Gao, Xin Yang, Ying Fu, Yu Jiang, and Jianguang Sun. 2018. Vulseeker: A semantic learning based vulnerability seeker for cross-platform binary. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 896–899. doi:10.1145/3238147.3240480
 - [24] Lian Gao, Yu Qu, Sheng Yu, Yue Duan, and Heng Yin. 2024. SigmaDiff: Semantics-Aware Deep Graph Matching for Pseudocode Diffing. *Proceedings 2024 Network and Distributed System Security Symposium* (2024). doi:10.14722/ndss.2024.23208
 - [25] A. Graves and J. Schmidhuber. 2005. Frameworkwise phoneme classification with bidirectional LSTM networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, Vol. 4. 2047–2052 vol. 4. doi:10.1109/IJCNN.2005.1556215
 - [26] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2020. GraphCodeBERT: Pre-training Code Representations with Data Flow. *CoRR* abs/2009.08366 (2020). arXiv:2009.08366 <https://arxiv.org/abs/2009.08366>
 - [27] Yixin Guo, Pengcheng Li, Yingwei Luo, Xiaolin Wang, and Zhenlin Wang. 2022. Exploring gnn based program embedding technologies for binary related tasks. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*. 366–377. doi:10.1145/3524610.3527900
 - [28] Irfan Ul Haq and Juan Caballero. 2021. A survey of binary code similarity. *Acm computing surveys (csur)* 54, 3 (2021), 1–38. doi:10.1145/3446371
 - [29] Haojie He, Xingwei Lin, Ziang Weng, Ruijie Zhao, Shuitao Gan, Libo Chen, Yuede Ji, Jiashui Wang, and Zhi Xue. 2024. Code is not natural language: Unlock the power of semantics-oriented graph representation for binary code similarity detection. In *33rd USENIX Security Symposium (USENIX Security 24)*, PHILADELPHIA, PA.
 - [30] Kaiyan He, Yikun Hu, Xuehui Li, Yunhao Song, Yubo Zhao, and Dawu Gu. 2024. Strtune: Data Dependence-Based Code Slicing for Binary Similarity Detection With Fine-Tuned Representation. *IEEE Transactions on Information Forensics and Security* 19 (2024), 10233–10245. doi:10.1109/TIFS.2024.3484944
 - [31] Hex-Rays. [n.d.]. IDA Pro. <https://hex-rays.com/ida-pro/>.
 - [32] Jiyong Jang, David Brumley, and Shobha Venkataraman. 2011. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on Computer and communications security*. 309–320. doi:10.1145/2046707.2046742
 - [33] Ang Jia, Ming Fan, Wuxia Jin, Xi Xu, Zhaohui Zhou, Qiyi Tang, Sen Nie, Shi Wu, and Ting Liu. 2023. 1-to-1 or 1-to-n? Investigating the Effect of Function Inlining on Binary Similarity Analysis. *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–26. doi:10.1145/3561385
 - [34] Ang Jia, Ming Fan, Xi Xu, Wuxia Jin, Haijun Wang, and Ting Liu. 2024. Cross-Inlining Binary Function Similarity Detection. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)* (2024), 2758–2770. doi:10.1145/3597503.3639080
 - [35] Ling Jiang, Junwen An, Huihui Huang, Qiyi Tang, Sen Nie, Shi Wu, and Yuqun Zhang. 2024. BinaryAI: Binary Software Composition Analysis via Intelligent Binary Source Code Matching. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)* (2024), 2771–2783. doi:10.1145/3597503.3639100
 - [36] Ling Jiang, Junwen An, Huihui Huang, Qiyi Tang, Sen Nie, Shi Wu, and Yuqun Zhang. 2024. BinaryAI: Binary Software Composition Analysis via Intelligent Binary Source Code Matching. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) (ICSE '24). Association for Computing Machinery, New York, NY, USA, Article 224, 13 pages. doi:10.1145/3597503.3639100
 - [37] Wesley Jin, Sagar Chaki, Cory Cohen, Arie Gurfinkel, Jeffrey Havrilla, Charles Hines, and Priya Narasimhan. 2012. Binary function clustering using semantic hashes. In *2012 11th International Conference on Machine Learning and Applications*, Vol. 1. IEEE, 386–391.
 - [38] Maurice George Kendall and Jean Dickinson Gibbons. 1962. Rank correlation methods. (1962).
 - [39] Geunwoo Kim, Sanghyun Hong, Michael Franz, and Dokyung Song. 2022. Improving cross-platform binary analysis using representation learning via graph alignment. *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (2022). doi:10.1145/3533767.3534383

- [40] Quoc V. Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *International Conference on Machine Learning*.
- [41] Siyuan Li, Yongpan Wang, Chaopeng Dong, Shouguo Yang, Hong Li, Hao Sun, Zhe Lang, Zuxin Chen, Weijie Wang, Hongsong Zhu, and Limin Sun. 2023. LibAM: An Area Matching Framework for Detecting Third-Party Libraries in Binaries. *ACM Transactions on Software Engineering and Methodology* 33 (2023), 1 – 35. doi:10.1145/3625294
- [42] Weilong Li, Jintian Lu, Rui Xiao, Pengfei Shao, and Shuyuan Jin. 2024. RCFG2Vec: Considering Long-Distance Dependency for Binary Code Similarity Detection. 2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE) (2024), 770–782. doi:10.1145/3691620.3695070
- [43] Xuezixiang Li, Yu Qu, and Heng Yin. 2021. Palmtree: Learning an assembly language model for instruction embedding. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3236–3251. doi:10.1145/3460120.3484587
- [44] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*. PMLR, 3835–3845.
- [45] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated Graph Sequence Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). <http://arxiv.org/abs/1511.05493>
- [46] Xiang Ling, Lingfei Wu, Saizhuo Wang, Tengfei Ma, Fangli Xu, Alex X Liu, Chunming Wu, and Shouling Ji. 2021. Multilevel graph matching networks for deep graph similarity learning. *IEEE Transactions on Neural Networks and Learning Systems* 34, 2 (2021), 799–813. doi:10.1109/TNNLS.2021.3102234
- [47] Zachary C. Lipton, John Berkowitz, and Charles Elkan. 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. arXiv:1506.00019 [cs.LG] <https://arxiv.org/abs/1506.00019>
- [48] Lingyuan Liu and Mengxiang Zhang. 2025. Exp4Fuse: A Rank Fusion Framework for Enhanced Sparse Retrieval using Large Language Model-based Query Expansion. In *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics, 163–173. doi:10.18653/v1/2025.findings-acl.9
- [49] Zhijie Liu, Qiyi Tang, Sen Nie, Shi Wu, Liang Feng Zhang, and Yutian Tang. 2025. KEENHash: Hashing Programs into Function-Aware Embeddings for Large-Scale Binary Code Similarity Analysis. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA036 (June 2025), 24 pages. doi:10.1145/3728911
- [50] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized Explainer for Graph Neural Network. *Advances in Neural Information Processing Systems* 33 (2020).
- [51] Dongsheng Luo, Tianxiang Zhao, Wei Cheng, Dongkuan Xu, Feng Han, Wenchao Yu, Xiao Liu, Haifeng Chen, and Xiang Zhang. 2024. Towards Inductive and Efficient Explanations for Graph Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 8 (2024), 5245–5259. doi:10.1109/TPAMI.2024.3362584
- [52] Lannan Luo, Jiang Ming, Dinghao Wu, Peng Liu, and Sencun Zhu. 2014. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2014). doi:10.1145/2635868.2635900
- [53] Zhenhao Luo, Pengfei Wang, Baosheng Wang, Yong Tang, Wei Xie, Xu Zhou, Danjun Liu, and Kai Lu. 2023. VulHawk: Cross-architecture Vulnerability Detection with Entropy-based Binary Code Search.. In *NDSS*. doi:10.14722/ndss.2023.24415
- [54] Andrea Marcelli, Mariano Graziano, Xabier Ugarte-Pedrero, Yanick Fratantonio, Mohamad Mansouri, and Davide Balzarotti. 2022. How machine learning is solving the binary function similarity problem. In *31st USENIX Security Symposium (USENIX Security 22)*. 2099–2116.
- [55] Luca Massarelli, Giuseppe Antonio Di Luna, Fabio Petroni, Roberto Baldoni, and Leonardo Querzoni. 2019. Safe: Self-attentive function embeddings for binary similarity. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19–20, 2019, Proceedings 16*. Springer, 309–329. doi:10.1007/978-3-030-22038-9_15
- [56] Luca Massarelli, Giuseppe A Di Luna, Fabio Petroni, Leonardo Querzoni, Roberto Baldoni, et al. 2019. Investigating graph embedding neural networks with unsupervised features extraction for binary analysis. In *Proceedings of the 2nd Workshop on Binary Analysis Research (BAR)*. 1–11. doi:10.14722/bar.2019.23020
- [57] Microsoft Research. 2024. Z3 Theorem Prover. <https://github.com/Z3Prover/z3/tree/z3-4.8.7>.
- [58] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *International Conference on Learning Representations*.
- [59] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).
- [60] Reza Mirzazadeh, Mohammad Hossein Moattar, and Majid Vafaei Jahan. 2015. Metamorphic malware detection using linear discriminant analysis and graph similarity. In *2015 5th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 61–66. doi:10.1109/ICCKE.2015.7365862
- [61] National Security Agency. [n. d.]. Ghidra. <https://ghidra-re.org/>.

- [62] Kexin Pei, Zhou Xuan, Junfeng Yang, Suman Jana, and Baishakhi Ray. 2023. Learning Approximate Execution Semantics From Traces for Binary Function Similarity. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2776–2790. doi:10.1109/TSE.2022.3231621
- [63] Zackary Rackauckas. 2024. Rag-Fusion: A New Take on Retrieval Augmented Generation. *International Journal on Natural Language Computing* 13, 1 (Feb. 2024), 37–47. doi:10.5121/ijnlc.2024.13103
- [64] Jingyi Shi, Yufeng Chen, Yang Xiao, Yuekang Li, Zhengzi Xu, Sihao Qiu, Chi Zhang, Keyu Qi, Yeting Li, Xingchu Chen, Yanyan Zou, Yang Liu, and Wei Huo. 2025. A Large Scale Study of AI-based Binary Function Similarity Detection Techniques for Security Researchers and Practitioners. arXiv:2511.01180 [cs.CR] <https://arxiv.org/abs/2511.01180>
- [65] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. 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*. Association for Computational Linguistics, 1556–1566. doi:10.3115/v1/P15-1150
- [66] Hao Wang, Zeyu Gao, Chao Zhang, Zihan Sha, Mingyang Sun, Yuchen Zhou, Wenyu Zhu, Wenju Sun, Han Qiu, and Xi Xiao. 2024. CLAP: Learning Transferable Binary Code Representations with Natural Language Supervision. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 503–515. doi:10.1145/3650212.3652145
- [67] Hao Wang, Zeyu Gao, Chao Zhang, Mingyang Sun, Yuchen Zhou, Han Qiu, and Xi Xiao. 2024. 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*. 149–161. doi:10.1145/3650212.3652117
- [68] Huaijin Wang, Pingchuan Ma, Shuai Wang, Qiyi Tang, Sen Nie, and Shi Wu. 2023. sem2vec: Semantics-aware assembly tracelet embedding. *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–34. doi:10.1145/3569933
- [69] Hao Wang, Wenjie Qu, Gilad Katz, Wenyu Zhu, Zeyu Gao, Han Qiu, Jianwei Zhuge, and Chao Zhang. 2022. jTrans: Jump-Aware Transformer for Binary Code Similarity Detection. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1–13. doi:10.1145/3533767.3534367
- [70] Junzhe Wang, Matthew Sharp, Chuxiong Wu, Qiang Zeng, and Lannan Luo. 2023. Can a Deep Learning Model for One Architecture Be Used for Others? Retargeted-Architecture Binary Code Analysis. In *USENIX Security Symposium*.
- [71] Jialai Wang, Chao Zhang, Longfei Chen, Yi Rong, Yuxiao Wu, Hao Wang, Wende Tan, Qi Li, and Zongpeng Li. 2024. Improving ML-based Binary Function Similarity Detection by Assessing and Deprioritizing Control Flow Graph Features. In *USENIX Security Symposium*.
- [72] Wenhan Wang, Ge Li, Bo Ma, Xin Xia, and Zhi Jin. 2020. Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 261–271. doi:10.1109/SANER48275.2020.9054857
- [73] Yongpan Wang, Hong Li, Xiaojie Zhu, Siyuan Li, Chaopeng Dong, Shouguo Yang, and Kangyuan Qin. 2025. BinEnhance: An Enhancement Framework Based on External Environment Semantics for Binary Code Search. In *Proceedings of the Network and Distributed System Security Symposium*. doi:10.14722/ndss.2025.240568
- [74] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. 2017. Sampling matters in deep embedding learning. In *Proceedings of the IEEE international conference on computer vision*. 2840–2848. doi:10.1109/ICCV.2017.309
- [75] Qitian Wu, Wentao Zhao, Chenxiao Yang, Hengrui Zhang, Fan Nie, Haitian Jiang, Yatao Bian, and Junchi Yan. 2023. SGFormer: Simplifying and Empowering Transformers for Large-Graph Representations. In *Advances in Neural Information Processing Systems (NeurIPS)*. doi:10.52202/075280-2826
- [76] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018). arXiv:1810.00826 [cs.LG] doi:10.48550/arXiv.1810.00826
- [77] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 363–376. doi:10.1145/3133956.3134018
- [78] Xi Xu, Qinghua Zheng, Zheng Yan, Ming Fan, Ang Jia, and Ting Liu. 2021. Interpretation-enabled Software Reuse Detection Based on a Multi-Level Birthmark Model. In *Proceedings of the 43rd International Conference on Software Engineering (Madrid, Spain) (ICSE '21)*. IEEE Press, 873–884. doi:10.1109/ICSE43902.2021.00084
- [79] Xi Xu, Qinghua Zheng, Zheng Yan, Ming Fan, Ang Jia, Zhaohui Zhou, Haijun Wang, and Ting Liu. 2023. PatchDiscovery: Patch Presence Test for Identifying Binary Vulnerabilities Based on Key Basic Blocks. *IEEE Transactions on Software Engineering* 49, 12 (2023), 5279–5294. doi:10.1109/TSE.2023.3332732
- [80] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. 2014. Modeling and Discovering Vulnerabilities with Code Property Graphs. In *2014 IEEE Symposium on Security and Privacy*. 590–604. doi:10.1109/SP.2014.44
- [81] Jia Yang, Cai Fu, Xiao-Yang Liu, Heng Yin, and Pan Zhou. 2021. Codee: A tensor embedding scheme for binary code search. *IEEE Transactions on Software Engineering* 48, 7 (2021), 2224–2244. doi:10.1109/TSE.2021.3056139
- [82] Shouguo Yang, Long Cheng, Yicheng Zeng, Zhe Lang, Hongsong Zhu, and Zhiqiang Shi. 2021. Asteria: Deep learning-based AST-encoding for cross-platform binary code similarity detection. In *2021 51st Annual IEEE/IFIP International*

- Conference on Dependable Systems and Networks (DSN)*. IEEE, 224–236. doi:10.1109/DSN48987.2021.00036
- [83] Shouguo Yang, Chaopeng Dong, Yang Xiao, Yiran Cheng, Zhiqiang Shi, Zhi Li, and Limin Sun. 2023. Asteria-Pro: Enhancing Deep Learning-based Binary Code Similarity Detection by Incorporating Domain Knowledge. *ACM Transactions on Software Engineering and Methodology* 33, 1 (2023), 1–40. doi:10.1145/3604611
- [84] Tengpeng Yang. 2026. Artifacts and Additional Technical Details for Revisiting Graph Representations for ML-Based Binary Code Similarity Detection: A Systematic Study. https://github.com/yangtt57/empirical_graphs_used_in_ML-BCSD
- [85] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNNExplainer: generating explanations for graph neural networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Vol. 32. Curran Associates Inc., Red Hook, NY, USA, Article 829, 12 pages. https://proceedings.neurips.cc/paper_files/paper/2019/hash/d80b7040b773199015de6d3b4293c8ff-Abstract.html
- [86] Zeping Yu, Rui Cao, Qiyi Tang, Sen Nie, Junzhou Huang, and Shi Wu. 2020. Order matters: Semantic-aware neural networks for binary code similarity detection. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 1145–1152. doi:10.1609/aaai.v34i01.5466
- [87] Zeping Yu, Wenxin Zheng, Jiaqi Wang, Qiyi Tang, Sen Nie, and Shi Wu. 2020. Codecmr: Cross-modal retrieval for function-level binary source code matching. *Advances in Neural Information Processing Systems* 33 (2020), 3872–3883.
- [88] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. 2021. On Explainability of Graph Neural Networks via Subgraph Explorations. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 12241–12252. <https://proceedings.mlr.press/v139/yuan21c.html>
- [89] Fangfang Zhang, Dinghao Wu, Peng Liu, and Sencun Zhu. 2014. Program logic based software plagiarism detection. In *2014 IEEE 25th international symposium on software reliability engineering*. IEEE, 66–77. doi:10.1109/ISSRE.2014.18
- [90] Yuntao Zhang, Binxing Fang, Zehui Xiong, Yanhao Wang, Yuwei Liu, Chao Zheng, and Qinnan Zhang. 2024. A Semantics-Based Approach on Binary Function Similarity Detection. *IEEE Internet of Things Journal* 11, 15 (2024), 25910–25924. doi:10.1109/JIoT.2024.3389014
- [91] Yuntao Zhang, Yanhao Wang, Yuwei Liu, Zhengyuan Pang, and Binxing Fang. 2022. PDG2Vec: Identify the Binary Function Similarity with Program Dependence Graph. In *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. 537–548. doi:10.1109/QRS57517.2022.00061
- [92] Li Zhou, Marc Dacier, and Charalambos Konstantinou. 2025. ReGraph: A Tool for Binary Similarity Identification. In *Proceedings of the 34th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Clarion Hotel Trondheim, Trondheim, Norway) (*ISSTA Companion '25*). Association for Computing Machinery, New York, NY, USA, 6–10. doi:10.1145/3713081.3731728
- [93] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. arXiv:1909.03496 [cs.SE] <https://arxiv.org/abs/1909.03496>

Received 2026-01-30; accepted 2026-06-25