Selective Knowledge Distillation: Fusing LLM Semantic Strengths with DNN Efficiency for Binary Code Similarity Detection

Shize Zhou[1], Peiyu Liu[1]*, Lirong Fu[2], Tong Ye[3], Wenhai Wang[1]*
[1]Zhejiang University, China
[2]Hangzhou Dianzi University, China
[3]vivo AI Lab, China
{shizezhou, liupeiyu, zdzzlab}@zju.edu.cn, tongye@vivo.com, flr@hdu.edu.cn

Abstract

Binary Code Similarity Detection (BCSD) plays a vital role in various security applications, including vulnerability identification, malware analysis, and code plagiarism detection. With the growing adoption of deep neural networks (DNNs), substantial progress has been made in recognizing and classifying similar code segments. However, DNN-based BCSD methods often exhibit low accuracy and robustness because they struggle to capture fine-grained and high-level program semantics. In contrast, such semantics are typically captured through natural language interpretations of source code by large language models (LLMs). Yet, LLM-based BCSD methods are constrained by their large model sizes and high inference latency. To alleviate these limitations, this paper proposes BinSKD. The key idea is to leverage an LLM-based BCSD method as the teacher model and transfer its knowledge of high-level program semantics to various DNN-based student models. Specifically, to avoid propagating errors from the teacher to the student, we introduce selective distillation, selecting targets with accurate semantics according to their detection retrieval. In addition, to mitigate the noise introduced by a number of negative samples during distillation, we further propose discrepancy-weighted sampling to focus on the samples where the student's prediction notably deviates from the teacher's. Our experiments show that BinSKD yields Recall@1 improvements of 14.5%–91.2% for DNN-based BCSD methods and enables HermesSim to match the teacher's performance with orders-of-magnitude efficiency.

1 Introduction

Binary Code Similarity Detection (BCSD) is a fundamental technique in software development and maintenance analysis, enabling binary code comparison when source code is unavailable (Gao et al., 2024). It underpins a wide range of applications, including bug search (Vadayath et al., 2022; Yang et al., 2023b; Luo et al., 2023), patch analysis (Zhan et al., 2024; Wu et al., 2025), and malware detection (Xu et al., 2023b; See et al., 2025). For example, by utilizing BCSD, developers can quickly identify and track vulnerable code introduced through code reuse, enabling more effective assessment of vulnerability impact (Dong et al., 2024; Zhou et al., 2025). Besides, BCSD accelerates security patch validation via pre- and post-patch binary comparison (Wu et al., 2025). Beyond these applications, BCSD is also widely used for code cloning detection and software plagiarism identification (Qasem et al., 2023; Fu et al., 2022).

With the rapid development of DNNs, BCSD techniques have made remarkable progress and are increasingly adopted in practice. Despite this progress, DNN-based BCSD methods still suffer from notable limitations, such as low discriminability and robustness (Fu et al., 2024). This is because they usually rely on syntax-centric features, such as lexical patterns (Massarelli et al., 2019; Wang et al., 2022), control-flow structures (Gao et al., 2018; He et al., 2024), or data-flow information (Guo et al., 2022), which are inherently too coarse-grained to represent program functionality (Wang et al., 2023).

More recently, LLM-based methods have emerged as the SOTA in BCSD, as they can effectively capture high-level semantics such as function purpose and logic of binary code by leveraging natural language supervision (Wang et al., 2024a; Zhang et al., 2025). Unlike low-level lexical or structural features, these high-level semantics remain stable even when the underlying binary code changes across different optimization levels. However, LLM-based BCSD methods involve model parameters two orders of magnitude larger than those of typical DNN-based BCSD methods, which in turn demand higher inference latency. Conse-

*Corresponding authors.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 26002–26014

quently, their deployment is constrained in large-scale binary analysis pipelines.

This motivates us to explore how to combine the strengths of both methods, leveraging the high-level semantics captured by LLMs while retaining the efficiency and scalability of DNN-based models. Potential methods, such as concatenating embeddings from different models or aligning them through contrastive learning, cannot work effectively when there are substantial disparities between the embedding spaces, limiting the generality of these methods. Therefore, a promising method to address this issue is relational knowledge distillation, which focuses on learning the relationships between samples in semantic space rather than individual embeddings. Compared to the existing knowledge distillation methods (Park et al., 2019; Hu et al., 2025), BinSKD presents a novel selective distillation in a relational framework, addressing two key challenges in BCSD.

Challenge 1: Knowledge transferred from the teacher model is not always reliable. The teacher model does not consistently outperform the student model across all samples, and transferring knowledge from such cases can degrade the effectiveness of distillation. From 3,000 manually inspected queries, the teacher model fails to rank true positives higher than the student model in 47.3% of cases. Among these, 10.4% are even worse. To address this challenge, we propose a selective distillation strategy. In particular, BinSKD selectively distills reliable relational knowledge from the teacher. A relation is deemed reliable if, for a given query function, the teacher model assigns higher ranks to candidate functions compiled from the same source code than the student model does. Through this strategy, we prevent the propagation of errors and ensure that accurate semantic knowledge is transferred to the student model.

Challenge 2: The imbalance between positive and negative samples introduces substantial noise into the distillation process. In BCSD scenarios, positive samples compiled from homologous source codes is far fewer than negative samples from different source codes (Wang et al., 2024b). Regularization methods such as focal loss can weight "hard" samples, but they rely on the sample's own prediction confidence and fail to account for samples where the student's prediction deviates from the teacher's. To address this challenge, we propose a discrepancy–weighted sampling strategy. In this strategy, a sample is assigned a higher weight if, for a given query function, its reciprocal rank under the student model deviates more from that under the teacher. This strategy allows us to prioritize samples where the student model benefit most from the teacher guidance.

In summary, our contributions are as follows[1]:

- Conceptual Level: We advocate a new focus on universally enhancing various DNN-based BCSD methods by leveraging the semantic strength of LLMs, while preserving lightweight nature and low inference time.
- Technical Level: We propose the first knowledge distillation framework tailored for the BCSD domain. In particular, we design a novel selective distillation and discrepancy-weighted sampling strategy to transfer accurate semantic knowledge and implement a targeted distillation process.
- Empirical Level: Our evaluation demonstrates that BinSKD is both general and effective in enhancing DNN-based BCSD methods. It improves Recall@1 by 14.5%–91.2% over the methods with different model architectures. Furthermore, when BinSKD is applied to the SOTA DNN-based BCSD method, it achieves accuracy comparable to the LLM-based teacher method while achieving 107× faster inference and 290× fewer parameters.

2 Motivation

DNN-based BCSD methods typically extract various types of features from binary code, including lexical-, control structure-, and data flow-level features. These features are then processed by graph neural networks (GNNs) (He et al., 2024), convolutional neural networks (CNNs) (Yu et al., 2020), recurrent neural networks (RNNs) (Massarelli et al., 2019), or Transformer-based models (Wang et al., 2022) to generate embeddings. However, prior studies have shown that low-level features are highly sensitive to compilation settings, such as optimization levels. This sensitivity leads to unstable performance and degraded accuracy (Guo et al., 2022; Wang et al., 2025). As a concrete example in Appendix A, functions compiled under O0 and O2 exhibit substantial differences in their low-level features. As a result, the DNN-based BCSD method HermesSim fails to correctly match them, producing a false negative.

[1] https://github.com/todoni123/BinSKD

Although LLM-based BCSD methods are capable of capturing high-level program semantics, their main limitation lies in the substantial cost of inference. These models (Wang et al., 2024a; Zhang et al., 2025) typically contain hundreds of millions of parameters—two orders of magnitude larger than DNN-based models. Consequently, this makes LLM-based BCSD methods restrictive in large-scale binary analysis. For instance, once a new vulnerability is discovered in the upstream code, we need to check millions of downstream functions, a process that could take days. Such a time expenditure is unsustainable in large-scale binary analysis scenarios (Liu et al., 2025).

3 Methodology

3.1 Overview

To fuse the LLM semantic strengths with the efficiency of DNNs, we propose a knowledge distillation framework, BinSKD (as shown in Fig. 1). Specifically, the LLM-based BCSD method is employed as a teacher model, and the DNN-based BCSD method serves as a student. Each model generates its own binary function embeddings. To transfer high-level semantic knowledge from the teacher to the student, we leverage relational representations of these binary code embeddings. The knowledge distillation process works by progressively minimizing the distillation loss, guiding the student model towards the relational topological structure of the teacher and enabling it to gradually learn the semantics contained in these relations.

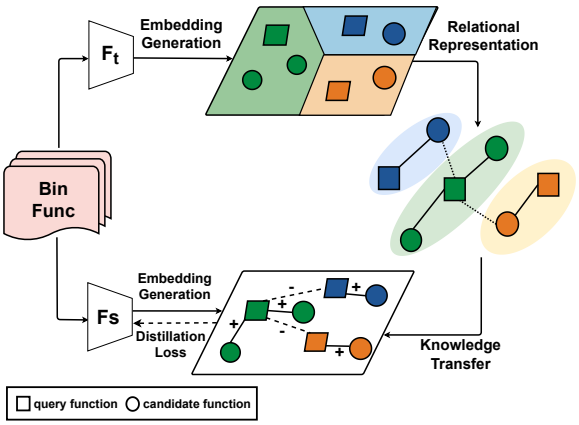

Figure 1: The pipeline of BinSKD. The elements with the same color represent the embeddings of positive samples compiled from the same source code.

3.2 Design

Fig.2 illustrates the design of BinSKD. Given a batch of binary functions, both the heterogeneous teacher F_t and student F_s process the inputs in parallel to produce their respective embeddings. The teacher embeddings are obtained offline from a pre-trained LLM-based BCSD method, while the student embeddings are generated by a DNN-based BCSD method. Next, each model computes pairwise similarities based on its own embeddings to construct a relation matrix, which is subsequently normalized to ensure consistent comparison across models. After normalization, we apply selective distillation to emphasize regions of the matrix where the teacher achieves superior retrieval performance. Furthermore, to achieve a targeted distillation and suppress a number of noisy negatives, discrepancy–weighted sampling assigns higher weights to samples exhibiting larger ranking discrepancies between the student and the teacher. The distillation loss is then computed by applying Huber loss (Meyer, 2021) to the weighted matrices and averaging over all pairwise samples. Finally, the distillation loss is integrated with the student's original task loss to jointly optimize performance.

3.2.1 Embedding Generation

In the embedding generation stage, BinSKD follows the preprocessing steps of the underlying BCSD methods. The cosine similarity between embeddings and their corresponding hard labels is used to compute a task-specific loss L_{task}, ensuring that the student model preserves its performance on the original task while participating in knowledge distillation (Line 4 in Algorithm 1).

3.2.2 Relational Representation

Since BCSD focuses on similarity relationships between functions rather than individual function embeddings, we represent knowledge using pairwise cosine similarities between function embeddings, resulting in relational matrices for both the teacher and the student. From a technical perspective, representing knowledge through relational structures alleviates discrepancies arising from inconsistent semantic spaces across different models. Furthermore, to focus on the relative distances among the matrices, we apply z-score normalization to standardize the relational matrices (Line 7).

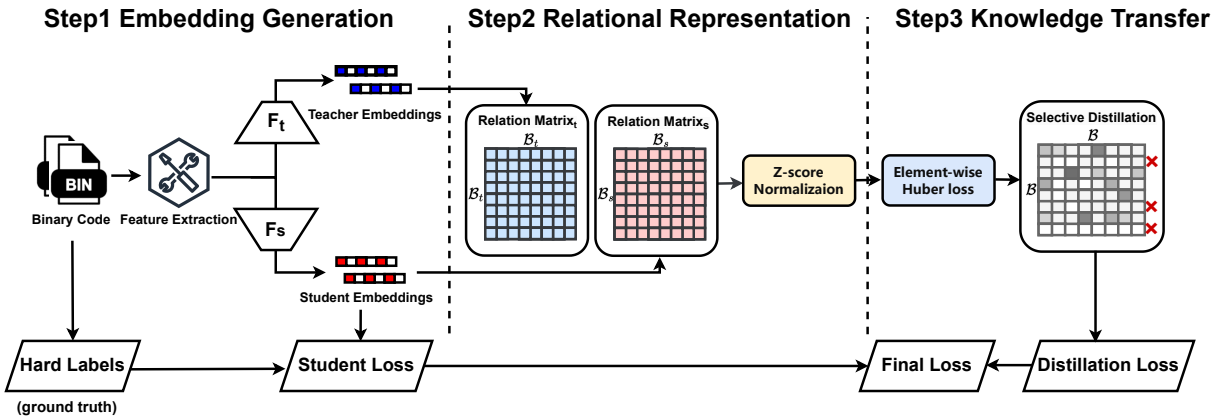

Figure 2: Design of BinSKD. Step 1: Generate embeddings using the teacher and student models, and compute the student's original task loss. Step 2: Construct and normalize the relation matrix. Step 3: Exclude the rows where the teacher model performs worse than the student (indicated by red crosses). For the remaining rows, assign weights to different samples (indicated by the color intensity). Finally, calculate the distillation loss and combine it with the student loss to compute the final loss.

3.2.3 Knowledge Transfer

To better transfer the teacher model's knowledge to the student model, we propose two strategies in this step: (1) Selective Distillation and (2) Discrepancy-weighted Sampling.

(1) Selective Distillation. To avoid propagating errors from the teacher to the student, BinSKD selectively distills reliable relational knowledge. We quantify the reliability of the teacher's knowledge by measuring the retrieval quality of query results. Following a widely adopted practice in this domain (Marcelli et al., 2022), we use mean reciprocal rank (MRR) as the metric, where higher value indicate that positive samples are ranked closer to the top, reflecting stronger semantic discrimination.

Each row i of the relation matrix corresponds to a query function f_i. For each row, we compute the $MRR^{(i)}$ of its positive samples for both teacher (T) and student (S) matrices:

$$\mathrm{MRR}^{(i)} = \frac{1}{|P_i|} \sum_{p \in P_i} \frac{1}{\mathrm{rank}^{(i)}(p)}, \tag{1}$$

where P_i is the set of positive samples for query f_i, and $\mathrm{rank}^{(i)}(p)$ denotes the rank of sample p among all candidates in row i.

To ensure reliable knowledge transfer, we restrict distillation to queries where the teacher outperforms the student in terms of retrieval. This selection criterion can be formally expressed as

$$\mathrm{MRR}_T^{(i)} > \mathrm{MRR}_S^{(i)}, \tag{2}$$

only rows that satisfy this condition are selected for knowledge distillation (Line 10–12).

(2) Discrepancy-weighted Sampling. In BCSD scenario, positive samples compiled from homologous source codes is far fewer than that of negative samples from different source codes. To mitigate the impact of noisy negatives, we introduce a discrepancy-weighted sampling strategy (Line 13-15). Since higher-ranked negative samples in the student model are more likely to cause false positives, we apply an inverse ranking design to increase the contribution of top-ranked samples. Each element (i, j) in matrix is assigned a weight based on the difference between its teacher and student inverse rankings in similarity detection order:

$$w_{ij} = \begin{cases} \left| \dfrac{1}{\mathrm{rank}_T^{(i)}(j)} - \dfrac{1}{\mathrm{rank}_S^{(i)}(j)} \right|, & \text{if } j \notin P_i, \\ 1, & \text{if } j \in P_i. \end{cases} \tag{3}$$

where $\mathrm{rank}^{(i)}(j)$ denotes the rank of sample j among all candidates in row i.

To make the training process more stable, we rescale each row by computing the weighted average of its elements to obtain $\tilde{w}_{ij}$ (Line 16-17). Finally, we calculate the distance-wise distillation loss L_{KD} as follows:

$$L_{\mathrm{KD}} = \frac{1}{|\mathcal{B}|} \sum_{\substack{(i,j) \in \mathcal{B} \\ \mathrm{MRR}_T^{(i)} > \mathrm{MRR}_S^{(i)}}} \tilde{w}_{ij} \cdot l_\delta\Big(\psi(t_i, t_j), \psi(s_i, s_j)\Big), \tag{4}$$

where t_i, t_j and s_i, s_j denote the embeddings produced by the teacher and student models, ψ denotes the normalized cosine similarity between the embeddings, and l_δ represents the Huber loss.

Algorithm 1: BinSKD

Input : Binary code R_b, Hard labels $labels$

Output : Final loss L

STEP 1. Embedding Generation:

$E_t \leftarrow F_t(R_b)$ `// Teacher embeddings`

$E_s \leftarrow F_s(R_b)$ `// Student embeddings`

$L_{\text{task}} \leftarrow \text{ComputeTaskLoss}(E_s, labels)$

STEP 2. Relational Representation:

$(M_t, M_s) \leftarrow \text{pairwise\_cosine\_similarity}(E_t, E_s)$

$(N_t, N_s) \leftarrow \text{z\_score\_normalize}(M_t, M_s)$

STEP 3. Knowledge Transfer:

(1) Selective Distillation:

foreach *row* i *in* (M_t, M_s) **do**

$(\text{MRR}_T^{(i)}, \text{MRR}_S^{(i)}) \leftarrow \text{ComputeMRR}(M_t[i], M_s[i])$

$$W_{\text{mask}}^{(i)} = \begin{cases} 1, & \text{MRR}_T^{(i)} > \text{MRR}_S^{(i)}, \\ 0, & \text{otherwise.} \end{cases}$$

(2) Discrepancy-weighted Sampling:

foreach *element* (i, j) **do**

$$w_{ij} = \begin{cases} 1, & \text{if } j \in P_i, \\ \left| \frac{1}{\text{rank}_T^{(i)}(j)} - \frac{1}{\text{rank}_S^{(i)}(j)} \right|, & \text{if } j \notin P_i. \end{cases}$$

`//` P_i `is the positive samples for query` i

foreach *row* i **do**

$\tilde{w}_{ij} \leftarrow w_{ij} / \sum_j w_{ij}$

$L_{\text{N}} \leftarrow \text{Huber\_loss}(N_t, N_s)$

$L_{\text{KD}} \leftarrow \text{mean}(L_{\text{N}} \odot W_{\text{mask}} \odot \tilde{w})$

$L \leftarrow L_{\text{task}} + \lambda_{\text{KD}} \cdot L_{\text{KD}}$

During training, to jointly optimize task performance and maintain consistency with the student's original objective, we combine the distillation loss with task-specific loss (e.g., *triplet_loss*). Therefore, the overall objective is formulated as:

$$L = L_{\text{task}} + \lambda_{\text{KD}} \cdot L_{\text{KD}}, \tag{5}$$

where L_{task} is the task-specific loss for the student model, L_{KD} is the knowledge distillation loss, and λ_{KD} is a tunable hyperparameter that balances the two terms.

4 Evaluation

To comprehensively evaluate the effectiveness of BinSKD, we design a series of experiments from multiple perspectives. To assess the general enhancements delivered by BinSKD across a range of cutting-edge DNN-based BCSD methods, we evaluate its performance on similar function detection (§4.1). We also conduct ablation studies to verify the contributions of our tailored design (§4.2). Furthermore, to demonstrate the efficiency of BinSKD, we measure its time cost, including both training and inference overhead (§4.3). To investigate whether BinSKD effectively transfers high-level program semantic knowledge, we provide representation visualizations (§4.4). Finally, to validate its robustness in real-world scenarios, we evaluate BinSKD on a downstream vulnerable function search task (§4.5).

Implementation Details. To evaluate the generality of BinSKD across different DNN architectures, the student cover a diverse set of DNN-based models, including GNN-, RNN-, CNN-, and Transformer-based models. For the teacher model, we select the LLM-based BCSD method CLAP (Wang et al., 2024a), as it is one of the most representative LLM-based methods, and its publicly available training datasets allow us to ensure that evaluation is free from potential data leakage. Finally, to ensure optimal performance of BinSKD, the hyperparameter λ_{KD} is selected via grid search. For detailed descriptions of the datasets and further explanations of the baseline methods, please refer to Appendix B.

4.1 Similar Function Detection

In real-world binary code search tasks, the size of the function pool is often much larger than the pool size of the experiments (Wang et al., 2025). To evaluate the impact of increasing pool size on BinSKD, we set the pool sizes to 2, 16, 32, 128, 512, 1024, 2048, 4096, and 8192. As shown in Fig. 3, the Recall@1 of all baselines decline notably as the pool size increases, reflecting the growing difficulty of similar function detection in larger candidate sets. Encouragingly, BinSKD demonstrates stable performance, with its improvements becoming more pronounced as the pool size grows. Specifically, HermesSim, SAFE, jTrans, and BinaryAI achieve Recall@1 improvements of 14.5%, 39.3%, 40.7%, and 91.2%, when the pool size reaches 8192. Notably, after being enhanced by BinSKD, HermesSim achieves a Recall@1 of 0.750, which even surpasses that of the teacher model CLAP 0.744.

To further evaluate the effectiveness of BinSKD across settings with varying levels of difficulty, we perform evaluations under six different combinations of optimization levels (as shown in Table 1). Overall, BinSKD consistently improves DNN-based BCSD methods across all settings. For example, in the harder setting (e.g., O0 vs. O3),

Table 1: Comparison of different methods under per-pair optimization setting. (Pool size = 8192)

Method	MRR						Recall@1					
	O0,O1	**O0,O2**	**O0,O3**	**O1,O2**	**O1,O3**	**O2,O3**	**O0,O1**	**O0,O2**	**O0,O3**	**O1,O2**	**O1,O3**	**O2,O3**
HermesSim	0.676	0.648	0.559	0.880	0.813	0.810	0.600	0.579	0.489	0.841	0.771	0.767
HermesSim + BinSKD	**0.817**	**0.784**	**0.691**	**0.892**	**0.819**	**0.815**	**0.761**	**0.733**	**0.633**	**0.859**	**0.776**	**0.771**
SAFE	0.099	0.067	0.061	0.521	0.396	0.679	0.052	0.028	0.025	0.442	0.324	0.653
SAFE + BinSKD	**0.185**	**0.147**	**0.121**	**0.734**	**0.584**	**0.814**	**0.104**	**0.081**	**0.066**	**0.672**	**0.525**	**0.787**
jTrans	0.102	0.087	0.081	0.753	0.661	0.812	0.043	0.039	0.031	0.678	0.592	0.786
jTrans + BinSKD	**0.494**	**0.414**	**0.360**	**0.810**	**0.699**	**0.822**	**0.302**	**0.292**	**0.243**	**0.746**	**0.624**	**0.794**
BinaryAI	0.036	0.031	0.028	0.194	0.141	0.543	0.015	0.014	0.012	0.120	0.086	0.498
BinaryAI + BinSKD	**0.088**	**0.071**	**0.061**	**0.491**	**0.342**	**0.629**	**0.045**	**0.033**	**0.024**	**0.399**	**0.272**	**0.597**
CLAP	0.805	0.777	0.688	0.872	0.776	0.867	0.748	0.722	0.627	0.843	0.730	0.838

HermesSim's MRR and Recall@1 improved by 23.6% and 29.4%, respectively. In the easier setting (e.g., O2 vs. O3), the improvements were smaller, with MRR and Recall@1 increasing by 0.617% and 0.522%, as the baseline metrics were already high. These results demonstrate that BinSKD remains robustness, especially in hard settings.

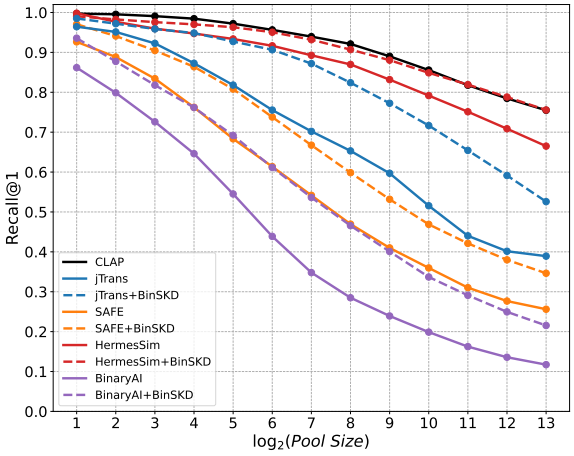

Figure 3: Recall@1 on different pool sizes under the cross-optimization (XO) setting. (Poolsize = 8192)

4.2 Ablation Study

To isolate the effect of BinSKD, we first compare against regularization strategy (Lin et al., 2017). The results show that regularization strategy has a limited effect, with only a 2.8% improvement in MRR, which is much smaller than 11.0% achieved with BinSKD (as shown in Table 2). Furthermore, to fairly demonstrate the contribution of each component to BinSKD, we conducted ablation experiments on the individual components under comparable distillation-loss scaling. First, applying relation knowledge distillation (RKD) (Park et al., 2019) to HermesSim results in an MRR of only 0.069, significantly lower than its baseline performance of 0.716. This disparity underscores the limitations of directly applying RKD to the BCSD domain, where incorrect teacher guidance can disrupt the student model's well-learned parameters, leading to performance degradation. In contrast, selective distillation plays a critical role in enhancing model performance. Specifically, after incorporating selective distillation, the MRR increases to 0.707, while discrepancy-weighted sampling improves it to 0.301. However, these two components are indispensable, only when they work together can BinSKD effectively transfer valuable semantic knowledge, further boosting the MRR to 0.795.

Table 2: Ablation study with different strategies. RS = Regularization Strategy (focal loss), DwS = Discrepancy-weighted Sampling, SD = Selective Distillation. (Pool size = 8192)

Method	MRR	Recall@1	Recall@5	Recall@10
HermesSim	0.716	0.655	0.783	0.836
HermesSim + RS	0.736	0.683	0.803	0.847
HermesSim + BinSKD	**0.795**	**0.750**	**0.847**	**0.882**
HermesSim + RKD	0.069	0.055	0.080	0.095
HermesSim + RKD + DwS	0.301	0.242	0.361	0.436
HermesSim + RKD + SD	0.707	0.634	0.735	0.796

Furthermore, to enable a controlled and quantitative evaluation of the relationship between teacher quality and student improvement, we deliberately degraded CLAP by introducing Gaussian Noise (Morris et al., 2023; Chen et al., 2024), rather than using other models. This controlled setting allows us to directly examine whether the student can selectively benefit from reliable teacher signals while remaining robust to noisy or misleading supervision. Given a teacher embedding t_x, we generate a noisy version as

$$t_x^{\text{noisy}} = t_x + \alpha \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1), \tag{6}$$

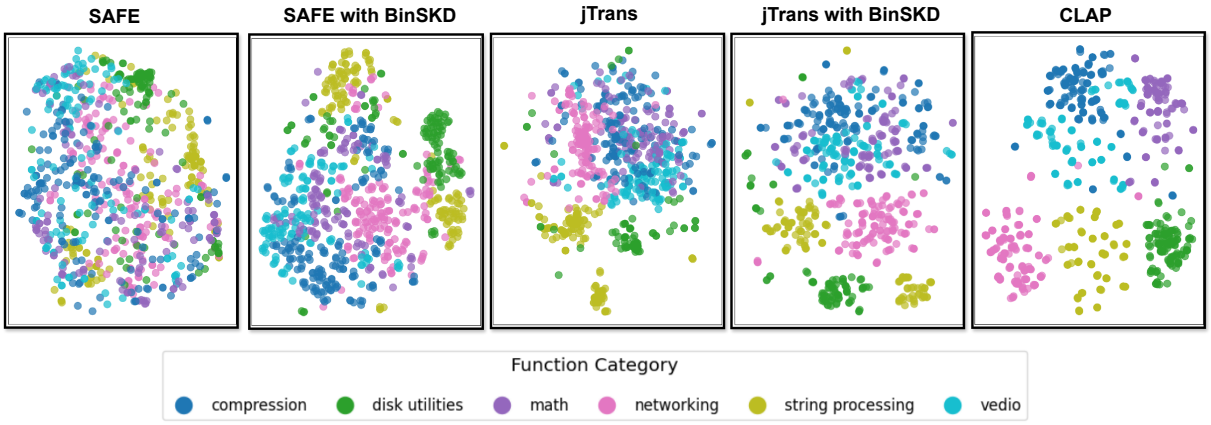

Figure 4: 2D t-SNE visualization of embedding vectors from different methods.

where higher α values correspond to greater degradation of the teacher model. As shown in Table 3, we found that the MRR of HermesSim with weaken teachers decreased by 5.53%, 8.43%, and 10.7%, as the α increased. The positive correlation confirms that the improvement of student model is attributed to semantic knowledge transfer from the teacher.

Table 3: Impact of teacher quality on student model performance. GN = Gaussian Noise. (Pool size = 8192)

Student	Teacher	MRR	Recall@1	Recall@5	Recall@10
HermesSim	CLAP	0.795	0.750	0.847	0.882
	CLAP + GN ($\alpha = 0.05$)	0.751	0.697	0.821	0.867
	CLAP + GN ($\alpha = 0.1$)	0.728	0.672	0.803	0.850
	CLAP + GN ($\alpha = 0.2$)	0.710	0.648	0.793	0.841

4.3 Time Efficiency

To demonstrate the efficiency of BinSKD, we selected HermesSim as the representative model, as it achieves the best performance among all baselines and attains accuracy comparable to the teacher model after enhancement. Table 4 reports the training time per epoch. The knowledge distillation introduced by BinSKD amounting to merely 1/3 of the student training cost.

Table 4: Training time per epoch (batch size = 80) across different stages.

Method	Student Training	Knowledge Distillation
HermesSim	144s	–
HermesSim+BinSKD	144s	51.6s

For inference, Table 5 shows that BinSKD enhances HermesSim while remaining highly lightweight, representing 290× fewer parameters and 107× faster inference than CLAP. Since BinSKD introduces no additional network layers and does not rely on auxiliary networks with independent parameters, it consistently improves the accuracy of DNN-based methods while preserving their lightweight nature and low inference time.

Table 5: Inference time for 1,000,000 functions and model parameter counts. h = hours, s = seconds.

Method	Inferring Time	Parameters
CLAP	5.51h	112M
HermesSim	185s	388K
HermesSim+BinSKD	185s	388K

4.4 Representation Visualization

To investigate whether BinSKD successfully transfers high-level program semantic knowledge to DNN-based BCSD methods, we perform intrinsic evaluations to analyze how the assembly code is represented. For this experiment, we consider six distinct functionality categories and manually select 1,000 binary functions from real-world software that represent each of these functionalities. To visualize the results, we employ the t-SNE (van der Maaten and Hinton, 2008) algorithm to map high-dimensional function embeddings onto a two-dimensional space. By effectively transforming complex data into a more interpretable form, t-SNE provides an intuitive illustration of the relationships among functions and thus reflects the quality of the embeddings.

As shown in Fig. 4, the visualizations reveal that the embeddings produced by jTrans and SAFE display a highly disordered pattern. This trend is consistent across both methods, indicating that their embeddings lack high-level program semantics. On

Table 6: Result of different methods on vulnerability search (Δ denotes improvement after BinSKD).

CVE	Function	SAFE		HermesSim		jTrans		BinaryAI	
		ΔMRR	ΔBugs	ΔMRR	ΔBugs	ΔMRR	ΔBugs	ΔMRR	ΔBugs
2025-21856	device_add	0.039→0.062	2→3	0.088→0.122	8→10	0.116→0.148	4→14	0.050→0.156	3→15
2024-1086	nf_hook_slow	0.085→0.139	4→11	0.151→0.158	14→15	0.150→0.150	11→11	0.018→0.016	1→6
2024-56631	sg_release	0.104→0.141	4→8	0.089→0.114	5→7	0.125→0.157	10→15	0.043→0.091	1→3
2022-3565	del_timer	0.046→0.067	1→7	0.121→0.129	5→8	0.097→0.149	5→12	0.001→0.022	0→1
2017-16939	xfrm_user_rcv_msg	0.125→0.139	8→12	0.159→0.159	15→15	0.137→0.153	13→15	0.034→0.098	1→5
2017-18270	keyring_search_aux	0.017→0.170	2→8	0.177→0.191	12→15	0.140→0.141	11→11	0.001→0.033	0→5
2016-5195	madvise	0.081→0.145	5→12	0.156→0.154	14→13	0.130→0.159	9→15	0.006→0.065	2→5
2014-4014	generic_permission	0.082→0.085	4→9	0.133→0.136	13→15	0.058→0.083	2→5	0.007→0.081	2→8
2014-9644	crypto_lookup_template	0.120→0.168	11→11	0.030→0.156	4→15	0.056→0.064	9→12	0.085→0.091	3→5
2014-9870	copy_thread	0.114→0.044	5→9	0.145→0.152	12→14	0.136→0.159	11→15	0.034→0.072	1→5

The numbers shown before and after the "→" indicate the metrics before and after applying BinSKD.

the other hand, when BinSKD is applied to transfer semantic knowledge from CLAP to the DNN-based BCSD methods, the outputs of the enhanced methods align more closely within the same vector space based on their functionality, resulting in similar distribution patterns. This observation highlights the role of BinSKD in transferring high-level program semantic knowledge to DNN-based BCSD methods, thereby universally enhancing their performance.

4.5 Vulnerability Search

To examine the robustness and practical effectiveness of BinSKD, we evaluate its impact on real-world vulnerability search performance. To facilitate a realistic and comprehensive evaluation, we selected 16 vmlinux binaries across different versions and compilation settings as an application dataset. From these, we focus on 10 highly exploitable vulnerable functions. These include the critical CVE-2024-1086, which allows remote attackers to bypass network packet filtering via the *nf_hook_slow* function, the notorious CVE-2016-5195, known as the "Dirty COW" vulnerability, which enables local privilege escalation through a race condition in the madvise system call, and CVE-2022-3565, which can cause system instability or denial-of-service due to improper timer handling in the *del_timer* function.

It's important to note that the purpose of our experiments is not to conduct large-scale vulnerability search, but rather to evaluate the effectiveness of BinSKD in enhancing downstream tasks in real-world scenarios. For each query function, we identify the ground truth by performing similarity searches and manually examining the top 50 retrievals from all methods, in line with the common practice in the field (Wang et al., 2024b). As shown in Table 6, we report the number of vulnerable functions within the top 50 retrievals and their corresponding MRR. The results indicate that BinSKD markedly enhances both the number of identified vulnerable functions and the ranking of function matches. This improvement dramatically reduces the manual effort required for vulnerability analysis. For example, in the case of CVE-2016-5195, SAFE initially identifies only 5 vulnerable functions that correctly pinpointed the vulnerability. However, after applying BinSKD, 12 vulnerable functions are successfully identified. This demonstrates BinSKD's ability to substantially improve the vulnerability search performance of various DNN-based BCSD methods.

5 Related Work

In the field of BCSD, numerous studies have focused on improving model performance (Yang et al., 2023a; Xu et al., 2023a; Wang et al., 2023, 2025). These methods can be categorized into two major types: (1) whether they are general-purpose methods, and (2) whether they involve extra time overhead after the embedding generation phase. We have summarized the characteristics of these methods in Table 7.

Specialized vs. General Methods. Some works are tailored to specific DNN-based BCSD models or architectures. For example, Asteria-Pro extends Asteria (Yang et al., 2021) by leveraging specific TreeLSTM-based representations, while DIEMPH focuses on Transformer-based methods. In contrast, general-purpose methods aim to improve a wide range of BCSD models. BinUSE employs under-

constrained symbolic execution to eliminate false positives during equivalence checking, and BinEnhance constructs an enhanced embedding graph using inter-function relations to augment original baseline embeddings.

Extra vs. Non-extra Overhead Methods. Some methods improve BCSD performance at the cost of additional time overhead, either by enhancing original generated embeddings or by reranking candidate functions during retrieval. For instance, BinEnhance introduces extra computation to transform original embeddings into improved ones. BinUSE incurs overhead from symbolic execution, and Asteria-Pro leverages call relationships between query and candidate functions. Both methods reorder the results through additional overhead. In contrast, non-extra overhead methods focus on improving the intrinsic capability of the model itself. DIEMPH enhances data quality to better suit Transformer-based methods, while BinSKD improves model performance via selective knowledge distillation.

Overall, BinSKD represents a fundamentally different enhancement paradigm. Unlike existing methods, which add post-hoc reranking or embedding refinement, BinSKD directly enhances the model, avoiding extra inference and reranking costs. BinSKD is complementary and can be combined with these enhancement methods.

Table 7: Comparison of BCSD enhancement methods.

Method	General	Non-extra overhead
Asteria-Pro (Yang et al., 2023a)	✗	✗
DIEMPH (Xu et al., 2023a)	✗	✓
BinUSE (Wang et al., 2023)	✓	✗
BinEnhance (Wang et al., 2025)	✓	✗
BinSKD	✓	✓

6 Conclusion

In this paper, we point out a key limitation of existing DNN-based BCSD methods: the insufficient capability to capture high-level program semantics. Although LLM-based BCSD methods can address this issue, they are hindered by the high number of model parameters and substantial inference costs, making them less practical in large-scale binary analysis. To overcome these limitations, this paper proposes BinSKD, a selective knowledge distillation method. BinSKD effectively transfers semantic knowledge from LLM-based BCSD methods to enhance the accuracy of DNN-based BCSD methods, while maintaining their efficiency. Comprehensive experiments show that BinSKD consistently improves the performance of various cutting-edge DNN-based BCSD methods. These results demonstrate that our method makes BCSD more effective in real-world tasks such as vulnerability search. We believe our work offers a practical direction for advancing BCSD research.

Limitations

Our study focuses on the cross-optimization scenario because the teacher model we employed supports only the x64 architecture. If the teacher model were capable of handling cross-architecture settings, BinSKD would naturally transfer improvements to those scenarios as well. In short, BinSKD is not designed for any specific scenario but instead provides a general enhancement framework.

A future direction to overcome the limitation of scenario is to incorporate multiple teachers trained on diverse compiler architectures and feature representations. Since different teacher models capture complementary aspects of feature learning, dynamically weighting their contributions according to their respective strength could provide richer and more reliable knowledge.

Ethical Considerations

BCSD techniques can be applied for both defensive and offensive purposes. Although they may potentially be misused, for instance to facilitate vulnerability exploitation, this work is conducted with the objective of advancing scientific research. We believe that exposing vulnerabilities is a fundamental step for software security. Accordingly, we introduce a new BCSD enhancement technique to improve the vulnerability detection performance.

Acknowledgments

This work was partly supported by NSFC under No. 62502133 and No. 62302443, the Fellowship of China National Postdoctoral Program for Innovative Talents (BX20230307), the Fundamental Research Funds for the Central Universities+2025ZFJH02.

References

Yiyi Chen, Heather Lent, and Johannes Bjerva. 2024. Text embedding inversion security for multilingual language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7808–7827, Bangkok, Thailand. Association for Computational Linguistics.

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. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 791–802.

Lirong Fu, Shouling Ji, Changchang Liu, Peiyu Liu, Fuzheng Duan, Zonghui Wang, Whenzhi Chen, and Ting Wang. 2022. Focus: Function clone identification on cross-platform. *International Journal of Intelligent Systems*, 37(8):5082–5112.

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. *Preprint*, arXiv:2410.07537.

Jian Gao, Xin Yang, Ying Fu, Yu Jiang, and Jiaguang Sun. 2018. VulSeeker: A semantic learning based vulnerability seeker for cross-platform binary. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 896–899. ISSN: 2643-1572.

Zeyu Gao, Hao Wang, Yuanda Wang, and Chao Zhang. 2024. Virtual compiler is all you need for assembly code search. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3040–3051, Bangkok, Thailand. Association for Computational Linguistics.

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*, ICPC '22, page 366–377, New York, NY, USA. Association for Computing Machinery.

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)*, pages 1759–1776, Philadelphia, PA. USENIX Association.

Shengxiang Hu, Guobing Zou, Song Yang, Shiyi Lin, Yanglan Gan, Bofeng Zhang, and Yixin Chen. 2025. Large language model meets graph neural network in knowledge distillation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(16):17295–17304.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007.

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).

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*.

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)*, pages 2099–2116, Boston, MA. USENIX Association.

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*, pages 309–329. Springer.

Gregory P. Meyer. 2021. An alternative probabilistic interpretation of the huber loss. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5257–5265.

John Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander Rush. 2023. Text embeddings reveal (almost) as much as text. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12448–12460, Singapore. Association for Computational Linguistics.

Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. 2019. Relational knowledge distillation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3962–3971.

Abdullah Qasem, Mourad Debbabi, Bernard Lebel, and Marthe Kassouf. 2023. Binary function clone search in the presence of code obfuscation and optimization over multi-cpu architectures. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '23, page 443–456, New York, NY, USA. Association for Computing Machinery.

August See, Moritz Mönnich, and Mathias Fischer. 2025. Enhancing binary code similarity analysis for software updates: A contextual diffing framework. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '25, page 1724–1740, New York, NY, USA. Association for Computing Machinery.

Jayakrishna Vadayath, Moritz Eckert, Kyle Zeng, Nicolaas Weideman, Gokulkrishna Praveen Menon, Yanick Fratantonio, Davide Balzarotti, Adam Doupé, Tiffany Bao, Ruoyu Wang, Christophe Hauser, and Yan Shoshitaishvili. 2022. Arbiter: Bridging the static and dynamic divide in vulnerability discovery on binary programs. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 413–430, Boston, MA. USENIX Association.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Hao Wang, Zeyu Gao, Chao Zhang, Zihan Sha, Mingyang Sun, Yuchen Zhou, Wenyu Zhu, Wenju Sun, Han Qiu, and Xi Xiao. 2024a. Clap: Learning transferable binary code representations with natural language supervision. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, page 503–515, New York, NY, USA. Association for Computing Machinery.

Hao Wang, Zeyu Gao, Chao Zhang, Mingyang Sun, Yuchen Zhou, Han Qiu, and Xi Xiao. 2024b. 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*, ISSTA 2024, page 149–161, New York, NY, USA. Association for Computing Machinery.

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*, ISSTA 2022, page 1–13, New York, NY, USA. Association for Computing Machinery.

Huaijin Wang, Pingchuan Ma, Yuanyuan Yuan, Zhibo Liu, Shuai Wang, Qiyi Tang, Sen Nie, and Shi Wu. 2023. Enhancing DNN-based binary code function search with low-cost equivalence checking. *IEEE Transactions on Software Engineering*, 49(1):226–250.

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 *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society.

Hongwei Wu, Jianliang Wu, Ruoyu Wu, Ayushi Sharma, Aravind Kumar Machiry, and Antonio Bianchi. 2025. Veribin: Adaptive verification of patches at the binary level. *Proceedings 2025 Network and Distributed System Security Symposium*.

Xiangzhe Xu, Shiwei Feng, Yapeng Ye, Guangyu Shen, Zian Su, Siyuan Cheng, Guanhong Tao, Qingkai Shi, Zhuo Zhang, and Xiangyu Zhang. 2023a. Improving binary code similarity transformer models by semantics-driven instruction deemphasis. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, page 1106–1118, New York, NY, USA. Association for Computing Machinery.

Xiangzhe Xu, Zhou Xuan, Shiwei Feng, Siyuan Cheng, Yapeng Ye, Qingkai Shi, Guanhong Tao, Le Yu, Zhuo Zhang, and Xiangyu Zhang. 2023b. Pem: Representing binary program semantics for similarity analysis via a probabilistic execution model. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 401–412, New York, NY, USA. Association for Computing Machinery.

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)*, pages 224–236.

Shouguo Yang, Chaopeng Dong, Yang Xiao, Yiran Cheng, Zhiqiang Shi, Zhi Li, and Limin Sun. 2023a. Asteria-pro: Enhancing deep learning-based binary code similarity detection by incorporating domain knowledge. *ACM Trans. Softw. Eng. Methodol.*, 33(1).

Shouguo Yang, Zhengzi Xu, Yang Xiao, Zhe Lang, Wei Tang, Yang Liu, Zhiqiang Shi, Hong Li, and Limin Sun. 2023b. Towards practical binary code similarity detection: Vulnerability verification via patch semantic analysis. *ACM Trans. Softw. Eng. Methodol.*, 32(6).

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):1145–1152.

Qi Zhan, Xing Hu, Zhiyang Li, Xin Xia, David Lo, and Shanping Li. 2024. Ps3: Precise patch presence test based on semantic symbolic signature. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA. Association for Computing Machinery.

Bolun Zhang, Zeyu Gao, Hao Wang, Yuxin Cui, Siliang Qin, Chao Zhang, Kai Chen, and Beibei Zhao. 2025. Binquery: A novel framework for natural language-based binary code retrieval. *Proc. ACM Softw. Eng.*, 2(ISSTA).

Shize Zhou, Lirong Fu, Peiyu Liu, and Wenhai Wang. 2025. Binega: Enhancing dnn-based binary code similarity detection through efficient graph alignment. In *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 488–499.

A Motivation Case

To provide a concrete illustration of the sensitivity of low-level features to compilation settings, we present a real-world motivating example based on the function `COVER_strict_cmp`. Figure 5 shows the linear representation, control flow graph (CFG), and data flow graph (DFG) of the function compiled with optimization levels O0 and O2. Although the two binaries are semantically equivalent, their low-level representations differ substantially across compilation settings. Specifically, compiler optimizations such as instruction reordering, dead-code elimination, and register allocation significantly alter the instruction sequences as well as the corresponding control- and data-flow structures. Such discrepancies pose challenges for DNN-based BCSD methods that rely heavily on these features. In this case, the DNN-based BCSD method HermesSim fails to correctly match the two semantically identical functions, resulting in a false negative. This example highlights that, despite their efficiency, DNN-based BCSD methods can be misled by compilation-induced variations, motivating the need for high-level semantics.

B Implementation Details

B.1 Evaluation Setup

All the experiments are run on a Linux server running Ubuntu 20.04.6 equipped with an Intel(R) Xeon(R) Platinum 8468 processor, 1TB of RAM, and Nvidia A100 GPUs. To ensure fairness and statistical significance, each metric value is the average of 3,000 random function queries.

Basic dataset. We use the basic dataset, consisting of 26 open-source projects and a total of 404,427 functions, in §4.1 to §4.4, except for the inference time in §4.3, which is evaluated using the application dataset. These projects cover a diverse range of utilities, data-processing tools, and operating-system support software, including widely used packages such as OpenSSL, BusyBox, and zstd. All programs are compiled for the x64 architecture using GCC with four standard optimization levels (O0–O3). We divide the basic datasets at the project level into training, validation, and testing sets, with an ratio of 7:1:2. To ensure no potential data leakage, duplicate functions are manually removed.

Application dataset. To evaluate the practical utility of BinSKD in real-world vulnerability search task §4.5, we construct an application dataset. This dataset includes four different kernel versions, each compiled with four optimization levels on the GCC x64 architecture, resulting in 16 distinct vmlinux binary files and a total of 1,021,892 functions. As the core of the Linux operating system, vmlinux plays a critical role in real-world production environments, powering a wide range of devices and services. Due to its complexity and widespread deployment, vmlinux is prone to vulnerabilities, making accurate and efficient vulnerability search essential in practice. It should be emphasized that this dataset is used to evaluate BinSKD's capability in locating vulnerable functions, rather than to conduct a comprehensive vulnerability search analysis.

B.2 Application Scope

Teacher Model. To serve as the teacher model, we adopt an LLM-based BCSD method CLAP (Wang et al., 2024a) that significantly surpasses traditional DNN-based BCSD methods.

- **CLAP (Wang et al., 2024a).** CLAP employs natural language supervision to align binary code (i.e., assembly code) with their semantic explanations from LLM. By doing so, it captures high-level program semantics. In our experiments, we produce embeddings using CLAP's official implementation.[2]

Student Models. To validate the generality and robustness of BinSKD, we apply it to four cutting-edge DNN-based BCSD methods built upon different models. They have been shown to suffer from limited accuracy due to their choice of coarse-grained features and are expected to be improved.

- **HermesSim (He et al., 2024).** It uses a GNN architecture with a novel semantics-oriented graph (SOG) of binary code. We generate embeddings following its official codebase.[3]
- **SAFE (Massarelli et al., 2019).** It uses an RNN architecture with attention mechanisms to derive embeddings directly from disassembled binary code. The embeddings used here are generated based on the official repository.[4]
- **jTrans (Wang et al., 2022).** It uses a Transformer-based model to capture control-

[2] https://github.com/Hustcw/CLAP
[3] https://github.com/NSSL-SJTU/HermesSim
[4] https://github.com/gadiluna/SAFE

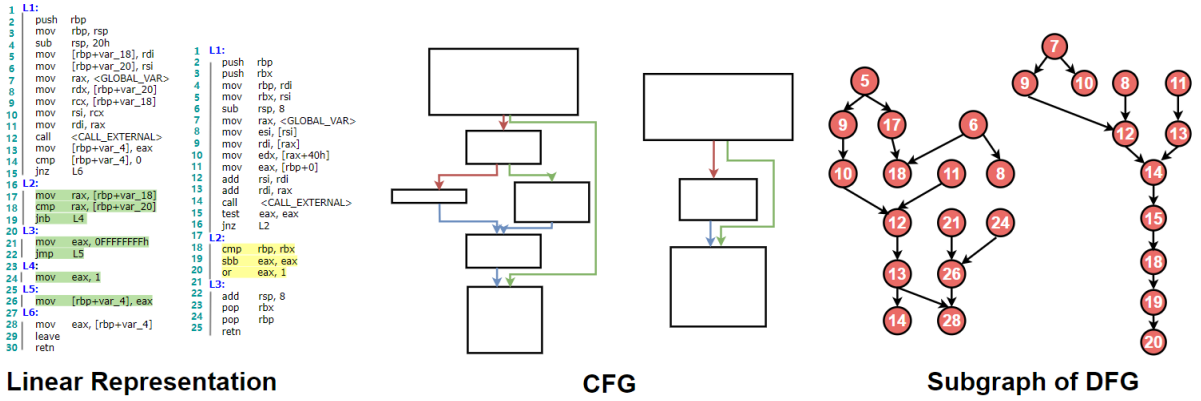

Figure 5: Linear representation, CFG, and DFG of the real-world function `COVER_strict_cmp` compiled with O0 (left) and O2 (right). The red graph highlights a subgraph of the DFG. Each node corresponds to an instruction in the function. Node numbers indicate the position of each instruction in the linear representation.

flow within binary code. Embeddings are generated using its official codebase.[5]

- **BinaryAI (Yu et al., 2020).** It applies a CNN to the CFG adjacency matrix for embedding generation, and uses BERT for token- and block-level pre-training. Since the original implementation is not publicly available, we reimplemented the embedding process based on the configurations described in the paper.

These selections do not imply that our teacher–student configuration is fixed. Owing to the relational representation learned by BinSKD, the proposed framework can seamlessly adapt to a wide range of BCSD methods built upon different neural network architectures. This adaptability enables BinSKD to deliver consistent and cost-effective enhancement across diverse methods.

B.3 Configuration of Hyperparameter

As outlined in the section 3, the distillation loss L_{KD} is integrated with the task-specific loss function L_{task}. The two losses are balanced by a tunable hyperparameter λ_{KD}, which controls the relative contribution of the teacher.

In practice, determining the optimal value of λ_{KD} requires empirical tuning and validation across datasets to achieve proper weighting. To investigate the sensitivity of λ_{KD}, we varied λ_{KD} in increments of 0.2 within the range $[0, 15]$ and plotted the corresponding results (as shown in Fig. 6). Notably, with increasing λ_{KD}, the model's MRR initially rises sharply, then drops slightly, and eventually converges to a stable value that remains noticeably higher than the performance without the distillation loss (i.e., at $\lambda_{\text{KD}} = 0$). This behavior reflects the fact that, at very large λ_{KD}, the student model depends almost on the soft labels provided by the teacher model, which typically carry richer information compared to the original hard labels. This is because hard labels merely indicate binary relationships—positive or negative—while soft labels capture the teacher model's more nuanced assessment of sample similarity. Owing to this convergence property, the tuning of λ_{KD} is rather broad. Even with relatively large values, the performance degradation remains minimal, indicating that BinSKD incurs limited computational overhead for hyperparameter.

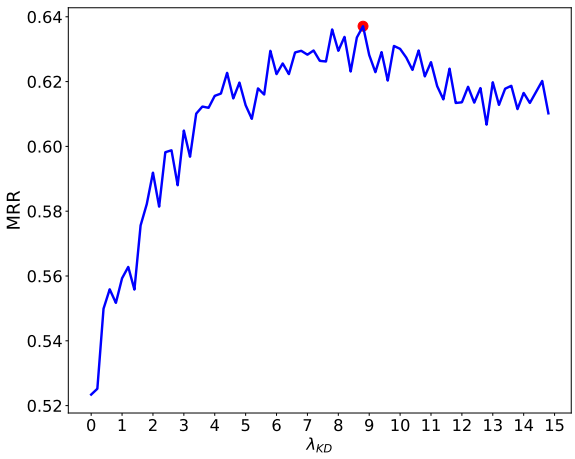

Figure 6: MRR changes w.r.t. different λ_{KD} using the BinSKD to boost the performance of BinaryAI. The optimal result is achieved when $\lambda_{\text{KD}} = 8.8$.

[5] https://github.com/vul337/jTrans/tree/main