LENA: Llama-based Embeddings of Neutralized Assembly for Cross-compiler/optimization Binary Code Similarity Detection

Mohammadhossein Amouei, Benjamin C. M. Fung, Philippe Charland, and Jun Meng

***Abstract*—Binary code similarity detection (BCSD) is fundamental to software security, yet remains challenging under cross-compiler and cross-optimization settings. Existing methods often rely on contrastive objectives that are sensitive to false negatives, while recent alternatives may learn redundant embeddings that limit fine-grained discrimination. In this paper, we present LENA, a BCSD framework that adapts a Llama backbone through an optimization-translation task and trains a lightweight projector using a hybrid objective that combines contrastive learning with variance and covariance regularization. Experiments on x86-64 benchmarks show that LENA outperforms state-of-the-art baselines, improving average Recall@1 by 10.45% on GCC, 5.36% on Clang, and 8.55% in cross-compiler settings, while also improving average R-Precision in vulnerability detection. We further provide a supporting tie-aware evaluation analysis and introduce Tie-sensitive Reciprocal Rank (TsRR), which provides deterministic reciprocal-rank-based reporting when tied similarity scores occur and reduces ambiguity in BCSD evaluation.**

***Index Terms*—Binary code similarity detection, self-supervised learning, assembly code embeddings, compiler-induced variations, tied similarity scores, vulnerability detection.**

I. Introduction

BINARY Code Similarity Detection (BCSD) refers to the process of comparing the binary representations of functions, typically compiled into machine or assembly code, to determine whether they perform the same or very similar operations. In this context, two functions are considered similar if they exhibit equivalent functional behavior, i.e., they implement comparable logic or computational tasks, even if their low-level code differs due to variations in compilation parameters or instruction set conventions. BCSD tools are critical for reverse engineers across multiple domains in computer security and software engineering. They help detect reused code (including potential intellectual property violations) [1], [2], identify known-vulnerable functions [3], [4], [5], and support malware analysis by revealing similarities among malicious binaries [6], [7], [8].

In real-world settings, however, the same source-level functionality can yield widely different binary representations. Distinct compilers (e.g., GCC vs. Clang) employ different instruction selection, register allocation, and control-flow structuring strategies, while optimization levels (O0–O3) apply transformations such as inlining, loop unrolling, constant folding, and dead code elimination [9]. As a result, even semantically identical functions may differ dramatically in syntax and structure. While some state-of-the-art (SOTA) BCSD approaches perform well under controlled conditions, they often degrade significantly in realistic cross-compiler and cross-optimization scenarios, particularly when applied to large search spaces where the combinatorial explosion of variants increases the difficulty of retrieval [3], [4], [5].

Recent methods increasingly employ *contrastive learning* to train function embeddings that abstract away compiler-induced variability. In these frameworks, positive pairs are formed from functions known to share semantics (e.g., compiled from the same source), while negative pairs are assumed to be dissimilar. Prior approaches to this challenge generally fall into two categories of contrastive learning. The first utilizes conventional contrastive objectives, such as triplet or margin-based losses (e.g., Trex [10], jTrans [11]), to capture semantic features and control-flow patterns. However, these conventional methods are highly sensitive to negative sample quality and are particularly vulnerable to *false negatives*—samples that are semantically similar but treated as dissimilar during training [12], [13], [14]. This issue is severe in BCSD, where partial code reuse, identical helper functions, and minor compiler variations frequently produce semantically related binaries that lack explicit positive labels.

To mitigate the impact of false negatives, more recent methods like BinCola [15] and CLAP [16] have adopted batch-objective styles, such as InfoNCE [17] or NT-Xent [18]. By contrasting each sample against a large batch of negatives simultaneously, these approaches distribute and dilute the penalty of any single false negative. Nevertheless, while large-batch objectives improve robustness against false negatives, they do not guarantee full utilization of the embedding dimensions. This often leads to the learning of redundant embeddings, which limits feature diversity and ultimately weakens the model's ability to distinguish fine-grained semantic differences between binary functions [19]. Conversely, *non-contrastive, regularizer-based* objectives from self-supervised learning in vision [20], [21] eliminate the dependence on negatives entirely and promote feature diversity by discouraging collapse and reducing dimensional redundancy. However, these techniques risk losing the valuable semantic signals that negative samples provide [22]. Although hybrid approaches that integrate contrastive objectives with regularizer-based

M. Amouei, B. C. M. Fung, and J. Meng are with the School of Information Studies, McGill University, Montreal, QC, Canada e-mail: ({mohammadhossein.amouei, jun.meng}@mail.mcgill.ca; ben.fung@mcgill.ca).

P. Charland is with Mission Critical Cyber Security Section, Defence R&D Canada, QC, Canada e-mail: (philippe.charland@drdc-rddc.gc.ca).

Preprint submitted to IEEE Transactions on Software Engineering

non-contrastive terms have shown promise in vision and language [23], [24], they remain underexplored in BCSD.

A third, and largely overlooked, challenge in BCSD lies in evaluation. To our knowledge, all prior work assumes that similarity scores produce a strict total order among candidates and reports metrics such as Recall@k or Mean Reciprocal Rank (MRR). In practice, however, *tied scores* are common in embedding-based retrieval, due to floating-point precision, quantization, or embedding collapse. In information retrieval, ties are known to distort ranking- and cutoff-based metrics, sometimes exaggerating differences between systems or unfairly favoring one model over another [25], [26]. However, to our knowledge, no previous BCSD study has systematically analyzed the frequency or impact of ties, nor adopted tie-aware metrics to address this gap.

To address these issues, in this study, we propose an approach to tackle key challenges in BCSD. Our study not only proposes an approach for generating robust and discriminative embeddings of assembly code using a hybrid training objective (contrastive and regularizer-based), but also introduces a novel tie-sensitive evaluation metric to address the issue of fair evaluation under the presence of tied similarity scores.

In our experiments, LENA consistently outperformed the best SOTA baseline, CLAP, across multiple evaluation settings. In cross-optimization experiments, LENA 1B improved average Recall@1 by 9.59% on GCC and 4.65% on Clang, while LENA 3B achieved gains of 10.45% and 5.36%, respectively. In cross-compiler evaluation, LENA 1B and LENA 3B exceeded CLAP by 6.67% and 8.55%, respectively. In the vulnerability detection study, LENA also achieved stronger retrieval effectiveness than CLAP: on Asteria-Pro, LENA 1B increased average R-Precision from 0.4079 to 0.4398, while on BinKit-CVE, LENA 3B improved it from 0.5553 to 0.6115; under a top-20 retrieval budget, LENA retrieved substantially more vulnerable functions than CLAP on both datasets.

In summary, our contributions are as follows:

- We proposed LENA[1], a new BCSD framework built on Llama 3.2 backbones (1B and 3B). LENA first fine-tunes the backbone with an optimization-translation task to reduce optimization-induced variability, and then learns robust assembly embeddings for clone search.
- We designed a lightweight representation learning pipeline that summarizes each function using mean and standard-deviation token embeddings and feeds them to an MLP projector trained with a hybrid objective combining contrastive learning and variance–covariance regularization. This design improves robustness while remaining computationally efficient.
- We conducted, to the best of our knowledge, the first systematic study of tied similarity scores in BCSD. We then introduced *tie-sensitive reciprocal rank* (*TsRR*[2]), a new tie-aware evaluation metric for BCSD. TsRR accounts for tie-group severity while preserving the desirable properties of reciprocal-rank-based measures, providing a more principled basis for comparing BCSD methods under tied scores.

[1]https://github.com/McGill-DMaS/LENA
[2]https://github.com/McGill-DMaS/TsRR

The remainder of this paper is organized as follows. Section II provides background on BCSD, self-supervised representation learning, and tied-score evaluation. Section III describes the proposed LENA framework. Section IV presents the TsRR metric. Section V details the experimental setup and empirical results. Section VI discusses practicality, cross-ISA generalization, threats to validity, and limitations. Section VII reviews related work, and Section VIII concludes the paper.

II. Background and Problem Definitions

A. Cross-compiler/optimization BCSD

When a source-level function is compiled, the resulting binary representation can vary significantly due to both cross-compiler and cross-optimization differences. Different compilers, such as GCC or Clang, implement distinct strategies for tasks such as instruction selection, register allocation, calling conventions, and control-flow structuring. This means that even when the same source function is compiled by different compilers, the generated assembly code can look markedly different. In addition, the optimization level (e.g., O0, O1, O2, O3) applied during compilation introduces further variability by applying code transformations such as inlining, loop unrolling, constant folding, and dead code elimination. As a result, a single source function may yield multiple, syntactically diverse binary representations that nevertheless implement the same underlying semantics.

To robustly detect semantic equivalence between binary functions at the function level despite these variations, several studies have proposed techniques to learn a representation that abstracts away the differences introduced by different compilers and optimization settings. Formally, let $\mathscr{F}$ denote the set of all binary functions and define an embedding function

$$\phi : \mathscr{F} \to \mathbb{R}^d, \tag{1}$$

which maps a binary function $f \in \mathscr{F}$ into a d-dimensional vector space. The objective is to learn ϕ such that for any two binary functions f and g:

- **Semantic Consistency:** If f and g are compiled from the same source (i.e., they are semantically equivalent despite differences in their assembly representations), then their embeddings should be close in the vector space:

$$\|\phi(f) - \phi(g)\| \approx 0 \tag{2}$$

- **Semantic Discrimination:** Conversely, if f and g originate from different source functions (i.e., they are semantically different), then their embeddings should be well-separated:

$$\|\phi(f) - \phi(g)\| \geq m \quad \text{for some } m > 0. \tag{3}$$

However, learning a perfect representation that fully satisfies these conditions is challenging [27]. First, the inherent complexity of the compilation process introduces a multitude of variations even among semantically equivalent functions [9], [27]. Minor differences in optimization heuristics or incidental

changes in code generation can lead to significantly different assembly outputs, making it difficult for the embedding function to capture the true semantic essence [28]. Second, the mapping from source code to binary code is inherently many-to-many. A single source function can produce a wide variety of syntactic forms, while similar syntactic patterns can sometimes correspond to different semantics [27], [29]. Additionally, noise in the training data, such as mislabeled function pairs or variations arising from compiler-specific idiosyncrasies, can further complicate the learning process [28]. Finally, the non-linearity and high dimensionality of the space of binary functions demand that the embedding function navigates a complex landscape, where semantic similarity does not always correlate with straightforward syntactic resemblance [8].

These challenges highlight that learning robust binary function embeddings remains far from solved. Existing methods often succeed in capturing coarse-grained similarities but struggle with subtle semantic variations introduced by compiler optimizations, inlining, or code transformations. As a result, the field requires approaches that not only abstract away syntactic noise but also provide reliable and semantically faithful embeddings that generalize across compilers and optimization levels.

B. Contrastive and Non-Contrastive Learning

Several recent studies have leveraged contrastive learning to learn robust embeddings for cross-compiler and cross-optimization BCSD [11], [10], [15]. In these approaches, an anchor sample x is paired with a positive sample x^+ known or assumed to be semantically similar, while negative samples x^- are selected to be dissimilar. The goal is to pull the representations of x and x^+ closer in the embedding space, while pushing apart the representations of x and x^-. This is typically achieved using a contrastive loss function, for example:

$$\mathscr{L} = -\log \frac{e^{\text{sim}(f(x),f(x^+))}}{e^{\text{sim}(f(x),f(x^+))} + \sum_{x^-} e^{\text{sim}(f(x),f(x^-))}}, \qquad (4)$$

where $f(\cdot)$ denotes an encoder mapping inputs to a feature space, and $\text{sim}(a,b)$ is a similarity measure such as cosine similarity.

However, studies show that the selection of bad negative samples in contrastive learning introduces significant challenges [12], [13], [14]. In BCSD, the process of negative sample selection is prone to incorporating false negatives, i.e., samples that are semantically similar. These false negatives can misguide the learning process and result in less discriminative embeddings. One way to mitigate this issue is to increase the number of negatives significantly. For this purpose, loss functions such as NT-Xent [18] have been proposed. These loss functions contrast each sample against all of the others in a large batch, thereby reducing the impact of individual false negatives.

Another direction is to eliminate the need for negatives altogether. To this end, non-contrastive, regularizer-based loss functions have been proposed in the computer vision literature. These losses eschew explicit negatives by coupling an invariance term, encouraging different views of the same sample to map to similar embeddings, with one or more regularization terms that prevent collapse and promote feature decorrelation [20], [21]. Such objectives aim to achieve three goals simultaneously: (1) decreasing the distance between positive views, (2) preventing trivial collapse, and (3) increasing the amount of encoded information in the feature vectors.

Nonetheless, both solutions come with limitations. While NT-Xent and related contrastive objectives improve robustness against bad negatives, they do not guarantee the effective use of all feature dimensions [19]. On the other hand, regularizer-based loss functions miss valuable semantic signals that could otherwise be learned from negative samples [22]. Recent studies suggest that combining these complementary approaches can notably enhance representation learning, by leveraging the strengths of both paradigms [23], [24]. However, the use of regularizer-based loss functions, and their hybrid variants, remains largely unexplored in BCSD.

C. Tied Similarity Scores

In information retrieval (IR), a *tie* occurs when multiple items receive identical scores for a query. Ties complicate rank-based evaluation because metrics such as Reciprocal Rank (RR), Recall@k, Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain (NDCG) implicitly assume a strict total order. When tied items are broken arbitrarily, e.g., by implementation details or document IDs, the resulting scores may reflect tie-breaking policies rather than genuine model quality.

This issue is directly relevant to *BCSD*, where binary functions are embedded into vector spaces and ranked using similarities such as cosine similarity. Although these scores are continuous, ties can still arise due to floating-point precision, quantization, or embedding collapse. However, existing BCSD evaluations typically rely on conventional rank-based metrics and do not account for tied scores, risking misleading or unstable comparisons.

Tie-aware alternatives exist in IR, but each has limitations. Tie-aware RR (ta-RR) [25] averages over possible positions within a tie group, but can over-reward degenerate no-ranking cases where all candidates receive the same score. Pessimistic RR (PRR) [26] assigns the worst possible position within the tie group, but can be overly conservative and obscure meaningful partial discrimination. These limitations motivate a metric that preserves classical RR when no ties exist, assigns fair partial credit within ties, and penalizes large tie groups so that indiscriminate rankings cannot outperform genuinely informative ones.

III. Approach

Fig. 1 shows an overview of our proposed method. The proposed training approach comprises two complementary stages. First, we fine-tune an open-source Large Language Model (LLM) to “undo” compiler optimizations by translating assembly code produced at higher optimization levels (e.g., O2) back into its O0 form; this not only strips

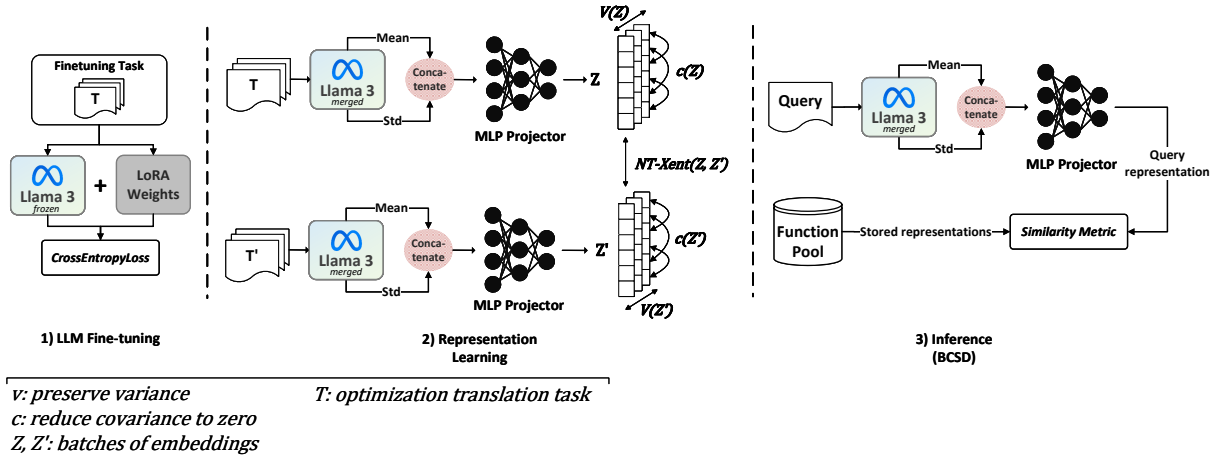

Fig. 1: Overview of the LENA framework. The approach consists of three stages: (1) large language model fine-tuning with LoRA weights on optimization translation, (2) representation learning using pooled embeddings from Llama processed through a projector optimized with a hybrid loss (NT-Xent with variance and covariance regularization), and (3) inference for BCSD by comparing stored and query representations.

away performance-oriented transformations, but also yields semantically meaningful token embeddings. Second, we use those embeddings in a lightweight MLP projector that learns to aggregate LLM-provided information into a fixed-length function representation, one that is robust to both compiler differences and optimization settings. Detailed formulations, training procedures, and hyperparameter choices are presented in the following sections.

A. Model Selection

LLMs are deep neural networks, typically based on the Transformer architecture, that have been pre-trained on vast amounts of text data to learn rich representations of language. By leveraging self-supervised objectives, such as next-token prediction or masked language modeling, these models capture complex syntactic, semantic, and contextual patterns, enabling them to perform a wide range of natural language understanding and generation tasks with minimal task-specific fine-tuning. Their ability to generalize across domains and to be adapted to specialized tasks has made LLMs a cornerstone in modern natural language processing research and applications.

For this study, we selected the open-source Llama 3.2 3B model [30] due to its widespread recognition, ease of access, and support for lightweight fine-tuning workflows. The lightweight nature of the backbone is particularly important in the context of BCSD, where practical pipelines must scale to very large corpora of functions. Consequently, we avoided heavier LLM variants that would significantly increase computational costs and hinder scalability. Our methodology is also model-agnostic: with minor adjustments, alternative LLMs could be substituted in place of Llama 3.2 3B. In preliminary experiments, we also evaluated other compact LLMs, including Qwen 3 4B [31] and Gemma 2 2B [32], but observed that Llama 3.2 consistently yielded stronger performance in our setting. Additionally, we experimented with an even lighter Llama 1B variant as a lower-cost alternative to the pipeline. Although the 1B model performed slightly worse than the 3B configuration, it still consistently outperformed prior state-of-the-art approaches.

B. LLM Fine-tuning

To prepare Llama for representation learning, we first designed a specialized fine-tuning task, **Optimization Translation**. More specifically, we fine-tuned the model to abstract away the effects of different compiler optimization levels by identifying the core semantic components of a function regardless of whether it was compiled at O0, O1, O2, O3, or Os. Consequently, we anticipated that Llama 3.2 would learn richer, more accurate embeddings of binary code that capture semantic invariance across optimizations.

In the Optimization Translation task, we intend to translate functions compiled at different optimization levels within the same compiler back to O0 as the reference representation. Since O0 closely resembles the original source code and lacks additional transformations from optimization techniques such as function inlining, this approach encourages the model to focus on the essential structure and key components of the original function. By using O0 as a baseline, we aim to train the model to prioritize core function elements, reducing reliance on artifacts introduced by higher optimization levels.

Inspired by [11], we prepared functions to capture both the structure and flow of the binary code. Fig. 2 shows an example of converting a binary function into formatted text. In this format, each basic block was enclosed within braces {} and labeled with a unique index, starting from 0. For instance, `0:{instruction sequence}` denotes the first basic block. To represent control flow, we appended the index of the target basic block to the jump instruction in the calling block, creating a clear, indexed map of execution paths between blocks. This structured approach enables the model to understand the sequence and connections within the binary code more effectively.

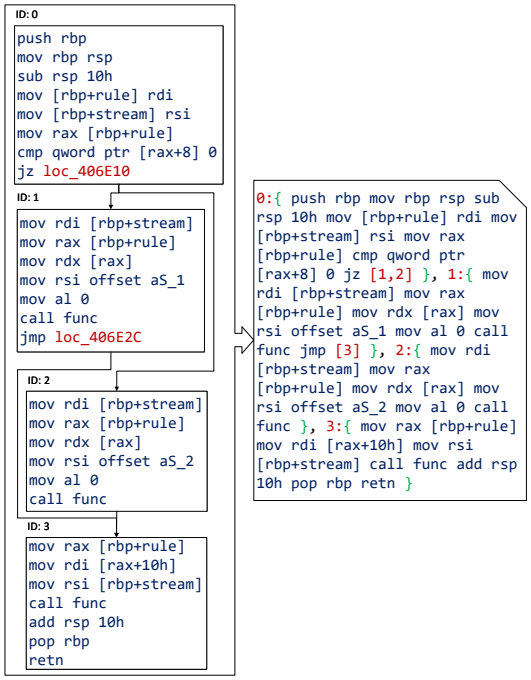

Fig. 2: An example of converting a binary function into formatted text. In this representation, each basic block is enclosed in braces and labeled with a unique index, while control flow information is embedded by appending the target block's index to jump instructions.

In our corpus, each function was originally compiled at five distinct optimization levels, resulting in a prohibitively large dataset for fine-tuning Llama within our available computational budget and time constraints. To strike a balance between representational richness and efficiency, we restricted our Optimization Translation task to the O2 level. O2 captures the majority of performance-oriented transformations found at higher settings (e.g., O3 or Os) while avoiding their more aggressive alterations, making it an ideal "sweet spot" for our objective. Incorporating additional optimization levels could further enhance representation quality; however, we defer this extension to future work, pending access to greater computational resources.

To fine-tune Llama on the optimization translation task, we used the following prompt format:

```
<|start_header_id|>user<|end_header_id|>
The O0 form of [instructions of the function at O2]

<|start_header_id|>assistant<|end_header_id|>
[instructions of the function at O0]
```

In this prompt, the O2 function's instructions were provided as input, and the model was tasked with generating the equivalent O0 instructions.

To efficiently fine-tune Llama, we leveraged Low-Rank Adaptation (LoRA) [33], a parameter-efficient tuning approach. LoRA introduces low-rank matrices into the model, allowing for fine-tuning without modifying most of the original weights. After fine-tuning, the learned LoRA adapters were merged into the Llama parameters, resulting in a unified model. This model is indicated by the merged tags in Fig. 1.

C. Representation Learning

Decoder-only LLMs (e.g., GPT-style models like GPT-2/3, Llama) do not inherently produce a fixed-length sentence embedding by design. However, several practical strategies are used to derive sentence or paragraph embeddings from their token-level outputs. These include mean pooling of hidden states [34], max pooling [35], extracting the last token embedding [36], [37], attention-based pooling [36], [38], and trainable projection layers [38]. In our experiments, we also evaluated Principal Component Analysis (PCA)-based pooling as an additional dimensionality-reduction baseline. All direct pooling approaches produced unsatisfactory embeddings, prompting us to design and train a simple MLP projector better suited to our downstream task.

Training projectors typically involves passing the LLM's final hidden states into a smaller neural network and optimizing it with objectives such as contrastive learning. However, with a context window of 1,024 tokens, the last hidden layer of Llama 3.2 produces an extremely high-dimensional representation. Since both contrastive and non-contrastive regularizer-based training objectives require large batch sizes, the sheer size of these embeddings rendered projector training on the full Llama's last hidden state impractical, given our limited hardware.

To address the aforementioned problem, we propose an alternative approach that is based on statistical properties of the last hidden state. In this approach, for each binary function, we extracted two embeddings from Llama: (1) the mean embedding of the function tokens, which captures the average semantic content and (2) the standard deviation vector of the token embeddings, providing information on variability across tokens. These embeddings were then used as inputs to an MLP projector to generate a unified function embedding based on this pair of embeddings.

The MLP projector consisted of four fully connected layers with Exponential Linear Unit (ELU) activations and an intermediate Batch Normalization layer. It first expanded the concatenated mean–std embeddings (dimension 8 192) through a residual-sized layer, and then progressively reduced the dimensionality to 4 096. The final bias-free linear layer produced the projected embedding used for self-supervised training. All linear layers were initialized with Kaiming normalization, scaled by the appropriate gain factor, to ensure stable convergence.

To train the projector, we adopted a hybrid framework that combines contrastive objectives with variance and covariance regularizers to promote rich embeddings. More specifically, we first align, within each compiler, every optimized version (O1, O2, O3, Os) with its O0 baseline. We then extend this to cross-compiler pairs by matching the O0 variants produced by different compilers (GCC and Clang) and compiler versions. This scheme encourages the projector to internalize both cross-optimization and cross-compiler invariances. For each view, we construct prompts in the following format:

```
<|start_header_id|>user<|end_header_id|>
The O0 form of [instructions of the query function]

<|start_header_id|>assistant<|end_header_id|>
```

For the contrastive objective, we adopted the normalized temperature-scaled cross-entropy loss (NT-Xent) introduced in SimCLR [18]. Given two batches of embeddings Z and Z', we first computed pairwise cosine similarities and scaled them by a temperature parameter τ. For each positive pair (z_i, z'_i), the loss took the following form:

$$\mathscr{L}_{\text{NT-Xent}} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp\left(\text{sim}(z_i,z'_i)/\tau\right)}{\sum_{j=1}^{N}\exp\left(\text{sim}(z_i,z'_j)/\tau\right)}, \tag{5}$$

where $\text{sim}(z_i,z'_j) = \frac{z_i^\top z'_j}{\|z_i\|\|z'_j\|}$ denotes the cosine similarity between embeddings.

To further regularize the learned representations, we borrowed the variance and covariance penalties from VICReg [21]. The original VICReg objective consists of three terms: invariance, variance, and covariance. The invariance term enforces similarity between paired embeddings, the variance term prevents feature collapse by ensuring that the standard deviation of each embedding dimension exceeds a margin γ, and the covariance term reduces linear dependencies across dimensions by penalizing off-diagonal entries of the covariance matrix. Formally,

$$\mathscr{L}_{\text{inv}} = \frac{1}{N}\sum_{i=1}^{N}\|z_i - z'_i\|_2^2, \tag{6}$$

$$\mathscr{L}_{\text{var}} = \frac{1}{d}\sum_{j=1}^{d}\max\left(0, \gamma - \sqrt{\text{Var}(Z_j)}\right), \tag{7}$$

$$\mathscr{L}_{\text{cov}} = \frac{1}{d}\sum_{i\neq j}\text{Cov}(Z_i,Z_j)^2, \tag{8}$$

$$\mathscr{L}_{\text{VICReg}} = \lambda_{\text{inv}}\mathscr{L}_{\text{inv}} + \lambda_{\text{var}}\mathscr{L}_{\text{var}} + \lambda_{\text{cov}}\mathscr{L}_{\text{cov}}, \tag{9}$$

where d is the embedding dimensionality, and Z_j denotes the j-th feature across the batch.

Since NT-Xent already enforced alignment, we omitted the $\mathscr{L}_{\text{inv}}$ term from VICReg. However, we retained both variance and covariance terms, as covariance regularization alone could be trivially minimized by reducing feature variance. The variance penalty ensured that features maintained sufficient spread across the batch, while the covariance penalty reduced redundancy by discouraging correlated dimensions.

Our final training objective was expressed as:

$$\mathscr{L} = \alpha\,\mathscr{L}_{\text{NT-Xent}} + (1-\alpha)\left(\lambda_{\text{var}}\mathscr{L}_{\text{var}} + \lambda_{\text{cov}}\mathscr{L}_{\text{cov}}\right), \tag{10}$$

where $\alpha \in [0,1]$ balanced the contrastive and regularizer-based components, and $\lambda_{\text{var}}, \lambda_{\text{cov}}$ controlled their relative weighting. This hybrid loss was used to train the MLP projector.

Intuitively, NT-Xent encouraged embeddings of semantically equivalent functions (e.g., different optimization levels of the same function) to be close together, while pushing apart non-matching samples within the batch. The variance and covariance terms reduced redundancy across features, encouraging more diverse and informative embeddings.

IV. Evaluation Metrics

A. Tie-sensitive Reciprocal Rank (TsRR)

To address the tie issue when calculating RR, Saha et al. [25] proposed ta-RR to handle these scenarios more fairly by replacing the single "position of the first relevant item" with the expected rank of the first relevant document when ties occur.

Formally, let r_{pre} denote the number of documents ranked before the first tie group that contains at least one relevant document, and let L be the random variable representing the position (within that group) of the first relevant document. Then, the ta-RR is defined as

$$\text{ta-RR} = \frac{1}{r_{\text{pre}} + E[L]}, \tag{11}$$

where the expected position within the tie group is computed as

$$E[L] = \sum_{l=1}^{|G|-k+1} l \cdot P(L=l). \tag{12}$$

Here, G denotes the tie group containing the relevant document(s), $|G|$ is its size, and k is the number of relevant documents within G.

Although ta-RR improves fairness by awarding partial credit within tied groups, it can lead to a paradoxical situation. A system that assigns the same score to all documents (and thus makes no real distinctions) can sometimes achieve a higher score than a system that actually ranks items, but places the relevant ones toward the bottom.

The core challenge lies in computing the expected rank for a relevant item in a fully tied scenario, where all items share the same rank. In such cases, the expected rank may appear relatively favorable. In contrast, a system that actively distinguishes items but places a relevant document in the second half of the ranked list may yield a significantly lower (worse) reciprocal rank, even though it demonstrates genuine ranking efforts.

For example, suppose R documents are retrieved, with exactly one relevant document among them. If all R documents are tied (i.e., $r_{\text{pre}} = 0$), then each position from 1 to R is equally likely for the relevant document. Consequently, the expected position is

$$E[L] = \frac{1+2+\cdots+R}{R} = \frac{R+1}{2}, \tag{13}$$

and the ta-RR becomes

$$\text{ta-RR} = \frac{1}{0+\frac{R+1}{2}} = \frac{2}{R+1}. \tag{14}$$

Now, consider a system with no ties that places the first relevant document at rank r. In this case, $r_{\text{pre}} = r-1$ and $E[L] = 1$, so that

$$\text{ta-RR} = \frac{1}{r}. \tag{15}$$

If the relevant document is ranked in the latter half of the list (i.e., if

$$r > \frac{R+1}{2}), \tag{16}$$

then

$$\frac{1}{r} < \frac{2}{R+1}. \tag{17}$$

Thus, a "no-ranking" (fully tied) approach can appear to perform better than a genuine ranking system that demotes the relevant document.

This paradox is not limited to the extreme case of a fully tied (no-ranking) system. In fact, it reveals a more general limitation of ta-RR: whenever two systems yield the same value of $r_{\text{pre}} + E[L]$, they are assigned the same score, even if the underlying tie sizes, and hence the certainty about the first relevant document's position, are very different.

Formally, for a tie group of size n containing $k \geq 1$ relevant documents, the expected position of the first relevant document is given by

$$E[L] = \frac{n+1}{k+1}, \tag{18}$$

so that

$$\text{ta-RR} = \frac{1}{r_{\text{pre}} + \frac{n+1}{k+1}}. \tag{19}$$

Consider two runs $(r_{\text{pre}}^{(1)}, n_1)$ and $(r_{\text{pre}}^{(2)}, n_2)$. If

$$r_{\text{pre}}^{(1)} + \frac{n_1+1}{k+1} = r_{\text{pre}}^{(2)} + \frac{n_2+1}{k+1}, \tag{20}$$

then both systems obtain identical ta-RR scores, regardless of whether the relevant document appears in a small tie group or a large one.

This is problematic because smaller ties entail more certainty: the probability mass of the first relevant position is concentrated earlier in the ranking. By contrast, larger ties spread this probability more widely. Indeed, the variance of the random variable L under the random tie-breaking model is

$$\text{Var}(L) = \frac{k(n+1)(n-k)}{(k+1)^2(k+2)}, \tag{21}$$

which increases strictly with tie size n when k is fixed. Thus, ta-RR may assign the same score to two systems with equal $r_{\text{pre}} + E[L]$, even though the system with the smaller tie group provides much greater certainty about the relevant item's placement.

Another approach to handling ties is the pessimistic Reciprocal Rank (PRR) [26], which assigns each relevant item the worst possible position within its tied group. While PRR ensures a conservative evaluation, it can still conflate groups of different sizes: if two tie groups end at the same rank, they receive identical PRR scores, even though one group may contain many more documents than the other or may have a different expected rank.

To address the limitations of both ta-RR and PRR, we introduce Tie-sensitive Reciprocal Rank (TsRR). TsRR blends the statistically grounded expected rank of a relevant item within its tie group (as in ta-RR) with the conservative worst-case rank (as in PRR), while also penalizing larger tie groups. Specifically, we weight the expected and worst ranks by the fraction of irrelevant items that share the tie with any relevant documents relative to all retrieved irrelevant documents. This proportional adjustment ensures that larger ties, where many irrelevant items obscure the relevant ones, incur a steeper penalty, preserving sensitivity to both ranking effort and tie size.

Formally, let $|G_{\text{irr}}|$ be the number of irrelevant documents tied with the first relevant item, and let N_{irr} be the total number of irrelevant documents retrieved:

$$|G_{\text{irr}}| = |G| - k. \tag{22}$$

We then define

$$\tau = \frac{|G_{\text{irr}}|}{N_{\text{irr}}}, \tag{23}$$

which quantifies the fraction of all retrieved irrelevants that fall into the tie group.

Given that

$$L_{\max} = |G| - k + 1, \tag{24}$$

represents the worst-case position of the first relevant document within the tie, we introduce the tie-adjusted expected rank:

$$E_\tau[L] = (1-\tau)\,E[L] + \tau L_{\max}. \tag{25}$$

Finally, Tie-sensitive Reciprocal Rank is defined as

$$\text{TsRR} = \frac{1}{r_{\text{pre}} + E_\tau[L]}. \tag{26}$$

B. Recall@k

In scenarios where results are tied, evaluating retrieval metrics such as Recall@k requires adjustments to account for the ambiguity introduced by ties. Traditional Recall@k assumes a strict ordering of results, which may lead to misleading interpretations when ties exist. Following the approach proposed by [39], we adopt a tie-aware Recall@k metric that incorporates probabilistic expectations inspired by hypergeometric distribution principles.

Consider a ranked list where items are grouped into tie groups. For a given tie group g that contains one or more relevant documents, let

- s_g denote the size of group g,
- TP_g denote the number of true positives in g,
- m_g denote the number of ranks already consumed by groups strictly preceding g.

Let k be the cutoff rank for evaluation. Then, Recall@k is defined as

$$\text{Recall@}k = \frac{\sum_{g \in \mathcal{G}_k} E[TP_g]}{\text{Total Relevant}}, \tag{27}$$

where $\mathcal{G}_k$ is the set of tie groups that intersect with the top-k ranks. For each such group g, the expected number of true positives is computed as

$$E[TP_g] = \min\{s_g, (k - m_g)\} \times \frac{TP_g}{s_g}. \tag{28}$$

This formulation, which mirrors the hypergeometric distribution, assumes that items within each tie group are ordered uniformly at random. The ratio $\frac{TP_g}{s_g}$ reflects the probability of selecting a true positive from group g, and the truncation $\min\{s_g, (k-m_g)\}$ ensures that only the slots available before the cutoff contribute. "Total Relevant" refers to the total number of relevant documents for the query in the entire collection.

TABLE I: Projects used in train and test sets

Set	Projects
Train	binutils, gsl, recutils, xorriso, inetutils, glpk, findutils, libidn, units, ccd2cue
Test	cppi, a2ps, libiconv, gdbm, nettle, gcal, dap, cpio, tar, direvent, patch, gawk, gmp, spell, wdiff, which, hello, cflow, datamash, gnudos, osip, libtool, readline, grep, gss, macchanger, gsasl, bool, gzip, sed, time, lightning, sharutils, libmicrohttpd, libtasn1, gnu-pw-mgr, libunistring, coreutils, enscript, texinfo

V. Empirical Study

A. Research Questions

In our empirical study, we seek to answer the following questions:

- **RQ1:** *How effective is our representation learning technique?*
- **RQ2:** *How does LENA's accuracy in cross-optimization and cross-compiler BCSD compare to that of current SOTA models?*
- **RQ3:** *How does varying the pool size impact LENA's performance in BCSD?*
- **RQ4:** *How effective is LENA in identifying real-world vulnerabilities?*
- **RQ5:** *What are the occurrence patterns and impacts of tied similarity scores in BCSD systems?*

RQ1 evaluates LENA's representation learning technique versus alternative last-token, mean, max, attention, and PCA pooling methods in its ability to capture meaningful features of binary code. RQ2 benchmarks LENA's accuracy in cross-optimization and cross-compiler BCSD tasks for x86-64 against state-of-the-art models to assess its robustness across different optimization levels and compilers. RQ3 studies how varying the pool size affects LENA's BCSD performance, providing insights into its scalability in large candidate pools. RQ4 evaluates LENA's effectiveness in identifying real-world vulnerabilities. Finally, RQ5 investigates the occurrence patterns of tied similarity scores in BCSD systems and analyzes how frequently such ties arise in practice. It then examines whether the theoretical limitations of RR and ta-RR, namely RR uncertainty and ta-RR's inability to distinguish rankings with different tie sizes, actually manifest in real BCSD evaluations.

B. Experimental Environment

All training and experiments in this study were conducted on a high-performance server featuring a 32-core AMD Ryzen Pro 3975WX CPU clocked at 3.50 GHz, 500 GB of RAM, and four NVIDIA RTX A6000 GPUs, operating on Windows Server 2022 Datacenter. IDA Pro version 8.4 [3] was employed for all disassembly tasks.

C. Datasets

The dataset used in this study is a subset of BinKit 2.0 [40], a large-scale benchmark specifically designed for binary code analysis and similarity detection tasks. This dataset contains binary code samples compiled from a variety of open-source projects using different compilers, such as GCC and Clang, across multiple versions. Each source code sample is compiled with a range of optimization levels (e.g., O0, O1, O2, O3, and Os) and target architectures (e.g., x86, x86-64, ARM, and MIPS), capturing variability introduced by compiler settings. In line with prior work [28], [11], [41], [15] on cross-compiler and cross-optimization BCSD, we specifically select the x86-64 architecture and optimization levels O0 through Os to ensure comparability with existing literature and to provide a representative evaluation setting. Moreover, while BinKit includes other ISAs, scaling LLM fine-tuning across them exceeds our available resources. Nevertheless, LENA's design is architecture-agnostic, and cross-architecture evaluation is a key direction for future research.

To ensure meaningful evaluation and eliminate information leakage, we performed the train-test split at the project level rather than at the function level. Splitting at the function level could allow functions from the same project to appear in both the training and test sets. This introduces contextual similarities, such as shared code structures, naming conventions, or dependencies, that could unintentionally assist the model during testing, resulting in inflated performance metrics. By splitting at the project level, we ensure that functions from the same project are exclusively assigned to either the training set or the test set, avoiding such leakage. Given the substantial variance in the number of functions per project, we first grouped projects by their size. Subsequently, we randomly selected a set of projects to achieve a roughly 70–60% training and 30–40% testing proportion. Table I shows the projects used in each set.

For the training set, we included binaries compiled using all available versions of Clang and GCC in the dataset, targeting the x86-64 architecture. For the test set, we restricted the binaries to those compiled with the latest available versions of GCC (11.2.0) and Clang (13.0.0), while still targeting the x86-64 architecture. Moreover, for the validation set, we randomly sampled 1,024 functions from the test set and removed them from the corresponding test set to prevent overlap.

Functions were labeled based on their source code origin and function names. Specifically, functions originating from the same source file and bearing the same function name were assigned the same label. Additionally, functions with identical assembly code at the O0 optimization level were considered equivalent, even if their function names differed. Functions named *main* and compiler-generated functions not linked to any source code were excluded from the dataset. To prevent data leakage, any functions appearing in the training set were explicitly removed from the test set. It is important to note that all results reported in the following sections are obtained

[3] https://docs.hex-rays.com/release-notes/8_4

using the test functions. The training set was solely used for model training and not for evaluation purposes.

D. Counterpart Selection

In our study, we selected a comprehensive set of state-of-the-art BCSD methods that utilize contrastive representation learning, together with the established Asm2Vec [28] baseline. Table II compares LENA with its counterparts. Each model employs different architectural designs and loss functions. The Asm2Vec method is a classic baseline that generates function embeddings by modeling control-flow graph execution traces with a PV-DM–style Word2Vec architecture.

Trex [10], jTrans [11], BinCola [15], and CLAP [16] represent four recent state-of-the-art approaches to binary code similarity detection that rely on contrastive learning but differ in their architectures and training strategies. Trex employs a hierarchical Transformer pre-trained on micro-trace data using a masked language modeling (MLM)–like objective and fine-tuned for function similarity through an additional MLP with a pairwise margin-based contrastive loss function that uses one positive and multiple negatives. jTrans introduces a jump-aware Transformer that integrates control-flow information, pre-trained with MLM and jump prediction tasks, and fine-tuned using a triplet loss objective. BinCola adopts a simpler design by applying the InfoNCE loss on hand-crafted function-level features such as basic block statistics. CLAP, in contrast, utilizes a RoBERTa-based backbone to encode assembly code and aligns it with natural language explanations through InfoNCE, thereby achieving cross-modal representation learning tailored to large-scale assembly–text data.

In addition to these state-of-the-art baselines, we trained multiple variants of our proposed model, LENA, to better understand both the effect of model scale and the contribution of the hybrid loss. Specifically, we considered two backbone sizes, denoted LENA 1B and LENA 3B. For each backbone, we trained two ablation variants using a single loss component: LENA 1B (NT-Xent) and LENA 3B (NT-Xent), which were trained using only the NT-Xent loss, and LENA 1B (VICReg) and LENA 3B (VICReg), which were trained using only the VICReg loss. These variants served as ablation baselines, allowing us to compare the hybrid objective directly against its individual components at both model scales. The main versions of our model, referred to as LENA 1B and LENA 3B, employed the hybrid loss that combined NT-Xent with variance–covariance regularization.

In our experiments, we fine-tuned all the models except CLAP on our training dataset to adapt them to the distribution and nuances of our benchmark. CLAP was not further fine-tuned because it had already been trained on the full Ubuntu repository, which encompasses our dataset as a subset, making additional training unnecessary.

E. Hyperparameters

For fine-tuning the Llama backbone, we used a parameter-efficient LoRA setup with rank $r = 4$ and scaling factor $\alpha = 4$. LoRA adapters were inserted only into the feed-forward projections (`gate_proj`, `up_proj`, and `down_proj`), while the attention projections were left unchanged. We set LoRA dropout to 0 and did not adapt bias terms. Training was conducted in `bfloat16` precision, without 4-bit quantization, and with a maximum input length of 2048 tokens. Optimization used AdamW with a learning rate of 3×10^{-4}, 10 warmup steps, a cosine scheduler, and gradient clipping at 0.3. The per-device batch size was 32 with no gradient accumulation. The model was trained for 0.5 epochs. Following our instruction-tuning format, the loss was computed only over response tokens, so that the model was optimized to generate the target normalized assembly rather than reproduce the prompt.

For training the MLP projector, we performed a grid search specifically over the loss-term coefficients and the batch size. The ranges explored were:

- $\lambda_{\text{inv}} \in \{5, 10, 15, 20, 25\}$
- $\lambda_{\text{var}} \in \{5, 10, 15, 20, 25\}$
- $\lambda_{\text{cov}} \in \{1, 5, 10, 15, 20, 25\}$
- $\alpha \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$
- batch size $\in \{512, 1024, 2048\}$

For the VICReg-only setting, the best coefficients were $\lambda_{\text{inv}} = 25$, $\lambda_{\text{var}} = 25$, and $\lambda_{\text{cov}} = 10$. For the hybrid loss, the best configuration was $\lambda_{\text{var}} = 10$, $\lambda_{\text{cov}} = 5$, and $\alpha = 0.6$. Across all settings, the optimal batch size was 512.

The projector was trained using the AdamW optimizer with a learning rate of 2×10^{-5}, a cosine learning-rate scheduler, and Kaiming normal weight initialization to ensure stable convergence.

F. RQ1: Representation Learning Effectiveness

To evaluate the effectiveness of our proposed MLP projector, we compared the embeddings generated by the projector with five alternative embedding strategies derived from the Llama model: Last Token, Mean Pooling, Max Pooling, Attention Pooling, and PCA Pooling. The objective of these experiments was to assess the effectiveness of LENA's embedding approach and determine whether the proposed projector provided advantages over simpler or commonly used pooling-based alternatives.

For the comparison, we generated six types of embeddings for the test set of functions:

- *MLP Projector*: Produced by feeding the mean and standard deviation vectors extracted from Llama 3B into the MLP projector.
- *Last Token*: Extracted directly as the representation of the last token from the Llama 3B model.
- *Mean Pooling*: Computed as the average of all token embeddings from the Llama 3B model.
- *Max Pooling*: Computed by taking the element-wise maximum over all token embeddings.
- *Attention Pooling*: Computed by applying a learned attention mechanism over token embeddings to produce a weighted representation.
- *PCA Pooling*: Computed by applying principal component analysis to the token embeddings to obtain a compact pooled representation.

In this experiment, we grouped functions based on their labels, where each label represented a distinct semantic class.

TABLE II: Summary of counterpart models and their training objectives.

Model	Architecture	Input Features	Pre-training/Fine-tuning Tasks; Representation learning Loss
Asm2Vec	PV-DM / Word2Vec–style	CFG traces	PV-DM Word2Vec objective
Trex	Hierarchical Transformer	Assembly code	MLM; Pairwise margin-based contrastive loss
jTrans	Jump-aware BERT Transformer	Formatted Assembly code	MLM & Jump Prediction; Triplet loss
BinCola	Shallow representation model	Hand-crafted function features	InfoNCE contrastive loss
CLAP	RoBERTa + Projector	Formatted Assembly code	MLM & Jump Prediction; InfoNCE contrastive loss
LENA	Llama + Projector	Formatted Assembly code	Opt. translation; NT-Xent + Variance + Covariance

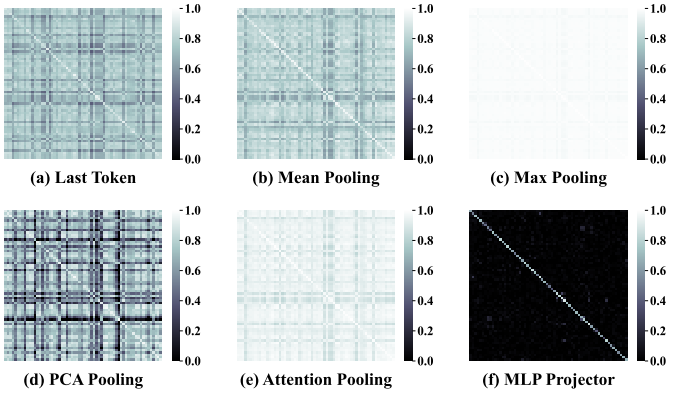

Fig. 3: Heatmap visualization of similarity scores of 64 function groups. The diagonal elements represent intra-group similarity, indicating that functions within the same group are closely clustered (lighter colors), while the off-diagonal elements show inter-group similarity.

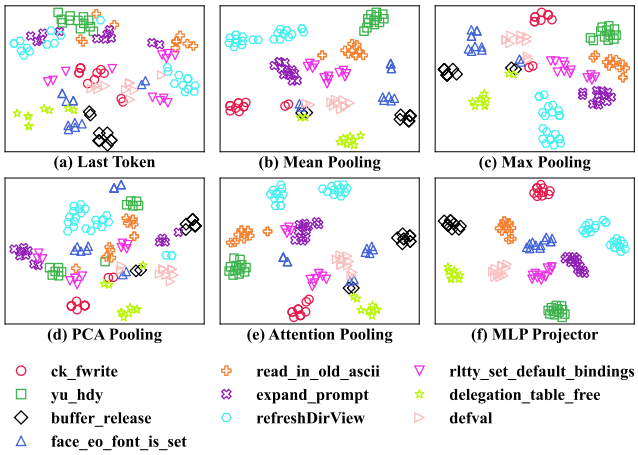

Fig. 4: t-SNE visualization of embeddings for 10 randomly selected function groups.

To evaluate embedding quality, we randomly selected 64 groups and measured two types of cosine similarity for each embedding type: the average similarity between functions within the same group (intra-group similarity) and the average similarity between functions in one group and those in other groups (inter-group similarity).

We evaluated both intra- and inter-group similarities because high-quality embeddings should simultaneously maximize cohesion among semantically equivalent functions and maintain clear separation across different semantic groups. Considering only one of these aspects would provide an incomplete assessment. High intra-group similarity without sufficiently low inter-group similarity could indicate that the embeddings captured within-group semantics but failed to distinguish different functions, which could lead to a higher rate of false positives. Conversely, low intra-group similarity would suggest that the embeddings were unable to cluster semantically equivalent functions effectively.

The similarity scores were visualized using heatmaps, as shown in Fig. 3, to provide an intuitive representation of intra-group and inter-group relationships. In these heatmaps, the diagonal corresponds to similarities among functions within the same semantic group, whereas the off-diagonal regions represent similarities between functions from different groups. A desirable embedding should therefore exhibit a clear contrast: a lighter diagonal, indicating strong intra-group cohesion, and darker off-diagonal regions, indicating good inter-group separation. It is important to note, however, that darker off-diagonal regions alone do not necessarily imply better embeddings; rather, the overall quality of the representation is reflected in the contrast between the diagonal and the rest of the matrix.

As shown in Fig. 3, the MLP projector produced the clearest and most desirable structure among all methods. Its heatmap exhibited a bright diagonal and consistently dark off-diagonal regions, indicating that semantically equivalent functions were mapped close together while unrelated functions remained well separated. In contrast, the baseline pooling strategies were substantially less effective. Last Token and PCA Pooling showed weak diagonal prominence and relatively dark horizontal and vertical banding patterns, suggesting poor intra-group cohesion and unstable separation across groups. Mean Pooling and Attention Pooling provided modest improvement, with somewhat more visible diagonal structures, but still retained considerable off-diagonal similarity, indicating overlap between distinct semantic groups. Max Pooling performed worst in terms of discriminative structure: its heatmap was nearly uniform and overly bright, suggesting that most functions were mapped to highly similar representations regardless of their semantics. Overall, these results show that simple pooling-based alternatives and direct token extraction failed to produce sufficiently discriminative embeddings, whereas the proposed MLP projector learned representations with substantially stronger cohesion and separation properties.

To further illustrate the differences in representational quality, we selected 10 random groups of functions and visualized

TABLE III: Cross-optimization binary similarity detection results with a pool size of 5,000 for GCC compiled binaries. The p-values were computed using the Wilcoxon test. The p column indicates whether the p-value for the comparison between LENA and each counterpart is less than 0.05 (✓indicates $p < 0.05$).

(a) Recall@1 results.

Method	O0-O1	O0-O2	O0-O3	O0-Os	O1-O2	O1-O3	O1-Os	O2-O3	O2-Os	O3-Os	Avg.	p
Asm2Vec	0.0000	0.0002	0.0000	0.0004	0.0002	0.0002	0.0000	0.0002	0.0000	0.0004	0.0002	✓
Trex	0.3501	0.2905	0.2624	0.3073	0.6395	0.5755	0.5799	0.7791	0.5597	0.4903	0.4834	✓
jTrans	0.5996	0.5484	0.5231	0.5955	0.7487	0.7146	0.7600	0.8228	0.7590	0.7241	0.6796	✓
BinCola	0.0691	0.0535	0.0427	0.0524	0.2693	0.2174	0.2409	0.6520	0.2405	0.1941	0.2032	✓
CLAP	0.7562	0.6747	0.6644	0.6889	0.7871	0.7721	0.7814	0.9341	0.8544	0.8257	0.7739	✓
LENA 1B (NT-Xent)	0.8179	0.7628	0.7399	0.8125	0.8985	0.8630	0.9058	0.9397	0.9102	0.8698	0.8520	✓
LENA 1B (VICReg)	0.8239	0.7736	0.7495	0.8071	0.9030	0.8585	0.9164	0.9339	0.9036	0.8596	0.8529	✓
LENA 1B	**0.8468**	0.7897	0.7688	0.8288	0.9094	0.8778	0.9270	**0.9427**	0.9214	0.8856	0.8698	✓
LENA 3B (NT-Xent)	0.8407	0.7920	0.7667	0.8247	0.9099	0.8738	0.9275	0.9360	0.9123	0.8745	0.8658	✓
LENA 3B (VICReg)	0.8285	0.7877	0.7602	0.8327	0.9109	0.8720	0.9360	0.9338	0.9131	0.8719	0.8647	✓
LENA 3B	0.8438	**0.8151**	**0.7882**	**0.8491**	**0.9176**	**0.8813**	**0.9383**	0.9399	**0.9247**	**0.8862**	**0.8784**	–

(b) TsRR results.

Method	O0-O1	O0-O2	O0-O3	O0-Os	O1-O2	O1-O3	O1-Os	O2-O3	O2-Os	O3-Os	Avg.	p
Asm2Vec	0.0016	0.0019	0.0021	0.0019	0.0017	0.0017	0.0015	0.0020	0.0016	0.0023	0.0018	✓
Trex	0.4541	0.3791	0.3404	0.3986	0.6991	0.6356	0.6607	0.7979	0.6323	0.5659	0.5564	✓
jTrans	0.6873	0.6358	0.6093	0.6741	0.7960	0.7691	0.8126	0.8567	0.8027	0.7749	0.7418	✓
BinCola	0.1345	0.1075	0.0939	0.1073	0.3562	0.2906	0.3220	0.6880	0.3233	0.2631	0.2686	✓
CLAP	0.8130	0.7401	0.7356	0.7500	0.8263	0.8143	0.8204	0.9492	0.8868	0.8602	0.8196	✓
LENA 1B (NT-Xent)	0.8682	0.8207	0.8020	0.8626	0.9244	0.8952	0.9369	0.9506	0.9343	0.9005	0.8895	✓
LENA 1B (VICReg)	0.8692	0.8271	0.8057	0.8562	0.9253	0.8880	0.9422	0.9439	0.9248	0.8870	0.8869	✓
LENA 1B	0.8900	0.8442	0.8262	0.8763	0.9307	0.9056	0.9510	**0.9530**	0.9414	**0.9124**	0.9031	✓
LENA 3B (NT-Xent)	0.8886	0.8485	0.8252	0.8784	0.9329	0.9022	0.9512	0.9477	0.9332	0.9017	0.9010	✓
LENA 3B (VICReg)	0.8794	0.8429	0.8163	0.8810	0.9322	0.8977	0.9565	0.9442	0.9312	0.8947	0.8976	✓
LENA 3B	**0.8923**	**0.8663**	**0.8474**	**0.8938**	**0.9374**	**0.9073**	**0.9568**	0.9509	**0.9427**	0.9114	**0.9106**	–

their embeddings using t-SNE, as shown in Fig. 4. Each color/marker combination corresponds to a distinct semantic group, and the embeddings were generated using Last Token (Fig. 4a), Mean Pooling (Fig. 4b), Max Pooling (Fig. 4c), PCA Pooling (Fig. 4d), Attention Pooling (Fig. 4e), and the MLP Projector (Fig. 4f). The t-SNE plots provided a qualitative view of how well each embedding method preserved cohesion within semantic groups while maintaining separation across different groups.

As shown in Fig. 4, the embeddings produced by the MLP projector exhibited the clearest overall structure, with most function classes forming compact clusters that were well separated from one another. Mean Pooling also produced relatively clean groupings, but several classes remained less compact and a few were positioned closer to neighboring groups. Max Pooling yielded better separation than the weaker baselines for some classes, yet its clusters were still less uniformly compact than those of the MLP projector. In contrast, Last Token embeddings were the most scattered, with substantial overlap and fragmented class structure across the space. PCA Pooling provided partial improvement, but several clusters remained diffuse or intermixed. Attention Pooling showed moderate clustering behavior, though multiple groups near the center still displayed overlap and weaker boundaries than in the MLP projector.

Answer: The results of our experiments show that the proposed representation learning technique in LENA was substantially more effective than direct token extraction or simple pooling-based alternatives. The MLP projector consistently produced embeddings with higher intra-group similarity and lower inter-group similarity, suggesting stronger cohesion within function groups and clearer separation across different semantic groups. Compared with last-token extraction, mean pooling, max pooling, PCA pooling, and attention pooling, the proposed projector yielded the most discriminative representation space.

G. RQ2: LENA's Effectiveness in Cross-Optimization and Cross-Compiler BCSD

To evaluate LENA's effectiveness compared to SOTA approaches, we conducted a series of clone search experiments under both cross-optimization and cross-compiler settings. For the cross-optimization evaluation, we considered all combinations of optimization levels O0, O1, O2, O3, and Os. These experiments were carried out using binaries produced by both GCC and Clang, with each query evaluated against a pool of 5,000 functions. For the cross-compiler performance evaluation, we maintained the same optimization level combinations, but selected query functions compiled with GCC, while the pool comprised functions compiled with Clang. In all experiments, we measured Recall@1 and TsRR to compare LENA with its counterparts. Moreover, to assess statistical significance, we employed the Wilcoxon signed-rank test and indicate in the p column of our tables whether the

TABLE IV: Cross-optimization binary similarity detection results with a pool size of 5,000 for Clang compiled binaries. The p-values were computed using the Wilcoxon test. The p column indicates whether the p-value for the comparison between LENA and each counterpart is less than 0.05 (✓indicates $p < 0.05$).

(a) Recall@1 results.

Method	O0-O1	O0-O2	O0-O3	O0-Os	O1-O2	O1-O3	O1-Os	O2-O3	O2-Os	O3-Os	Avg.	p
Asm2Vec	0.0000	0.0002	0.0002	0.0004	0.0004	0.0002	0.0004	0.0000	0.0002	0.0000	0.0002	✓
Trex	0.1583	0.1504	0.1543	0.1694	0.8270	0.8050	0.7404	0.8856	0.7618	0.7402	0.5392	✓
jTrans	0.4824	0.4850	0.4784	0.5079	0.8490	0.8383	0.8008	0.8667	0.8073	0.7980	0.6914	✓
BinCola	0.0471	0.0470	0.0436	0.0503	0.6156	0.5513	0.4784	0.7462	0.5612	0.4986	0.3639	✓
CLAP	0.7367	0.6989	0.6890	0.7080	0.8967	0.8940	0.8570	0.9746	0.9197	0.9096	0.8284	✓
LENA 1B (NT-Xent)	0.7632	0.7460	0.7538	0.7894	0.9539	0.9486	0.9117	0.9735	0.9196	0.9141	0.8674	✓
LENA 1B (VICReg)	0.7587	0.7558	0.7494	0.7865	0.9578	0.9431	0.9161	0.9734	0.9111	0.9017	0.8654	✓
LENA 1B	0.7811	0.7644	0.7598	0.7978	0.9615	0.9552	0.9174	0.9751	0.9221	**0.9148**	0.8749	✓
LENA 3B (NT-Xent)	0.7698	0.7597	0.7591	0.7992	0.9578	0.9508	0.9189	0.9767	0.9235	0.9096	0.8725	✓
LENA 3B (VICReg)	0.7656	0.7602	0.7509	0.7984	0.9642	**0.9597**	0.9187	**0.9775**	0.9176	0.9115	0.8724	✓
LENA 3B	**0.7817**	**0.7836**	**0.7722**	**0.8142**	**0.9644**	0.9595	**0.9304**	0.9746	**0.9251**	0.9146	**0.8820**	–

(b) TsRR results.

Method	O0-O1	O0-O2	O0-O3	O0-Os	O1-O2	O1-O3	O1-Os	O2-O3	O2-Os	O3-Os	Avg.	p
Asm2Vec	0.0017	0.0024	0.0016	0.0021	0.0019	0.0015	0.0020	0.0016	0.0018	0.0017	0.0018	✓
Trex	0.2316	0.2244	0.2226	0.2524	0.8635	0.8448	0.7868	0.9131	0.7923	0.7793	0.5911	✓
jTrans	0.5828	0.5760	0.5797	0.6054	0.8778	0.8734	0.8333	0.8947	0.8397	0.8302	0.7493	✓
BinCola	0.0942	0.0905	0.0877	0.1076	0.6643	0.6030	0.5358	0.7864	0.6052	0.5519	0.4127	✓
CLAP	0.7922	0.7582	0.7500	0.7640	0.9160	0.9133	0.8794	0.9808	0.9345	0.9259	0.8614	✓
LENA 1B (NT-Xent)	0.8173	0.8043	0.8075	0.8418	0.9660	0.9624	0.9312	0.9795	0.9346	0.9298	0.8974	✓
LENA 1B (VICReg)	0.8098	0.8071	0.8011	0.8353	0.9672	0.9567	0.9336	0.9794	0.9281	0.9202	0.8938	✓
LENA 1B	0.8306	0.8162	0.8126	0.8475	0.9705	0.9669	0.9355	0.9809	0.9365	0.9308	0.9028	✓
LENA 3B (NT-Xent)	0.8239	0.8152	0.8140	0.8491	0.9690	0.9625	0.9365	0.9818	0.9388	0.9278	0.9019	✓
LENA 3B (VICReg)	0.8167	0.8123	0.8055	0.8472	**0.9732**	**0.9694**	0.9356	**0.9823**	0.9332	0.9286	0.9004	✓
LENA 3B	**0.8334**	**0.8363**	**0.8280**	**0.8648**	0.9731	**0.9694**	**0.9454**	0.9805	**0.9404**	**0.9325**	**0.9104**	–

improvement of LENA 3B over each baseline and other LENA variants is significant at the 0.05 level.

On GCC binaries (Tables IIIa and IIIb), the classical baseline Asm2Vec performed poorly across both metrics, with average Recall@1 and TsRR values close to zero. Recent SOTA methods, including Trex, jTrans, BinCola, and CLAP, achieved considerably higher scores. Among them, CLAP was the strongest baseline, with averages of 0.7739 in Recall@1 and 0.8196 in TsRR. All LENA variants surpassed CLAP on average. Importantly, the hybrid variants consistently achieved the strongest overall results within each model scale, outperforming their NT-Xent-only and VICReg-only counterparts on average. The best overall performance was obtained by LENA 3B, which reached 0.8784 in average Recall@1 and 0.9106 in average TsRR. LENA 3B achieved the best Recall@1 on 8 of the 10 optimization pairs and the best TsRR on 8 of the 10 pairs, while LENA 1B achieved the top TsRR on O2–O3 and O3–Os. These results showed that combining NT-Xent with variance–covariance regularization was more effective overall than using either objective alone.

On Clang binaries (Tables IVa and IVb), the trend was similar. Asm2Vec again yielded negligible scores, while the stronger baselines provided competitive performance. CLAP achieved average scores of 0.8284 in Recall@1 and 0.8614 in TsRR. The best overall performance was again obtained by LENA 3B, which reached 0.8820 in average Recall@1 and 0.9104 in average TsRR. More importantly, the hybrid loss again produced the strongest average performance overall: both LENA 1B and LENA 3B outperformed their corresponding NT-Xent-only and VICReg-only variants in average Recall@1 and TsRR. Although individual optimization pairs occasionally favored a single-loss variant, the hybrid objective was the most consistently effective across settings. Overall, these results further supported the benefit of combining alignment and regularization objectives rather than relying on either one in isolation.

Tables Va and Vb report the Recall@1 and TsRR results for cross-compiler, cross-optimization retrieval with a pool size of 5,000 functions. This setting was the most challenging since functions compiled with different compilers and optimization levels introduced greater variability.

In terms of Recall@1 (Table Va), Asm2Vec and BinCola performed poorly, with averages of 0.0002 and 0.1431, respectively. More competitive methods such as Trex and jTrans achieved average scores of 0.3522 and 0.6000. CLAP provided the strongest baseline, reaching 0.7339 on average. Among the ablations, the NT-Xent-only variants performed noticeably worse in this setting, whereas the VICReg-only variants remained much stronger, suggesting that variance–covariance regularization played an important role under compiler shift. However, the hybrid objective still achieved the best overall performance, indicating that the contrastive and regularization terms were complementary rather than redundant. The full models performed best overall, with LENA 1B reaching

TABLE V: Cross-compiler and cross-optimization binary similarity detection results with a pool size of 5,000. The p-values were computed using the Wilcoxon test. The p column indicates whether the p-value for the comparison between LENA and each counterpart is less than 0.05 (✓indicates $p < 0.05$).

(a) Recall@1 results.

Method	O0-O1	O0-O2	O0-O3	O0-Os	O1-O2	O1-O3	O1-Os	O2-O3	O2-Os	O3-Os	Avg.	p
Asm2Vec	0.0006	0.0004	0.0002	0.0002	0.0004	0.0000	0.0000	0.0002	0.0000	0.0004	0.0002	✓
Trex	0.2203	0.2115	0.2140	0.2388	0.4223	0.4119	0.4666	0.4433	0.4671	0.4259	0.3522	✓
jTrans	0.5224	0.4999	0.5029	0.5339	0.6374	0.6343	0.6770	0.6616	0.6837	0.6470	0.6000	✓
BinCola	0.0429	0.0372	0.0321	0.0491	0.1780	0.1599	0.2248	0.2351	0.2559	0.2159	0.1431	✓
CLAP	0.7293	0.6958	0.6953	0.7061	0.7224	0.7169	0.7425	0.7741	0.7826	0.7741	0.7339	✓
LENA 1B (NT-Xent)	0.7456	0.7339	0.7363	0.7714	0.6482	0.6314	0.6809	0.6514	0.6800	0.6377	0.6917	✓
LENA 1B (VICReg)	0.7649	0.7497	0.7470	0.7807	0.7978	0.7980	0.8427	0.8035	0.8281	0.7890	0.7901	✓
LENA 1B	0.7754	0.7660	0.7611	0.7955	0.8085	0.7963	0.8448	0.8158	0.8396	0.8026	0.8006	✓
LENA 3B (NT-Xent)	0.7492	0.7368	0.7377	0.7740	0.6458	0.6523	0.7107	0.6582	0.6876	0.6500	0.7002	✓
LENA 3B (VICReg)	0.7781	0.7731	0.7695	0.7983	0.8203	0.8104	0.8549	0.8243	0.8453	0.8119	0.8086	✓
LENA 3B	**0.7922**	**0.7837**	**0.7753**	**0.8120**	**0.8271**	**0.8191**	**0.8656**	**0.8263**	**0.8647**	**0.8284**	**0.8194**	–

(b) TsRR results.

Method	O0-O1	O0-O2	O0-O3	O0-Os	O1-O2	O1-O3	O1-Os	O2-O3	O2-Os	O3-Os	Avg.	p
Asm2Vec	0.0024	0.0020	0.0017	0.0018	0.0021	0.0015	0.0017	0.0020	0.0017	0.0019	0.0019	✓
Trex	0.3116	0.3033	0.2946	0.3334	0.5097	0.4984	0.5629	0.5296	0.5587	0.5135	0.4416	✓
jTrans	0.6017	0.5919	0.5896	0.6261	0.7090	0.7048	0.7453	0.7342	0.7435	0.7131	0.6759	✓
BinCola	0.0874	0.0826	0.0769	0.1020	0.2479	0.2230	0.3145	0.3105	0.3417	0.2895	0.2076	✓
CLAP	0.7891	0.7586	0.7587	0.7668	0.7758	0.7714	0.7951	0.8264	0.8335	0.8264	0.7902	✓
LENA 1B (NT-Xent)	0.8081	0.7966	0.7989	0.8288	0.7214	0.7099	0.7586	0.7255	0.7513	0.7133	0.7612	✓
LENA 1B (VICReg)	0.8140	0.8044	0.8001	0.8312	0.8415	0.8425	0.8816	0.8475	0.8689	0.8323	0.8364	✓
LENA 1B	0.8301	0.8229	0.8185	0.8474	0.8508	0.8437	0.8857	0.8628	0.8816	0.8478	0.8491	✓
LENA 3B (NT-Xent)	0.8087	0.7970	0.7987	0.8312	0.7219	0.7234	0.7802	0.7307	0.7588	0.7246	0.7675	✓
LENA 3B (VICReg)	0.8298	0.8256	0.8205	0.8484	0.8620	0.8534	0.8928	0.8649	0.8843	0.8530	0.8535	✓
LENA 3B	**0.8471**	**0.8414**	**0.8324**	**0.8629**	**0.8696**	**0.8643**	**0.9019**	**0.8737**	**0.9021**	**0.8714**	**0.8667**	–

0.8006 and LENA 3B achieving the highest average Recall@1 of 0.8194, together with the best result on every optimization pair.

The TsRR evaluation (Table Vb) confirmed and strengthened these findings. CLAP again set a strong baseline with an average of 0.7902, while LENA 1B and LENA 3B increased this to 0.8491 and 0.8667, respectively. As in Recall@1, the VICReg-only variants substantially outperformed the NT-Xent-only variants in this setting, but the hybrid LENA variants again achieved the strongest overall results. LENA 3B obtained the best TsRR on all optimization pairs, with consistent and statistically significant improvements over every baseline.

Answer: LENA consistently outperformed the strongest state-of-the-art baselines across cross-optimization and cross-compiler BCSD settings. Moreover, the hybrid objective achieved the best overall performance, outperforming both NT-Xent-only and VICReg-only variants across model scales and evaluation settings, showing that the hybrid combination of NT-Xent and variance–covariance regularization was more effective than either loss alone.

H. RQ3: Impact of Pool Size on LENA's Performance

To evaluate the robustness of LENA against varying pool sizes, we conducted experiments using the most challenging optimization setting (O0, O3) across three conditions: GCC, Clang, and a cross-compiler scenario (GCC, Clang). We performed these experiments with pool sizes of 1, 5, 50, 500, and 5,000, and measured both recall@1 and recall@5 to assess how performance is impacted as the pool size changes.

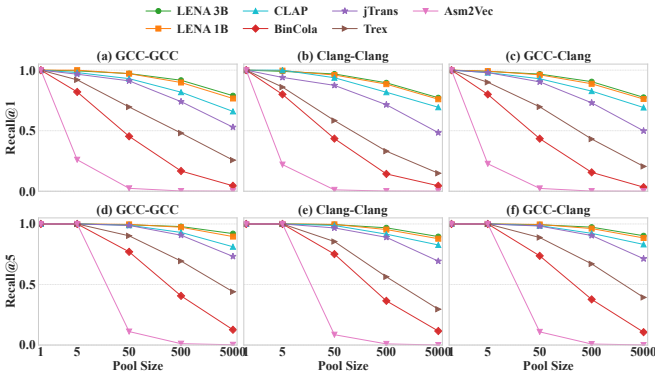

Fig. 5: Effect of pool size on recall metrics. As the pool size increases, both Recall@1 and Recall@5 for LENA degrade more gradually than those of its counterparts.

Figure 5 shows the results of our experiments. We observed that the performance of Asm2Vec, Trex, jTrans, and BinCola drops significantly as the pool size increases, indicating that these approaches struggle to maintain discriminative power when the search space becomes larger. In contrast, both LENA and CLAP exhibit substantially more stable behavior, retaining

TABLE VI: R-Precision values across different query optimization levels.

Dataset	O0	O1	O2	O3	Avg.
Asteria-Pro's Dataset					
CLAP	0.3234	0.4284	0.4423	0.4376	0.4079
LENA 1B	**0.3588**	0.4768	**0.4691**	**0.4547**	**0.4398**
LENA 3B	0.3253	**0.4816**	0.4520	0.4466	0.4264
BinKit-CVE Dataset					
CLAP	0.5349	0.5756	0.5579	0.5529	0.5553
LENA 1B	0.5594	0.6152	0.5997	0.6158	0.5975
LENA 3B	**0.5881**	**0.6252**	**0.6136**	**0.6192**	**0.6115**

higher levels of accuracy across increasing pool sizes. The performance degradation of LENA and CLAP is comparable for the Clang and cross-compiler scenarios, whereas LENA demonstrates slightly greater robustness in the GCC setting.

Answer: Our experiments show that LENA maintains strong performance as the pool size increases, with only a modest decline compared to other baselines whose accuracy drops sharply. While CLAP exhibits comparable robustness under Clang and cross-compiler settings, LENA demonstrates slightly higher stability for GCC.

I. RQ4: LENA's Effectiveness in Vulnerability Detection

To answer this question, we conducted our experiments on two Common Vulnerabilities and Exposures (CVE) datasets: Asteria-Pro's dataset [42] and BinKit-CVE [43]. From the Asteria-Pro dataset, we selected five previously unseen projects containing 127 CVEs and their corresponding patches, which were compiled using GCC at four optimization levels (O0–O3). From the BinKit-CVE dataset, we selected seven unseen projects covering 14 CVEs, compiled with both GCC and Clang at the same four optimization levels (O0–O3).

Next, we designed a challenging experimental setup to rigorously evaluate the models. We first extracted function embeddings for both datasets using LENA and CLAP. In each trial, we selected vulnerable functions at one optimization level as queries, while constructing the repository by mixing all remaining functions (both vulnerable and non-vulnerable) across the other optimization levels. For the BinKit-CVE dataset, this repository additionally included functions compiled with both GCC and Clang. Each vulnerable query function was then matched against the repository, and the retrieved results were ranked. A retrieved function was considered a positive if it was associated with the same CVE as the query. Note that if a function was linked to multiple CVEs, it was queried multiple times, once for each CVE. Finally, with a minor adaptation of tie-aware Recall@k, by setting k equal to the total number of positives in the repository, we measured R-Precision for both approaches.

Table VI reports the R-Precision values across different query optimization levels for both datasets. Overall, LENA consistently outperformed CLAP in nearly all settings. On the Asteria-Pro dataset, both LENA variants surpassed CLAP on all query optimization levels, with LENA 1B achieving the best overall average R-Precision of 0.4398 compared to 0.4079 for CLAP. Interestingly, LENA 1B slightly outperformed LENA 3B on this dataset, suggesting that the smaller model may have generalized better to the unseen Asteria-Pro binaries. LENA 1B achieved the highest scores on O0, O2, and O3, while LENA 3B performed best on O1. On the BinKit-CVE dataset, LENA again outperformed CLAP across all query settings, with LENA 3B achieving the strongest overall performance and increasing the average R-Precision from 0.5553 to 0.6115. To validate the robustness of these improvements, we combined the results from both datasets and conducted a Wilcoxon signed-rank test, which confirmed that the performance gains of LENA over CLAP were statistically significant ($p < 0.05$).

TABLE VII: Number of correctly retrieved CVEs in the top-20 retrieval budget across different query optimization levels.

Dataset	O0	O1	O2	O3	Total
Asteria-Pro's Dataset					
CLAP	459.00	665.50	642.33	636.17	2403.00
LENA 1B	580.00	721.50	686.50	688.67	2676.67
LENA 3B	537.50	714.67	666.67	647.00	2565.84
Gain (LENA 1B)	+121.00	+56.00	+44.17	+52.50	**+273.67**
Gain (LENA 3B)	+78.50	+49.17	+24.34	+10.83	**+162.84**
BinKit-CVE Dataset					
CLAP	174.00	192.00	203.00	202.00	771.00
LENA 1B	186.00	217.00	217.00	229.00	849.00
LENA 3B	194.00	219.00	219.00	223.00	855.00
Gain (LENA 1B)	+12.00	+25.00	+14.00	+27.00	**+78.00**
Gain (LENA 3B)	+20.00	+27.00	+16.00	+21.00	**+84.00**

To better understand the practical impact of the observed improvements in R-Precision, we repeated the experiments using an alternative evaluation. Instead of measuring R-Precision, we allocated a retrieval budget of 20 for each query and calculated the total number of correctly retrieved CVEs within this budget. This analysis translated the relative ranking gains into concrete retrieval outcomes, highlighting how many additional vulnerabilities LENA could recover compared to CLAP under realistic constraints.

Table VII presents the number of correctly retrieved vulnerable functions within a top-20 retrieval budget. The results show that both LENA variants consistently retrieved more relevant vulnerable functions than CLAP across all optimization levels in both datasets. On the Asteria-Pro dataset, LENA 1B achieved the strongest overall result, retrieving 2676.67 vulnerable functions in total, which was 273.67 more than CLAP. LENA 3B also improved over CLAP, with a total gain of 162.84 correct retrievals. On the BinKit-CVE dataset, LENA 3B achieved the best overall performance, retrieving 855 vulnerable functions compared to CLAP's 771, yielding 84 additional correct retrievals, while LENA 1B retrieved 849, corresponding to a gain of 78. These results show that the improvements in R-Precision translated into substantial practical gains in vulnerability retrieval.

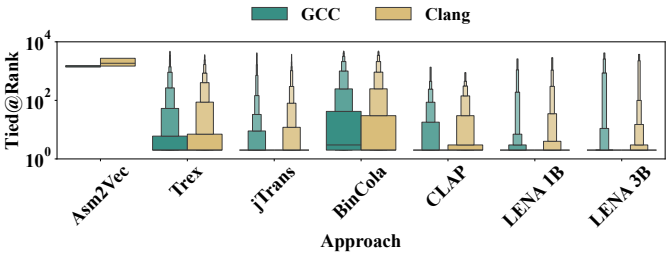

Fig. 6: Boxen plot of the rank positions of tied groups containing the first relevant result.

Answer: Our experiments on two independent CVE datasets, Asteria-Pro's dataset and BinKit-CVE, showed that LENA was effective in identifying real-world vulnerabilities. Compared to CLAP, LENA consistently achieved higher R-Precision across query optimization levels, with statistically significant improvements. On Asteria-Pro, the best average R-Precision increased from 0.4079 with CLAP to 0.4398 with LENA 1B, while on BinKit-CVE it increased from 0.5553 to 0.6115 with LENA 3B. More importantly, under a top-20 retrieval budget, LENA retrieved substantially more vulnerable functions, with gains of up to +273.67 on Asteria-Pro and +84 on BinKit-CVE. These results suggest that LENA not only improved ranking quality, but also translated these gains into tangible benefits for real-world vulnerability retrieval.

J. RQ5: Occurrence Pattern and Impacts of Tie in BCSD

In earlier sections, we discussed that tied similarity scores in BCSD systems can occur and may distort ranking-based evaluation metrics such as MRR and Recall@k. In this section, we characterize where and how these ties arise in ranked search results. We first examine the ranking positions at which tie groups appear, with particular attention to the top-ranked results where evaluation metrics are most sensitive. We then analyze the frequency and size of tie groups whose rank falls within the top-10 results, focusing on practically relevant retrieval positions commonly used in BCSD evaluation.

The rank at which a tie occurs is a critical factor in determining its effect on evaluation. A tie becomes especially problematic when it overlaps an evaluation cutoff. For example, if a tied group starts at or before rank k, but the number of higher-ranked items plus the size of the tied group exceeds k, then Recall@k depends on how items within the tied group are ordered. In such cases, the metric value is not uniquely determined unless an explicit tie-handling rule is specified. We therefore begin our analysis by quantifying the ranking positions of tied groups before examining their frequency and size.

To this end, we conducted a comprehensive clone search experiment using seven BCSD approaches, including the two variants of LENA. For each set of functions in our test set, we examined all combinations of optimization-level pairs for the query and the search pool. Each function was used as a query against a pool of 5,000 functions. For every query, we identified the tied group containing the first relevant result and recorded the starting rank of that group.

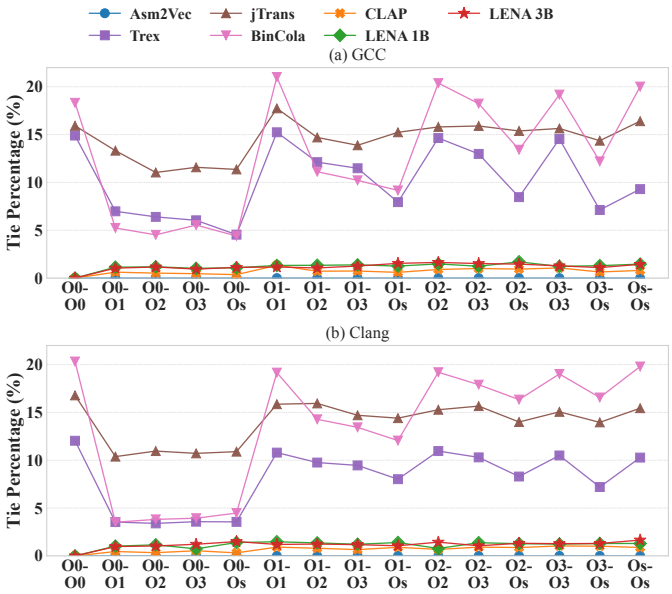

Fig. 7: Frequency of top-10 tied similarity scores for the first relevant result across evaluated BCSD approaches. A tie event is counted when the tied group containing the first relevant result appears within the top-10 retrieval results and includes at least one irrelevant candidate.

Fig. 6 presents the distribution of tied-group ranks using a boxen plot, which provides a detailed view of the rank distribution beyond the median and quartiles. Across most approaches, tied groups tend to appear within the top 100 ranks, although their exact distributions vary across methods. More importantly, a noticeable fraction of ties occurs within the top-10 positions, where Recall@1 through Recall@10 and reciprocal-rank-based metrics are most sensitive. This result shows that ties are not confined to low-ranked candidates with limited evaluation impact; instead, they can occur in precisely the ranking region most commonly used to compare BCSD systems.

Having established where ties occur, we next measure how frequently the first relevant result is tied with one or more irrelevant results within the top-10 ranked positions. For each query, we counted a tie event when the tied group containing the first relevant result had a starting rank of at most 10 and also contained at least one irrelevant result. This captures cases in which early-rank evaluation may depend on arbitrary or implementation-specific ordering among candidates with identical similarity scores.

Fig. 7 shows that all evaluated approaches experience tied similarity scores to varying degrees. Asm2Vec exhibits almost no tied scores, with tie occurrences close to 0% across all settings. CLAP, LENA 1B, and LENA 3B also show relatively low tie frequencies, each remaining below 3% of queries. In contrast, Trex, jTrans, and BinCola exhibit substantially higher tie rates, ranging from approximately 5% to over 20%, depending on the evaluation configuration. These results suggest that, although some modern approaches produce relatively stable similarity rankings, tied scores remain a non-

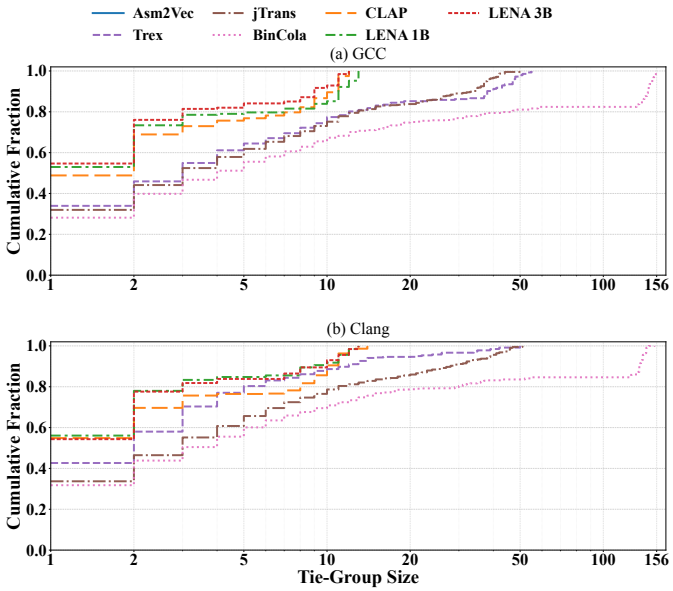

Fig. 8: ECDF of tie-group sizes for the first relevant result, measured over tie cases observed within the top-10 retrieval results. The x-axis denotes the number of irrelevant candidates tied with the first relevant result. Curves closer to the upper-left indicate less severe ties. Asm2Vec has no visible curve because it produced no tied first-relevant results in the top-10 retrieval results.

negligible phenomenon in BCSD evaluation, particularly for several widely used baselines.

At first glance, these tie rates may appear surprising because high-precision floating-point vectors would seem unlikely to produce identical similarity scores. However, this intuition mainly applies to randomly generated vectors, whereas BCSD embeddings are structured representations learned under strong invariance constraints. In practice, several factors can lead to tied scores. First, representation learning methods, particularly contrastive learning, may produce local representation collapse, where multiple inputs are mapped to nearly identical embeddings when the model fails to preserve fine-grained distinctions. Such phenomena have been documented in prior studies of contrastive learning and representation collapse [21], [19]. Second, many binary functions are nearly identical at the instruction level and differ only in a small number of instructions or operands. Because transformer-based models aggregate token information through attention and pooling operations, these small variations may contribute only marginally to the final embedding, resulting in identical or indistinguishable similarity scores. Third, some BCSD approaches, such as jTrans, intentionally apply instruction normalization or abstraction to improve cross-compiler robustness. While beneficial for generalization, these transformations can also remove subtle distinctions between functions, increasing the likelihood of ties.

These ties are not merely numerical artifacts; they reflect realistic challenges in security-oriented BCSD tasks. Prior work has shown that vulnerable functions and their patched counterparts often differ only through small code modifications [44]. Distinguishing such subtle differences is therefore essential in applications such as known vulnerability detection [42]. The problem may become even more pronounced when functions are truncated to fit model input limits, further reducing the observable differences between candidate functions.

Finally, beyond rank position and frequency, the size of the tied group determines the extent to which a tie can affect evaluation metrics. Small tied groups may have limited impact, especially when they occur lower in the ranking. In contrast, larger tied groups can substantially change reciprocal-rank-based scores. For example, if two items are tied around rank 10, the difference between assigning the relevant item rank 10 or rank 11 is relatively small. However, if the tied group contains five irrelevant results in addition to the relevant one, the assigned reciprocal rank can vary much more substantially depending on the tie-breaking policy.

To evaluate tied-group sizes, we measured the number of irrelevant results in each top-10-ranked tied group that included the first relevant result. Fig. 8 reports the empirical cumulative distribution function (ECDF) of these tied-group sizes, where the x-axis denotes the tied-group size and the y-axis shows the cumulative fraction of tie events whose size is at most that value. This representation captures not only the typical tied-group size but also the tail behavior of each approach.

The results show that CLAP, LENA 1B, and LENA 3B generally produce small tied groups: most of their top-10 tie events involve only a few irrelevant results, and nearly all are resolved within relatively small group sizes. In contrast, Trex and jTrans exhibit larger tied groups, with a non-negligible fraction extending to several dozen tied candidates. BinCola shows the most severe behavior, with a heavy-tailed distribution in both GCC and Clang settings and some tied groups exceeding 100 candidates. Thus, the methods with higher tie frequencies also tend to produce larger tied groups, compounding their effect on ranking-based evaluation. Together, the rank, frequency, and size analyses show that tie events can meaningfully influence BCSD evaluation and should therefore be handled explicitly when reporting ranking-based performance.

The empirical analysis above shows that ties in BCSD rankings can occur in evaluation-sensitive positions and, for some methods, involve large groups of irrelevant candidates. This makes tie handling a practical concern for ranking-based evaluation. The simulation study in our supplementary materials provides a complementary metric-level analysis: as shown in Scenarios 1 and 4, ta-RR can fail to distinguish rankings that differ in their tie structure, while standard RR becomes ambiguous because its value depends on arbitrary tie-breaking. Specifically, when the first relevant result belongs to a tied group, RR can vary between the best-case rank (i.e., the group rank) and the worst-case rank (i.e., PRR). Empirically, we did not observe Scenario 1 to be a practical issue in our BCSD experiments. However, we found that Scenario 4 does occur in real BCSD evaluations, where ta-RR assigns identical scores to tied groups with different sizes. This motivates an evaluation strategy that explicitly accounts for both tied-group rank and tied-group size.

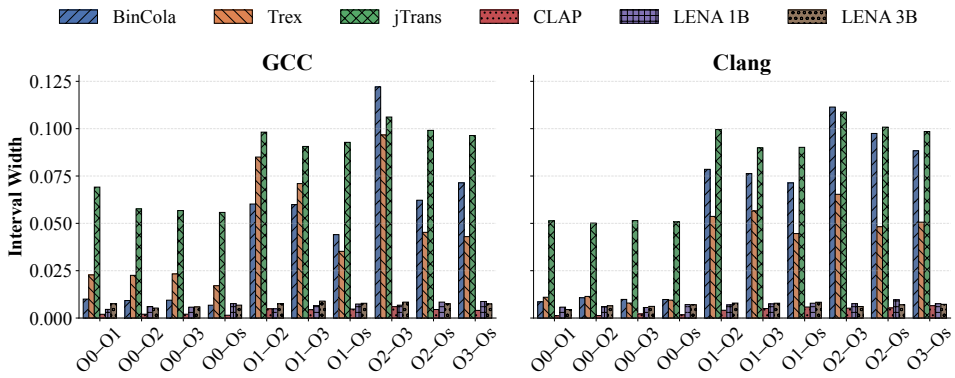

Fig. 9: RR fluctuation intervals in cross-optimization BCSD experiments for GCC and Clang. The interval width represents the difference between the best-case rank (group rank) and the worst-case rank (PRR) when tied similarity scores occur. Larger intervals indicate greater uncertainty in the RR score caused by ties. Results are reported across different optimization-level pairs for six BCSD models using 5,000 queries with a candidate pool size of 5,000.

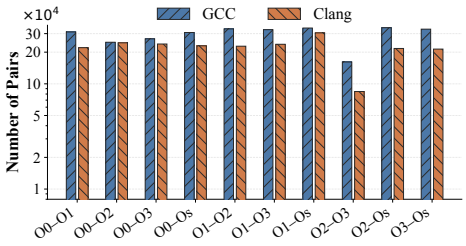

Fig. 10: Number of result pairs exhibiting Scenario 4 in ta-RR across cross-optimization BCSD experiments for GCC and Clang. Each bar shows the number of pairs that share the same expected rank under ta-RR but have different tie-group sizes.

To evaluate the occurrence of Scenario 4 and quantify the uncertainty of RR in practice, we conducted a cross-optimization BCSD experiment. Specifically, we executed 5,000 queries with a candidate pool size of 5,000 for six BCSD models. We deliberately excluded Asm2Vec from this experiment because it produced almost no ties and would therefore act as an outlier that does not meaningfully contribute to the analysis of tie-related behavior. During the experiment, we recorded the expected ranks and tie-group sizes used by ta-RR, as well as the worst-case, best-case, and assigned ranks associated with RR.

We do not report TsRR values in this analysis. As demonstrated in our simulation study, Scenario 4 cannot occur under TsRR because the metric uniquely determines a score based on both the ranking position and the tie-group size. Consequently, including TsRR in this experiment would not provide additional insight. Instead, our analysis focuses on identifying the frequency of Scenario 4 occurrences in real BCSD evaluations and quantifying the practical uncertainty introduced by RR.

To quantify the uncertainty of RR, we measured the fluctuation between its best- and worst-case values. For each experiment, we computed the average group rank (i.e., the best-case RR) and the PRR (i.e., the worst-case RR), and calculated the width of the interval between them. This interval represents the range within which the RR score can vary depending on how ties are resolved. Importantly, these measurements include both tied and non-tied retrieval results, so the reported values reflect the overall behavior of RR in realistic BCSD evaluations.

Figure 9 shows the breakdown of these interval widths across different cross-optimization scenarios for both GCC and Clang compilers. Each bar represents the average fluctuation range between the best-case and worst-case RR values for a given BCSD model. Larger intervals indicate greater uncertainty in the reported RR score due to tied similarity values. As shown in the figure, several BCSD models exhibit noticeable RR fluctuations. For instance, when querying functions compiled at O2 against a pool compiled at O3, the MRR of BinCola and jTrans can fluctuate from approximately 10% to nearly 12.5%. Such variation can substantially overestimate or underestimate an approach's actual performance. This highlights the practical impact of tie-induced uncertainty in RR-based evaluation.

To evaluate the occurrence of Scenario 4, we grouped all retrieval results based on their expected rank as measured by ta-RR. Within each group, we then counted the number of possible pairs of results that share the same expected rank but have different tied group sizes. These pairs may correspond either to different queries evaluated using the same BCSD model or to results obtained from two different models. Figure 10 summarizes the results of this analysis.

We observe that, in almost all experimental settings except the O2–O3 configuration, between approximately 200,000 and slightly over 300,000 such pairs exist. These pairs correspond to cases where ta-RR assigns identical scores despite the underlying rankings having different tie sizes. In total, these cases account for approximately 0.06% of all evaluated pairs. Although this indicates that Scenario 4 occurs relatively infrequently in our current BCSD experimental setup, its presence confirms that the issue is not purely theoretical. Rather, it reflects a structural limitation of ta-RR: the metric can map ranking outcomes with different tie structures to the same score, even when the underlying tie-group sizes differ.

Answer: Tied similarity scores in BCSD systems appear with different frequencies and characteristics across approaches. In our experiments, Asm2Vec produced no observable top-10 tie events, while CLAP, LENA 1B, and LENA 3B showed consistently low tie frequencies. In contrast, Trex, jTrans, and BinCola exhibited higher rates of top-10 ties, with BinCola showing the most pronounced variability across optimization-level settings. The observed ties also differ in size: CLAP and the LENA variants generally produced small tied groups, whereas Trex and jTrans showed heavier-tailed size distributions, and BinCola produced the largest tied groups, with some cases involving more than 100 irrelevant candidates tied with the first relevant result. These patterns suggest that, for some methods, ties may affect evaluation not only because they occur near evaluation cutoffs, but also because they can involve many candidates. Their impact is mainly metric-level: standard RR becomes ambiguous when tied candidates can be ordered differently, and Scenario 4 from the supplementary simulation study also appears in real BCSD evaluations, where ta-RR assigns identical scores to tied groups with different sizes. Although such cases represent only about 0.06% of all evaluated pairs in our experiments, their presence suggests that the simulated limitations can arise in practice. Overall, tied similarity scores are not equally problematic across all systems, but for several baselines they occur often enough, and sometimes with large enough tied groups, to justify explicit tie-aware reporting.

K. Ablation Study

To better understand the contribution of the main design choices in LENA, we conducted an ablation study along three axes: (1) the effect of combining the mean and standard deviation representations, (2) the impact of backbone fine-tuning, and (3) the use of alternative LLM backbones.

1) Contribution of Mean and Standard Deviation Representations: To assess the individual and combined contributions of the mean and standard deviation vectors, we conducted a controlled ablation study. In each ablation condition, we generated function representations by retaining one of these vectors and setting the other to zero. We then evaluated each variant on the most challenging clone-search tasks, matching O0-compiled functions against their counterparts compiled at O1, O2, O3, and Os under both GCC and Clang.

Table VIII summarizes the Recall@1 performance under different feature ablation settings. The results showed that each component contributed to the final representation: removing either the mean vector or the standard deviation vector consistently reduced performance. Across all settings, the mean vector appeared to provide a stronger individual signal than the standard deviation vector, but neither component alone was sufficient to match the performance of the full representation. The best results were consistently achieved when both were retained. For example, for Llama 1B under GCC, Recall@1 for O0–O1 dropped from 0.8468 with the full representation to 0.4049 and 0.1890 under the two ablation settings. A similar trend was observed under Clang, where the score decreased from 0.7811 to 0.4356 and 0.1400, respectively. The same pattern held across the remaining optimization levels and for the Llama 3B backbone. Overall, these results suggested that the mean and standard deviation vectors captured complementary information, and that both were important for strong cross-optimization clone-search performance.

TABLE VIII: Recall@1 results of different feature combinations when querying O0-compiled assembly functions against target functions compiled under the listed optimization levels with a pool size of 5,000.

Model	Compiler	Opt. Level	mean	std	mean + std
Llama 1B	GCC	O1	0.4049	0.1890	**0.8468**
		O2	0.3370	0.1526	**0.7897**
		O3	0.3244	0.1313	**0.7688**
		Os	0.3806	0.1554	**0.8288**
	Clang	O1	0.4356	0.1400	**0.7811**
		O2	0.4253	0.1413	**0.7644**
		O3	0.4179	0.1345	**0.7598**
		Os	0.4553	0.1459	**0.7978**
Llama 3B	GCC	O1	0.5014	0.2186	**0.8438**
		O2	0.4293	0.1719	**0.8151**
		O3	0.3992	0.1628	**0.7882**
		Os	0.4621	0.1964	**0.8491**
	Clang	O1	0.5222	0.1706	**0.7817**
		O2	0.5140	0.1640	**0.7836**
		O3	0.4891	0.1622	**0.7722**
		Os	0.5290	0.1668	**0.8142**

2) Impact of Backbone Fine-Tuning: To isolate the contribution of backbone fine-tuning to BCSD performance, we conducted a controlled comparison in which the fine-tuned Llama backbone was replaced with the original zero-shot Llama model. Keeping the training dataset, data splits, loss functions, optimizer settings, learning-rate schedule, batch size, and number of epochs identical, we trained only the pooler from scratch on top of the frozen zero-shot backbone. We then repeated the clone-search experiment under the same evaluation conditions and computed Recall@1.

Table IX highlights the impact of backbone fine-tuning on BCSD performance. Across both GCC and Clang, fine-tuning the Llama backbone consistently improved Recall@1 over the zero-shot alternative. For example, for Llama 1B under GCC, Recall@1 increased from 0.8132 to 0.8468 for O0–O1, and from 0.7666 to 0.7897 for O0–O2. Under Clang, the same model improved from 0.7565 to 0.7811 for O0–O1 and from 0.7728 to 0.7978 for O0–Os. Similar gains were observed for Llama 3B. In particular, under GCC, Recall@1 increased from 0.7746 to 0.8151 for O0–O2 and from 0.7728 to 0.8491 for O0–Os, while under Clang it improved from 0.7564 to 0.7836 for O0–O2 and from 0.7803 to 0.8142 for O0–Os. These results showed that, although the zero-shot backbone already provided reasonably strong representations, backbone fine-tuning consistently improved discrimination across optimization levels and yielded the best overall clone-search performance.

TABLE IX: Recall@1 results of the fine-tuned versus zero-shot Llama backbone when querying O0-compiled assembly functions against target functions compiled under the listed optimization levels with a pool size of 5,000.

Model	Compiler	Opt. Level	Finetune	Zero-shot
Llama 1B	GCC	O1	**0.8468**	0.8132
		O2	**0.7897**	0.7666
		O3	**0.7688**	0.7549
		Os	**0.8288**	0.8012
	Clang	O1	**0.7811**	0.7565
		O2	**0.7644**	0.7447
		O3	**0.7598**	0.7414
		Os	**0.7978**	0.7728
Llama 3B	GCC	O1	**0.8438**	0.8228
		O2	**0.8151**	0.7746
		O3	**0.7882**	0.7552
		Os	**0.8491**	0.7728
	Clang	O1	**0.7817**	0.7615
		O2	**0.7836**	0.7564
		O3	**0.7722**	0.7493
		Os	**0.8142**	0.7803

TABLE X: Recall@1 results of different LLM backbones when querying O0-compiled assembly functions against target functions compiled under the listed optimization levels with a pool size of 5,000.

Model	Compiler	O1	O2	O3	Os
Gemma 2B	GCC	0.8177	0.7655	0.7411	0.8131
	Clang	0.7502	0.7328	0.7317	0.7765
Qwen 4B	GCC	0.8145	0.7750	0.7434	0.8091
	Clang	0.7637	0.7541	0.7425	0.7895
Llama 1B	GCC	**0.8468**	0.7897	0.7688	0.8288
	Clang	0.7811	0.7644	0.7598	0.7978
Llama 3B	GCC	0.8438	**0.8151**	**0.7882**	**0.8491**
	Clang	**0.7817**	**0.7836**	**0.7722**	**0.8142**

3) Alternative LLM Backbones: To examine whether the effectiveness of LENA generalized beyond the Llama family, we evaluated the framework with two additional decoder-only LLM backbones: Gemma 2 2B and Qwen 3 4B. For a fair comparison, these models were fine-tuned under the same training setup used for Llama, including the same dataset, data splits, loss function, optimizer configuration, learning-rate schedule, batch size, and number of epochs. The only model-specific adjustments were adapting the prompt format to match the corresponding chat template and resizing the projector input dimension to match the hidden representation size of each backbone.

Table X reports the Recall@1 results across different optimization levels and compilers. Overall, the results showed that LENA generalized well across backbone families, as both Gemma 2B and Qwen 4B achieved strong clone-search performance under GCC and Clang. However, the Llama-based variants remained consistently stronger in most settings. Under GCC, Llama 1B achieved the best Recall@1 on O1 with 0.8468, while Llama 3B performed best on O2, O3, and Os with 0.8151, 0.7882, and 0.8491, respectively. Under Clang, Llama 3B achieved the best performance across all optimization levels, reaching 0.7817, 0.7836, 0.7722, and 0.8142 on O1, O2, O3, and Os, respectively. Among the alternative backbones, Qwen 4B consistently outperformed Gemma 2B, although both remained competitive. These results suggest that the proposed framework is not tied to a specific LLM family.

VI. Discussion

A. Practicality of LENA

While LENA achieved substantial improvements in binary code similarity detection accuracy, it is also important to examine the computational implications of using large language models. Existing BCSD approaches such as jTrans and CLAP rely on encoder-only transformers derived from BERT and RoBERTa, which contain approximately 110M–125M parameters [45], [46], [11], [16]. In contrast, LENA employs Llama 3.2 backbones with 1B and 3B parameters [30]. As a result, LENA is inherently more computationally demanding than smaller encoder-based baselines.

Nevertheless, the Llama variants used in our framework remain modest compared with many modern large language models and were designed with efficient deployment in mind. Official ExecuTorch deployment results showed that, on an Android OnePlus 12 device, the Llama 3.2 1B bf16 model achieved 19.2 decode tokens/s, 60.3 prefill tokens/s, and a 1.0 s time-to-first-token, while the 3B bf16 model achieved 7.6 decode tokens/s, 21.2 prefill tokens/s, and a 3.0 s time-to-first-token. The same report also indicated memory footprints of approximately 3.2 GB and 7.4 GB RSS for the 1B and 3B bf16 variants, respectively [47]. Although these measurements were obtained in a lightweight mobile deployment environment rather than in our BCSD pipeline, they provided concrete evidence that the Llama 3.2 1B/3B family remained practically deployable under constrained hardware settings.

To further improve efficiency during training and adaptation, LENA used the optimized implementation provided by Unsloth. In its official documentation, Unsloth reported about 2× faster fine-tuning and roughly 70% lower memory usage for Llama 3.2-based workflows compared with standard implementations [48]. While such figures depend on the specific setup and training recipe, they nevertheless supported the practicality of adapting billion-parameter LLM backbones on single-GPU systems.

It is also important to consider the operational context of BCSD systems. Binary similarity detection is commonly used in offline security analysis workflows, such as vulnerability discovery, malware analysis, and firmware auditing [27]. In these settings, function embeddings are typically extracted once and stored in an index for subsequent retrieval. Therefore, the cost of embedding extraction is amortized over many similarity queries, meaning that encoder runtime does not directly determine online query latency.

TABLE XI: Cross-ISA Recall@1 results of LENA 3B. XO denotes cross-optimization evaluation and XC+XO denotes the combined cross-compiler and cross-optimization setting. ARM and MIPS binaries were drawn from the BinKit dataset using the same train/test split as in the x86 experiments.

Arch	XO		XC + XO
	GCC	Clang	
MIPS	0.7526	0.7717	0.6793
ARM	0.7761	0.7882	0.7095
x86	**0.8784**	**0.8820**	**0.8194**

Finally, LENA provides a practical scalability trade-off through its two model variants. LENA 1B offers a lighter configuration with lower computational requirements while still outperforming prior BCSD approaches in our experiments. LENA 3B provides further accuracy gains at additional computational cost. This two-tier design allows practitioners to select a model according to the available hardware budget and the desired accuracy–efficiency trade-off.

B. Cross-ISA Generalization

To assess whether our approach was inherently limited to x86-64, we conducted additional cross-architecture experiments on ARM and MIPS using our 3B variant. To keep the evaluation directly comparable to the main study, we followed the same train/test split protocol and sourced the ARM and MIPS binaries from BinKit, the same dataset used for the x86 experiments. Importantly, we did not fine-tune the underlying LLM on the target Instruction Set Architecture (ISA); instead, we froze our backbone and trained only the lightweight adapter. This created a deliberately challenging transfer setting that tested whether the proposed framework could generalize beyond x86 without architecture-specific LLM adaptation.

The results (Table XI) showed that performance remained strongest on x86, which was expected given that the main focus of this work was cross-compiler and cross-optimization BCSD in the x86 setting, consistent with prior literature. Nevertheless, the model still achieved substantial performance on ARM and MIPS. In the cross-optimization setting, the average Recall@1 reached 0.7761/0.7882 on ARM under GCC/Clang and 0.7526/0.7717 on MIPS under GCC/Clang. In the more challenging combined cross-compiler + cross-optimization setting, it achieved 0.7095 on ARM and 0.6793 on MIPS, compared with 0.8194 on x86. These findings indicated that the proposed framework was not exclusively tied to x86 syntax and retained a meaningful degree of transferability across ISAs even without full LLM fine-tuning.

At the same time, the drop relative to x86 confirmed that ISA shift remained a challenging source of distribution mismatch. Differences in instruction vocabularies, register sets, calling conventions, and compiler code generation patterns likely reduced alignment when the backbone had been adapted primarily in an x86-centered setting. Thus, while these results demonstrated promising cross-ISA generalization, they also suggested that full retraining or more extensive ISA-aware adaptation would likely further improve performance on non-x86 binaries.

Overall, these additional experiments refined, rather than contradicted, the scope of our contribution. Our primary objective was to study cross-compiler and cross-optimization BCSD, and x86 was selected to align with the dominant evaluation setting in the literature and enable fair comparison with prior work. However, the ARM and MIPS results showed that the proposed framework was not restricted to x86 and could generalize reasonably well across architectures even under a limited adaptation regime.

C. Adopting TsRR in BCSD evaluation.

Although our empirical analysis shows that the edge cases where ta-RR fails occur in only a small fraction of our current BCSD evaluations, approximately 0.06% of all examined pairs, the theoretical distinction between TsRR and ta-RR remains important. To the best of our knowledge, tie-aware reciprocal-rank metrics have not been used in prior BCSD evaluations, partly because the occurrence and impact of tied similarity scores have not been systematically studied. As BCSD evaluation practices become more rigorous, it is therefore useful to adopt a reciprocal-rank metric with well-defined behavior in the presence of ties.

We propose using TsRR as a direct replacement for MRR in settings where tied similarity scores may occur. This transition is practical because TsRR preserves the main interpretability properties of reciprocal-rank-based metrics: it is bounded in $[0,1]$, higher values indicate better ranking quality, and in the absence of tied similarity scores, it reduces exactly to standard RR and therefore to MRR after averaging across queries. Thus, TsRR remains compatible with conventional MRR-based evaluation when ties do not occur, while providing deterministic behavior when they do.

For reproducibility and ease of adoption, we recommend reporting TsRR using a standardized evaluation template consisting of three components: (1) TsRR as the primary reciprocal-rank-based metric; (2) Tie-aware recall@k, with the chosen values of k explicitly stated; and (3) tie statistics, including tie frequency, tied-group size, and tied-group rank distributions. This reporting template separates retrieval success from tie behavior. Recall@k indicates whether relevant results appear within a target cutoff, TsRR summarizes rank-sensitive performance under ties, and the accompanying tie statistics explain whether differences in TsRR are associated with fewer, smaller, or lower-ranked tied groups.

Reporting MRR alongside TsRR is less informative when ties are present because the standard RR value depends on how tied candidates are ordered. In such cases, MRR may vary under different tie-breaking policies, whereas TsRR assigns a unique score by incorporating both the rank and the size of the tied group. This makes TsRR a more suitable reciprocal-rank-based metric for tied rankings while preserving compatibility with existing evaluation frameworks. More broadly, although we introduce TsRR in the context of BCSD, the same reporting principle can apply to other software engineering retrieval tasks where ranked outputs may contain tied similarity scores.

D. Threats to Validity

As with any empirical study, our results were subject to several threats to validity.

1) Internal Validity: Our ground truth relied on debug-information-based mappings, which may become incomplete or inaccurate under compiler optimizations. In addition, any compilation or labeling errors in BinKit [40] could bias the reported results.

2) External Validity: Our evaluation mainly covered x86 binaries, a limited set of compilers and optimization levels, and a small number of LLM backbones. Although we included additional experiments on ARM, MIPS, Gemma, and Qwen, the findings may not fully generalize to other architectures, toolchains, or deployment settings.

3) Construct Validity: We evaluated the method using retrieval-oriented metrics such as Recall@1, reciprocal-rank-based measures, and R-Precision. These metrics captured clone-search and vulnerability-retrieval effectiveness, but not all aspects of practical security analysis, such as false positive burden or vulnerable-versus-patched discrimination.

E. Limitations and Future Work

Although LENA demonstrated strong performance, several limitations remained. First, our main study was still largely x86-centered, despite the additional ARM and MIPS experiments, so broader multi-ISA evaluation remained future work. Second, while LENA performed well in vulnerability retrieval, we did not directly evaluate its ability to distinguish vulnerable functions from their patched counterparts, which is an important practical scenario in security analysis. Third, although the framework generalized reasonably well to Gemma, Qwen, and limited cross-ISA transfer, we considered only a small set of decoder-only LLMs and adaptation settings. Broader evaluation across model families, scales, and architecture-specific adaptation strategies would further clarify the robustness and portability of the approach.

VII. Related Work

BCSD plays a pivotal role in software security, enabling tasks such as vulnerability detection [5], [4], [3], malware analysis [7], [49], [6], and software plagiarism detection [1]. The goal of BCSD is to identify semantically similar binary functions despite differences introduced by various compilers, optimization levels, or architectures. Recent advancements leverage deep learning to address this problem, offering promising solutions but leaving critical gaps in cross-optimization and cross-compiler settings. Below, we situate our approach with respect to the most closely related lines of work and clarify how our design decisions address limitations in prior art.

A. Representation Learning for BCSD

a) Pre-transformer and non-LLM encoders: Early neural BCSD systems such as *Asm2Vec* [28] learned function representations by modeling control-flow-guided instruction traces with Word2Vec-style objectives. While effective under controlled settings, these architectures struggle to capture the long-range dependencies and structural variance introduced by different compilers and optimization levels.

b) Transformer encoders with contrastive objectives: More recent methods employ Transformer encoders and contrastive learning to directly encourage invariance across compiler/optimization variants of the same function. *Trex* [10] adopts a hierarchical Transformer pre-trained with an MLM-like objective on micro-traces and then fine-tuned with a margin-based pairwise loss using one positive and multiple negatives. *jTrans* [11] introduces a jump-aware Transformer that explicitly encodes control-flow and is fine-tuned with a triplet loss. *BinCola* [15] uses a lighter-weight representation built from hand-crafted function features trained with InfoNCE, and *CLAP* [16] leverages a RoBERTa backbone aligned to natural-language rationales via InfoNCE to improve robustness and scalability. Collectively, these systems hinge on the availability and quality of negative samples; false negatives (semantically similar yet labeled as dissimilar) and small-batch training can hamper representation quality [12], [13], [14].

c) Regularizer-based and hybrid self-supervision: In computer vision and language, non-contrastive regularizer-based objectives (e.g., variance/covariance penalties) have reduced dependency on negatives while maintaining informative features [20], [21]. However, purely non-contrastive objectives can under-utilize the semantic signals that negatives provide [22]. Hybrid objectives that combine contrastive alignment with variance/covariance regularization have recently shown strong empirical behavior, yet they remain underexplored in BCSD. Our work adopts such a hybrid objective, pairing a contrastive term to align compiler/optimization variants with variance and covariance regularizers to preserve spread and de-correlate features, thereby exploiting the complementary benefits of both paradigms.

d) LLMs for assembly representation: Prior BCSD models typically rely on BERT/RoBERTa-like encoders or task-specific Transformers [10], [11], [50], [41], [16]. In contrast, we start from a modern decoder-only LLM and specialize it to assembly through a targeted auxiliary task, *Optimization Translation*, which encourages invariance by translating optimized code back toward its O0 form. We then derive compact, statistics-based token summaries (mean and standard deviation) from the LLM and train a lightweight MLP projector under the hybrid objective. This design yields unified embeddings that are resilient to compiler- and optimization-induced variability while remaining computationally tractable.

B. Evaluation Under Rank Ties

Most BCSD studies evaluate retrieval quality using metrics such as Recall@k and RR/MRR, which emphasize the position of the first correct match and implicitly assume a strict total order over similarity scores [28], [10], [11], [50], [41], [16]. In practice, however, embedding-based retrieval can produce tied scores due to floating-point granularity, quantization, or local representation collapses, making conventional rank-based metrics sensitive to arbitrary tie-breaking.

Tie-aware evaluation has been studied more broadly in information retrieval. Prior work has proposed optimistic

expected-rank formulations, such as tie-aware RR [25], as well as pessimistic variants, such as PRR [26]. More general tie-aware alternatives also exist for cumulative ranking metrics such as Average Precision and Normalized Discounted Cumulative Gain (NDCG) [51]. However, these metrics target a different retrieval setting, typically involving multiple relevant items and emphasizing overall ranked-list quality rather than the first correct match. Since BCSD evaluation is commonly centered on Recall@k and MRR-style first-hit effectiveness, our focus in this work is specifically on the family of reciprocal-rank-based metrics.

Within this family, optimistic tie-aware formulations can over-credit indiscriminate systems that assign identical scores to many candidates, while pessimistic formulations can over-penalize and obscure meaningful distinctions. Our work introduces a *tie-sensitive* reciprocal-rank variant that remains within the RR/MRR evaluation paradigm while incorporating tie-group size explicitly. TsRR interpolates between expected-rank and worst-case behavior, penalizes larger tie groups, reduces to classical RR in the absence of ties, and preserves monotonic, deterministic scoring under tied rankings.

C. Positioning and Contributions

Relative to the above, our approach differs along two main dimensions that, to our knowledge, have not been jointly explored in BCSD. First, LENA specializes a modern decoder-only LLM to assembly code through an optimization-translation task that explicitly targets cross-optimization invariance. Second, it trains a compact projector with a *hybrid* objective that combines contrastive alignment with variance/covariance regularization, improving robustness while reducing embedding redundancy. In addition, we provide a supporting tie-aware evaluation analysis to ensure that reciprocal-rank-based comparisons remain well defined when tied similarity scores occur. Empirically, LENA achieves consistent improvements over competitive Transformer-based baselines across cross-optimization, cross-compiler, and vulnerability-retrieval settings, while the tie analysis serves to clarify evaluation behavior rather than define the central contribution of the paper.

VIII. Conclusion

In this work, we introduced LENA, a BCSD framework that combines LLM specialization through an optimization-translation fine-tuning task with a lightweight projector trained using a hybrid contrastive–regularizer objective. Across cross-optimization, cross-compiler, and vulnerability-retrieval evaluations, LENA consistently outperformed state-of-the-art baselines, demonstrating the effectiveness of the proposed representation learning strategy for robust binary function embeddings. As a supporting evaluation contribution, we also analyzed tied similarity scores in BCSD and introduced TsRR for deterministic reciprocal-rank-based reporting when tied scores occur. While our current implementation is limited to the x86-64 architecture and relies on BinKit-based ground truth, future work will extend LENA to other architectures, further refine vulnerability detection capabilities, and explore additional fine-tuning strategies.

Acknowledgment

This research is supported by Defence Research and Development Canada (contract no. W7701-217332), NSERC Discovery Grants (RGPIN-2024-04087), NSERC DND Supplements (DGDND-2024-04087), NSERC-CSE Research Community Grants (ALLRP 598786-24), and Canada Research Chairs Program (CRC-2019-00041).

Researchers funded through the NSERC-CSE Research Communities Grants do not represent the Communications Security Establishment Canada or the Government of Canada. Any research, opinions or positions they produce as part of this initiative do not represent the official views of the Government of Canada.

This work used ChatGPT (OpenAI, model version 5) to aid in refining the language and presentation of the work. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication. No original scientific content, data, or results were generated by the AI tool; all such parts were authored and verified by the human authors.

References

[1] L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu, "Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection," *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1157–1177, 2017.

[2] A. Hemel, K. T. Kalleberg, R. Vermaas, and E. Dolstra, "Finding software license violations through binary code clone detection: A retrospective," *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 24–25, Jul. 2021. [Online]. Available: https://doi.org/10.1145/3468744.3468752

[3] D. Zhao, H. Lin, L. Ran, M. Han, J. Tian, L. Lu, S. Xiong, and J. Xiang, "CVSkSA: Cross-architecture vulnerability search in firmware based on kNN-SVM and attributed control flow graph," *Software Quality Journal*, vol. 27, no. 3, pp. 1045–1068, Sep. 2019.

[4] L. Yu, Y. Lu, Y. Shen, H. Huang, and K. Zhu, "BEDetector: A two-channel encoding method to detect vulnerabilities based on binary similarity," *IEEE Access*, vol. 9, pp. 51 631–51 645, 2021.

[5] Z. Luo, P. Wang, B. Wang, Y. Tang, W. Xie, X. Zhou, D. Liu, and K. Lu, "VulHawk: Cross-architecture vulnerability detection with entropy-based binary code search," in *Proceedings of the Network and Distributed System Security Symposium*. Reston, VA: Internet Society, 2023.

[6] M. Q. Li, B. C. M. Fung, P. Charland, and S. H. H. Ding, "A novel and dedicated machine learning model for malware classification," in *Proceedings of the 16th International Conference on Software Technologies*. SCITEPRESS - Science and Technology Publications, 2021.

[7] H. Sun, H. Shu, F. Kang, and Y. Guang, "ModDiff: Modularity similarity-based malware homologation detection," *Electronics*, vol. 12, no. 10, p. 2258, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/10/2258

[8] L. Massarelli, G. A. Di Luna, F. Petroni, R. Baldoni, and L. Querzoni, "SAFE: Self-attentive function embeddings for binary similarity," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, R. Perdisci, C. Maurice, G. Giacinto, and M. Almgren, Eds. Cham: Springer, 2019, pp. 309–329.

[9] X. Ren, M. Ho, J. Ming, Y. Lei, and L. Li, "Unleashing the hidden power of compiler optimization on binary code difference: An empirical study," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, ser. PLDI 2021. New York, NY, USA: Association for Computing Machinery, 2021, pp. 142–157. [Online]. Available: https://doi.org/10.1145/3453483.3454035

[10] K. Pei, Z. Xuan, J. Yang, S. Jana, and B. Ray, "Trex: Learning execution semantics from micro-traces for binary similarity," *arXiv preprint arXiv:2012.08680*, 2020.

[11] H. Wang, W. Qu, G. Katz, W. Zhu, Z. Gao, H. Qiu, J. Zhuge, and C. Zhang, "jTrans: Jump-Aware Transformer for Binary Code Similarity Detection," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2022. New York, NY, USA: Association for Computing Machinery, 2022, pp. 1–13. [Online]. Available: https://doi.org/10.1145/3533767.3534367

[12] Y. Kalantidis, M. B. Sariyildiz, N. Pion, P. Weinzaepfel, and D. Larlus, "Hard negative mixing for contrastive learning," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 21 798–21 809. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/f7cade80b7cc92b991cf4d2806d6bd78-Paper.pdf

[13] C.-Y. Chuang, J. Robinson, Y.-C. Lin, A. Torralba, and S. Jegelka, "Debiased contrastive learning," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 8765–8775. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/63c3ddcc7b23daa1e42dc41f9a44a873-Paper.pdf

[14] Z. Long, G. Killick, L. Zhuang, R. McCreadie, G. A. Camarasa, and P. Henderson, "Elucidating and overcoming the challenges of label noise in supervised contrastive learning," *arXiv preprint arXiv:2311.16481*, 2023.

[15] S. Jiang, C. Fu, S. He, J. Lv, L. Han, and H. Hu, "BinCola: Diversity-sensitive contrastive learning for binary code similarity detection," *IEEE Transactions on Software Engineering*, vol. 50, no. 10, pp. 2485–2497, 2024.

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

[17] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[18] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020, pp. 1597–1607.

[19] L. Jing, P. Vincent, Y. LeCun, and Y. Tian, "Understanding dimensional collapse in contrastive self-supervised learning," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://arxiv.org/abs/2110.09348

[20] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, Jul. 2021, pp. 12 310–12 320. [Online]. Available: https://proceedings.mlr.press/v139/zbontar21a.html

[21] A. Bardes, J. Ponce, and Y. LeCun, "VICReg: Variance-invariance-covariance regularization for self-supervised learning," in *International Conference on Learning Representations*, Apr. 2022. [Online]. Available: https://inria.hal.science/hal-03541297

[22] Z. D. Guo, B. A. Pires, K. Khetarpal, D. Schuurmans, and B. Dai, "Representation learning via non-contrastive mutual information," *arXiv preprint arXiv:2504.16667*, 2025.

[23] G. Mialon, R. Balestriero, and Y. LeCun, "Variance covariance regularization enforces pairwise independence in self-supervised representations," *arXiv preprint arXiv:2209.14905*, 2022.

[24] Q. Garrido, Y. Chen, A. Bardes, L. Najman, and Y. LeCun, "On the duality between contrastive and non-contrastive self-supervised learning," in *International Conference on Learning Representations*, 2023.

[25] S. Saha, D. Roy, and M. Mitra, "On modifying evaluation measures to deal with ties in ranked lists," in *Proceedings of the 22nd ACM/IEEE Joint Conference on Digital Libraries*, ser. JCDL '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 12:1–12:4. [Online]. Available: https://doi.org/10.1145/3529372.3533291

[26] M. Berrendorf, E. Faerman, L. Vermue, and V. Tresp, "On the ambiguity of rank-based evaluation of entity alignment or link prediction methods," *arXiv preprint arXiv:2002.06914*, 2020.

[27] I. U. Haq and J. Caballero, "A survey of binary code similarity," *ACM Computing Surveys*, vol. 54, no. 3, Apr. 2021. [Online]. Available: https://doi.org/10.1145/3446371

[28] S. H. H. Ding, B. C. M. Fung, and P. Charland, "Asm2Vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 472–489.

[29] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 363–376. [Online]. Available: https://doi.org/10.1145/3133956.3134018

[30] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The Llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[31] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei *et al.*, "Qwen2.5 Technical Report," 2025.

[32] Gemma Team and Google DeepMind, "Gemma: Open models based on Gemini research and technology," 2024. [Online]. Available: https://arxiv.org/abs/2403.08295

[33] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[34] N. Muennighoff, H. Su, L. Wang, N. Yang, F. Wei, T. Yu, A. Singh, and D. Kiela, "Generative representational instruction tuning," in *ICLR 2024 Workshop: How Far Are We From AGI*, 2024. [Online]. Available: https://openreview.net/forum?id=8cQrRO9iFe

[35] S. Myers, T. A. Miller, Y. Gao, M. M. Churpek, A. Mayampurath, D. Dligach, and M. Afshar, "Lessons learned on information retrieval in electronic health records: a comparison of embedding models and pooling strategies," *Journal of the American Medical Informatics Association*, vol. 32, no. 2, pp. 357–364, 2025.

[36] Y. Tang and Y. Yang, "Pooling and attention: What are effective designs for LLM-based embedding models?" *arXiv preprint arXiv:2409.02727*, 2024.

[37] C. Tao, T. Shen, S. Gao, J. Zhang, Z. Li, Z. Tao, and S. Ma, "LLMs are also effective embedding models: An in-depth overview," *arXiv preprint arXiv:2412.12591*, 2024.

[38] C. Lee, R. Roy, M. Xu, J. Raiman, M. Shoeybi, B. Catanzaro, and W. Ping, "NV-Embed: Improved techniques for training LLMs as generalist embedding models," in *International Conference on Learning Representations*, 2025.

[39] F. McSherry and M. Najork, "Computing information retrieval performance measures efficiently in the presence of tied scores," in *Advances in Information Retrieval*, C. Macdonald, I. Ounis, V. Plachouras, I. Ruthven, and R. W. White, Eds. Berlin, Heidelberg: Springer, 2008, pp. 414–421.

[40] D. Kim, E. Kim, S. K. Cha, S. Son, and Y. Kim, "Revisiting binary code similarity analysis using interpretable feature engineering and lessons learned," *IEEE Transactions on Software Engineering*, pp. 1–23, 2022.

[41] W. Zhu, H. Wang, Y. Zhou, J. Wang, Z. Sha, Z. Gao, and C. Zhang, "kTrans: Knowledge-aware transformer for binary code embedding," *arXiv preprint arXiv:2308.12659*, 2023.

[42] S. Yang, C. Dong, Y. Xiao, Y. Cheng, Z. Shi, Z. Li, and L. Sun, "Asteria-Pro: Enhancing deep learning-based binary code similarity detection by incorporating domain knowledge," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 1, Nov. 2023. [Online]. Available: https://doi.org/10.1145/3604611

[43] M. Amouei, B. C. M. Fung, and P. Charland, "FIN: Boosting binary code embedding by normalizing function inlinings," *Journal of Systems and Software*, vol. 231, p. 112603, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121225002729

[44] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 2201–2215. [Online]. Available: https://doi.org/10.1145/3133956.3134072

[45] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423/

[46] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[47] PyTorch ExecuTorch Team, "ExecuTorch Llama model deployment guide," 2024. [Online]. Available: https://github.com/pytorch/executorch/blob/main/examples/models/llama/README.md

[48] D. Han and M. Han, "Unsloth: Efficient LLM fine-tuning and inference," 2024. [Online]. Available: https://github.com/unslothai/unsloth

[49] C. Molloy, P. Charland, S. H. H. Ding, and B. C. M. Fung, "JARV1S: Phenotype clone search for rapid zero-day malware triage and functional decomposition for cyber threat intelligence," in *2022 14th International Conference on Cyber Conflict: Keep Moving! (CyCon)*, 2022, pp. 385–403.

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

[51] K. He, F. Cakir, S. A. Bargal, and S. Sclaroff, "Hashing as tie-aware learning to rank," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2018.

Mohammadhossein Amouei is a Ph.D. candidate in the School of Information Studies at McGill University, Canada. His research focuses on applying machine learning techniques to address challenges in cybersecurity. His doctoral work aims to develop robust methods for automated binary analysis to enhance threat detection and vulnerability assessment. He holds a Master's degree in Artificial Intelligence and a Bachelor's degree in Computer Engineering both from Shahrood University of Technology, Iran.

Benjamin C. M. Fung is a Canada Research Chair in Data Mining for Cybersecurity, a Full Professor with the School of Information Studies, and an Associate Member with the School of Computer Science at McGill University in Canada. He received a Ph.D. degree in computing science from Simon Fraser University in Canada in 2007. He has over 190 refereed publications that span the research forums of data mining, privacy protection, cybersecurity, services computing, and building engineering with h-index 63. His data mining works in crime investigation and authorship analysis have been reported by media worldwide. Prof. Fung is a licensed Professional Engineer of software engineering in Ontario, Canada.

Philippe Charland is a Defence Scientist at Defence Research and Development Canada – Valcartier Research Centre, in the Mission Critical Cyber Security Section. His research interests encompass software reverse engineering and computer forensics. As a member of the Systems Vulnerabilities and Lethality Group, his research focuses on binary-level program comprehension to accelerate the reverse engineering process involved in malicious software analysis and classification, as well as for mission assurance. Mr. Charland holds a bachelor and a master's degree in Computer Science, both from Concordia University, Montreal, Canada.

Jun Meng is a Ph.D. student in the School of Information Studies at McGill University. He received his Bachelor's degree in Mathematics from the University of Waterloo. His research focuses on natural language processing, deep learning, and distributed AI systems, with particular interests in hypernetworks, meta-learning, and domain generalization for adaptive machine learning models.

APPENDIX A
AXIOMATIC PROPERTIES OF TSRR

Following the axiomatic tradition in IR evaluation, we require any reciprocal-rank variant M to satisfy the following desiderata:

(i) *Range boundedness:* $0 \le M \le 1$.
(ii) *Strict monotonicity:* If the first relevant document moves to an earlier rank (or a tighter tie), then M strictly increases.
(iii) *Reduction to classical RR:* In the absence of ties, $M = 1/r$, where r is the rank of the first relevant document.
(iv) *Balanced tie handling:* Award partial credit within ties, but penalize larger tie groups so that indiscriminate "no-ranking" cannot outperform genuine but imperfect rankings.

Proposition A.1. *TsRR satisfies (i)–(iv) above, and in the extreme case of a single tie covering all R retrieved documents (with exactly one relevant document), TsRR reduces to the PRR.*

Proof. **(i) Range boundedness.** Since $r_{\text{pre}} \ge 0$, $E[L] \ge 1$, $L_{\max} \ge 1$, and $0 \le \tau \le 1$, it follows that

$$r_{\text{pre}} + E_\tau[L] \;\ge\; 1, \tag{29}$$

hence $0 < \text{TsRR} = 1/(r_{\text{pre}} + E_\tau[L]) \le 1$.

(ii) Strict monotonicity. Decreasing r_{pre} (moving the tie group up) or decreasing $E_\tau[L]$ (breaking the tie in favor of the relevant document) strictly decreases the denominator $r_{\text{pre}} + E_\tau[L]$, and thus strictly increases TsRR.

(iii) Reduction to classical RR. When there are no ties, $|G| = k$ so $\tau = 0$ and $E[L] = 1$. Thus

$$E_\tau[L] = 1, \quad \text{TsRR} = \frac{1}{r_{\text{pre}} + 1} = \frac{1}{r}, \tag{30}$$

recovering the standard reciprocal rank.

(iv) Balanced tie handling. By interpolating between the expected rank $E[L]$ (as in ta-RR) and the worst-case rank $L_{\max}$ (as in PRR), weighted by τ, TsRR (a) grants partial credit within ties and (b) increasingly penalizes larger tie groups (since τ grows with $|G_{\text{irr}}|$). Consequently, indiscriminate ties cannot outperform a system that genuinely distinguishes documents.

Reduction to PRR under full tie. If all R documents form a single tie and exactly one is relevant, then

$$r_{\text{pre}} = 0, \quad |G| = R, \quad k = 1, \quad N_{\text{irr}} = R - 1 \implies \tau = 1, L_{\max} = R,$$

so

$$E_\tau[L] = R, \quad \text{TsRR} = \frac{1}{R}, \tag{31}$$

which matches the pessimistic reciprocal rank $\text{PRR} = 1/R$. □

The sensitivity to tie size is precisely where TsRR departs from ta-RR. Whereas ta-RR assigns the same score whenever $r_{\text{pre}} + E[L]$ is equal, TsRR introduces the additional adjustment factor τ that grows with the number of irrelevants tied to the first relevant item. As a result, the effective denominator

$$r_{\text{pre}} + E_\tau[L] \;=\; r_{\text{pre}} + E[L] + \tau\,(L_{\max} - E[L]), \tag{32}$$

is strictly larger for larger tie groups, even when the value of $r_{\text{pre}} + E[L]$ is identical. Consequently, TsRR assigns *higher* scores to runs with smaller ties. Thus, TsRR breaks the indifference of ta-RR: two systems that look equivalent under ta-RR because they share the same $r_{\text{pre}} + E[L]$ are correctly distinguished under TsRR, with preference given to the system whose relevant item lies in a smaller, more informative tie group.

APPENDIX B
SIMULATION-DRIVEN EVALUATION OF TSRR BEHAVIOR

Even if certain edge cases, such as situations where two ranking outputs share the same expected rank but differ in tie-group size, arise only infrequently in real-world corpora, a robust evaluation metric must handle them consistently. Frequency of occurrence cannot justify biased scoring: a metric that assigns identical scores to configurations with markedly different levels of uncertainty fails to reflect true performance differences. Therefore, fairness demands that variation in tie-group size or rank position, however rare, be properly accounted for to provide a more accurate and trustworthy assessment of ranking-system behavior.

In real-world corpora, many of the tie–rank configurations needed to expose the boundary behavior of reciprocal-rank-based metrics may occur only rarely, making it difficult to evaluate TsRR comprehensively using standard BCSD test collections alone. To address this limitation, we conduct a simulation-based study in which we construct controlled synthetic ranking scenarios. These scenarios are intended as analytical edge cases rather than empirical models of the most common ranking patterns in BCSD. Their purpose is to isolate when RR, PRR, ta-RR, and TsRR agree, when they diverge, and when pathological cases arise.

The completeness of these scenarios follows from the structure of the tied-ranking problem. For a single relevant document under ties, the behavior of reciprocal-rank-based metrics is governed by two primitive quantities: (i) the number of items ranked before the tie group containing the relevant document, denoted r_{pre}, and (ii) the size of that tie group, denoted $S = |G|$. Equivalently, the group rank is $R = r_{\text{pre}} + 1$, and the expected position of the relevant document under random tie-breaking is $r_{\text{pre}} + E[L]$, where L is the position of the relevant document within its tie group. Therefore, any tied-ranking configuration can be represented as a point in the two-dimensional space (r_{pre}, S). Qualitative changes in metric behavior can arise only from changes in rank position, changes in tie size, or interactions between the two.

Based on this observation, we examine four primary trajectory scenarios:

1. Tie size decreases from fully tied to fully ordered while the group rank worsens.
2. Tie size decreases from fully tied to fully ordered while the group rank remains at its best.
3. Tie size remains constant while the group rank improves.
4. The expected rank $r_{\text{pre}} + E[L]$ remains constant while tie size increases.

Together, these scenarios form a compact but systematic coverage set over the two factors that determine tied

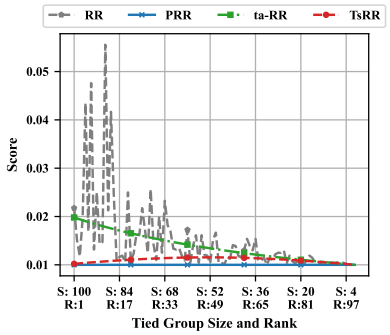

Fig. 11: Scenario 1: Tie size $S = |G|$ decreases from fully tied to fully ordered while the group rank R worsens. Comparison of RR, PRR, ta-RR, and TsRR in this edge case.

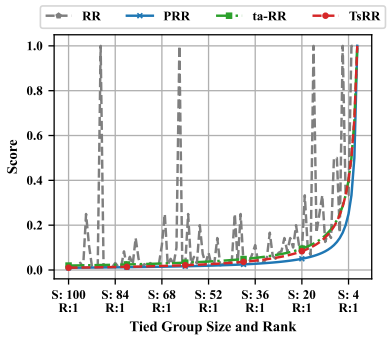

Fig. 12: Scenario 2: Tie size $S = |G|$ decreases from fully tied to fully ordered while the group rank R remains at its best. Comparison of RR, PRR, ta-RR, and TsRR in this edge case.

reciprocal-rank behavior: rank position and tie-group size. Scenarios 2 and 3 isolate the two individual axes: reducing tie uncertainty at a fixed best rank and improving rank position at a fixed tie size. Scenario 1 captures their interaction when the two axes move in opposite directions, i.e., the system becomes more discriminative but places the relevant item worse. Scenario 4 captures an equivalence class under expected-rank-based scoring, where two configurations have the same expected rank but different tie-group sizes and hence different levels of uncertainty. This mapping allows us to test agreement cases, monotonicity cases, divergence cases, and pathological cases in which a metric fails to distinguish rankings with different uncertainty levels. Thus, the simulations are complete with respect to the qualitative boundary behaviors induced by (r_{pre}, S), although they are not intended to enumerate all empirical ranking trajectories that may occur in real BCSD corpora.

In addition to the four primary scenarios, we isolate the individual effects of tie size and rank by comparing TsRR on pairs of configurations in which:

(i) tie sizes differ while r_{pre} is held constant;
(ii) r_{pre} differs while tie sizes are held constant.

The practical frequency with which these behaviors arise in real BCSD evaluations is examined separately in RQ5. In all simulation experiments, we assume a retrieval set of 100 documents, comprising one relevant document and 99 irrelevant documents.

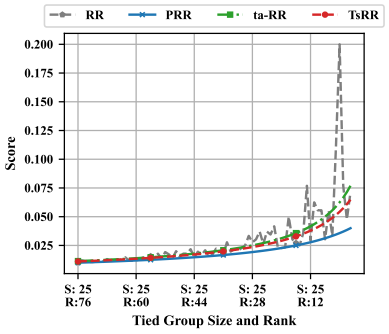

Fig. 13: Scenario 3: Tie size $S = |G|$ remains constant while the group rank R improves. Comparison of RR, PRR, ta-RR, and TsRR in this edge case.

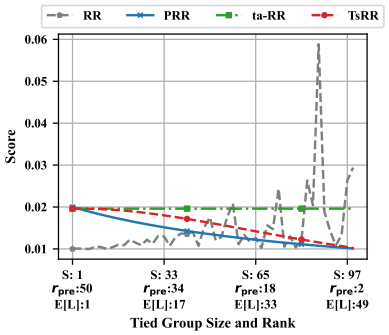

Fig. 14: Scenario 4: Expected rank $r_{\text{pre}} + E[L]$ remains constant while tie size $S = |G|$ increases. Comparison of RR, PRR, ta-RR, and TsRR in this edge case.

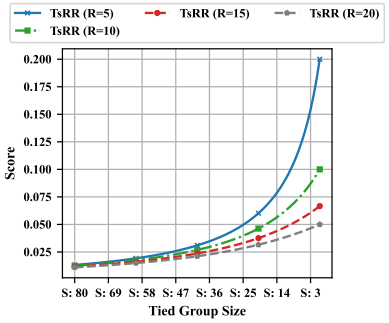

Fig. 15: Monotonicity Experiment 1: Four fixed ranks R with systematically decreasing tie sizes $S = |G|$. TsRR increases monotonically as S decreases for constant R.

Fig. 11 presents the results for Scenario 1, in which the ranking system initially treats all retrieved documents as tied, then gradually achieves full discrimination, but ultimately positions the relevant document last. As the plot illustrates, RR fluctuates unpredictably because the position of the relevant item within the tied group is random. PRR remains constant because the worst possible rank does not change, while ta-RR paradoxically assigns a higher score to a system that never differentiates among documents than to one that places the

TABLE XII: Coverage of qualitative metric behaviors by the four simulation scenarios.

Scenario	Controlled variation	Metric behavior tested	Purpose
Scenario 1	S decreases while r_{pre} worsens	Conflicting effects between tie reduction and rank degradation	Tests whether a metric can balance improved discrimination against worse placement of the relevant item.
Scenario 2	S decreases while r_{pre} remains optimal	Tie-size effect at best rank	Tests whether a metric rewards reduced uncertainty when the relevant tie group remains at the top.
Scenario 3	r_{pre} improves while S remains fixed	Rank-position effect at fixed uncertainty	Tests whether a metric monotonically rewards improved placement independent of tie size.
Scenario 4	$r_{\text{pre}} + E[L]$ remains constant while S changes	Equal expected rank but different tie uncertainty	Tests whether a metric distinguishes rankings that ta-RR treats as equivalent despite different uncertainty levels.

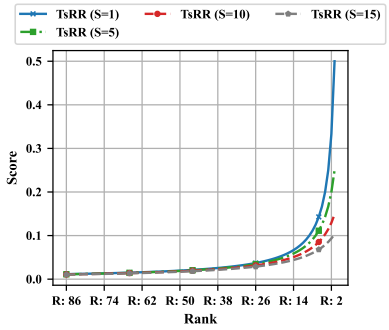

Fig. 16: Monotonicity Experiment 2: Four fixed tie sizes $S = |G|$ with systematically improving ranks R. TsRR increases monotonically as R improves for constant S.

relevant item last. This behavior is undesirable: a complete tie over the entire retrieval set reflects a total failure to prioritize the relevant document, even though its expected rank may appear less severe than the worst possible rank. In contrast to ta-RR, TsRR attains its minimum when all documents are tied. As the tie group begins to shrink, TsRR increases, peaking when the system can distinguish half of the retrieved items even though it still relegates the relevant document to the lower half. This initial increase reflects the fact that partial discrimination is better than no discrimination. From that peak onward, as the system moves from partial discrimination toward the second failure point of ranking the relevant document last, TsRR declines until it returns to its minimum at the worst case. By jointly accounting for both tie-group size and rank position, TsRR captures the trade-off between uncertainty reduction and ranking precision.

Figs. 12 and 13 present the results for Scenarios 2 and 3, respectively. These two scenarios isolate the two monotonic dimensions of tied ranking. In Scenario 2, the relevant tie group remains at the best rank while the tie size decreases. A metric should therefore assign higher scores as uncertainty decreases, because fewer irrelevant documents are indistinguishable from the relevant document. In Scenario 3, the tie size remains fixed while the group rank improves. A metric should therefore assign higher scores as the relevant group moves closer to the top of the ranked list. In both cases, RR continues to fluctuate unpredictably, whereas PRR, ta-RR, and TsRR exhibit clear upward trends as tie size or ranking performance improves. Notably, TsRR consistently falls between ta-RR and PRR, striking a middle ground: it is neither as lenient as ta-RR nor as stringent as PRR.

In the final scenario, we analyze cases where the expected rank $r_{\text{pre}} + E[L]$ is held constant, but the size of the tie group containing the relevant document changes. While the expected rank remains unchanged, smaller tie groups reduce the positional uncertainty of the relevant document, making the configuration inherently more favorable. A well-designed metric should therefore assign higher scores when fewer irrelevant documents are tied with the relevant one. As illustrated in Fig. 14, ta-RR fails to differentiate between tie groups of different sizes when their expected ranks are equal, whereas PRR and TsRR correctly reward the improvement resulting from reduced tie size. This case is particularly important because it exposes a failure mode that is invisible to metrics based only on expected rank.

To empirically validate TsRR's monotonicity, we conducted two controlled experiments. In the first experiment, we fixed four distinct rank positions and systematically reduced the size of the tie group; the results are shown in Fig. 15. In the second experiment, we fixed four tie-group sizes and systematically improved rank positions; these findings appear in Fig. 16. In both cases, TsRR behaves monotonically: for a constant tie-group size, a better rank yields a higher TsRR score, and for a constant rank, a smaller tie group likewise produces a higher TsRR score. These monotonicity experiments complement the four trajectory scenarios by confirming that TsRR responds consistently to isolated improvements along each of the two primitive dimensions of tied-ranking behavior.

APPENDIX C
KNOWN VULNERABILITY DETECTION DATASETS

This appendix provides the complete list of known vulnerabilities used in our vulnerability-detection experiments. We include these details to make the construction of the evaluation set transparent and to clarify which projects and CVEs are used as unseen test cases. Table XIII reports the 127 CVEs collected from five unseen projects in the main vulnerability-detection dataset. Table XIV reports the additional CVEs

used from the BinKit-CVE dataset. These datasets allow us to evaluate whether binary-similarity methods can retrieve vulnerable functions when the query and candidate binaries are compiled under different compiler and optimization settings.

TABLE XIII: CVE details for the vulnerability detection experiments. The table maps each project to its associated CVEs. In total, we collected 127 CVEs from five unseen projects.

Project	CVEs
libpng	CVE-2015-7981, CVE-2015-8540, CVE-2015-8126, CVE-2015-8472
FreeType	CVE-2012-1142, CVE-2014-9664, CVE-2010-3855, CVE-2014-9662, CVE-2017-8287, CVE-2014-9669, CVE-2014-9661, CVE-2014-9747, CVE-2012-1141, CVE-2017-7858, CVE-2012-1132, CVE-2012-1129, CVE-2014-9663, CVE-2014-9746, CVE-2014-9660, CVE-2010-2805, CVE-2012-1140, CVE-2012-1135, CVE-2014-9656, CVE-2012-5668, CVE-2014-9659, CVE-2012-1144, CVE-2014-9658
OpenSSL	CVE-2014-0221, CVE-2015-0209, CVE-2014-3508, CVE-2016-6302, CVE-2015-1789, CVE-2015-0205, CVE-2015-0288, CVE-2014-8275, CVE-2015-3196, CVE-2014-3510, CVE-2016-2176, CVE-2016-2108, CVE-2014-0198, CVE-2014-3569, CVE-2016-6306, CVE-2015-0292, CVE-2016-6303, CVE-2014-3513, CVE-2015-1793, CVE-2015-1790, CVE-2014-0224, CVE-2014-3512, CVE-2016-2105, CVE-2016-2182, CVE-2016-2177, CVE-2014-3507, CVE-2015-1788, CVE-2015-0293, CVE-2015-3195, CVE-2016-2109, CVE-2016-0705, CVE-2015-0206, CVE-2016-2178, CVE-2016-2180, CVE-2010-5298, CVE-2016-6304, CVE-2014-3568, CVE-2016-0797, CVE-2013-0166, CVE-2015-0286, CVE-2016-2106, CVE-2016-0798, CVE-2014-0160, CVE-2015-0289, CVE-2015-1791, CVE-2014-8176
libxml2	CVE-2015-5312, CVE-2015-8317, CVE-2015-8241, CVE-2015-7498, CVE-2016-5131, CVE-2013-2877, CVE-2015-7500, CVE-2015-7941, CVE-2015-7499
tcpdump	CVE-2017-12995, CVE-2017-13039, CVE-2017-12992, CVE-2017-13015, CVE-2017-13043, CVE-2017-12896, CVE-2017-11542, CVE-2017-13004, CVE-2017-12989, CVE-2017-13046, CVE-2017-12998, CVE-2017-13009, CVE-2017-13045, CVE-2017-13029, CVE-2017-13008, CVE-2017-13040, CVE-2017-13688, CVE-2017-13690, CVE-2017-13012, CVE-2017-12985, CVE-2017-13017, CVE-2017-13050, CVE-2017-12902, CVE-2017-13000, CVE-2017-12901, CVE-2017-13687, CVE-2017-13028, CVE-2017-13689, CVE-2017-13725, CVE-2017-12899, CVE-2017-12990, CVE-2017-13021, CVE-2017-13034, CVE-2017-13002, CVE-2017-13049, CVE-2017-13033, CVE-2017-13032, CVE-2017-13051, CVE-2017-12898, CVE-2017-13035, CVE-2017-13007, CVE-2017-11541, CVE-2017-13031, CVE-2017-12897, CVE-2017-12993

TABLE XIV: CVEs from the unseen projects in the BinKit-CVE dataset.

Project	CVEs
a2ps-4.14	CVE-2015-8107
tar-1.34	CVE-2022-48303
sharutils-4.15.2	CVE-2018-1000097
cpio-2.12	CVE-2010-4226, CVE-2019-14866, CVE-2021-38185
cflow-1.7	CVE-2023-2789
patch-2.7.5	CVE-2016-10713, CVE-2018-6951, CVE-2018-6952, CVE-2019-13636, CVE-2018-20969, CVE-2019-20633
libmicrohttpd-0.9.75	CVE-2023-27371