BDLF-Qwen3: Enhanced Cross-Architecture Binary Function Similarity Detection Through Binary Dynamic Layer Fusion

Yuanda Wang[1], Ji Zhou[3], Xinhui Han[1]*, Chao Zhang[2]*

[1]Wangxuan Institute of Computer Technology (WICT), Peking University, China
[2]Institute for Network Sciences and Cyberspace, Tsinghua University, China
[3]Faculty of Engineering, Monash University, Australia
yuandawang958@stu.pku.edu.cn, hanxinhui@pku.edu.cn, chaoz@tsinghua.edu.cn, ji.zhou@monash.edu

Abstract

Binary code analysis is essential for software security across various instruction set architectures. Cross-architecture binary function similarity detection faces significant challenges due to substantial differences in instruction sets and architectural conventions. Existing approaches struggle to capture relationships between code abstraction levels and lack comprehensive cross-architecture datasets for effective evaluation. Inspired by human cognitive processes of dynamically integrating multi-level information, we propose Binary Dynamic Layer Fusion (BDLF), a novel neural architecture that enhances cross-architecture similarity detection through adaptive layer-wise feature integration. BDLF leverages Qwen3's multilingual code understanding and introduces dynamic weight generation to optimally combine representations from all previous layers. We also construct Cross-Bin, a high-quality cross-architecture binary function dataset. BDLF-Qwen3 employs two-stage training: partial fine-tuning with pairwise similarity learning followed by BDLF enhancement with InfoNCE contrastive learning. Experiments demonstrate BDLF-Qwen3 significantly outperforms state-of-the-art methods, achieving 36-65% improvement in Recall@10 across diverse CPU architectures.

1 Introduction

Binary code analysis plays a critical role in modern cybersecurity, enabling tasks such as malware detection, vulnerability discovery, and intellectual property protection (David, Partush, and Yahav 2017; Xu et al. 2017). Modern software ecosystems routinely deploy applications across diverse instruction set architectures (ISAs), including x86, ARM, MIPS, and other platforms. This architectural diversity has reached unprecedented scale, with IoT deployments alone totaling 18.8 billion connected devices globally (IoT Business News 2024), creating complex mixed-architecture environments spanning x86 servers, ARM mobile devices, and MIPS network infrastructure.

Recent supply chain attacks highlight the critical security implications of this complexity. The SolarWinds attack demonstrates the need for cross-architecture analysis, as the 18,000+ affected customers operated mixed environments spanning Windows, Linux, and mobile platforms, while supply chain attacks such as Applied Materials resulted in $250 million losses across multiple processor architectures(Kalapatapu 2025). Without cross-architecture binary similarity detection, security teams cannot effectively correlate attacks against x86 development systems with compromises of ARM-based embedded devices, significantly extending breach detection and remediation times.

Cross-architecture binary function similarity detection has emerged as a fundamental problem in this domain (Pewny et al. 2015; Zuo et al. 2018). The core challenge lies in bridging the semantic gap between architecturally distinct binary representations while preserving the essential functional characteristics that define code similarity. Existing approaches primarily rely on static feature extraction methods, including control flow graphs (CFGs) (Feng et al. 2016), intermediate representations (IRs) (David, Partush, and Yahav 2016), and traditional neural embeddings (Ding, Fung, and Charland 2019). However, these methods struggle with several fundamental limitations: (1) they fail to capture dynamic semantic relationships between different abstraction levels of code representation, (2) they often overlook the complementary information available across architectural boundaries, and (3) they lack the adaptive capability to optimally integrate multi-level features based on specific similarity detection contexts.

Recent advancement in natural language processing, particularly large language models (LLMs), has demonstrated remarkable capabilities in understanding code semantics across different programming languages (Chen et al. 2021; Wang et al. 2023). The emergence of multilingual code models like Qwen3 (Yang et al. 2025) presents new opportunities for cross-architecture binary analysis. However, directly applying these models to binary code analysis faces significant challenges due to the fundamental differences between high-level source code and low-level assembly instructions, as well as the need for architecture-specific adaptations.

Inspired by human cognitive processes that dynamically integrate information from multiple abstraction levels when analyzing code, we propose **Binary Dynamic Layer Fusion (BDLF)**, a novel neural architecture that enhances cross-architecture binary function similarity detection through adaptive layer-wise feature integration. As illustrated in Figure 1, our comprehensive framework addresses the limi-

*Corresponding authors

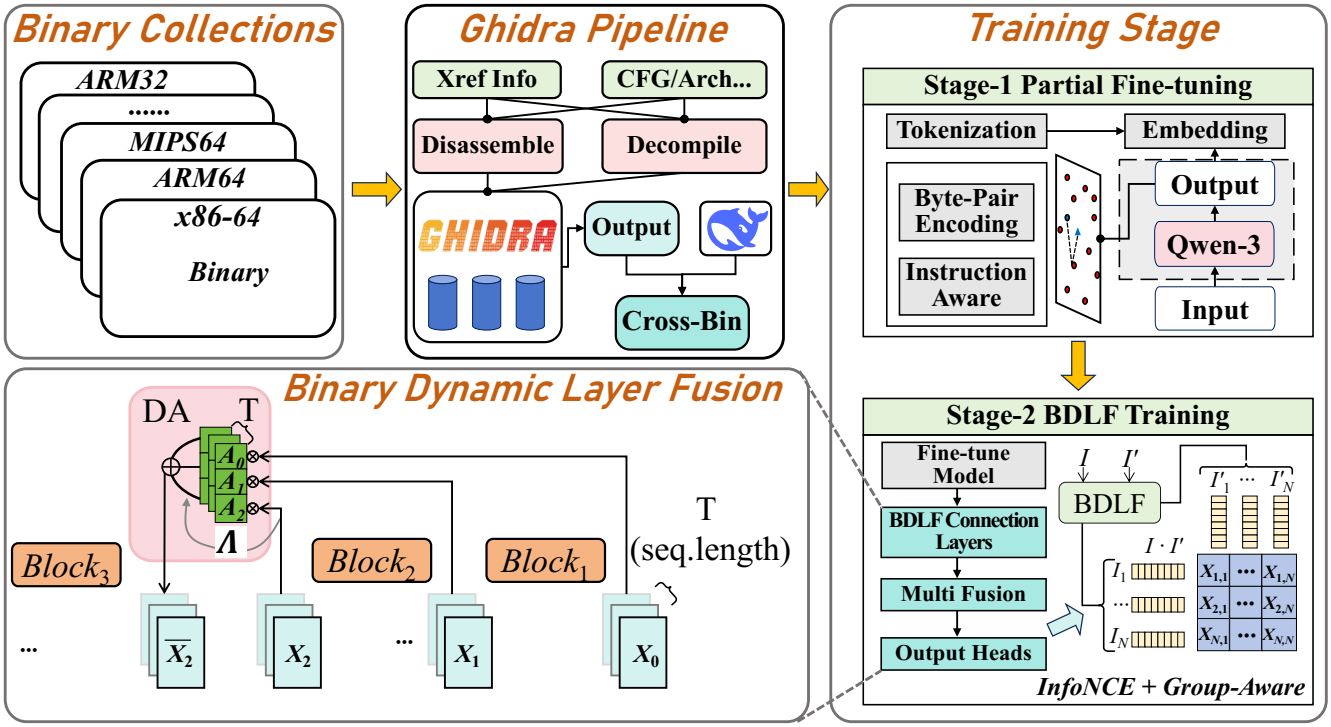

Figure 1: Overview of BDLF-Qwen3. The Ghidra pipeline constructs the Cross-Bin dataset, followed by two-stage training: Stage 1 pairwise similarity fine-tuning and Stage 2 BDLF enhancement via InfoNCE contrastive learning.

tations of existing static methods by introducing dynamic weight generation mechanisms that optimally combine representations from all transformer layers based on the specific characteristics of input binary functions.

The key insight underlying our approach is that different transformer layers capture different code semantics, from low-level syntactic patterns in the early layers to high-level semantic abstractions in deeper layers (Goldberg 2016). Rather than using only final layer representations, BDLF dynamically weights and fuses features from all layers, adaptively emphasizing the most relevant semantic levels for each task. Our framework processes binary collections through Ghidra's analysis pipeline to construct the Cross-Bin dataset, followed by a two-stage training that progressively builds cross-architecture understanding.

Our contributions are summarized as follows:

- We build **Cross-Bin**, a comprehensive cross-architecture binary function dataset containing 547,000 unique functions across six major instruction set architectures with 156,000 semantically equivalent function groups. Our automated pipeline leverages Ghidra's enhanced cross-reference (Xref) processing and DeepSeek-V3 for high-quality semantic annotations.
- We propose **BDLF**, a novel neural architecture that dynamically integrates multi-layer transformer representations for enhanced cross-architecture binary function similarity detection. To our knowledge, this is the first work to introduce BDLF mechanisms on Qwen3-Embedding for cross-architecture binary analysis, establishing a new training framework that combines pairwise similarity learning with contrastive optimization.
- We propose a **two-stage training framework** that enhances the detection of similarity between architecture binary functions. Our comprehensive experimental evaluation demonstrates that this two-stage approach achieves superior performance, with BDLF-Qwen3 outperforming state-of-the-art methods by 36-65% in Recall@10 across large-scale cross-architecture scenarios and maintaining robust zero-shot generalization capabilities on unseen RISC-V architecture.

2 Related Work

2.1 Binary Code Similarity Detection

Binary code similarity detection (BCSD) addresses the fundamental challenge of identifying functionally equivalent code across different ISAs, compilation settings, and optimization levels (Li, Qu, and Yin 2021; Huang, Youssef, and Debbabi 2017; Kim et al. 2022). Traditional approaches employed static analysis techniques such as CFGs and hand-crafted features (Flake 2004; Dullien and Rolles 2005), while early cross-architecture solutions like Multi-MH (Pewny et al. 2015), Esh (David and Yahav 2014), and discovRE (Eschweiler et al. 2016) relied on intermediate representations and graph matching algorithms (Feng et al. 2016; Xu et al. 2017).

Recent neural approaches have demonstrated significant potential for cross-architecture analysis (Massarelli et al.

2019; Liu et al. 2018; Feng et al. 2020; Hu et al. 2016; Wang et al. 2024). Methods like TREX (Pei et al. 2020), jTrans (Wang et al. 2022), and BinDiffNN (Ullah and Oh 2021) address the BCSD problem through transfer learning, transformer architectures, and neural program embeddings respectively. Graph-based neural networks including HBin-Sim (Wang et al. 2021), GMN (Li et al. 2019), and Order Matters (Yu et al. 2020) have advanced the field through hierarchical matching networks and semantic-aware graph representations.

2.2 Multi-Stage Training and Feature Fusion

Multi-stage training has emerged as a powerful paradigm for developing robust representation learning models, particularly in embedding and similarity detection tasks (Yu et al. 2024). In text embedding, the GTE series (Li et al. 2023) demonstrates the effectiveness of progressive contrastive learning that combines large-scale unsupervised pre-training with supervised fine-tuning for enhanced generalization. The BGE family (Xiao et al. 2024) further advances multi-stage approaches through unified training across dense, sparse, and multi-vector representations. Recent work has extended multi-stage paradigms to foundation model adaptation, with Qwen3-Embedding (Zhang et al. 2025) implementing sophisticated three-stage pipelines that effectively balance model generalization and task-specific performance.

Feature fusion strategies have evolved from simple concatenation to sophisticated attention-based mechanisms and dynamic weighting schemes (Lahat, Adali, and Jutten 2015; Atrey et al. 2010). Multi-scale feature fusion through pyramid networks (Lin et al. 2017) and progressive alignment methods (Zhao et al. 2017) have achieved significant improvements across various domains (Liu et al. 2022; Dosovitskiy et al. 2020). However, existing approaches typically employ static fusion strategies that lack the adaptivity required for complex cross-architectural scenarios, where optimal feature combinations may vary significantly based on the specific instruction sets, compilation settings, and semantic contexts.

3 Method

In this section, we provide a detailed introduction of BDLF-Qwen3 framework, including the Cross-Bin dataset and training pipeline for effective cross-architecture binary function similarity detection through Binary Dynamic Layer Fusion.

3.1 Dataset Engine of Cross-Bin

Recent advances in foundation model training have demonstrated that data quality and diversity are more critical than sheer quantity (Seawead et al. 2025). Following this principle, we build Cross-Bin (Figure 2), a comprehensive cross-architecture binary function dataset specifically designed for multi-modal similarity detection across diverse instruction set architectures. Rather than pursuing maximum scale, we prioritize careful curation of high-quality, semantically-rich binary functions that capture the full spectrum of real-world code patterns.

Our collection strategy encompasses three complementary approaches ensuring comprehensive architectural coverage across six major ISAs (x86-32/64, ARM32/64 and MIPS32/64). We systematically collect **real-world applications**, **cross-compiled open source software**, and **security-critical samples** following established research practices (Luo et al. 2023; Kim et al. 2022).

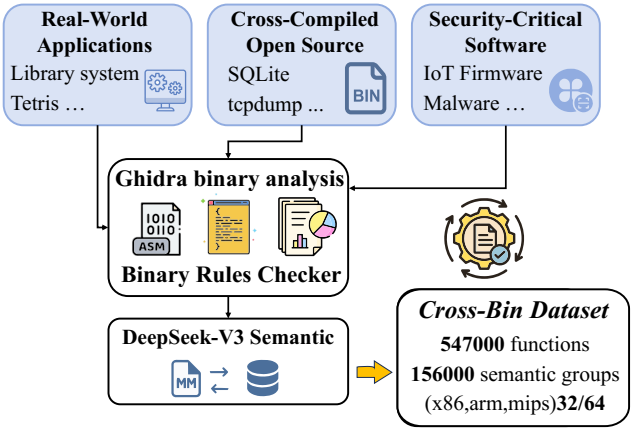

Figure 2: Cross-Bin dataset construction pipeline.

XRef-Enhanced Quality Improvement. Our automated pipeline leverages Ghidra's binary analysis capabilities with DeepSeek-V3 (Liu et al. 2024; Eagle and Nance 2020) for comprehensive semantic annotation. For each function, Ghidra extracts assembly instructions, pseudo-code representations, and cross-reference information, including function calls, data references, and control dependencies. Our XRef analysis module resolves indirect calls to actual function names, replaces raw memory addresses with meaningful symbols, and provides context-aware parameter type inference. This process transforms low-quality pseudo-code with cryptic addresses into readable code with resolved symbols and inferred types, enabling more accurate semantic understanding.

Semantic Annotation Generation. DeepSeek-V3 generates natural language descriptions and parameter annotations using the XRef-enhanced representations. Our prompts leverage both pseudo code details and high-level semantic context from cross-reference analysis, enabling architecturally-neutral semantic understanding while preserving architecture-specific implementation details. Cross-Bin contains 547,000 unique functions across six architectures with 156,000 semantically-equivalent function groups. Quality assurance includes automated consistency checking and expert validation of cross-architecture equivalence.

3.2 BDLF-Qwen3 Architecture

We present BDLF, a novel method for cross-architecture binary function similarity detection. Our approach dynamically integrates multi-layer representations from Qwen3-Embedding-0.6B(Yang et al. 2025) to capture hierarchical semantic features across different instruction set architectures.

Problem Formulation. Given a set of binary functions $\mathcal{F} = \{f_1, f_2, ..., f_n\}$ compiled for different architectures $\mathcal{A} = \{a_1, a_2, ..., a_m\}$, our goal is to learn an embedding function $\phi : \mathcal{F} \times \mathcal{A} \to \mathbb{R}^d$ such that semantically equivalent functions have similar representations regardless of their target architecture.

Base Representation. For a binary function f with architecture a, we first extract its textual representation x_f (assembly and pseudo-code) and compute the base embedding:

$$\mathbf{h}_0 = \text{Qwen3}(x_f) + \alpha \cdot \mathbf{e}_a \tag{1}$$

where $\mathbf{e}_a \in \mathbb{R}^d$ is a learnable architecture embedding and α is a scaling factor.

Dynamic Layer Fusion and Aggregation. The core of BDLF is a dynamic fusion mechanism that adaptively combines representations from multiple layers. For layer $l \in \{1, ..., L\}$, we compute:

$$\mathbf{w}_l = \text{softmax}(\mathbf{W}_g \sigma(\mathbf{W}_h \mathbf{h}_{l-1})) \tag{2}$$

where $\mathbf{w}_l \in \mathbb{R}^l$ represents the attention weights over all previous layers. The layer update is then:

$$\mathbf{h}_l = \text{LN}\left(\sum_{i=0}^{l-1} w_l^{(i)} \mathbf{h}_i + \mathbf{W}_f \sigma(\mathbf{W}_t \mathbf{h}_{l-1})\right) \tag{3}$$

Unlike static approaches, our dynamic weights $\mathbf{w}_l$ are computed based on the current hidden state, allowing the model to adaptively select relevant information from different semantic levels.

After processing through L BDLF layers, we obtain a set of layer representations $\mathcal{H} = \{\mathbf{h}_0, \mathbf{h}_1, ..., \mathbf{h}_L\}$. To effectively capture multi-scale semantic information, we employ self-attention and the final embedding is obtained through pooling and projection.

3.3 Training Strategy

We employ a two-stage training strategy that progressively builds cross-architecture understanding. The complete procedure is summarized in Algorithm 1.

Stage 1: Base Model Fine-tuning. In the first stage, we partially fine-tune Qwen3-Embedding using pairwise similarity regression on function pairs. Given pairs $\{(x_i^{(1)}, x_i^{(2)}, y_i)\}_{i=1}^B$ where $y_i \in [0, 1]$ indicates similarity:

$$\mathcal{L}_{\text{stage1}} = \frac{1}{B} \sum_{i=1}^{B} \| \cos(\mathbf{e}_i^{(1)}, \mathbf{e}_i^{(2)}) - y_i \|^2 \tag{4}$$

Stage 2: BDLF Enhancement. The second stage introduces BDLF layers and optimizes using InfoNCE loss. For anchor x_i with group g_i:

$$\mathcal{L}_{\text{BDLF}} = -\log \frac{\exp(\text{sim}(\mathbf{f}_i, \mathbf{f}_{i^+})/\tau)}{\sum_{k \neq i} \exp(\text{sim}(\mathbf{f}_i, \mathbf{f}_k)/\tau)} \tag{5}$$

where $i^+ \in \{j : g_j = g_i, j \neq i\}$ and τ is temperature.

Algorithm 1: BDLF-Qwen3 Training Procedure

Require: Dataset $\mathcal{D}$, Model $\mathcal{M}_{\text{base}}$, Hyperparameters Θ
Ensure: Trained BDLF model $\mathcal{M}_{\text{BDLF}}$

// Stage 1: Base Model Training
Initialize $\mathcal{M}_{\text{base}} \leftarrow$ Qwen3-Embedding-0.6B
for epoch $e = 1$ to E_1 **do**
  for batch $\mathcal{B} \in \mathcal{D}_{\text{pairs}}$ **do**
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{stage1}}(\mathcal{M}_{\text{base}}(\mathcal{B}))$
    Update $\mathcal{M}_{\text{base}}$ via gradient descent
  end for
end for
// Stage 2: BDLF Enhancement
Initialize BDLF layers on $\mathcal{M}_{\text{base}}$
for epoch $e = 1$ to E_2 **do**
  for batch $\mathcal{B} \in \mathcal{D}_{\text{groups}}$ **do**
    $\mathbf{h}_0 \leftarrow \mathcal{M}_{\text{base}}(\mathcal{B}) + \mathbf{E}_{\text{arch}}$
    for layer $l = 1$ to L **do**
      $\mathbf{w}_l \leftarrow \text{DynamicWeights}(\mathbf{h}_{l-1})$
      $\mathbf{h}_l \leftarrow \text{BDLFLayer}(\mathbf{h}_{l-1}, \{\mathbf{h}_i\}_{i=0}^{l-1}, \mathbf{w}_l)$
    end for
    $\mathbf{f} \leftarrow \text{MultiGranularityAggregation}(\{\mathbf{h}_l\}_{l=0}^{L})$
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{BDLF}}(\{\mathbf{f}_i\}, \{g_i\})$
    Update model via gradient descent
  end for
end for
return $\mathcal{M}_{\text{BDLF}}$

Through this two-stage training process, BDLF-Qwen3 first establishes a solid foundation in binary code understanding via pairwise similarity learning, then enhances its cross-architecture capabilities through dynamic layer fusion.

4 Experiments

We evaluate BDLF-Qwen3 on both function and instruction-level similarity tasks across diverse architectures. Extensive experiments demonstrate the superiority of our multi-stage training and dynamic layer fusion strategy.

4.1 Experimental Details

Experiments are performed on a machine equipped with Ubuntu 20.04 LTS. The machine has an Intel CPU (Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz), four NVIDIA GPUs (A100 PCIE) and 754GB RAM, and is installed with LLVM 12.0.1, GCC 7.5.0, libipt 2.0.0 (commit 892e12c5), Ghidra 11.3.2, and Python 3.12.4. Python is equipped with transformers 4.2.9, gensim 4.3.2 and PyTorch 2.4.1.

BDLF-Qwen3 is implemented on the Qwen3-Embedding-0.6B foundation using the Cross-Bin dataset, which is partitioned into a training set of 437,613 functions (2,184 binaries) and a test set of 109,387 functions (546 binaries). The training employs a two-stage process: **Stage 1** consists of partial fine-tuning with pairwise similarity learning for 1 epoch at a 3×10^{-5} peak learning rate, utilizing a progressive strategy that advances from sequences under 512 tokens to a 2:1 mixture of 0–512 and 512–1024 token-length data. **Stage 2** introduces BDLF layers optimized with InfoNCE loss for 2 epochs using a 1.5×10^{-5}

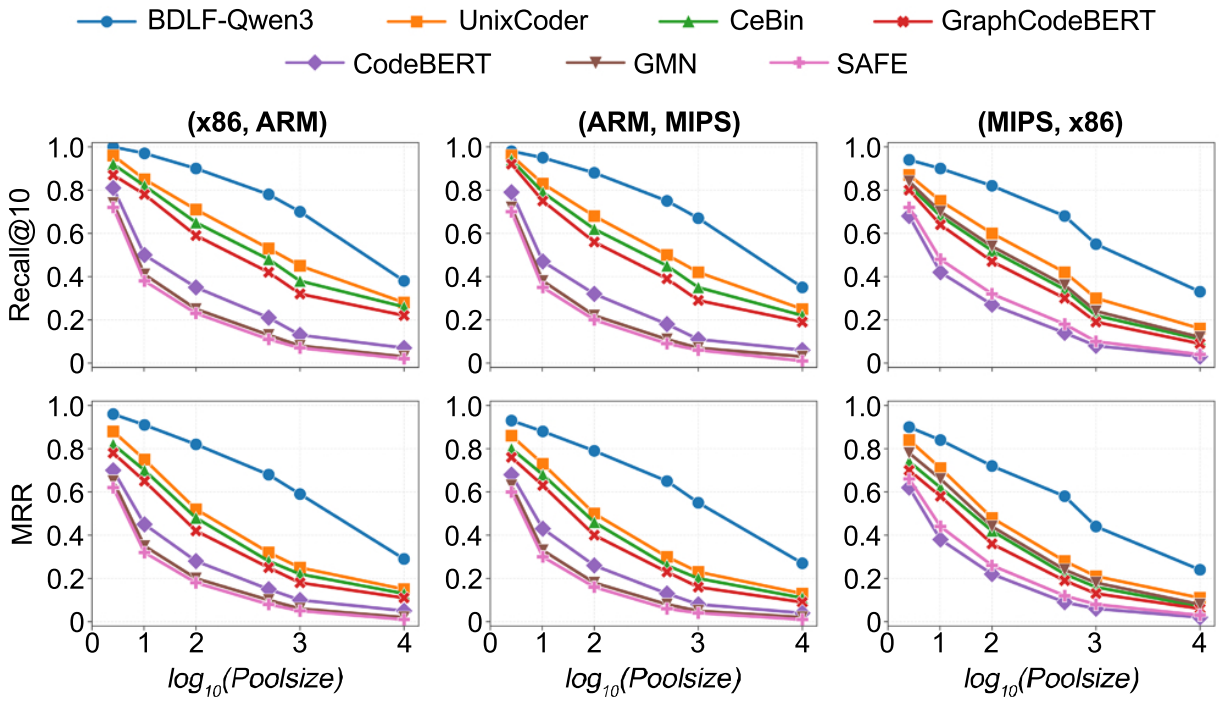

Figure 3: Cross-architecture binary similarity detection performance across different poolsizes. The top row shows Recall@10 results and the bottom row shows MRR results for three cross-architecture scenarios.

learning rate and temperature $\tau = 0.07$. Dynamic weights for the fusion mechanism are generated by a 2-layer MLP with ReLU activation and a 0.1 dropout rate, using a batch size of 32.

4.2 Experimental Results

Binary Similarity Detection Performance. We evaluate the cross-architecture binary similarity detection performance of our proposed BDLF-Qwen3 model against six state-of-the-art baselines: UnixCoder, CeBin, GraphCodeBERT, CodeBERT, GMN, and SAFE. The experiments are conducted across three distinct cross-architecture scenarios: x86→ARM, ARM→MIPS, and MIPS→x86. To comprehensively assess the scalability and robustness of each approach, we evaluate performance across varying poolsizes ranging from 16 to 10,000 functions.

The experimental results, detailed in Figure 3, demonstrate that BDLF-Qwen3 superiorly outperforms all baseline models across all evaluated configurations. For small pool sizes (16–32), BDLF-Qwen3 achieves Recall@10 scores consistently above 0.900, maintaining a 2.1%–4.3% lead over the strongest baseline. As the database scales, the performance advantage of our model becomes significantly more pronounced. At the largest pool size of 10,000, BDLF-Qwen3 retains a robust Recall@10 of up to 0.383, whereas the performance of baselines like SAFE and GMN degrades sharply to below 0.050. Specifically, in the x86→ARM scenario with a 10,000 pool size, BDLF-Qwen3 achieves a Recall@10 of 0.383 and an MRR of 0.293, representing a substantial improvement over UnixCoder (0.284 Recall@10 and 0.151 MRR). This sustained performance across large-scale databases underscores the robust semantic capturing capabilities of our LLM-based approach compared to traditional GNN or static embedding-based methods.

Semantic Consistency Analysis. To evaluate the quality of learned representations, we conduct a semantic consistency analysis using t-SNE visualization. We collect 500 binary functions across 10 semantic categories: networking, mathematics, encryption, sorting, search, authentication, file management, audio, graphics, and time operations (50 functions per category).

Figure 4 presents the t-SNE visualizations for all evaluated models, revealing significant differences in semantic understanding capabilities. Traditional methods like Trex exhibit poor semantic consistency with heavily overlapped embeddings and no discernible clustering patterns, while GraphCodeBERT and CeBin show marginal improvements with loose clusters for certain categories. UnixCoder achieves better semantic separation with more distinct clustering patterns. Our BDLF-Qwen3 model demonstrates dramatically superior semantic consistency, revealing well-separated, compact clusters for each semantic category with minimal inter-cluster overlap. Quantitative assessment using silhouette coefficients confirms this observation: BDLF-Qwen3 achieves a score of 0.72, significantly outperforming UnixCoder (0.45), CeBin (0.31), GraphCodeBERT (0.18), and Trex (0.09). This superior semantic understanding enables our model to focus on high-level algorithmic patterns

rather than low-level architectural artifacts.

Zero-Shot New Architecture Generalization. To evaluate the generalization capability of our model to completely unseen architectures, we conduct a zero-shot evaluation on RISC-V architecture. We compile 1000 binary functions using RISC-V 64-bit GCC toolchain (`riscv64-linux-gnu-gcc`). The models trained on x86/ARM/MIPS architectures are directly applied to RISC-V binaries without any parameter updates or fine-tuning.

We evaluate zero-shot performance on cross-architecture search where RISC-V functions serve as queries to search in x86/ARM/MIPS databases. For this challenging task, we use a poolsize of 1000 functions to assess how well the models can match RISC-V binaries against functions compiled for completely different architectures.

Models	Recall@1	Recall@10	MRR
SAFE	0.02	0.06	0.04
GMN	0.03	0.09	0.06
GraphCodeBERT	0.12	0.24	0.17
CeBin	0.15	0.32	0.21
UnixCoder	0.18	0.38	0.27
BDLF-Qwen3	**0.41**	**0.58**	**0.45**

Table 1: Zero-shot cross-architecture search performance on RISC-V (Poolsize=1000)

Table 1 presents the zero-shot search performance when RISC-V functions are used as queries. Despite never encountering RISC-V architecture during training, BDLF-Qwen3 demonstrates remarkable generalization capability, achieving Recall@10 of 0.58 and Recall@1 of 0.41 for poolsize of 1000. This represents a 53% improvement in Recall@10 and 128% improvement in Recall@1 over UnixCoder, the second-best performing model. The MRR score of 0.45 further confirms BDLF-Qwen3's ability to rank correct matches highly, outperforming UnixCoder by 67%. Traditional approaches like SAFE and GMN show severe performance degradation with Recall@10 below 0.10, confirming their heavy reliance on architecture-specific patterns and inability to generalize to unseen instruction sets.

Cross-Bin Dataset Quality Validation. To validate the quality of the Cross-Bin dataset and the effectiveness of our dataset engine pipeline, we conduct comprehensive quality assessment experiments focusing on annotation accuracy and data efficiency.

Human Evaluation. We randomly sample 1,000 functions from Cross-Bin and have two security experts independently evaluate the quality of semantic annotations. The experts assess: (1) accuracy of function descriptions, (2) correctness of parameter type inference, and (3) cross-architecture semantic consistency. Table 2 presents the evaluation results with Cohen's Kappa (κ) coefficient measuring inter-annotator agreement (Warrens 2015).

The evaluation demonstrates high annotation quality with strong inter-annotator agreement. The Cohen's Kappa values indicate substantial to almost perfect agreement across all criteria ($\kappa > 0.7$), with cross-architecture consistency

Evaluation Criteria	Expert 1	Expert 2	Cohen's Kappa
Function Description	91.2%	93.5%	0.82
Parameter Inference	86.8%	88.4%	0.73
Cross-Arch Consistency	93.7%	94.8%	0.86
Average Score	90.6%	92.2%	0.80

Table 2: Human evaluation results on Cross-Bin annotations

achieving the highest agreement ($\kappa = 0.86$). The relatively lower but still substantial agreement on parameter inference ($\kappa = 0.73$) is expected given the inherent ambiguity in type recovery from binary code. The overall Cohen's Kappa of 0.80 confirms that our dataset engine pipeline produces reliable and consistent labels suitable for training cross-architecture models.

Impact of XRef. To directly demonstrate the effectiveness of XRef enhancement, we train BDLF-Qwen3 on Cross-Bin with and without XRef-enhanced information. We evaluate both configurations on the challenging large-scale scenario (poolsize=10,000) where reference recovery quality becomes critical. Table 3 reveals that XRef enhancement yields substantial performance gains, with Recall@10 improving by 11.4% and MRR by 10.2%. These improvements are particularly significant in large-scale scenarios where resolving cryptic addresses to meaningful symbols becomes crucial for distinguishing among thousands of candidate functions.

Configuration	Recall@10	MRR
Cross-Bin w/o XRef	0.335	0.263
Cross-Bin w/ XRef	**0.378**	**0.293**
Performance Gain	+11.4%	+10.2%

Table 3: Performance comparison with and without XRef enhancement (Poolsize=10,000)

Cross-Architecture Vulnerability Discovery. To validate BDLF-Qwen3's practical security applications, we use 1,200 binary functions across four CWE types: CWE-119 (Buffer Overflow), CWE-120 (Buffer Copy), CWE-416 (Use After Free), and CWE-476 (NULL Pointer Dereference). Each category contains 300 functions compiled from identical source code across x86-64, ARM64, and MIPS64 with optimization levels -O0 to -O2.

Method	CWE-119	CWE-120	CWE-416	CWE-476	Avg
UnixCoder	0.68	0.71	0.74	0.69	0.705
BDLF-Qwen3	**0.94**	**0.92**	**0.97**	**0.95**	**0.945**

Table 4: Cross-Architecture Vulnerability Detection Performance (Recall@10)

The evaluation follows a query-database paradigm where vulnerable functions from one architecture query semantically equivalent functions in other architectures, simulating real-world vulnerability discovery across heterogeneous deployments. Table 4 shows BDLF-Qwen3 achieves substantial 29-38% improvements over UnixCoder, with exceptional performance on memory-related vulnerabilities

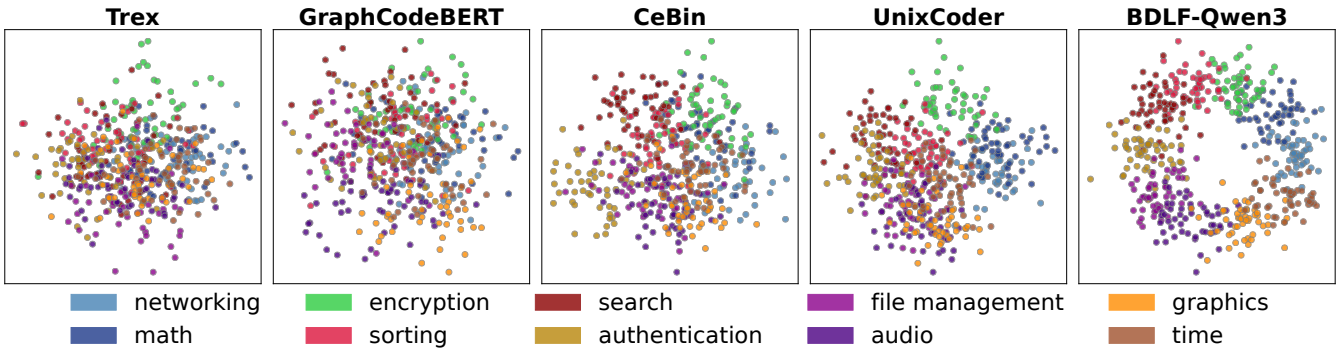

Figure 4: t-SNE visualization of semantic consistency across different models. Each point represents a binary function colored by its semantic category. BDLF-Qwen3 shows clear semantic clustering with well-separated categories.

(CWE-416: 0.97, CWE-119: 0.94).

Ablation Study. To validate the effectiveness of key components in BDLF-Qwen3, we conduct ablation studies focusing on the dynamic layer fusion mechanism and training strategies.

Dynamic Weight Generation. We first investigate the importance of dynamic weight generation by comparing it against static alternatives. Table 5 shows the performance comparison using different weight generation strategies on the challenging poolsize=1,000 scenario. The dynamic weights achieve 13.6% higher Recall@10 than fixed uniform weights (0.67 vs 0.59), confirming that input-dependent layer fusion is essential for effective cross-architecture representation.

Weight Strategy	Recall@10	MRR
Fixed Uniform Weights	0.59	0.47
Learned Static Weights	0.60	0.48
Dynamic Weights (BDLF)	**0.67**	**0.53**

Table 5: Ablation study on weight generation strategies (Poolsize=1,000)

Training Strategy Comparison Across Base Models. We investigate the effectiveness of our two-stage training approach by comparing it against single-stage alternatives across different base models. Table 6 shows the performance comparison using ranking-focused metrics on the challenging poolsize=1,000. NDCG@10 evaluates position-aware ranking quality crucial for analysts examining top results, while MAP assesses performance across all semantically equivalent functions when single source functions correspond to multiple binary variants across architectures.

The results demonstrate that our two-stage approach outperforms single-stage alternatives. Stage 2 only performs significantly worse with NDCG@10 drops of 32.3%, 11.4%, and 13.8% respectively, as base models lack binary-specific understanding needed for effective dynamic fusion. Stage 1 only achieves reasonable performance but shows NDCG@10 improvements of 29.0%, 15.9%, and 12.1% when adding Stage 2, highlighting the necessity of combining foundational learning with dynamic feature aggregation.

Base Model	Training Strategy	NDCG@10	MAP
GraphCodeBERT	Stage 1 Only (Fine-tuning)	0.31	0.18
	Stage 2 Only (BDLF)	0.21	0.13
	Two-stage (Ours)	**0.40**	**0.25**
UnixCoder	Stage 1 Only (Fine-tuning)	0.44	0.27
	Stage 2 Only (BDLF)	0.39	0.21
	Two-stage (Ours)	**0.51**	**0.31**
Qwen3-Embedding	Stage 1 Only (Fine-tuning)	0.58	0.43
	Stage 2 Only (BDLF)	0.50	0.37
	Two-stage (Ours)	**0.65**	**0.49**

Table 6: Training strategy comparison across different base models (Poolsize=1,000)

5 Conclusion

We presented BDLF, a neural architecture that adaptively integrates multi-layer transformer representations for cross-architecture binary similarity detection. Our key contributions include: (1) **Cross-Bin**, an XRef-enhanced dataset of 547,000 functions annotated via DeepSeek-V3; (2) the **BDLF mechanism** for dynamic layer-wise feature fusion; and (3) a **two-stage training framework**. BDLF-Qwen3 outperforms state-of-the-art methods by 36–65% in large-scale scenarios and demonstrates robust zero-shot generalization to RISC-V. Future work will focus on efficient large-scale indexing and integration with professional reverse engineering suites to enhance practical security analysis in heterogeneous environments.

6 Limitation

Despite its performance, BDLF-Qwen3 faces several limitations. First, **dataset bias** exists as Cross-Bin predominantly features open-source software, potentially lacking representativeness for proprietary code or malware with anti-analysis protections. Second, the model faces **scalability challenges**: performance drops as the database scales (Recall@10 from 0.905 to 0.383 at 10k pool size), and the BDLF mechanism's linear memory scaling with layer count imposes computational overhead. Industrial deployment at a million-function scale would require further advances in efficient indexing and model compression.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under grant U24A20337.

References

Atrey, P. K.; Hossain, M. A.; El Saddik, A.; and Kankanhalli, M. S. 2010. Multimodal fusion for multimedia analysis: a survey. *Multimedia systems*, 16(6): 345–379.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

David, Y.; Partush, N.; and Yahav, E. 2016. Statistical similarity of binaries. *Acm sigplan notices*, 51(6): 266–280.

David, Y.; Partush, N.; and Yahav, E. 2017. Similarity of binaries through re-optimization. In *Proceedings of the 38th ACM SIGPLAN conference on programming language design and implementation*, 79–94.

David, Y.; and Yahav, E. 2014. Tracelet-based code search in executables. *Acm Sigplan Notices*, 49(6): 349–360.

Ding, S. H.; Fung, B. C.; and Charland, P. 2019. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *2019 ieee symposium on security and privacy (sp)*, 472–489. IEEE.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Dullien, T.; and Rolles, R. 2005. Graph-based comparison of executable objects (english version). *Sstic*, 5(1): 3.

Eagle, C.; and Nance, K. 2020. *The Ghidra Book: The Definitive Guide*. no starch press.

Eschweiler, S.; Yakdan, K.; Gerhards-Padilla, E.; et al. 2016. Discovre: Efficient cross-architecture identification of bugs in binary code. In *Ndss*, volume 52, 58–79.

Feng, Q.; Zhou, R.; Xu, C.; Cheng, Y.; Testa, B.; and Yin, H. 2016. Scalable graph-based bug search for firmware images. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 480–491.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

Flake, H. 2004. Structural comparison of executable objects. In *Detection of intrusions and malware & vulnerability assessment, GI SIG SIDAR workshop, DIMVA 2004*, 161–173. Gesellschaft für Informatik eV.

Goldberg, Y. 2016. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57: 345–420.

Hu, Y.; Zhang, Y.; Li, J.; and Gu, D. 2016. Cross-architecture binary semantics understanding via similar code comparison. In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, volume 1, 57–67. IEEE.

Huang, H.; Youssef, A. M.; and Debbabi, M. 2017. Binsequence: Fast, accurate and scalable binary code reuse detection. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 155–166.

IoT Business News. 2024. State of IoT 2024: Number of connected IoT devices growing 13 https://iotbusinessnews.com/2024/09/04/26399-state-of-iot-2024-number-of-connected-iot-devices-growing-13-to-18-8-billion-globally/. Accessed: 2025-04-19.

Kalapatapu, P. 2025. Top 15 software supply chain attacks: Case studies. https://outshift.cisco.com/blog/top-10-supply-chain-attacks. Accessed: 2025-05-20.

Kim, D.; Kim, E.; Cha, S. K.; Son, S.; and Kim, Y. 2022. Revisiting binary code similarity analysis using interpretable feature engineering and lessons learned. *IEEE Transactions on Software Engineering*, 49(4): 1661–1682.

Lahat, D.; Adali, T.; and Jutten, C. 2015. Multimodal data fusion: an overview of methods, challenges, and prospects. *Proceedings of the IEEE*, 103(9): 1449–1477.

Li, X.; Qu, Y.; and Yin, H. 2021. Palmtree: Learning an assembly language model for instruction embedding. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, 3236–3251.

Li, Y.; Gu, C.; Dullien, T.; Vinyals, O.; and Kohli, P. 2019. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, 3835–3845. PMLR.

Li, Z.; Zhang, X.; Zhang, Y.; Long, D.; Xie, P.; and Zhang, M. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Lin, T.-Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; and Belongie, S. 2017. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2117–2125.

Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Liu, B.; Huo, W.; Zhang, C.; Li, W.; Li, F.; Piao, A.; and Zou, W. 2018. αdiff: cross-version binary code similarity detection with dnn. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, 667–678.

Liu, Z.; Hu, H.; Lin, Y.; Yao, Z.; Xie, Z.; Wei, Y.; Ning, J.; Cao, Y.; Zhang, Z.; Dong, L.; et al. 2022. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 12009–12019.

Luo, Z.; Wang, P.; Wang, B.; Tang, Y.; Xie, W.; Zhou, X.; Liu, D.; and Lu, K. 2023. VulHawk: Cross-architecture vulnerability detection with entropy-based binary code search. In *NDSS*.

Massarelli, L.; Di Luna, G. A.; Petroni, F.; Baldoni, R.; and Querzoni, L. 2019. Safe: Self-attentive function embeddings

for binary similarity. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 309–329. Springer.

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

Pewny, J.; Garmany, B.; Gawlik, R.; Rossow, C.; and Holz, T. 2015. Cross-architecture bug search in binary executables. In *2015 IEEE Symposium on Security and Privacy*, 709–724. IEEE.

Seawead, T.; Yang, C.; Lin, Z.; Zhao, Y.; Lin, S.; Ma, Z.; Guo, H.; Chen, H.; Qi, L.; Wang, S.; et al. 2025. Seaweed-7b: Cost-effective training of video generation foundation model. *arXiv preprint arXiv:2504.08685*.

Ullah, S.; and Oh, H. 2021. Bindiff nn: Learning distributed representation of assembly for robust binary diffing against semantic differences. *IEEE Transactions on Software Engineering*, 48(9): 3442–3466.

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

Wang, H.; Qu, W.; Katz, G.; Zhu, W.; Gao, Z.; Qiu, H.; Zhuge, J.; and Zhang, C. 2022. Jtrans: Jump-aware transformer for binary code similarity detection. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 1–13.

Wang, Y.; Jia, P.; Huang, C.; Liu, J.; and He, P. 2021. [Retracted] Hierarchical Attention Graph Embedding Networks for Binary Code Similarity against Compilation Diversity. *Security and Communication Networks*, 2021(1): 9954520.

Wang, Y.; Le, H.; Gotmare, A. D.; Bui, N. D.; Li, J.; and Hoi, S. C. 2023. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*.

Warrens, M. J. 2015. Five ways to look at Cohen's kappa. *Journal of Psychology & Psychotherapy*, 5.

Xiao, S.; Liu, Z.; Zhang, P.; Muennighoff, N.; Lian, D.; and Nie, J.-Y. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, 641–649.

Xu, X.; Liu, C.; Feng, Q.; Yin, H.; Song, L.; and Song, D. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 363–376.

Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yu, J.; Dai, Y.; Liu, X.; Huang, J.; Shen, Y.; Zhang, K.; Zhou, R.; Adhikarla, E.; Ye, W.; Liu, Y.; et al. 2024. Unleashing the power of multi-task learning: A comprehensive survey spanning traditional, deep, and pretrained foundation model eras. *arXiv preprint arXiv:2404.18961*.

Yu, Z.; Cao, R.; Tang, Q.; Nie, S.; Huang, J.; and Wu, S. 2020. Order matters: Semantic-aware neural networks for binary code similarity detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 1145–1152.

Zhang, Y.; Li, M.; Long, D.; Zhang, X.; Lin, H.; Yang, B.; Xie, P.; Yang, A.; Liu, D.; Lin, J.; et al. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *arXiv preprint arXiv:2506.05176*.

Zhao, H.; Shi, J.; Qi, X.; Wang, X.; and Jia, J. 2017. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2881–2890.

Zuo, F.; Li, X.; Young, P.; Luo, L.; Zeng, Q.; and Zhang, Z. 2018. Neural machine translation inspired binary code similarity comparison beyond function pairs. *arXiv preprint arXiv:1808.04706*.