

DOI: 10.3969/j.issn.1671-0673. 2026. 04. 008

基于大语言模型推理增强的函数级二进制 与源代码相似性检测

常金为, 孙佳佳, 尹小康, 张一箐, 朱肖雅, 李路凯

(信息工程大学, 河南 郑州 450001)

摘要: 二进制与源代码之间的相似性检测在软件成分分析、漏洞定位等领域应用广泛。然而, 现有方法在复杂编译环境下难以对齐不同模态的语义, 导致检索效果较差。为此, 提出基于大语言模型推理增强的相似性检测框架 BinThink。BinThink 利用大语言模型构建二进制函数和源代码函数的统一语义表示, 并结合语义引导机制将不同模态的函数映射到同一向量空间; 同时, 利用对比学习优化跨模态语义对齐能力, 提升复杂编译环境下的函数检索效果。实验结果表明, 在候选池大小为 100 时, BinThink 的 Recall@1 相较于现有方法 BinaryAI, 在跨编译优化与跨指令架构的函数检索任务中分别提升 7.8 和 8.5 个百分点。

关键词: 二进制与源代码相似性检测; 大语言模型; 函数级推理增强; 自适应推理; 标准化语义流
中图分类号: TP309; TP311.5 **文献标识码:** A **文章编号:** 1671-0673(2026)04-0436-07

Function-Level Binary-to-Source Code Similarity Detection Method Enhanced by Large Language Model Reasoning

CHANG Jinwei, SUN Jiajia, YIN Xiaokang, ZHANG Yizheng, ZHU Xiaoya, LI Lukai

(Information Engineering University, Zhengzhou 450001, China)

Abstract: Binary-to-source code similarity detection is widely used in software component analysis and vulnerability localization. However, existing methods struggle to align semantics across different modalities in complex compilation environments, leading to poor retrieval performance. To address these challenges, a similarity detection framework enhanced by Large Language Model reasoning (BinThink) is proposed. A Large Language Model is utilized in BinThink to construct a unified semantic representation of binary and source code functions, and a semantic guidance mechanism is combined to map functions from different modalities to the same vector space. Simultaneously, contrastive learning is leveraged to optimize cross-modal semantic alignment, improving function retrieval performance in complex compilation environments. Experimental results show that, when the candidate pool size is 100, BinThink achieves Recall@1 improvements of 7.8 and 8.5 percentage points compared with the existing method BinaryAI in function retrieval tasks across compilation optimization and instruction architectures, respectively.

Key words: binary-to-source code similarity detection; Large Language Model; function-level reasoning-enhancement; adaptive reasoning; standardized semantic flow

二进制与源代码相似性检测是计算机安全领域的重要研究问题, 广泛应用于漏洞定位、代码复

用识别和逆向工程等场景^[1-2]。例如, 在已知漏洞源代码的情况下, 若能够准确识别目标二进制程

收稿日期: 2026-03-05; 修回日期: 2026-03-27

作者简介: 常金为(2002—), 男, 硕士生, 主要研究方向为二进制安全。E-mail: cjl_email@163.com

序中是否包含相应编译代码,将有助于开展软件漏洞预警与风险分析。

然而,源代码与二进制代码之间存在显著的语义鸿沟。源代码保留了函数名、变量名和数据结构等高层语义信息,而二进制代码在编译过程中被转换为底层指令序列,大量高层语义信息被剥离。同时,不同优化等级和指令集架构还会进一步改变指令形式与程序结构,使得相同源代码生成的二进制呈现出明显差异。

现有研究主要包括基于静态分析和基于表征学习两类方法^[3-4]。前者依赖人工设计的常量、控制流和调用关系等特征,难以适应复杂编译环境下的结构变换;后者虽然能够借助图神经网络或序列模型自动学习代码表示^[5],但往往仍停留于指令符号模式或拓扑结构层面,容易受到寄存器分配、编译优化等底层实现细节干扰,缺乏对函数高层逻辑意图的显式建模。近年来,大语言模型(Large Language Model, LLM)展现出较强的代码理解能力^[6-8],能够有效提升跨架构检测的精度。但其生成式架构并不天然适配判别式检索任务,直接用于大规模相似性匹配时仍面临特征边界不清、推理开销较大以及可扩展性不足等问题。

针对上述问题,本文提出一种函数级二进制到源代码相似性检测框架 BinThink。该方法引入语义归一化中间层,对二进制函数中的变量、操作语义和逻辑结构进行统一建模,将底层异构指令转化为与架构无关的语义流,从而缓解编译差异带来的结构噪声,并增强二进制与源代码之间的深层语义对齐能力。在此基础上,本文进一步设计上下文增强型提示机制,并结合自适应推理与差异化训练策略,以提升模型在检索任务中的判别能力与计算效率。本文主要工作如下。

1) 提出 BinThink 框架,通过构建语义归一化中间层,将异构二进制指令转译为与架构无关的语义流,提升二进制到源代码相似性检测的准确性。

2) 设计面向检索任务的上下文增强提示与自适应推理机制,缓解大语言模型在向量匹配任务中的生成偏置、计算开销大和收敛慢等问题。

3) 在大规模真实数据集上的实验结果表明, BinThink 在跨优化等级和跨架构场景下均具有较好的鲁棒性与泛化能力,整体性能优于现有方法。

1 相关工作

二进制与源代码相似性检测旨在判定两者的

语义对应关系,是漏洞定位及逆向分析的关键技术^[9-10]。由于源代码保留了丰富的语法结构与符号信息,而二进制代码仅包含底层机器指令,二者在表示形式上存在显著差异。此外,编译优化、目标架构变化及符号剥离等因素还会进一步扭曲代码表层结构,加剧跨模态匹配难度。现有方法大致可分为基于静态特征的方法和基于表征学习的方法。

1.1 基于静态特征的方法

基于静态特征的方法主要依赖人工设计的特征进行匹配。代表性工作如 BinPro^[11]利用字符串常量、整数立即数和函数调用等特征,通过二分图匹配计算源代码与二进制代码间的对应关系; B2SFinder^[12]进一步结合字符串、常量和控制流等多类特征,并通过加权策略提升匹配精度; OSSPolice^[13]则利用字符串字面量、导出函数等特征识别开源代码复用关系。此类方法通常具有较好的可解释性和较高的计算效率,但其本质上依赖表层特征,当面对常量折叠、函数内联以及跨架构转换等复杂编译扰动时,容易因关键特征缺失或变形而性能下降^[14]。

1.2 基于表征学习的方法

基于表征学习的方法通过学习统一向量表示来缩小源代码与二进制代码之间的模态差异,可进一步分为基于图建模和基于序列建模两类。

基于图建模的方法通常将二进制函数表示为控制流图,并利用图神经网络提取结构语义。CodeCMR^[15]采用双塔架构,在源代码侧和二进制侧分别学习表示并进行对齐; Order Matters^[16]结合基本块级表示与消息传递机制建模函数语义; VulHawk^[17]通过将不同架构二进制提升为与架构无关的中间表示,并结合预训练任务学习函数嵌入。该类方法能够较好利用控制流结构信息,但也容易受图结构复杂性限制,在处理长距离依赖和细粒度操作语义时存在不足。

基于序列建模的方法则将汇编代码视为指令序列,借助自然语言处理模型捕捉上下文语义^[18]。SAFE^[19]利用自注意力机制学习函数嵌入表示; PalmTree^[20]基于 BERT 预训练指令表示,以增强对汇编语义的建模能力; BinaryAI^[21]通过将二进制反编译为类 C 伪代码,并结合对比学习实现与源代码的语义对齐。此类方法将具有复杂拓扑关系的程序表示线性化,往往会损失控制流结构信息^[22];同时,在跨架构场景下,不同指令集之间显著的分

总体来看,现有方法虽然在二进制相似性检测任务中取得了较大进展,但仍普遍存在对表层模式依赖较强、对深层逻辑语义建模不足的问题。如何在复杂编译环境下实现更稳健的跨模态语义对齐,仍是该方向需要进一步解决的关键问题。

2 BinThink 方法

2.1 问题定义

给定一个二进制函数集合 $B=\{b_1, b_2, \dots, b_N\}$ 和一个源代码函数集合 $S=\{s_1, s_2, \dots, s_M\}$, 二进制到源代码函数相似性检测的目标是学习一个相似度量函

数 $\text{Sim}(b_i, s_j) \in [0, 1]$ 。对于每一对互为编译关系的 (b_i, s_j) , 模型应使得 $\text{Sim}(b_i, s_j)$ 趋近于 1, 而对于非同源对 $(b_i, s_j), j \neq i$, 模型应使得 $\text{Sim}(b_i, s_j)$ 趋近于 0。

2.2 框架概览

本文提出 BinThink 框架,如图 1 所示。该框架首先利用 Qwen3-Coder-30B 模型作为推理引擎,通过自适应推理机制,将二进制代码转化为包含丰富语义意图的中间表征。之后,采用 Qwen3-7B 作为骨干网络,配合非线性投影层,将推理生成的语义流与源代码映射到统一的高维向量空间,实现跨模态特征对齐。

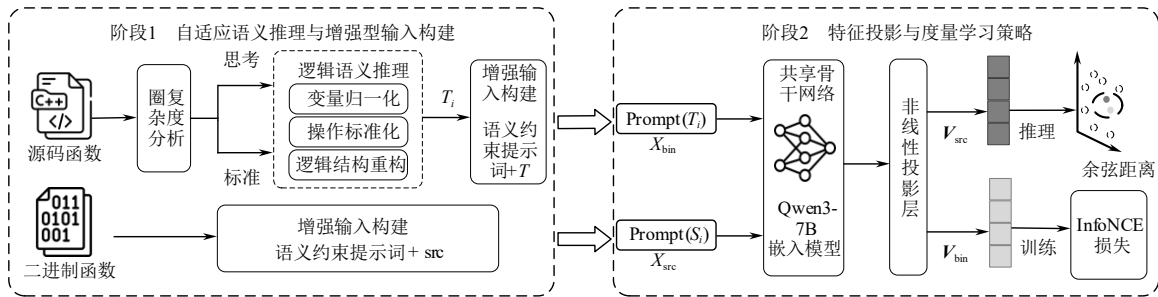


图1 总体框架图

2.3 自适应语义推理与增强型输入构建

为在深度语义理解与计算开销之间取得平衡,并实现跨模态数据的形式化对齐,BinThink 构建了一套从复杂度评估到增强型输入生成的处理流程。

2.3.1 复杂度评估与调度

根据函数的循环嵌套深度、条件分支数量等结构特征计算复杂度得分 $C(f)$ 。本文将 $C(f)$ 定义为边数 E 与节点数 V 的加权和,作为圈复杂度的近似度量。当 $C(f)$ 超过预设阈值 τ 时,判定该函数为高复杂度,系统激活 Qwen3 的思考模式;反之,对于低复杂度函数,则采用标准推理模式以节省资源。复杂度得分 $C(f)$ 定义如下:

$$C(f) = w_1 \cdot |E| + w_2 \cdot |V| \quad (1)$$

式中: E 为控制流边数; V 为基本块块数; w_1 和 w_2 为

权重系数,用于平衡节点规模与控制流分支对复杂度的贡献。

2.3.2 语义流生成

在推理过程中,模型生成语义流 T_i ,用于将底层指令序列转化为与架构无关的逻辑操作表示。其构建主要包括 3 个方面。1) 变量归一化,即识别寄存器在数据流中的语义角色,并统一标记为参数、计数器或返回值等逻辑变量;2) 操作语义标准化,即利用预定义词表对机器指令进行统一转译与降噪,将语义相近或由编译优化产生的指令模式映射为一致的原子操作;3) 逻辑结构重构,即结合控制流信息恢复循环、条件分支和函数调用等高层控制范式。图 2 展示了数组求和函数经推理后得到的标准语义流示例。

逻辑阶段与原子操作		主操作对象/属性	辅助操作源/参数	语义意图注解
[INIT]	ASSIGN	TGT: VAR_accum (RET_VAL)	SRC: CONST(0)	NOTE: Initialize accumulator
[CHECK]	BRANCH	COND: ARG_len == 0	DEST: EXIT	NOTE: Boundary check for empty array
[LOOP_HEAD]	ITERATE	TYPE: DO_WHILE	CTR: ITR_idx	NOTE: Main accumulation loop
[BODY]	READ_MEM	TGT: TMP_val	SRC: MEM[ARG_ptr]	NOTE: Load array element
[BODY]	CALC	TGT: VAR_accum	ACT: ADD OP: TMP_val	NOTE: Update sum
[BODY]	CALC	TGT: ARG_ptr	ACT: ADD OP: CONST(4)	NOTE: Pointer increment (Stride)
[BODY]	CALC	TGT: ARG_len	ACT: SUB OP: CONST(1)	NOTE: Decrement counter
[BODY]	BRANCH	COND: ARG_len > 0	DEST: LOOP_HEAD	NOTE: Loop continuation condition
[EXIT]	RETURN	VAL: VAR_accum		NOTE: Return final sum

图2 语义流示例

2.3.3 增强型输入

为适配嵌入模型输入形式并减弱 LLM 的生成

偏置,本文分别为二进制语义流和源代码函数设计了任务导向的增强提示。其目标是引导模型聚

焦深层语义理解与特征提取,而非代码补全或文本生成。

在二进制侧,将推理生成的语义流 T_i 嵌入提示模板,通过加入“关注数据流与控制逻辑、忽略格式噪声”等约束,引导模型从逻辑执行角度理解输入,形成增强表示 X_{bin} :

$$X_{bin} = \text{Prompt}(T_i) \quad (2)$$

在源码侧,对源代码 s_i 施加相似的任务约束,引导模型弱化变量命名、注释和语法风格等表层因素的影响,提取与二进制语义流对应的核心计算逻辑,形成增强表示 X_{src} :

$$X_{src} = \text{Prompt}(s_i) \quad (3)$$

2.4 特征投影与度量学习策略

在获得 X_{bin} 与 X_{src} 后,系统将其输入到基于 Qwen3-7B 的嵌入模型中。

2.4.1 共享投影与解耦微调机制

为提升相似性度量能力,本文在模型末端引入非线性投影层,将原始隐藏状态映射到统一的相似性空间,并通过计算 V_{bin} 与 V_{src} 的余弦相似度完成匹配。

训练阶段, BinThink 基于 QLoRA 采用差异化学习率策略。对基座模型及 LoRA 适配器使用较低学习率,以保留预训练知识;对投影层使用较高学习率,以加快二进制代码与源代码特征在统一度量空间中的对齐。

2.4.2 困难负样本挖掘与边界约束优化

为提升度量空间的区分度,优化目标采用带边界约束的 InfoNCE 损失函数。本文通过计算投影后的向量 V_{bin} 与 V_{src} 的余弦相似度来评估相似性。针对随机负样本梯度贡献不足的问题,在同一批次内在线挖掘困难负样本,即寻找与二进制锚点 b_i 相似度最高但非同源的源码函数 s_j (例如 `strcpy` 与 `strncpy`)。同时引入 Margin 约束 m , 强制正样本对的相似度必须显著高于最难负样本对的相似度,从而在特征空间中形成清晰的类间边界。

2.5 复杂度与误差传播分析

2.5.1 相关分析

本文提出框架的计算开销主要来源于大语言模型推理与下游嵌入映射。设待检测二进制函数的规模为 N , 指令序列平均长度为 L 。若直接应用全量 LLM 推理,其时间复杂度高达 $O(N \cdot L^2 \cdot d)$ (d 为隐藏层维度),且面临极大的显存开销 $O(P + b \cdot L \cdot d)$ (P 为模型参数量, b 为批大小)。为此, BinThink 引入自适应推理机制,仅对圈复杂度越界的

复杂样本 (占比 $\alpha \ll 1$) 触发 LLM 分析,将核心时间复杂度降至 $O(\alpha \cdot N \cdot L^2 \cdot d)$;同时,将 LLM 输出固化为轻量级的规范化文本 (语义流),避免在内存中持续维护海量稠密向量,显著降低了空间复杂度,有效提升模型在大规模代码库中的可扩展性。

2.5.2 推理误差传播影响

在基于 LLM 的检测范式中,局部推理偏差可能在下游引发级联误差。为缓解该问题, BinThink 设计了两级容错机制。首先,中间表示“语义流”通过标准化对底层指令进行抽象,过滤细粒度噪声。其次,下游对比学习在统一语义空间中进行全局优化,引导模型关注函数的高层语义意图而非局部细节。因此,即使中间语义流中存在少量结构性噪声,非线性投影层仍可将其稳定映射至正确的语义簇,有效阻断局部推理误差向整体匹配结果的传播,保障模型在复杂编译环境下的鲁棒性。

3 实验

3.1 实验设置

本文构建了大规模跨模态函数检索数据集,以评估 BinThink 在跨架构和跨优化场景下的鲁棒性。数据集来源于 GitHub 上多个高流行度 C/C++ 开源项目,包括 OpenSSL、Coreutils 和 SQLite 等,共包含 60 000 余对二进制函数与源代码函数。编译环境覆盖 O0~O3 这 4 种优化等级及 x86-32、x86-64、ARM64、MIPS32 等主流架构。

实验中, BinThink 采用 Qwen3-Coder-30B 作为推理模型、Qwen3-7B 作为嵌入模型,并基于 4-bit QLoRA 进行微调。硬件环境为 Intel Xeon Silver 4116、128 GB 内存和 2×RTX4090。主要超参数设置如下:训练轮次为 150,批次大小为 16,学习率分别为 2×10^{-5} 和 2×10^{-4} ,温度系数为 0.07, Margin 为 0.2。

3.2 评估指标

1) 平均倒数排名 (Mean Reciprocal Rank, MRR): 反映了真实结果在推荐列表中的位置, MRR 越高,位置越靠前。

$$R_{MRR} = \frac{1}{N} \sum_{i=1}^N \frac{1}{R_i^{\text{rank}}} \quad (4)$$

2) Recall@k: 衡量检索结果前 k 个候选中包含真实对应源码函数的比例,是检索任务中最核心的指标。

$$R_{\text{Recall}@k} = \frac{N_k}{N}$$

(5)

式中: N_k 为前 k 次命中数; N 为查询总数。

3.3 对比实验

本文选取 BinPro、B2SFinder、CodeCMR 和 BinaryAI 作为基线方法。其中 BinaryAI 以 Pythia-

410M 为骨干,并在本文数据集上完成预训练与微调;BinPro 和 B2SFinder 则严格依照原论文实现。

BinThink 在跨优化等级(Cross-Optimization, XO)和跨架构(Cross-Architecture, XA)两个场景下评估了各模型的检索性能。在候选池大小(Pool Size)分别为 100 和 10 000 的设置下,实验结果如表 1 所示。

表 1 不同方法在跨优化与跨架构任务上的实验结果

模型	方法类型	Recall@1 (XO)/%		Recall@1 (XA)/%		MRR(overall)	
		100	10 000	100	10 000	100	10 000
BinPro	Static Features	38.6	20.4	32.4	23.8	0.42	0.35
B2SFinder	Weighted Features	40.2	26.1	41.1	24.8	0.47	0.40
CodeCMR	GNN + DPCNN	64.3	39.3	62.5	38.6	0.71	0.56
BinaryAI	Transformer	81.6	53.5	80.2	51.7	0.83	0.72
BinThink	Reasoning LLM	89.4	62.3	88.7	59.4	0.91	0.78

实验结果表明,BinThink 在所有评估指标上均取得了最优性能。当候选池大小为 100 时,BinThink 在 XO 和 XA 任务上的 Recall@1 分别达到了 89.4% 和 88.7%,整体 MRR 高达 0.91。与 BinaryAI 相比,BinThink 在 XO 和 XA 任务上的 Recall@1 分别提高了 7.8 和 8.5 个百分点。

结果表明,BinThink 通过“自适应推理—增强表征—对比检索”框架,有效缓解了编译环境带来的结构噪声,提升了二进制代码与源代码之间的语义对齐能力。

从对比结果看,BinPro 和 B2SFinder 等基于静态特征的方法性能最低,说明浅层特征容易受编译优化影响。CodeCMR 借助图结构建模取得了一定提升,但对细粒度指令语义的刻画仍有限。BinaryAI 依靠大规模预训练进一步提高了性能,但在跨模态表征差异较大的场景下仍存在不足。相比之下,BinThink 通过显式语义推断直接建模底层汇编的功能意图,减少了对表层结构和反编译结果的依赖,因此取得了更优的检测效果。

在候选池大小为 100 时,BinThink 在 XO 和 XA 任务上的 Recall@1 分别达到 89.4% 和 88.7%,二者仅相差 0.7 个百分点,表明 BinThink 在不同编译优化等级和不同指令集架构下均具有较强的鲁棒性。相比之下,BinaryAI 在 XO 和 XA 任务上的 Recall@1 分别为 81.6% 和 80.2%,其在 XA 任务上的性能较 XO 任务下降 1.4 个百分点。

当候选池扩展至 10 000 时,所有方法性能均有所下降,但 BinThink 仍保持较强判别能力:在 XO 场景下 Recall@1 达到 62.3%,高于 BinaryAI, MRR 为 0.78,也优于 BinaryAI 的 0.72。结果说明,Bin-

Think 在大规模检索场景下仍能维持较清晰的特征边界和较好的抗干扰能力。

3.4 消融实验

为验证 BinThink 各个核心组件的贡献,本文设计了消融实验,重点关注模型选择和损失函数设计。实验结果如表 2 所示。

表 2 模型规模与推理模式消融实验

嵌入模型	推理模式	FNN 投影层	Recall@1(overall)/%		MRR(overall)	
			100	10 000	100	10 000
Qwen3-3B	否	是	65.2	39.2	0.76	0.56
Qwen3-7B	否	是	74.1	48.3	0.81	0.64
Qwen3-7B	是	否	61.4	37.5	0.73	0.57
Qwen3-7B	是	是	92.3	62.3	0.94	0.79

3.4.1 嵌入模型的规模与推理模块的影响

表 2 结果表明,在同为 Qwen3-7B 嵌入模型的条件,启用推理模式后 Recall@1 由 74.1% 提升至 92.3%,增幅达 18.2 个百分点。需要指出的是,该 overall 指标并非仅由 XO 和 XA 等困难样本构成,还包含一定比例的同架构、同优化等级的基础样本。由于 BinThink 在这些结构未显著变异的场景中表现出接近饱和的对齐精度,从而进一步抬升了整体加权性能。因此,该结果不仅反映了模型在复杂编译环境下的有效性,也体现了其在常规场景中的稳定表现。相比之下,单纯增大嵌入模型规模仅带来有限收益,说明思维链推理比参数扩展更能促进模型对二进制代码深层语义的理解。这表明,跨模态对齐不仅依赖表层特征匹配,更依赖对代码逻辑意图的显式建模。

此外,FNN 投影层在生成式表征向判别式检

索迁移中起到了关键作用。即使启用推理模式,若去除投影层而直接使用 LLM 隐藏状态进行检索,Recall@1 仍会降至 61.4%。这说明 LLM 原始特征空间并不适合直接用于相似度度量,而投影层能够有效将推理获得的语义信息映射到更适合检索的对比空间。

3.4.2 损失函数设计的演进

为了验证对比学习目标函数中各关键组件的有效性,本文逐步叠加了困难负样本挖掘与 Margin 约束,实验结果如表 3 所示。

表 3 损失函数各组件对性能的影响					
实验配置	损失函数	Recall@1 /%	MRR	平均正样本相似度	平均负样本相似度
A	标准 InfoNCE	75.0	0.78	0.637 5	0.044 9
B	A+难负样本挖掘	81.2	0.84	0.659 6	0.036 0
C	B+Margin 约束	89.1	0.91	0.730 9	0.006 0

实验配置 A 与 B 的对比表明,样本采样策略对模型收敛具有重要影响。仅采用标准 InfoNCE 时,Recall@1 为 75.0%;引入难负样本挖掘后,Recall@1 提升至 81.2%。这说明随机负样本区分难度较低,而难负样本挖掘能够促使模型关注细粒度语义差异,从而提升检索性能。

实验配置 B 与 C 的对比进一步表明,Margin 约束有助于优化特征空间结构。加入 Margin 后,Recall@1 提升至 89.1%,MRR 达到 0.91;同时,平均正样本相似度由 0.659 6 提升至 0.730 9,平均负样本相似度降至 0.006 0。结果说明,Margin 约束能够增强类内聚合和类间分离,从而提高模型的判别性与鲁棒性。

3.4.3 效率与开销分析

本文评估了自适应推理策略在性能与效率间的平衡效果。对比了 3 种策略:1)Full Reasoning(Qwen3-coder-30B);2)No Reasoning(仅使用 7B 嵌入);3)Adaptive(BinThink 自适应策略)。实验结果如表 4 所示。

表 4 不同推理策略下的性能与开销对比				
策略	Recall@1 (XO)/%	推理延迟/s	显存占用/GB	相对加速比
No Reasoning	82.1	1.8	14.5	25.0×
Full Reasoning	90.2	45.0	48.0*	1.0×
Adaptive	89.4	15.8	16.2	2.8×

表 4 结果表明,Full Reasoning 虽取得最高 Recall@1(90.2%),但平均推理延迟达 45 s,部署成本

较高(表格中*表示该策略的显存占用已达到当前实验环境可用显存总容量 48 GB,实际峰值显存需求可能高于 48 GB,受硬件条件限制无法进一步测得)。No Reasoning 虽将延迟降至 1.8 s、显存降至 14.5 GB,但 Recall@1 仅为 82.1%,精度下降明显。相比之下,BinThink(Adaptive)在较小精度损失下显著提升了效率:相较 Full Reasoning,其平均推理延迟降低约 65%,Recall@1 仅下降 0.8 个百分点;相较 No Reasoning,Adaptive 策略将 Recall@1 由 82.1% 提升至 89.4%,提高了 7.3 个百分点,即相对提升约 8.9%,同时仅增加了有限的计算开销。结果表明,BinThink 在检测精度与运行效率之间实现了更优平衡。

4 结束语

本文提出了 BinThink 框架,将二进制到源代码相似性检测建模为“自适应推理—增强表征—对比检索”过程,从而提升了跨模态语义对齐能力。实验结果表明,BinThink 在准确性、跨架构鲁棒性和大规模检索能力方面均优于现有方法。但是,BinThink 仍存在一定局限:1)深度推理仍带来较高计算开销,在超大规模代码库场景下的效率仍有提升空间;2)在高强度混淆或超长函数场景下,模型性能可能受到一定影响。未来将进一步探索轻量化推理与复杂场景下的鲁棒语义建模方法。

参考文献:

[1] 陈斌. 基于代码相似性的跨平台二进制漏洞检测技术研究[D]. 郑州: 信息工程大学, 2021.DOI: 10.27188/d.cnki.gzjxu.2021.000032.

[2] 方磊. 基于神经网络的二进制代码相似性检测关键技术研究[D]. 郑州: 信息工程大学, 2021.DOI: 10.27188/d.cnki.gzjxu.2021.000034.

[3] 魏有缘, 宋建华, 张龔. 二进制代码相似性检测方法综述[J]. 计算机科学, 2025, 52(6): 365-380.

[4] Haq I U, Caballero J. A survey of binary code similarity [J]. ACM Computing Surveys, 2022, 54(3): 1-38.

[5] Marcelli A, Graziano M, Ugarte-pedrero X, et al. How machine learning is solving the binary function similarity problem [C] //Proceedings of the 31st USENIX security symposium. Berkeley, USA: USENIX Association, 2022: 2099-2116.

[6] Feng Z Y, Guo D Y, Tang D Y, et al. CodeBERT: a pre-trained model for programming and natural languages [C] //Findings of the association for computational linguistics: EMNLP 2020. Stroudsburg, USA: ACL, 2020:

- 1536-1547.
- [7] Dong C P, Guo J D, Yang S G, et al. Advancing binary code similarity detection via context-content fusion and LLM verification [C] //Proceedings of the 40th IEEE/ACM international conference on automated software engineering. Piscataway, USA: IEEE, 2025:304-316.
- [8] Wang H J, Lin Z Q, Wang H J, et al. vSim: semantics-aware value extraction for efficient binary code similarity analysis [C] //Proceedings of the 2026 network and distributed system security symposium. San Diego, USA. Internet Society, 2026. DOI:10.14722/ndss.2026.240213.
- [9] 严尹彤,于璐,王泰彦,等.基于Jump-SBERT的二进制代码相似性检测技术研究[J].计算机科学,2024,51(5):355-362.
- [10] Zhou L, Dacier M, Konstantinou C. ReGraph: a tool for binary similarity identification [C] //Proceedings of the 34th ACM SIGSOFT international symposium on software testing and analysis. New York, USA: ACM,2025:6-10.
- [11] Miyani D, Huang Z, Lie D. BinPro: a tool for binary source code provenance [PP/OL]. arXiv (2017-11-02) [2026-02-17]. <https://doi.org/10.48550/arXiv.1711.00830>.
- [12] Yuan Z M, Feng M Y, Li F, et al. B2SFinder: detecting open-source software reuse in COTS software [C] //Proceedings of the 2019 34th IEEE/ACM international conference on automated software engineering. Piscataway, USA: IEEE, 2019:1038-1049.
- [13] Duan R A, Bijlani A, Xu M, et al. Identifying open-source license violation and 1-day security risk at large scale[C]//Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. New York, USA: ACM, 2017:2169-2185.
- [14] Ding S H H, Fung B C M, Charland P. Asm2vec: boosting static representation robustness for binary clone search against code obfuscation and compiler optimization [C]//Proceedings of the 2019 IEEE symposium on security and privacy. Piscataway, USA:IEEE, 2019:472-489.
- [15] Yu Z P, Zheng W X, Wang J Q, et al. CodeCMR: cross-modal retrieval for function-level binary source code matching [C] // Advances in neural information processing systems. Red Hook, USA: Curran Associates Inc., 2020,33:3872-3883.
- [16] Yu Z P, Cao R, Tang Q Y, et al. Order matters: semantic-aware neural networks for binary code similarity detection [C] //Proceedings of the AAAI conference on artificial intelligence. Palo Alto, USA: AAAI Press, 2020,34(1):1145-1152.
- [17] Luo Z H, Wang P F, Wang B S, et al. VulHawk: cross-architecture vulnerability detection with entropy-based binary code search [C] //Proceedings of the 2023 network and distributed system security symposium. Rosten, USA: Internet Society, 2023. DOI:10.14722/ndss.2023.24415.
- [18] Pei K X, Xuan Z, Yang J F, et al. Trex: learning execution semantics from micro-traces for binary similarity [PP/OL]. arXiv (2021-04-26) [2026-02-17]. <https://doi.org/10.48550/arXiv.2012.08680>.
- [19] Massarelli L, Di Luna G A, Petroni F, et al. SAFE: self-attentive function embeddings for binary similarity [C] // Detection of intrusions and malware, and vulnerability assessment. Cham, Switzerland: Springer, 2019:309-329.
- [20] Li X, Qu Y, Yin H. PalmTree: learning an assembly language model for instruction embedding [C] //Proceedings of the 2021 ACM SIGSAC conference on computer and communications security. New York: ACM, 2021:3236-3251.
- [21] Jiang L, An J W, Huang H H, et al. BinaryAI: binary software composition analysis via intelligent binary source code matching [C] //Proceedings of the IEEE/ACM 46th international conference on software engineering. New York, USA: ACM, 2024:1-13.
- [22] He H J, Lin X W, Weng Z A, et al. Code is not natural language: unlock the power of semantics-oriented graph representation for binary code similarity detection [C] // Proceedings of the 33rd USENIX security symposium. Berkeley, USA: USENIX Association, 2024:1759-1776.

(编辑: 邵 华)