计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(9): 2302−2318, 2026

基于大语言模型的源代码漏洞检测方法

曹明生[1,2] 丁桥隆[1,2] 张文清[1] 李延斌[3] 林 迪[1] 许 辉[4]

[1](电子科技大学信息与软件工程学院 成都 610054)

[2](网络与数据安全四川省重点实验室(电子科技大学) 成都 610054)

[3](山东大学软件学院 济南 250101)

[4](电子科技大学(深圳)高等研究院 深圳 518000)

(cms@uestc.edu.cn)

Source Code Vulnerability Detection Methods Based on Large Language Models

Cao Mingsheng[1,2], Ding Qiaolong[1,2], Zhang Wenqing[1], Li Yanbin[3], Lin Di[1], and Xu Hui[4]

[1](*School of Information and Software Engineering*, *University of Electronic Science and Technology of China*, *Chengdu* 610054)

[2](*Network and Data Security Key Laboratory of Sichuan Province*(*University of Electronic Science and Technology of China*),*Chengdu*610054)

[3](*School of Software*, *Shandong University*, *Jinan* 250101)

[4](*Shenzhen Institute for Advanced Study*, *University of Electronic Science and Technology of China*, *Shenzhen* 518000)

Abstract To address the limitations of large language models (LLMs) in source code vulnerability detection—namely insufficient utilization of program structural information, unstable reasoning processes, and high fine-tuning costs—we propose a vulnerability detection framework that integrates graph-based representations with LLM reasoning capabilities. The proposed method constructs data flow graphs (DFGs) from source code to model variable dependencies and semantic propagation paths, and aligns graph features with the semantic embedding space of LLMs through a cross-modal mapping module, enabling effective injection of structural knowledge into the language model. Furthermore, a progressive pretraining strategy based on code reconstruction tasks is designed to enhance the graph encoder's ability to capture vulnerability-related structural patterns. On this basis, structured reasoning distillation and parameter-efficient fine-tuning via low-rank adaptation (LoRA) are incorporated to transfer vulnerability analysis knowledge from teacher models to student models, improving detection performance while reducing training overhead. In addition, a chain-of-thought reasoning process tailored for vulnerability detection is introduced to improve interpretability and reasoning consistency. Experimental results demonstrate that the proposed method significantly improves both the accuracy and robustness of source code vulnerability detection while maintaining low fine-tuning cost.

Key words source code vulnerability detection; large language model; graph neural network; parameter-efficient fine-tuning; chain-of-thought reasoning

收稿日期: 2026-04-16; **修回日期**: 2026-07-16

基金项目: 国家自然科学基金项目(62471090, 62472259, 62576079); 中央高校基本科研业务费专项资金(ZYGX2024Z016); 四川省智能终端重点实验室(SCITLAB-20005); 泰山学者工程专项经费资助(tsqn202408026); 山东省自然科学基金项目(ZR2024QF098); 飞行器流体物理全国重点实验室基金(2025-APF-KFMS-21)

This work was supported by the National Natural Science Foundation of China (62471090, 62472259, 62576079), the Fundamental Research Funds for the Central Universities (ZYGX2024Z016), the Intelligent Terminal Key Laboratory of Sichuan Province (SCITLAB-20005), the Program of Taishan Young Scholars of Shandong Province (tsqn202408026), Shandong Provincial Natural Science Foundation (ZR2024QF098) and the National Key Laboratory of Aerospace Vehicle Fluid Physics Fund (2025-APF-KFMS-21).

通信作者: 许辉(huixu.kim@uestc.edu.cn)

摘　要　针对现有大语言模型（large language model, LLM）在源代码漏洞检测中对程序结构信息利用不足、推理稳定性较弱以及微调成本较高等问题，提出一种融合图结构表征与 LLM 推理能力的漏洞检测方法。该方法以源代码数据流图为基础，对变量依赖关系和语义传播路径进行建模，并通过跨模态映射将图结构特征对齐到语言模型语义空间，实现程序结构知识的有效注入。同时，设计基于代码重构任务的渐进式图编码器预训练策略，以增强模型对漏洞相关结构模式的感知能力。在此基础上，结合结构化推理蒸馏与低秩自适应参数高效微调机制，在降低训练开销的同时提升漏洞判别性能。进一步，引入面向漏洞检测的思维链推理流程，增强检测结果的可解释性与推理一致性。实验结果表明，该方法能够有效提升源代码漏洞检测的准确性和稳定性。

关键词　源代码漏洞检测；大语言模型；图神经网络；参数高效微调；思维链推理

中图法分类号　TP311.13；TP309

DOI: 10.7544/issn1000-1239.202660229　　**CSTR:** 32373.14.issn1000-1239.202660229

随着软件系统规模的不断扩张和开源组件的深度耦合，源代码中安全隐患的辐射传播速度越来越快，高效、准确的自动化源代码漏洞检测方案已成为研究热点。其中，基于大语言模型（large language model, LLM）的方案持续涌现，成为漏洞分析与代码审计的重要途径。目前，基于 LLM 的漏洞检测方案主要有：基于微调的 LLM 漏洞检测方案，如 Li 等人[1]通过双阶段低秩自适应微调提升了 LLM 的漏洞检测能力，Du 等人[2]提出了基于多任务指令微调的代码漏洞检测增强方法。基于提示（prompt）工程的检测方案，如 Steenhoek 等人[3]综合评估了 LLM 在提示驱动下的漏洞发现能力，Zhou 等人[4]进一步梳理了提示策略在真实场景中的应用局限。基于检索增强生成（retrieval-augmented generation, RAG）的漏洞检测方案，如 Daneshvar 等人[5]探索了基于 RAG 机制的漏洞上下文动态扩充与增强。上述方法在代码语义理解与上下文补充方面取得了一定进展，但大多仍以线性文本序列作为主要建模对象。实际上，源代码漏洞的产生往往与变量依赖、控制约束及危险操作触发等结构化因素密切相关，在真实工程环境中还常表现出跨语句、跨分支乃至跨模块的传播特征，因此仅依赖序列建模难以完整刻画其底层逻辑。基于此，研究者开始尝试引入程序图表示技术，以增强模型对程序结构信息的表达能力。

程序图表示技术能够显式刻画程序中变量的定义、传播与使用关系，从而为漏洞检测提供更贴近程序执行机制的结构先验。主要的代码图融合分析技术，如 Zhang 等人[6]提出的图对齐 LLM，以及 Liu 等人[7]结合代码属性图与语言模型的联合框架，在提升语义提取能力的同时，采用特定结构将高层拓扑特征进行聚合，以捕获跨语句、跨模块的长距离数据依赖。将图结构表征直接引入 LLM，在理论上能最大程度地弥补纯文本序列建模的缺陷，然而难以直接适配实际的代码审计环境，原因在于：LLM 的嵌入空间与高维图结构之间存在语义鸿沟，且全量融合结构特征的训练成本极高。因此，亟需适用于实际漏洞检测场景的图文融合检测方案，该方案面临以下挑战：

1）程序图结构与 LLM 序列表示之间存在模态鸿沟。漏洞检测所依赖的数据流与控制依赖关系通常以离散图结构形式存在，而预训练 LLM 主要面向一维标记序列进行建模，缺乏对程序图节点及其拓扑关系的直接表示能力。

2）图结构增强下的 LLM 适配面临较高训练成本。在将图结构信息引入漏洞检测任务后，模型不仅需要学习文本语义，还需要进一步建立结构信息与漏洞判定之间的关联。若直接对 LLM 进行全量微调，不仅显存与算力开销巨大，也不利于在实际安全分析场景中部署。

3）漏洞检测过程缺乏稳定的结构化推理约束。复杂漏洞的识别通常涉及控制流边界分析、污染源定位、数据传播追踪与危险操作确认等多步逻辑过程，而 LLM 自由生成时容易停留于表层语义匹配，难以形成一致、可验证的因果推演链条。

针对上述挑战，本文设计了一种融合结构感知微调与思维链提示的图对齐推理漏洞检测框架（graph-aligned reasoning for vulnerability detection, GAR-VD）。主要贡献包括 3 个方面：

1）提出了一种多模态图表征的注入机制，该机制基于数据流图提取拓扑特征，并通过图神经网络与交叉注意力适配器（adapter）将高维节点状态投影至 LLM 的连续潜空间，实现图文特征的空间对齐；

2)提出了图感知推理蒸馏与低秩适配(low-rank adaptation, LoRA)联合微调方法，通过单教师模型提取结构化漏洞分析轨迹作为监督信号，在极低的算力开销下使基座模型内化底层的因果推演逻辑；

3)提出了定制化的漏洞思维链(vulnerability chain-of-thought, Vuln-CoT)机制，通过五阶段自然语言指令约束模型严格执行控制流解析、污染源界定、数据流追踪、安全校验及槽点触发确认，显著提升了推理过程的准确性与可解释性。

1 相关工作

1.1 基于微调的 LLM 漏洞检测

随着 LLM 在代码理解与生成方面能力的提升，将预训练范式应用于源代码漏洞检测逐渐成为热点。与传统方法相比，该方法可复用模型在海量语料上积累的代码语义与结构知识，有效增强对隐蔽漏洞模式的识别能力。早期研究重点评估了 LLM 作为漏洞检测器的潜力与局限，实验显示，在零样本和少样本场景下 LLM 对部分漏洞任务的检测能力已超越传统工具基线，但推理稳定性欠佳且对提示扰动较为敏感[8]。进一步的系统评测则揭示，尽管 LLM 在代码语义建模方面具有优势，但在面对深层数据流交互和复杂逻辑依赖的真实漏洞场景时，其推理鲁棒性仍显不足[9]。

在此基础上，部分研究转向监督微调范式，通常将漏洞检测建模为函数级二分类任务，并结合参数高效微调机制实现任务适配。相关结果表明，微调后的 LLM 在多个主流基准数据集上的检测效果显著优于基座模型[1,10]。然而，随着研究推进，单一微调路径的局限性逐渐显现。文献 [6] 指出，即便经过深度微调，模型在处理复杂控制流或跨函数数据流依赖场景时，性能仍会明显下降，这表明仅依赖一维文本序列建模难以充分捕获程序的结构拓扑信息。与此同时，文献 [1] 指出，随着基座模型规模持续扩大，其在漏洞检测任务上的性能增益呈现递减趋势，单纯依赖参数扩展与微调优化已难以带来持续提升[1]。

为弥补上述不足，近期研究尝试在微调阶段引入结构化先验。Feng 等人[11] 通过融合程序图结构或领域规则，引导模型关注关键语义节点，从而克服纯文本建模的局限；Liu 等人[7] 进一步将代码语言模型与代码属性图相结合，实现了表层语义与底层控制流、数据流特征的联合建模，有效提升了检测效能。然而，这类方法通常依赖额外静态分析工具链，增加了计算开销与部署复杂度。

除检测性能外，任务范式的局限性亦逐渐引起学界的关注。Xu 等人[12] 指出，将漏洞检测简单抽象为二分类问题，难以满足真实安全审计中对漏洞位置、类别及成因分析的需求。受此启发，部分工作将研究范畴拓展至自动化漏洞修复。Du 等人[2] 的研究表明 LLM 在低复杂度漏洞修复中已展现出一定的应用潜力，但修复成效仍高度依赖于上游检测与定位阶段所提供的上下文质量。

将 LLM 直接作为端到端漏洞检测器，在推理稳定性与工程可扩展性方面仍面临挑战。为此，近期研究逐渐转向混合式检测架构，即将 LLM 嵌入传统程序分析流程，令其承担语义特征提取或辅助分析等角色，该路径被认为更具现实可行性[13]。由此可见，如何在完整的漏洞分析流程中合理界定 LLM 的角色与功能，仍是当前亟待突破的核心课题。

1.2 基于提示工程的漏洞检测

伴随 LLM 在代码理解与生成方面能力的提升，研究者开始探索不依赖额外微调的提示(prompt)驱动漏洞检测方法。该类方法利用模型的上下文学习能力，通过构建提示模板直接执行漏洞识别、分类及风险分析等任务。相较于监督微调方法，Prompt 方法在降低标注成本和提升跨域泛化能力方面展现出一定优势，但在输出稳定性、推理可信度及工程可用性仍存在明显局限。

早期研究主要关注 Prompt 驱动漏洞检测的可行性。已有实证结果表明，LLM 在零样本和少样本设定下能够依据自然语言指令对代码安全性作出判断，并在部分基准数据集上取得较好效果；但进一步研究发现，这类结果往往依赖特定数据分布与定制化提示设计，在真实场景中模型预测容易出现较强波动和过度自信现象[3]。与此同时，学界也开始反思现有评测范式。Nong 等人[13] 指出，部分研究依赖人工平衡数据集或小规模测试集，难以真实反映软件工程中的长尾漏洞分布，且预训练语料的不透明性还可能带来潜在数据泄露风险，从而影响评测结论的可靠性。Yang 等人[14] 进一步指出，将复杂漏洞检测简化为“代码—标签”判别任务，会忽略漏洞触发所依赖的上下文关系与多步推理过程，而这一问题在有限交互的 Prompt 机制下尤为突出。

为缓解上述局限，部分研究开始从流程重构角度重新定位 Prompt 在漏洞检测中的作用。Lu 等人[15] 认为，漏洞分析更适合被组织为多阶段递进过程，LLM 应主要承担语义理解和候选筛选等中间分析功能，

而非直接作为最终决策单元。Zhou 等人[4]系统比较了 Prompt 驱动与微调驱动 2 类方法，指出前者在迁移灵活性方面具有优势，但在检测稳定性、推理开销和部署可行性方面仍面临挑战。进一步地，相关研究发现，基础自然语言提示往往难以充分激发模型的深层漏洞推理能力；在 Prompt 中引入漏洞模式描述、参考样本或外部知识虽可提升检测精度，但也会加剧模型对提示扰动的敏感性。在面向工程实践的场景中，Mathews 等人[16]结合检索增强生成机制补充跨文件上下文信息，但实验表明该策略仍存在较高误报率，通常需要额外人工或规则校正。

此外，后续研究还尝试通过更复杂的 Prompt 机制提升检测质量。Mao 等人[17]利用多角色 Prompt 和协商式推理，引导模型在多轮交互中修正初始判断，在精确率和召回率上取得一定改进，但其推理开销较高，限制了大规模应用。Zhang 等人[18]则表明，在 Prompt 中引入 API 调用序列、数据流信息和链式思维等内容，虽能显著影响模型的检测行为，但此类增强策略在不同编程语言与漏洞类型下的效果波动较大，尚缺乏统一设计准则。

为降低 Prompt 自由推理带来的不确定性，近期研究开始关注 LLM 与传统程序分析技术的协同。Ding 等人[19]先利用静态分析工具标定潜在漏洞位置，再通过 Prompt 引导模型完成语义解释与结果筛选，从而在降低误报率的同时展现出混合式检测框架的工程潜力。在此基础上，Khare 等人[20]进一步引入形式化约束与一致性校验机制对模型输出进行验证，表明在安全关键任务中，检测结果的可解释性与可验证性应与性能指标并重。

1.3 基于检索增强生成的漏洞检测

随着 LLM 在代码安全分析中的应用日益深化，研究者逐渐认识到，仅依赖模型参数中隐含的知识，难以有效应对漏洞特征快速演化、代码上下文复杂以及模型幻觉等问题。在此背景下，检索增强生成机制被引入源代码漏洞检测任务，通过动态检索外部知识或相似代码样本，为模型提供额外上下文支撑，已成为该方向的重要研究路径。

现有研究首先从评测角度验证了 RAG 的必要性。Kouliaridis 等人[21]的基准测试表明不同 LLM 在漏洞检测任务上的性能差异显著，且对特定漏洞类型具有较强敏感性；在上下文信息不足时，该局限更加突出。Daneshvar 等人[5]进一步指出，在真实软件工程场景中，仅依赖 Prompt 驱动的 LLM 容易因全局上下文缺失而产生较高误报率，而检索机制则在一定程度上增强模型对跨文件依赖关系和全局安全背景的理解。

在方法设计层面，已有研究尝试将 RAG 与提示工程结合以提升检测性能。Mathews 等人[16]通过检索漏洞描述、历史样本及相关代码切片，为模型补充分析上下文，但实验结果表明，该方法仍存在将代码过度判定为不安全的问题，说明单纯引入 RAG 尚不足从根本上缓解误报问题。Sheng 等人[22]的研究表明将 RAG 与链式思维提示结合，能够在受控环境下进一步提升检测效果，但此类方法对数据集构造方式和检索语料质量依赖较高，仍面临信息泄露与跨域泛化不足等问题。

若将视角扩展至更广泛的安全知识推理任务，RAG 同样表现出显著潜力。Simoni 等人[23]指出，通过多检索器协同整合 CVE、CWE 及漏洞利用数据库，能够提升模型在安全问答与漏洞知识解析任务中的准确性，并在一定程度上缓解幻觉现象。类似地，引入溯源追踪与自我评估机制的 RAG 框架也被用于漏洞情报分析与缓解策略生成，在可信性与可解释性方面取得较好效果[24]。但总体而言，此类研究主要面向宏观安全知识分析，尚未直接触及细粒度的源代码级漏洞判定任务。

进一步地，RAG 还被拓展到漏洞检测之外的相关任务。Ding 等人[19]在已知漏洞位置或实例条件下，利用 RAG 动态检索补丁片段和修复案例，引导 LLM 生成修复策略。Du 等人[25]则通过多轮交互式推理框架，增强了模型对漏洞成因与修复逻辑的理解。Yoon 等人[26]提出多层次 RAG 架构，通过整合异构漏洞数据库提升漏洞情报分析的时效性与完备性，但其研究重点仍聚焦于漏洞情报层面，尚未深入至源代码级检测任务。

总体而言，当前将 RAG 直接应用于源代码级漏洞检测的研究仍然较少。代表性方法通常通过向量检索召回与目标代码语义相近的先验样本，并将其漏洞标签及分析逻辑作为增强上下文输入模型，从而激发 LLM 的上下文学习能力，在规避高成本微调成本的同时取得较优的检测效果[27]。这表明，合理设计的 RAG 框架能够有效提升 LLM 在源代码漏洞检测中的实用性，但其性能仍然受制于检索语料质量、上下文组织方式以及数据分布一致性等因素。

2 预备知识

2.1 漏洞检测的基本概念

软件漏洞检测贯穿程序生命周期，旨在识别代

码实现与系统配置阶段存在的安全隐患,并对其风险进行判定与量化。根据相关标准[28],漏洞是软硬件计算逻辑中的内在脆弱性,一旦被攻击者利用,可能导致系统机密性、完整性或可用性受损。与常规软件缺陷分析侧重程序行为是否偏离设计规范不同,漏洞检测更关注潜在攻击路径及其可能造成的安全后果,因此更强调对系统安全边界失效机制的刻画。

在源代码漏洞检测任务中,检测对象通常包括函数、代码片段、文件或补丁差分,输出形式可为漏洞存在判定、漏洞类型识别或漏洞位置定位。其复杂性主要来源于漏洞触发机制的非局部性,即安全问题往往不是单条语句异常,而是局部敏感操作与全局依赖链共同作用的结果。不可信数据或异常状态可能通过赋值、传参、指针别名等方式持续传播,并受到分支、循环、异常处理等控制流结构约束。因此,漏洞检测本质上依赖于对数据依赖关系和控制依赖关系的有效建模。

从技术路径看,现有方法主要包括静态分析、动态分析和混合分析。静态分析依赖代码特征和预设规则进行推断,适合开发前期使用,但易产生误报;动态分析通过程序运行行为识别漏洞,可更有效验证可利用性,但成本较高且依赖测试覆盖;混合分析则尝试在效率与准确性之间取得平衡。

随着深度学习在代码理解中的发展,漏洞检测又形成了序列建模、图建模和LLM驱动3类代表性路线。序列建模实现简单,但对程序结构关系刻画不足;图建模可表达跨语句依赖关系,但依赖图构建质量且泛化性受限;LLM驱动在语义理解和迁移能力上更具潜力,但仍需解决结构信息注入、上下文窗口受限和标注数据不足等问题。

2.2 图神经网络

图神经网络(graph neural network, GNN)是一类面向图结构数据的学习模型,其目标在于通过表征学习提取节点本身及其所处拓扑环境中的语义信息[29-30]。这类方法的基本出发点是“消息传递”机制,即在图结构上反复进行邻域信息聚合,使节点表示在层层传播过程中不断吸收更大范围的结构上下文。对于给定图$G=(V,E)$,其中节点集合V与边集合E分别表示实体与实体间的关系。在第k层网络中,GNN一般采用“聚合-更新”的方式来计算节点v的隐层表示:

$$\boldsymbol{h}_v^{(k)}=\phi\left(\boldsymbol{h}_v^{(k-1)},\ \bigoplus_{u\in\mathcal{N}(v)}\psi\left(\boldsymbol{h}_u^{(k-1)},\boldsymbol{h}_v^{(k-1)},e_{uv}\right)\right),\qquad(1)$$

式中,$\boldsymbol{h}_v^{(k)}$表示节点v在第k层的特征向量,$\mathcal{N}(v)$表示节点v的邻域集合,e_{uv}表示相应的边特征。其中,函数$\psi(\cdot)$用于生成由邻居节点传递而来的消息,$\oplus$表示一种满足排列不变性的聚合算子,而$\phi(\cdot)$对应更新函数,用来结合节点自身已有的历史状态以及聚合后的邻居信息。随着消息传递层数的增加,节点表示能够逐步吸收来自多跳邻域的结构信息,进而在表示空间中刻画图结构内部所包含的关系模式,并为后续关系推理和图级任务提供可用的表示基础。

为增强模型对复杂图结构的刻画能力,尤其是在不同邻接节点对中心节点贡献并不一致的情况下,图注意力网络(graph attention network, GAT)[30]被提出。传统图卷积网络在进行邻域信息聚合时,通常依赖给定图拓扑下的固定权重;与此不同,GAT将注意力机制引入图结构建模过程中,使模型能够在聚合邻域特征时,为不同节点分配动态的自适应权重。

2.3 LLM

语言模型(language model, LM)主要用于刻画自然语言或形式化语言(如程序源代码)序列的概率分布。对于一个给定的句子或代码片段,它既可以用来估计该序列出现的概率,也可以在已有历史上下文的条件下,对下一个词(Token)出现的可能性进行预测。近年来,随着计算能力的持续提升以及大规模语料资源的不断积累,LM[31-32]逐渐由早期的浅层统计建模方法演进为深层大规模预训练模型,并在代码理解、代码生成和复杂推理等任务中表现出更强的能力。

早期语言建模方法大多建立在马尔可夫假设之上,即认为当前词出现的概率只与其前面的$n-1$个词有关。这类依据统计共现频率构建的模型通常被称为N-gram模型。对于长度为T的序列$W=(w_1,w_2,\cdots,w_T)$,其整体概率通常可以近似写成各条件概率的连乘形式。统计LM在早期自然语言处理任务中曾发挥过一定作用,但这类方法对上下文范围的扩展并不友好。随着窗口长度的增大,模型参数数量会迅速增长,进而带来明显的维度负担,同时也难以处理代码中常见的长距离依赖关系。针对这一问题,后续研究开始转向神经语言模型,并引入词嵌入技术,将原本离散的符号表示映射到低维连续的稠密向量空间中。在此基础上,循环神经网络及其相关变体逐渐成为序列建模的重要方法。与传统统计模型相比,神经语言模型能够借助隐状态保留历史信息,在连续向量空间内表达语义相近性,因此在一定程度上改善了时序依赖建模效果。不过,受递归计算方式的限制,这类模型仍然难以有效应对长距离梯度衰减问题,同时在训练过程中也不易实现

高效并行。

Transformer 架构提出之后，基于纯注意力机制的建模方式逐渐摆脱了传统序列模型对计算顺序的强依赖，语言建模也由此进入预训练 LM 的发展阶段[31]。以 BERT 和 GPT 为代表的一类模型推动形成了新的研究范式，即先在大规模无监督语料上进行预训练，再面向具体下游监督任务完成微调。这一时期的模型参数规模通常已达到千万级至数亿级，因而能够较充分地学习上下文中的深层语义信息[33]。在此基础上，随着预训练模型参数量继续大幅扩展，并在规模上跨越某些关键区间，模型能力的提升开始不再局限于相对平缓的增长趋势，而是逐步表现出更强的复杂推理能力和零样本泛化能力。通常情况下，学术界将这类参数规模达到百亿乃至万亿量级的模型统称为 LLM[32-34]。

从当前主流的开源 LLM 来看，其整体架构设计大多已不再沿用传统的编码器-解码器结构，而是更多采用纯解码器（decoder-only）形式。这种架构与自回归语言生成方式之间具有较好的适配性。设输入的 Token 序列为 $X=(x_1,x_2,\cdots,x_n)$，自回归语言建模的目标是最大化该序列的联合概率，即通过链式法则将其分解为各时间步条件概率的乘积：

$$p(x_{1:n})=\prod_{t=1}^{n}p(x_t|x_{<t}), \tag{2}$$

式中，x_t表示第t个 Token；$p(x_t|x_{<t})$表示在已有上下文条件下生成x_t的条件概率。这种解码范式使得 LLM 能够以统一的形式处理文本生成、漏洞代码补全与条件推断等多种任务[35]。

3 基于 LLM 的源代码漏洞检测方法框架

3.1 GAR-VD 方法总体架构

针对传统静态分析在复杂代码场景下误报率较高、维护成本较大，以及纯文本 LLM 难以充分建模程序控制流与数据流结构的问题，本文提出一种图对齐推理漏洞检测框架 GAR-VD，如图 1 所示。该框架在不改变 LLM 自回归解码结构的前提下，引入程序结构感知与漏洞推理能力，以提升复杂源代码场景下的检测效果。

GAR-VD 整体可分为 2 个阶段。第 1 阶段为多模态图表征隐式注入阶段。该阶段首先借助静态分析工具提取代码数据流图（data flow graph, DFG），以刻画变量定义、传播与使用关系；随后通过 GNN 与交叉注意力适配器将图节点特征映射到与文本词向量对齐的连续潜空间，形成可被 LLM 接收的图 Token。在冻结 LLM 主体参数的条件下，适配器逐步完成图结构表征与语言嵌入空间的对齐，从而使模型在文本序列之外具备对程序内部结构信息的感知能力。

第 2 阶段为图感知推理蒸馏与 LoRA 联合微调阶段。在完成结构对齐后，框架进一步引入教师模型生成的高质量漏洞推理轨迹，并采用 LoRA 对基座模型进行参数高效微调，使其在接收文本 Token 与图 Token 联合输入的条件下学习结构化漏洞分析过程。通过该阶段训练，模型不仅能够利用程序结构先验，还能够逐步建立结构信息与漏洞判定结果之间的推理对应关系。

在检测阶段，GAR-VD 进一步结合漏洞思维链提示机制，对模型输出过程施加结构化约束，从而减少表层语义干扰与推理断裂问题，提升漏洞检测结果的稳定性与可解释性。

3.2 多模态图表征的隐式注入机制

LLM 在处理源代码时，其内置的自注意力机制主要依赖一维标记序列完成概率计算。受这种序列化建模方式影响，程序内部的控制逻辑与数据流转路径在输入阶段往往被压缩为线性表示，进而导致结构语义出现一定程度的降维损耗。为突破纯文本序列的表征维度限制，本文方法框架构建了多模态图表征的隐式注入阶段，旨在将图表征结构转换为

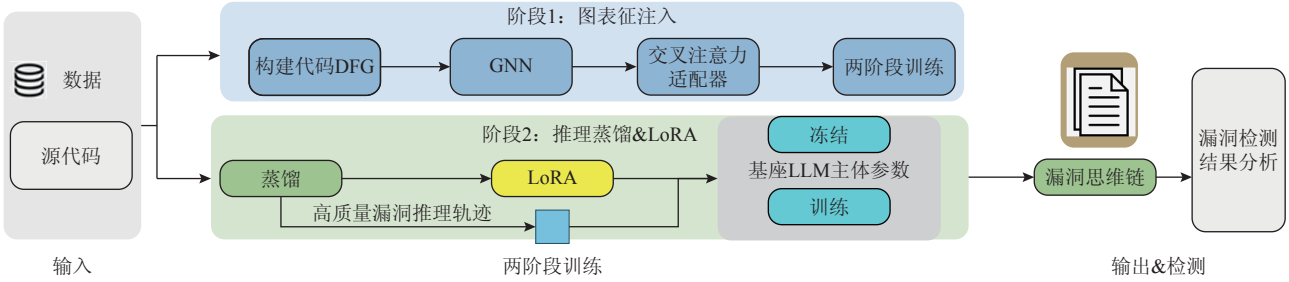


Fig. 1 Overall architecture of GAR-VD for source code vulnerability detection

图 1 GAR-VD 源代码漏洞检测总体框架

与预训练 LM 相兼容的连续向量表示。具体实施上，该阶段可划分为拓扑构建与特征对齐 2 个机制：首先，框架通过静态分析组件提取代码的依赖特征；进一步地，引入图特征编码模块与交叉注意力适配器，对高维节点状态进行聚合与映射，使离散结构信息能够对齐至文本词嵌入对应的连续潜空间。基于这一跨模态表征机制，异构信息得以在统一表示空间中建立对应关系，从而为模型联合优化与后续深层逻辑对齐提供基础。

源代码漏洞的触发本质上表现为不可信数据从输入源出发，沿程序内部的数据传播路径逐步传递，并最终在缺乏有效安全校验的情况下到达危险操作位置。相较于抽象语法树(abstract syntax tree, AST)主要刻画代码的语法层次结构与嵌套关系，DFG 能够更加直接地表征变量从定义、传播到使用的依赖链路，从而较为准确地对应漏洞检测任务中“污染源识别—数据传播跟踪—危险操作触达”的核心分析过程。Zhang 等人[6]的实验结果表明，在多种程序图表示中，DFG 的检测效果优于 AST，且在其基础上进一步与 AST 进行融合后，性能提升幅度并不明显，说明相较于语法结构信息，数据依赖信息对漏洞识别更具直接贡献。因此，本文在该阶段仅采用 DFG 作为程序结构信息的表示形式，以在简化图构建与编码流程的同时，突出与漏洞触发机制密切相关的数据依赖特征。

在构建 DFG 之前，首先对原始代码执行变量名标准化预处理。由于不同开发者在变量命名上通常存在较大差异，原始标识符中往往包含与漏洞语义无直接关联的词汇信息，容易引入额外的表示偏置，进而干扰模型对关键数据依赖关系的学习。为减弱此类非本质特征的影响，本文通过静态分析方法遍历代码的完整作用域，并将其中的用户自定义变量标识符统一替换为标准化符号序列，如 $var_1, var_2, \cdots$。该预处理操作有助于削弱非语义层面的特征扰动，降低模型在特征学习阶段的表示负担，并提升样本之间结构表示的一致性。需要说明的是，变量名标准化仅用于多模态图表征对齐及代码重构预训练阶段。在后续结构化推理蒸馏与漏洞检测任务适配阶段，模型使用保留原始变量名、函数名和 API 名称的源代码，以充分利用标识符所包含的功能语义和安全语义。

对于完成标准化处理后的代码，本文借助 Joern 工具提取其对应的 DFG。Joern 是一种面向程序分析的代码属性图构建与查询工具，能够对源代码进行静态解析，并在统一中间表示上恢复程序中的语法、控制与数据依赖信息。在本文中，重点利用 Joern 所提供的数据依赖分析能力，从代码属性图中抽取与变量定义、赋值、传递和使用相关的数据流边，进而构建代码对应的 DFG。设提取得到的数据流图为

$$G_{\mathrm{DFG}} = (V_{\mathrm{DFG}}, E_{\mathrm{DFG}}), \tag{3}$$

其中，节点集合 V_{DFG} 表示程序中的变量、表达式或相关语义单元，边集合 E_{DFG} 表示变量在定义、赋值、传递与使用过程中的数据依赖关系。通过该图结构，程序中跨语句、跨作用域的数据传播路径能够被显式保留下来，从而为后续的图表示学习提供结构化输入。

最终构建得到的 DFG 将作为后续图神经网络编码器的输入，为 LLM 提供面向漏洞检测任务的核心结构先验。由于该图结构能够较好贴合漏洞形成过程中污染数据的传播机理，因此模型在训练与推理过程中可以更加集中地关注与漏洞判定密切相关的数据依赖链路，减少无关语法结构特征带来的干扰。

数据流图 G_{DFG} 确立了源代码结构的数据依赖拓扑约束，但此类图表征仍难以直接兼容 LLM 基于一维序列的标准词嵌入空间。为解决这一问题，本文设计了跨模态特征映射网络，旨在将离散的图节点状态转化为 LM 可处理的连续向量表示。该网络由图特征编码模块与交叉注意力适配模块 2 部分构成，共同实现从拓扑空间到语义空间的表示映射与特征对齐，具体如图 2 所示。

在图特征编码阶段，框架采用 GNN 对数据流图 G_{DFG} 进行局部特征聚合计算。定义图节点的初始特征矩阵为 $\boldsymbol{H}^{(0)}$，网络基于边集合 E_{DFG} 执行多层信息传递。在每一次迭代中，目标节点通过聚合函数整合其连通邻居的结构与语义状态。假设网络包含 L 层传播计算，其最终输出的全局节点表征矩阵记为 $\boldsymbol{H}^{(L)} \in \mathbb{R}^{|V| \times d_{\mathrm{GNN}}}$。其中，$|V|$ 表示 DFG 中的节点总数，d_{GNN} 表示 GNN 预设的隐藏层特征维度。该输出矩阵反映了节点经过多层信息传播后的结构表示，可用于刻画程序控制边界与数据流转依赖的深层状态。

由于 GNN 的特征维度 d_{GNN} 通常独立于 LLM 的原生词嵌入维度 d_{LLM}，框架进一步引入交叉注意力(cross-attention)机制构建适配器模块，以实现 2 个异构特征空间之间的投影与对齐。具体而言，该适配模块定义了一组可学习的查询矩阵 $\boldsymbol{Q}_{\mathrm{g}} \in \mathbb{R}^{N_{\mathrm{g}} \times d_{\mathrm{LLM}}}$，其中 N_{g} 表示框架设定的固定图标记(graph tokens)数量。同时，将 GNN 输出的节点表征矩阵 $\boldsymbol{H}^{(L)}$ 经过线性层变换，生成对应的键矩阵 $\boldsymbol{K}_{\mathrm{g}}$ 与值矩阵 $\boldsymbol{V}_{\mathrm{g}}$：

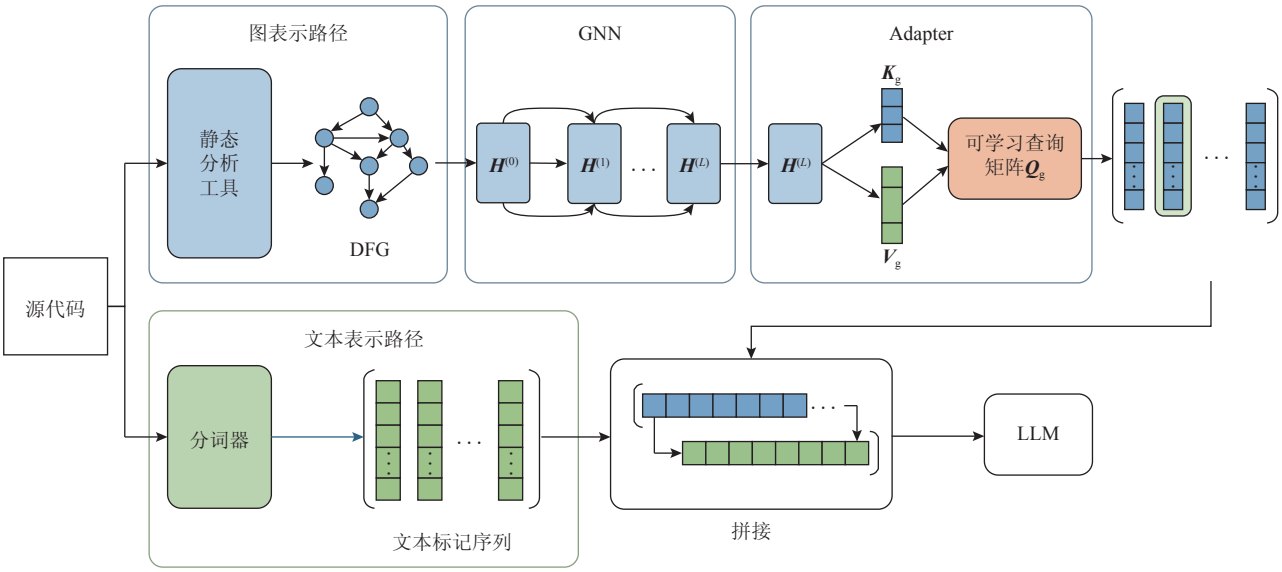


Fig. 2 Architecture of the cross-modal graph-text feature mapping network

图 2 跨模态图-文本特征映射网络结构

$$\boldsymbol{K}_{\mathrm{g}}=\boldsymbol{H}^{(L)}\boldsymbol{W}_{\mathrm{K}}, \tag{4}$$

$$\boldsymbol{V}_{\mathrm{g}}=\boldsymbol{H}^{(L)}\boldsymbol{W}_{\mathrm{V}}, \tag{5}$$

其中，$\boldsymbol{W}_{\mathrm{K}},\boldsymbol{W}_{\mathrm{V}}\in\mathbb{R}^{d_{\mathrm{GNN}}\times d_{\mathrm{LLM}}}$代表可学习的权重投影参数。

在此基础上，适配器模块通过标准的注意力机制计算图特征的映射张量：

$$\boldsymbol{X}_{\mathrm{g}}=Softmax\left(\frac{\boldsymbol{Q}_{\mathrm{g}}\boldsymbol{K}_{\mathrm{g}}^{\mathrm{T}}}{\sqrt{d_{\mathrm{LLM}}}}\right)\boldsymbol{V}_{\mathrm{g}}。 \tag{6}$$

上述计算输出的张量$\boldsymbol{X}_{\mathrm{g}}\in\mathbb{R}^{N_{\mathrm{g}}\times d_{\mathrm{LLM}}}$即为跨模态对齐后的图标记矩阵。借助点积计算与 Softmax 归一化，固定数量的图标记能够根据学习到的权重分布，自适应地从原始 DFG 中提取并压缩关键的安全拓扑信息。

相应地，对于输入代码的纯文本序列，框架通过标准分词器（tokenizer）得到文本标记序列$X_{\mathrm{t}}=(x_1,x_2,\cdots,x_{N_{\mathrm{t}}})$，其中$N_{\mathrm{t}}$表示文本序列的长度。随后，特征映射网络在序列维度上对图标记矩阵与文本标记矩阵进行拼接，从而构建面向 LLM 的联合输入张量。

$$\boldsymbol{X}=[\boldsymbol{X}_{\mathrm{g}};X_{\mathrm{t}}]\in\mathbb{R}^{(N_{\mathrm{g}}+N_{\mathrm{t}})\times d_{\mathrm{LLM}}} \tag{7}$$

通过上述网络设计，图表征数据被映射为与 Transformer 自注意力机制兼容的连续表示。基于这一联合输入范式，多模态信息得以在统一表示空间中建立对应关系，从而使基座模型在后续计算过程中能够同时结合局部文本特征与全局拓扑特征进行联合优化。

在完成跨模态特征映射网络的构建后，模型已能够将图结构表征映射到与 LLM 兼容的输入嵌入空间，并以前缀形式参与统一序列建模。然而，能够接入并不意味着已经实现有效对齐。由于 LLM 的主体参数Θ_{LLM}已在大规模代码与文本语料上完成预训练，若将随机初始化的图编码表示直接注入 LM，容易引入不稳定噪声，从而扰动其原有的代码语义分布。为此，本文进一步设计了一种基于代码重构任务的渐进式图编码器预训练策略，使图结构表示能够以平滑且受约束的方式接入 LM，并在条件生成任务中逐步完成与语言语义空间的对齐，具体如图 3 所示。

该预训练过程由 2 个连续子阶段构成：前期为接口预热阶段，后期为 LoRA 协同对齐阶段。前者用于首先建立图结构表示到语言嵌入空间的稳定映射；后者则在保持 LM 主干参数冻结的前提下，引入参数高效微调机制，使语言侧能够对图前缀进行有限适配，从而提升图结构信息的利用效率与模态对齐深度。

在上文联合输入范式下，图前缀表示作为条件信息参与 LM 的上下文建模，代码文本序列作为需要重构的目标序列。因此，本文将预训练任务建模为条件代码生成问题，其目标为：

$$P(X_{\mathrm{t}}\mid\boldsymbol{X}_{\mathrm{g}})=\prod_{i=1}^{N_{\mathrm{t}}}P(x_i\mid\boldsymbol{X}_{\mathrm{g}},x_{<i}), \tag{8}$$

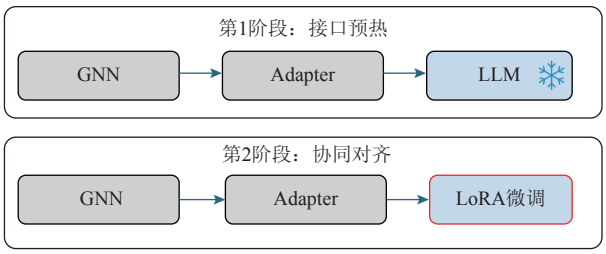


Fig. 3 Two-stage training pipeline diagram of GAR-VD

图 3 GAR-VD 两阶段训练流程图

其中, $x_{<i}=(x_1,x_2,\cdots,x_{i-1})$ 表示第i个代码标记之前的上下文序列。

基于上述定义, 代码重构损失采用掩码交叉熵形式, 仅对文本标记对应的位置进行监督:

$$\mathcal{L}_{\mathrm{G2C}}=-\sum_{i=1}^{N_{\mathrm{t}}}\log P(x_i\mid \boldsymbol{X}_{\mathrm{g}},x_{<i};\Theta),\tag{9}$$

其中, Θ表示当前训练子阶段中所有参与前向计算的参数集合。为避免图前缀位置对损失产生干扰, 在实现中对$\tilde{\boldsymbol{X}}_{\mathrm{g}}$所对应的输出位置施加损失掩码, 仅保留代码标记部分的预测误差。若记图前缀长度为N_{g}, 则更一般的掩码损失可表示为

$$\mathcal{L}_{\mathrm{G2C}}=-\sum_{j=1}^{N_{\mathrm{g}}+N_{\mathrm{t}}}m_j\log P(x_j\mid x_{<j}),\tag{10}$$

其中, 掩码变量m_j定义为

$$m_j=\begin{cases}0, & 1\leqslant j\leqslant N_{\mathrm{g}},\\ 1, & N_{\mathrm{g}}+1\leqslant j\leqslant N_{\mathrm{g}}+N_{\mathrm{t}}。\end{cases}\tag{11}$$

由此可见, 图标记部分仅承担条件前缀作用, 而代码标记部分才是真正的监督目标。

在接口预热阶段, LM 主干参数Θ_{LLM}保持冻结, 仅更新图神经网络参数Θ_{GNN}与适配器参数$\Theta_{\mathrm{Adapter}}$。因此, 该阶段的优化目标可写为

$$\min_{\Theta_{\mathrm{GNN}},\Theta_{\mathrm{Adapter}}}\mathcal{L}_{\mathrm{G2C}}(\Theta_{\mathrm{GNN}},\Theta_{\mathrm{Adapter}};\Theta_{\mathrm{LLM}}),\tag{12}$$

其中, Θ_{LLM}在该阶段视为固定参数, 不参与梯度更新。这一阶段的核心目的是在不扰动预训练 LM 内部表示空间的前提下, 使图结构表示首先获得稳定的前缀表达能力。换言之, 此阶段主要由图侧模块学习如何将抽象语法结构与数据依赖关系转换为 LM 能够识别的连续条件表示。

当图前缀经过前一阶段训练后具备初步稳定性, 框架进入 LoRA 协同对齐阶段。在该阶段中, 仍保持 LM 主干参数Θ_{LLM}冻结, 但在其若干线性投影层中引入低秩适配矩阵, 以参数高效微调的方式增强 LM 对图前缀的响应能力。设 LoRA 参数为Θ_{LoRA}, 则 LM 的有效参数可表示为

$$\Theta'_{\mathrm{LLM}}=\Theta_{\mathrm{LLM}}+\Delta\Theta(\Theta_{\mathrm{LoRA}}),\tag{13}$$

其中, Θ_{LLM}固定不变, 仅Θ_{LoRA}参与训练。于是, 第 2 子阶段的优化目标写为

$$\min_{\Theta_{\mathrm{GNN}},\Theta_{\mathrm{Adapter}},\Theta_{\mathrm{LoRA}}}\mathcal{L}_{\mathrm{G2C}}(\Theta_{\mathrm{GNN}},\Theta_{\mathrm{Adapter}},\Theta_{\mathrm{LoRA}};\Theta_{\mathrm{LLM}}),\tag{14}$$

其中, Θ_{LLM}同样视为固定参数, 不参与反向传播更新。

通过在语言侧引入低秩增量参数, 预训练过程由单纯依赖图侧模块的映射学习, 扩展为图编码器、适配器与语言侧轻量参数的协同优化过程。一方面, LoRA 仅引入少量可训练参数, 不会显著破坏预训练 LM 原有的代码知识; 另一方面, LM 能够在保持主干稳定的条件下, 对图前缀表示进行有限而有效的内部重整, 从而提升图结构信息在自回归生成过程中的利用率。

3.3 结构化推理蒸馏与 LoRA 微调

在图编码与跨模态特征隐式注入阶段完成后, 图表征数据已被转化为 LLM 可接收的连续表示。然而, 特征维度上的对齐仅为模型提供了处理程序结构信息的输入基础, 尚不足以支撑针对代码漏洞的因果推理过程。为使基座模型具备从底层拓扑结构推演安全缺陷触发机制的能力, 本文进一步引入图感知推理蒸馏与低秩自适应联合微调策略。该策略旨在将复杂的漏洞推演逻辑逐步内化到 LM 参数之中。具体而言, 框架采用单教师知识蒸馏机制提取结构化推理轨迹, 并将其作为显式监督信号以引导模型预测分布的调整; 同时, 利用低秩矩阵分解限制参数更新范围, 在降低整体优化资源开销的前提下, 实现图感知特征与输出标签之间的对齐。基于这一联合微调机制, 模型在安全审计任务中的决策空间得到进一步约束, 也为后续构建较一致的结构化推理方式提供了参数层面的支持。

在完成图特征空间的隐式对齐后, 基座模型已具备处理图表征结构数据的基础映射能力。然而, 为了在复杂漏洞检测任务中输出具有因果关联的逻辑分析过程, 模型仍需进一步习得深层的结构化推理范式。鉴于从零开始训练具备思维链能力的 LM 通常需要较大的算力开销以及高质量清洗语料, 本文采用知识蒸馏机制构造结构化推理监督信号。具体而言, 通过提取大参数推理模型生成的漏洞分析轨迹, 并将其作为显式监督信息, 用于引导基座模型输出分布的优化, 从而使其逐步具备面向漏洞检测任务的结构化推理能力。

在数据构造流程中, 框架定义单一的大规模推理 LM 为教师模型$\mathcal{M}_{\mathrm{teacher}}$, 其参数集合表示为$\Theta_{\mathrm{teacher}}$。为基于代码文本生成结构化漏洞分析轨迹, 框架进一步针对目标漏洞数据设计了结构化引导提示词P_{distill}, 如图 4 所示。该提示词在文本解码层面强制施加拓扑分析约束, 要求教师模型在输出最终判定前, 必须依照代码结构界定污染源边界, 并沿着数据流路径追踪变量状态的转移情况。

给定原始的源代码文本序列X_{t}, 教师模型$\mathcal{M}_{\mathrm{teacher}}$基于输入组合$(X_{\mathrm{t}},P_{\mathrm{distill}})$执行自回归解码计算, 输出中

```
你是一名资深源代码安全审计专家。你需要基于程序语义、控制关系和数据依赖关系，对给定代码进行严格审计。
你的回答必须遵循结构化漏洞分析流程，不允许只给出笼统结论。

请务必按照以下5个阶段完成分析：
[阶段1] 功能解析：说明代码片段的主要功能、关键变量、关键函数调用以及潜在敏感操作位置。
[阶段2] 污染源识别：指出所有可能由外部输入、用户输入、文件/网络输入、函数参数传入的可控数据源。
[阶段3] 数据流追踪：分析污染数据如何通过赋值、参数传递、返回值传播、条件分支等路径到达敏感操作。
[阶段4] 安全检查：检查在到达敏感操作之前，代码是否执行了边界检查、空值检查、长度限制、类型检查、转义
过滤、权限验证等安全约束。
[阶段5] 漏洞结论：基于前述分析，给出是否存在漏洞的最终判断，并说明漏洞形成或未形成的根本原因。

请严格使用如下输出格式：
[阶段1] . . .
[阶段2] . . .
[阶段3] . . .
[阶段4] . . .
[阶段5] . . .
[最终标签] 0/1
```

Fig. 4 Structured reasoning distillation prompt template

图 4 结构化推理蒸馏提示模板

间的图感知推理轨迹R与最终的漏洞预测标签$\hat{Y}$。该生成过程可形式化为条件概率的最大化求解：

$$\max P(R,\hat{Y}|X_{\mathrm{t}},P_{\mathrm{distill}};\Theta_{\mathrm{teacher}}), \tag{15}$$

其中，推理轨迹由离散的文本标记序列组成，可形式化定义为$R=(r_1,r_2,\cdots,r_m)$，m为该推理链的标记总长度。

由于LLM在处理复杂代码样本时仍可能产生局部逻辑偏差，若直接采用全量生成结果，容易向基座模型引入额外噪声。在此基础上，框架对生成结果执行严格的拒绝采样操作。设样本的真实漏洞判定标签为Y_{gt}，框架通过离散对比检验预测标签$\hat{Y}$与Y_{gt}的一致性。仅当$\hat{Y}=Y_{\mathrm{gt}}$时，对应推理轨迹R才被认定为有效表征并予以保留。由此得到的精炼蒸馏数据集D_{distill}可形式化定义为：

$$D_{\mathrm{distill}}=\{(X_{\mathrm{t}},R,Y_{\mathrm{gt}})\mid\hat{Y}=Y_{\mathrm{gt}}\}。 \tag{16}$$

通过上述生成与离散过滤机制，原始静态分类任务数据被扩展为包含图结构分析逻辑的序列化数据。由此构建的结构化蒸馏数据集D_{distill}为后续联合微调阶段提供了因果对齐依据，使基座模型能够在参数更新过程中学习包含控制流边界与数据流转移状态的漏洞分析路径，具体微调数据格式见图5。

由于基础模型通常具有数十亿级参数规模，对全局权重执行全量微调（full fine-tuning）将带来较高的显存与计算资源消耗。为此，本文框架引入低秩自适应技术，以实现参数高效微调（parameter-efficient fine-tuning）。该机制通过在冻结的预训练网络旁侧引入可训练的低秩分解矩阵，在保留基座模型原有语义表示能力的同时，降低了优化过程中的计算开销。

具体而言，定义LM中某一特定线性层的预训练权重矩阵为$\boldsymbol{W}_0\in\mathbb{R}^{d\times k}$。在微调阶段，该矩阵$\boldsymbol{W}_0$的参数被施加严格的锁定约束，框架不对其执行梯度传递与权重更新。相应地，模型在该网络层中并行引入2个低秩矩阵$\boldsymbol{A}\in\mathbb{R}^{r\times k}$，其中参数$r$刻画了矩阵分解的秩（rank），且满足$r\ll\min(d,k)$。在此机制下，该层参数的更新增量被建模为低秩矩阵的乘积$\Delta\boldsymbol{W}=\frac{\alpha}{r}\boldsymbol{B}\boldsymbol{A}$。对于给定的输入向量$\boldsymbol{x}$，线性层修改后的前向传播计算公式定义为：

$$\boldsymbol{h}=\boldsymbol{W}_0\boldsymbol{x}+\Delta\boldsymbol{W}\boldsymbol{x}=\boldsymbol{W}_0\boldsymbol{x}+\frac{\alpha}{r}\boldsymbol{B}\boldsymbol{A}\boldsymbol{x}。 \tag{17}$$

在参数初始化阶段，矩阵$\boldsymbol{A}$服从随机高斯分布，矩阵$\boldsymbol{B}$则被初始化为零矩阵。这样的初始化方式使训练起点处的输出增量满足$\Delta\boldsymbol{W}\boldsymbol{x}=\boldsymbol{0}$，从而保证模型在微调初期的输出分布与预训练阶段保持一致，并降低异构参数注入带来的特征偏移风险。

在漏洞逻辑蒸馏的实际部署中，框架将LoRA适配模块局部应用于Transformer架构的自注意力层。基于前序构建的联合输入序列S_{in}与目标输出序列S_{out}，模型的优化目标可以形式化为标准条件自回归建模问题。设目标输出序列由M个离散标记构成，即$S_{\mathrm{out}}=(y_1,y_2,\cdots,y_M)$，则模型在可训练LoRA参数集合$\Theta_{\mathrm{LoRA}}$上的联合交叉熵损失函数$\mathcal{L}_{\mathrm{SFT}}$定义为：

$$\mathcal{L}_{\mathrm{SFT}}=-\sum_{j=1}^{M}\log P(y_j|S_{\mathrm{in}},y_{<j};\boldsymbol{W}_0,\Theta_{\mathrm{LoRA}}), \tag{18}$$

其中，$y_{<j}$表示在生成第j个目标标记之前已经输出的前置标记序列。通过最小化该目标函数，反向传播产生的梯度仅更新低秩矩阵集合Θ_{LoRA}。在这一机制下，基座模型能够在较小的低维参数子空间内学习结构化漏洞推理路径，从而以较低成本实现向特定安全缺陷检测任务的知识迁移。

“Instruction”：你是一名资深源代码安全审计专家，请判断下面代码是否含有漏洞。

“Input”：

```
#define ATTR_RMTVALUE_MAPSIZE1
int
xfs_attr3_rmt_blocks(
        struct xfs_mount *mp,
        int                     attrlen)
...
...
                error = xfs_trans_roll(&args->trans, args->dp);
                if (error)
                        return (error);
        }
        return(0);
}
```

“Output”：

[阶段1] 功能解析

本段代码主要负责XFS文件系统中“远程扩展属性”（remote attributes，即数据过大无法存放在 Inode 内部，必须存放在独立的磁盘块中的属性）的读、写和校验操作。

...

[阶段2] 污染源识别

可控数据源：本代码处理的是底层文件系统数据块。所有的污染源均来自于外部输入的文件系统磁盘镜像（disk image）。

...

[阶段3] 数据流追踪

1. 系统在读取远程扩展属性块时，从磁盘读取数据到内存缓冲区bp中。

...

5. 污染数据 *rmt->rm_offset* 和 *rmt->rm_bytes* 通过 *be32_to_cpu*() 转换字节序后，直接参与条件分支的安全检查逻辑：*be32_to_cpu*(*rmt->rm_offset*)+*be32_to_cpu*(*rmt->rm_bytes*) > *XATTR_SIZE_MAX*。

[阶段4] 安全检查

在到达敏感的内部状态确认之前，代码在 *xfs_attr3_rmt_verify* 中执行了一系列安全约束：

1. 魔数校验：检查 *rm_magic*。

...

4. 总长度/偏移量限制（存在缺陷）：*be32_to_cpu*(*rmt->rm_offset*)+*be32_to_cpu*(*rmt->rm_bytes*) > *XATTR_SIZE_MAX*。代码试图确保偏移量加上字节数不超过内核允许的扩展属性最大长度（通常 *XATTR_SIZE_MAX* 为 64KB）。但此处在进行加法运算前，未进行整数溢出检查（integer overflow check）。

[阶段5] 漏洞结论

本代码存在高危安全漏洞。

...

[最终标签] 1

Fig. 5 Example of structured reasoning distillation fine-tuning data

图 5 结构化推理蒸馏微调数据示例

为了在脱离显式图结构输入的条件下，仍能够调用 LLM 在微调阶段习得的控制流与数据流先验知识，本文设计了漏洞思维链提示词模板。该机制通过结构化自然语言指令对模型的自回归解码过程进行约束，引导其在输出最终检测结论之前，依照预设的五阶段范式完成逻辑推演。

在提示词模板构建中，本文未引入复杂的外部调用栈信息或宏观上下文组装，而是采用直观的输入界定符，直接向模型暴露待测的核心源代码片段序列。这种相对简洁的输入边界能够减少冗余文本对注意力分配的干扰，使 LLM 的自注意力计算更加集中于目标指令逻辑本身，从而提升漏洞检测的判别效果。

模板的核心约束部署于特定的推理标签对内部，定义了推演轨迹的 5 个有序逻辑子序列。各子序列深度对应代码静态分析的物理目标：

1)控制流边界解析。约束模型首先提取目标代码中的条件分支、循环结构与嵌套层级，界定代码执行的拓扑控制域。

2)污染源界定。识别系统外部输入、未校验的参数传入或不安全的内存初始化操作，确立不可信数据的源头节点。

3)动态数据流追踪。要求模型严格依据变量依赖关系，推演污染数据在代码行之间的传播路径，并分析其跨域转移状态。该步骤在文本维度上实现了对数据流图有向边传播过程的对应刻画。

4)安全校验评估。审查数据流转路径上是否存在有效的数据清洗、边界截断或类型约束机制，评估污染状态的存续性。

5)敏感槽点触发确认。最终判定未被净化的污染数据是否能够触达具有高危权限的内存操作指令或系统调用接口，从而完成漏洞逻辑闭环。

在该推理协议约束下，LM 的解码过程需要依照预设步骤生成中间分析内容，而不能在未完成拓扑推演前直接输出最终结论。当结构化推演完成后，模板中的终端指令会进一步约束模型收敛到离散判定结果。具体而言，模型被限制为仅输出最终判定标签，即在特定判断标识后直接给出二元布尔值（是/否），具体提示词如图 6 所示。这样的终端输出形式有助于降低 LM 在结论生成阶段的语义歧义。

System：
你是一个严谨且专业的代码安全审计系统。请对下方的源代码片段进行深度的逻辑分析。你必须严格按照指定的5个阶段依次展开推演，在完成全量拓扑推演前，绝对禁止直接生成猜测性的结论 。

Input Variables：<Code >

Structured reasoning steps：
1. 控制流边界解析：请提取目标代码中的条件分支、循环结构与嵌套层级，界定代码执行的拓扑控制域 。
2. 污染源界定：请识别系统外部输入、未校验的参数传入或不安全的内存初始化操作，确立不可信数据的源头节点 。
3. 动态数据流追踪：请严格依据变量的依赖关系，推演污染数据在物理代码行间的传播路径，并定量分析其跨域转移状态 。
4. 安全校验评估：请审查数据流转路径上是否存在有效的数据清洗、边界截断或类型约束机制，评估污染状态的存续性 。
5. 敏感槽点触发确认：请最终判定未被净化的污染数据是否能够触达具有高危权限的内存操作指令或系统调用接口，从而完成漏洞逻辑闭环 。

Output Format:
最终判定：[是/否]

Fig. 6 Vulnerability chain-of-thought prompt template

图 6 漏洞思维链提示模板

4 实验设计及结果分析

在完成基于 GNN 与 LLM 的融合检测框架设计后，本文进一步通过系统实验量化评估该方法在源代码漏洞检测任务中的性能表现。本文围绕模型有效性与组件贡献度展开定量实证分析。整体评测流程相应划分为实验环境配置与结果对比分析 2 个阶段：首先通过界定多语言漏洞数据集、确立评价指标、设置对比基线模型以及规范 LLM 的推理参数，构建标准化测试基准；在此基础上，针对测试集样本执行联合检测，并利用消融实验分离图编码、结构化蒸馏与思维链约束等核心模块的性能收益。

4.1 实验设置

为验证本文方法在源代码漏洞检测任务中的有效性与稳健性，本文从数据集构建、实验配置、模型选择、评价指标以及对比基线方法等方面对实验设置进行说明。

本文选取 DiverseVul、Devign 和 CVEfixes 作为评测基准。这 3 个数据集均基于真实开源项目中的漏洞修复提交构建，涵盖多种常见漏洞类型。同时，它们在源代码规模、漏洞分布以及标注方式等方面存在一定差异，因此能够从不同角度评估模型在跨项目漏洞检测任务中的泛化能力。实验中，本文采用其公开的训练集与测试集划分方式，不对原始数据集作额外处理，不将测试集样本用于模型训练、超参数选择或蒸馏样本构造。针对本文涉及的结构化蒸馏过程，教师模型 DeepSeek-R1 仅基于训练集样本生成结构化推理轨迹，生成后的蒸馏数据仅用于训练阶段，不参与测试样本推理过程。

在模型选择方面，本文在跨模态特征映射网络中选用 2 层 GAT 作为 GNN 实现，用于对 DFG 进行结构编码。每层采用 8 个注意力头，隐藏维度设为 512，输出节点表示维度设为 1 024。随后，跨模态 Adapter 通过交叉注意力机制将图节点表示映射到 LLM 词嵌入空间，并生成固定数量的 Graph Token。本文默认 Graph Token 数量设为 16。本文采用 LLaMA3-8B 作为学生模型基座，并以 DeepSeek-R1 作为教师模型生成结构化漏洞推理轨迹。此外，为覆盖不同建模范式，本文选取了多类具有代表性的漏洞检测模型作为对比对象，包括传统基于图结构的检测模型（如 Devign、ReGVD、AMPLE）、基于预训练 LM 的方法（如 CodeBERT、UniXcoder、EPVD），以及参数规模不大的 LLM（如 LLaMA3-8B、Qwen2.5-Coder-7B）。上述模型涵盖了当前源代码漏洞检测中较具代表性的技术路线，能够用于分析不同模型范式及参数规模对检测性能的影响。

LoRA 微调中，LoRA 的秩参数 r 设为 8，缩放系数 α 设为 16，LoRA dropout 设为 0.05，bias 选项设为 none。训练过程中冻结基座模型全部原始参数，仅更新

LoRA 参数以及与任务相关的图编码和映射模块参数。优化器采用 AdamW, LoRA 部分的学习率设为 2×10^{-4}, 权重衰减系数设为 0.01, 预热比例设为 0.03, 训练轮数设为 3, 单卡 batch size 设为 1。

本文所有学生模型训练、图编码器训练和推理评估均在单卡消费级 GPU 环境下完成, 该配置采用一张 NVIDIA GeForce RTX 4090 GPU, 显存为 24 GB, CPU 为 14 核, 系统内存为 64 GB。软件环境为 Ubuntu 22.04、CUDA 12.8、PyTorch 2.9.1 和 Python 3.12。

在此基础上, 所有模型采用一致的数据划分和评价指标进行评测, 以保证对比实验的公平性。所有对比模型均采用其原始论文或官方实现中推荐的网络结构与训练配置, 本文不对模型本身进行结构性修改, 而仅通过方法层面的设计对训练与推理过程进行增强。为降低随机性, 本文实验分别采用随机种子 42、2 025 和 3 407 进行 3 次独立实验。

源代码漏洞检测任务可以形式化为二分类问题, 因此本文选取精确率(precision, P)、召回率(recall, R)、准确率(accuracy, A)以及 $F1$ 作为主要评价指标, 对模型性能进行衡量。设测试集中预测结果中, 真正例(true positive)数量为 TP, 真负例(true negative)数量为 TN, 假正例(false positive)数量为 FP, 假负例(false negative)数量为 FN。精确率用于衡量模型预测为漏洞的样本中真实漏洞所占的比例, 其定义为:

$$P=\frac{TP}{TP+FP}。\tag{19}$$

召回率用于衡量模型对真实漏洞样本的覆盖能力, 即被正确识别出的漏洞样本占所有真实漏洞样本的比例, 其定义为:

$$R=\frac{TP}{TP+FN}。\tag{20}$$

准确率用于衡量所有样本中预测正确的比例, 反映模型的整体判别正确率, 其定义为:

$$A=\frac{TP+TN}{TP+TN+FP+FN}。\tag{21}$$

$F1$ 是精确率和召回率的调和平均, 用于在两者之间取得平衡, 能够更加全面地反映模型的整体检测性能, 因此被设定为实验分析中的核心指标, 其定义为:

$$F1=\frac{2\times P\times R}{P+R}。\tag{22}$$

4.2 实验设计

为了系统分析 GAR-VD 在漏洞检测任务中的性能表现及其关键模块的实际作用, 本文设置如下 3 个研究问题。

问题 1: GAR-VD 的整体漏洞检测性能是否优于现有代表性方法?

该问题旨在验证 GAR-VD 在整体框架层面的有效性。为回答该问题, 本文设计多模型主实验, 在统一数据集划分、统一训练配置和统一评价指标条件下, 将 GAR-VD 与传统图结构模型、预训练代码模型以及通用 LLM 进行比较, 并进一步考察其与教师模型之间的性能差距。分别采用随机种子 42、2 025 和 3 407 进行 3 次独立运行, 报告精确率、召回率、$F1$ 和准确率的均值与样本标准差。

问题 2: GAR-VD 的性能提升主要来源于哪些核心模块?

该问题旨在分析完整框架中各组成模块的实际贡献。为回答该问题, 本文设计消融实验, 以完整 GAR-VD 为基准, 分别移除图-文联合输入、结构化推理蒸馏和漏洞思维链提示约束 3 个核心部分, 在保持其他设置不变的情况下比较平均 $F1$ 的变化幅度。通过这种方式, 可以定量分析输入层结构先验、训练阶段知识迁移机制以及推理阶段结构化约束对最终检测结果的影响。

问题 3: GAR-VD 相较于不同技术范式模型, 其优势究竟体现在哪里?

该问题不是单纯比较数值高低, 而是要解释性能差异背后的原因。前文已经指出, 当前漏洞检测方法主要包括纯图结构建模、纯代码文本建模以及图文融合的大模型检测 3 类路线; 同时也指出, 纯文本序列方法难以充分建模复杂代码中的数据流依赖关系, 而直接将高维图结构引入 LLM 又面临跨模态语义鸿沟、训练成本高和推理逻辑不稳定等问题。基于这一问题, 本文将在主实验和消融实验结果基础上, 分析 GAR-VD 为什么能够优于传统图模型、代码预训练模型和未经结构化对齐的通用 LLM, 从而解释其“结构输入—蒸馏迁移—推理约束”这一整体方案的合理性。

4.3 实验结果与分析

1)问题 1: GAR-VD 的整体漏洞检测性能是否优于现有代表性方法?

主实验对不同模型在 CVEfixes、DiverseVul 和 Devign 数据集上的漏洞检测性能进行了比较, 实验统计结果如表 1 所示。

从表 1 可以看出, GAR-VD 在 CVEfixes、DiverseVul 和 Devign 数据集上的 $F1$ 分别达到 63.24%±0.50%、65.09%±0.40% 和 67.22%±0.80%, 准确率分别达到 76.88%±0.38%、78.22%±0.42% 和 79.68%±0.48%。3 个

Table 1 Comparison of Detection Performance of Different Models Across Multiple Vulnerability Datasets

表 1 各模型在多漏洞数据集上的检测性能对比 %

模型/方法	CVEfixes				DiverseVul				Devign			
	P	*R*	*F*1	*A*	*P*	*R*	*F*1	*A*	*P*	*R*	*F*1	*A*
Devign	46.45±0.85	36.21±0.71	40.69±0.45	68.33±0.44	39.17±0.85	49.42±0.98	43.70±0.91	61.77±0.66	44.01±1.06	45.97±0.54	44.96±0.40	66.22±0.78
ReGVD	39.25±0.75	51.04±0.75	44.37±0.21	61.59±0.88	45.32±0.51	46.38±0.88	45.84±0.66	67.10±0.30	47.69±0.72	49.47±0.47	48.56±0.57	68.55±0.46
AMPLE	55.34±0.45	56.10±0.40	55.72±0.39	73.24±0.26	54.35±0.72	57.54±0.76	55.90±0.74	72.74±0.46	57.83±0.46	60.91±0.37	59.33±0.40	74.94±0.27
CodeBERT	48.97±0.76	51.20±0.40	50.06±0.54	69.35±0.49	50.34±0.66	52.97±1.31	51.62±0.88	70.19±0.42	54.40±0.41	52.43±0.74	53.40±0.39	72.54±0.22
UniXcoder	54.54±0.45	53.91±0.83	54.23±0.60	72.69±0.27	46.71±0.62	71.38±1.03	56.46±0.51	66.95±0.62	55.49±0.52	63.17±1.52	59.08±0.93	73.75±0.40
EPVD	56.48±0.39	57.92±0.28	57.19±0.26	73.98±0.22	57.11±0.32	60.80±0.55	58.90±0.40	74.52±0.21	59.51±0.33	62.75±0.45	61.09±0.29	76.01±0.19
LLaMA3-8B	43.21±0.61	58.18±0.79	49.59±0.55	64.50±0.57	51.65±1.65	53.41±1.31	52.52±1.49	71.00±1.02	45.16±0.55	70.52±0.62	55.06±0.51	65.46±0.57
Qwen2.5-Coder-7B	54.62±0.43	59.68±0.91	57.04±0.59	73.02±0.29	59.77±0.73	66.30±0.43	62.87±0.51	76.48±0.42	61.67±0.50	66.90±0.53	64.18±0.43	77.59±0.29
DeepSeek-R1	61.88±0.65	71.44±0.67	66.32±0.66	78.23±0.44	63.64±0.62	73.19±0.76	68.08±0.68	79.40±0.43	66.20±0.98	72.84±0.90	69.36±0.94	80.69±0.62
GAR-VD（本文）	60.47±0.65	66.27±0.51	63.24±0.50	76.88±0.38	62.75±0.87	67.61±0.43	65.09±0.40	78.22±0.42	65.13±0.74	69.45±0.90	67.22±0.80	79.68±0.48

注：“±”前后数值表示均值和标准差。

数据集上的 *F*1 标准差处于 0.40～0.80 个百分点，说明模型在不同随机初始化条件下的性能波动总体可控，但不同数据集上的波动程度仍存在差异。

在非教师模型中，Qwen2.5-Coder-7B 取得了最具竞争力的整体性能。与该模型相比，GAR-VD 在 CVEfixes、DiverseVul 和 Devign 上的平均 *F*1 分别提高 6.20、2.22 和 3.04 个百分点，准确率分别提高 3.86、1.74 和 2.09 个百分点。GAR-VD 的跨数据集平均 *F*1 为 65.18%，高于 Qwen2.5-Coder-7B 的 61.36%，平均提升 3.82 个百分点；平均准确率由 75.70% 提高至 78.26%，提升 2.56 个百分点。这表明，图结构信息、结构化推理蒸馏和 Vuln-CoT 能够在通用代码 LLM 的基础上进一步提升漏洞判别能力。

与代表性的专用漏洞检测模型 EPVD 相比，GAR-VD 在 3 个数据集上的 *F*1 分别提高 6.05、6.19 和 6.13 个百分点，跨数据集平均 *F*1 由 59.06% 提高至 65.18%，平均提升 6.12 个百分点，平均准确率提高 3.42 个百分点。与 LLaMA3-8B 相比，GAR-VD 的平均 *F*1 由 52.39% 提高至 65.18%，提升 12.79 个百分点，说明仅依赖代码文本序列不足以充分刻画复杂漏洞中的数据传播关系，而结构先验和漏洞推理知识能够为模型提供有效补充。

GAR-VD 与教师模型 DeepSeek-R1 的跨数据集平均 *F*1 分别为 65.18% 和 67.92%，相差 2.74 个百分点；平均准确率分别为 78.26% 和 79.44%，相差 1.18 个百分点。考虑到 DeepSeek-R1 仅用于离线生成结构化推理轨迹，而实际部署阶段仅需运行 8B 级学生模型，GAR-VD 在较低部署成本下取得了接近教师模型的平均检测性能。

因此，对于问题 1，可以得出结论：GAR-VD 在 3 个公开漏洞检测数据集上均取得了高于最强非教师基线的平均 *F*1 和准确率。总体而言，GAR-VD 能够在保持较低部署成本的条件下提高漏洞检测性能，并取得接近教师模型的平均检测结果。

2）问题 2：GAR-VD 的性能提升主要来源于哪些核心模块？

为分析 GAR-VD 各组成模块的实际贡献，本文围绕图-文联合输入、结构化推理蒸馏和漏洞思维链提示约束开展消融实验，并进一步设置仅标签微调和普通思维链提示 2 个对照变体，实验结果如表 2 所示。完整 GAR-VD 的平均 *F*1 为 65.18%。

Table 2 *F*1 Results of Ablation Experiments on the Core Components of the Proposed Method

表 2 本文方法核心组件消融实验 *F*1 结果

实验设置	平均 *F*1/%	性能下降百分点
完整 GAR-VD	65.18	
变体①：移除图-文联合输入	58.65	6.53
变体②：移除结构化推理蒸馏	53.43	11.75
变体③：仅标签微调	59.71	5.47
变体④：移除漏洞思维链提示约束	60.58	4.60
变体⑤：普通思维链约束	62.21	2.97

当移除图-文联合输入后，平均 *F*1 降至 58.65%，下降 6.53 个百分点。这表明，仅使用代码文本序列难以充分刻画变量依赖和数据传播路径，DFG 及其图特征注入能够为模型提供有效的程序结构先验。

当移除结构化推理蒸馏且不进行漏洞检测任务微调时,平均 $F1$ 降至 53.43%,降幅达到 11.75 个百分点,为所有变体中下降最大。仅使用漏洞标签进行 LoRA 微调时,平均 $F1$ 为 59.71%,较未进行任务微调提高 6.28 个百分点,但仍低于完整 GAR-VD 5.47 个百分点。该结果说明,标签监督能够使模型初步适配漏洞判别任务,而教师模型生成的结构化推理轨迹能够进一步传递污染源识别、数据流追踪和危险操作确认等中间推理知识,是性能提升的重要来源。

在推理阶段,移除漏洞思维链提示约束后,平均 $F1$ 降至 60.58%;使用普通思维链约束替代漏洞思维链时,平均 $F1$ 为 62.21%。普通思维链约束能够带来一定推理增益,但仍低于完整 GAR-VD 2.97 个百分点,说明面向漏洞检测过程设计的结构化提示比通用思维链更有利于约束模型的分析步骤并提高推理稳定性。

因此,对于问题 2,可以得出结论:GAR-VD 的性能优势来源于多个关键模块的协同作用,其中结构化推理蒸馏贡献最大,图-文联合输入次之,漏洞思维链提示约束进一步提升了推理稳定性。

3)问题 3:GAR-VD 相较于不同技术范式模型,其优势究竟体现在哪里?

结合主实验与消融实验结果,可以进一步回答问题 3。相较于传统图结构模型,GAR-VD 并没有停留在仅用程序图进行分类的层面,而是借助 LLM 补充了更强的语义理解与推理能力,因此在整体性能上明显更优。相较于预训练代码模型,GAR-VD 不仅利用了代码文本本身,还显式引入了 DFG 等结构先验,并通过跨模态映射缓解图结构与 LM 嵌入空间之间的语义鸿沟,因此能够更有效地建模漏洞形成中的变量传播路径和依赖关系。相较于未经对齐的通用 LLM, GAR-VD 又通过教师推理蒸馏和思维链约束增强了任务定向性与逻辑一致性,从而避免模型只做表层语义匹配。

此外, GAR-VD 的完整训练可在单张 RTX 4090 24 GB GPU 上完成,峰值显存占用为 22.1GB。由于训练过程中冻结 LLM 主体参数,仅更新 LoRA、GNN 和 Adapter 参数,显著降低了全量微调所需的显存与参数更新成本。在推理阶段,教师模型 DeepSeek-R1 仅用于离线生成结构化推理轨迹,不参与学生模型的实际部署。GAR-VD 仅需加载 8B 学生模型,平均推理显存占用为 15.7 GB,可在单张消费级 GPU 上完成部署,能够满足离线代码审计和批量漏洞检测场景的需求。

因此, GAR-VD 的优势不仅体现在检测性能上,还体现在结构信息利用、任务推理能力和部署成本之间的平衡。其通过“图-文联合输入—结构化推理蒸馏—漏洞思维链约束”形成完整检测闭环,在获得接近教师模型性能的同时,保持了较低的训练成本和较好的工程部署可行性。

5 讨论与展望

尽管本文围绕漏洞结构感知的 LLM 检测框架开展了较为系统的研究,并取得了一定实验效果,但面向真实软件工程环境,现有工作仍有进一步拓展空间。尤其是在大规模开源项目和持续迭代的工业代码场景中,漏洞形态、代码风格及依赖关系愈发复杂,这对模型的跨场景适应能力和长效应用能力提出了更高要求。

未来研究可从以下 2 个维度重点推进。其一,深化复杂工程上下文的建模能力。真实漏洞往往并不局限于单一函数或局部代码片段,而与跨文件调用、外部依赖及版本演化等因素密切相关,因此仅依赖局部结构信息仍难以完整还原漏洞的触发条件与传播链条。未来可在现有结构表征基础上,进一步引入跨函数、跨模块乃至跨仓库的上下文关联机制,以提升模型对复杂漏洞成因和影响范围的分析能力。其二,提升漏洞检测系统的持续演化能力。当前系统已构建了从代码输入、结构解析到模型推理和结果展示的完整流程,但整体上仍偏向离线研究验证。考虑到漏洞知识与利用方式的持续演变,后续可探索将增量知识更新、人工反馈校正与在线样本积累机制融入现有框架,从而增强系统在真实环境中的长效可用性。

6 结 论

针对 LLM 在源代码漏洞检测任务中存在的结构感知能力不足、易受表层语义干扰以及推理稳定性有限等问题,本文提出了一种图对齐推理漏洞检测框架 GAR-VD。该方法通过图-文联合输入增强模型对程序结构信息的表达能力,并结合结构化推理蒸馏与 LoRA 微调,将教师模型中的漏洞分析能力迁移到轻量化学生模型中;同时,在检测阶段引入漏洞思维链提示约束,以提升推理过程的稳定性与结果可解释性。

实验结果表明, GAR-VD 在多个公开漏洞检测数

据集上取得了优于现有代表性方法的性能，且消融实验进一步验证了图-文联合输入、结构化推理蒸馏和漏洞思维链提示约束3个模块的有效性。其中，结构化推理蒸馏对性能提升贡献最为显著，说明结构化漏洞分析知识能够有效增强模型对复杂漏洞模式的判别能力。

总体来看，本文从输入表示、模型训练和推理约束3个层面构建了面向源代码漏洞检测的结构化分析闭环，为LLM在软件安全审计场景中的应用提供了一种具有较强实用价值的技术路径。

作者贡献声明：曹明生提出了基于LLM的源代码漏洞检测核心方法框架；丁桥隆设计了整体模型架构，开展了模型训练实验与超参数调优，撰写论文；张文清设计了LLM的微调策略；李延斌参与设计了对比实验与评估指标方案；林迪进行了基线模型的对比测试与实验结果分析；许辉开展了整体研究的指导思想与技术路线，并对论文审阅修改。

参 考 文 献

[1] Li Jiayuan, Cui Lei, Yu Wenyan, et al. VulnTeam: A team collaboration framework for LLM-based vulnerability detection[C]// Proc of the 2025 Int Joint Conf on Neural Networks. Piscataway, NJ: IEEE, 2025: 1−8

[2] Du Xiaohu, Wen Ming, Zhu Jiahao, et al. Generalization-enhanced code vulnerability detection via multi-task instruction fine-tuning [C]//Proc of the Findings of the Association for Computational Linguistics: ACL 2024. Stroudsburg, PA, ACL, 2024: 10507−10521

[3] Steenhoek B, Rahman M M, Roy M K, et al. A comprehensive study of the capabilities of large language models for vulnerability detection[J]. arXiv preprint, arXiv: 2403.17218, 2024

[4] Zhou Xin, Zhang Ting, Lo D. Large language model for vulnerability detection: Emerging results and future directions[C]//Proc of the 2024 ACM/IEEE 46th Int Conf on Software Engineering: New Ideas and Emerging Results. Piscataway, NJ: IEEE, 2024: 47−51

[5] Daneshvar S S, Nong Y, Yang X, et al. VulScribeR: Exploring RAG-based vulnerability augmentation with LLMs[J]. ACM Transactions on Software Engineering and Methodology, 2025, 35(5): 1−26

[6] Zhang Ziyin, Yu Hang, Lee Sage, et al. GALLa: Graph aligned large language models for improved source code understanding[C]//Proc of the 63rd Annual Meeting of the Association for Computational Linguistics, Stroudsbrug, PA: ACL, 2025: 13784−13802

[7] Liu Ruitong, Wang Yanbin, Xu Haitao, et al. Source code vulnerability detection: Combining code language models and code property graphs[J]. arXiv preprint, arXiv: 2404.14719, 2024

[8] Jiang Zhonghao, Sun Weifeng, Gu Xiaoyan, et al. DFEPT: Data flow embedding for enhancing pre-trained model-based vulnerability detection[C]//Proc of the 15th Asia-Pacific Symp on Internetware. New York: ACM, 2024: 95−104

[9] Fitero-Dominguez D, Garcia-Lopez E, Garcia-Cabot A, et al. Enhanced automated code vulnerability repair using large language models[J]. Engineering Applications of Artificial Intelligence, 2024, 138: 109291

[10] Yang Aidan Z H, Tian Haoye, Ye He, et al. Security vulnerability detection with multitask self-instructed fine-tuning of large language models[J]. arXiv preprint, arXiv: 2406.05892, 2024

[11] Feng Ruijun, Pearce H, Liguori P, et al. CGP-Tuning: Structure-aware soft prompt tuning for code vulnerability detection[J]. IEEE Transactions on Software Engineering, 2025, 51(9): 2533−2548

[12] Xu Hanxiang, Wang Shenao, Li Ningke, et al. Large language models for cyber security: A systematic literature review[J]. arXiv preprint, arXiv: 2405.04760, 2024

[13] Nong Yu, Aldeen M, Cheng Long, et al. Chain-of-thought prompting of large language models for discovering and fixing software vulnerabilities[J]. arXiv preprint, arXiv: 2402.17230, 2024

[14] Yang Yanjing, Zhou Xin, Mao Runfeng, et al. DLAP: A deep learning augmented large language model prompting framework for software vulnerability detection[J]. Journal of Systems and Software, 2025, 219: 112234

[15] Lu Guilong, Ju Xiaolin, Chen Xiang, et al. GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning[J]. Journal of Systems and Software, 2024, 212: 112031

[16] Mathews N S, Brus Y, Aafer Y, et al. LLBezpeky: Leveraging large language models for vulnerability detection[J]. arXiv preprint, arXiv: 2401.01269, 2024

[17] Mao Zhenyu, Li Jialong , Jin Dongming, et al. Multi-role consensus through LLMs discussions for vulnerability detection[C]//Proc of the 2024 IEEE 24th Int Conf on Software Quality, Reliability and Security Companion. Piscataway, NJ: IEEE, 2024: 1318−1319

[18] Zhang Chenyuan, Liu Hao, Zeng Jiutian, et al. Prompt-enhanced software vulnerability detection using ChatGPT[C]//Proc of the 2024 IEEE/ACM 46th Int Conf on Software Engineering: Companion Proceedings. New York: ACM, 2024: 276−277

[19] Ding Hao, Liu Yizhou, Piao Xuefeng, et al. SmartGuard: An LLM-enhanced framework for smart contract vulnerability detection[J]. Expert Systems with Applications, 2025, 269: 126479

[20] Khare A, Dutta S, Li Z, et al. Understanding the effectiveness of large language models in detecting security vulnerabilities[C]//Proc of the 2025 IEEE Conf on Software Testing, Verification and Validation. Piscataway, NJ: IEEE, 2025: 103−114

[21] Kouliaridis V, Karopoulos G, Kambourakis G. Assessing the effectiveness of LLMs in Android application vulnerability analysis[C]// Proc of the Int Conf on Attacks and Defenses for Internet-of-Things. Cham: Springer, 2024: 139−154

[22] Sheng Ze, Wu Fenghua, Zuo Xiangwu, et al. LProtector: An LLM-driven vulnerability detection system[J]. arXiv preprint, arXiv: 2411.06493, 2024

[23] Simoni M, Saracino A, Vinod P, et al. MoRSE: Bridging the gap in cybersecurity expertise with retrieval augmented generation[C]//Proc of the 40th ACM/SIGAPP Symp on Applied Computing. New York: ACM, 2025: 1213−1222

[24] Fayyazi R, Trueba S H, Zuzak M, et al. ProveRAG: Provenance-driven vulnerability analysis with automated retrieval-augmented LLM[J]. arXiv preprint, arXiv: 2410.17406, 2024

[25] Du Xueying, Zheng Geng, Wang Kaixin, et al. Vul-RAG: Enhancing LLM-based vulnerability detection via knowledge-level RAG[J]. arXiv preprint, arXiv: 2406.11147 , 2024

[26] Yoon M J, Yoo S M, Park J H, et al. Toward a synergistic vulnerability analysis enhanced with a multi-level RAG model[C]// Proc of the 2024 Int Conf on AI x Data and Knowledge Engineering. Piscataway, NJ: IEEE, 2024: 129−130

[27] Qin Zhen, Zhuang Tianming, Zhu Guosong, et al. Survey of security attack and defense strategies for artificial intelligence model[J]. Journal of Computer Research and Development, 2024, 61(10): 2627−2648 (in Chinese)
(秦臻, 庄添铭, 朱国淞, 等. 面向人工智能模型的安全攻击和防御策略综述[J]. 计算机研究与发展, 2024, 61(10): 2627−2648)

[28] National Institute of Standards and Technology. National vulnerability database[EB/OL]. 2005[2025-12-16]. https://nvd.nist.gov/

[29] Scarselli F, Gori M, Tsoi A C, et al. The graph neural network model[J]. IEEE Transactions on Neural Networks, 2009, 20(1): 61−80

[30] Veličković P, Cucurull G, Casanova A, et al. Graph attention networks[J]. arXiv preprint, arXiv : 1710.10903, 2017

[31] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Proc of the Advances in Neural Information Processing Systems. Red Hook: Curran Associates, 2017, 30: 5998−6008

[32] Brown T B, Mann B, Ryder N, et al. Language models are few-shot learners[C]//Proc of the Advances in Neural Information Processing Systems. Red Hook: Curran Associates, 2020, 33: 1877−1901

[33] Jiang Yi, Yang Yong, Yin Jiali, et al. Survey on security and privacy risks in large language models[J]. Journal of Computer Research and Development, 2025, 62(8): 1979−2018 (in Chinese)
(姜毅, 杨勇, 印佳丽, 等. 大语言模型安全与隐私风险综述[J]. 计算机研究与发展, 2025, 62(8): 1979−2018)

[34] Ding Yi, Lin Nengjian, Jiang Yuntao, et al. Adversarial transfer attacks on weak block of SAM downstream models[J]. Journal of Computer Research and Development, 2025, 62(10): 2455−2467 (in Chinese)
(丁熠, 林能健, 蒋昀陶, 等. 针对 SAM 下游模型脆弱模块的对抗迁移攻击[J]. 计算机研究与发展, 2025, 62(10): 2455−2467)

[35] Ouyang Long, Wu J, Jiang Xu, et al. Training language models to follow instructions with human feedback[C]//Proc of the Advances in Neural Information Processing Systems. Red Hook: Curran Associates, 2022, 35: 27730−27744



Cao Mingsheng, born in 1986. PhD, associate professor, master supervisor. His main research interests include artificial intelligence, information security, and privacy protection.
曹明生, 1986 年生。博士, 副研究员, 硕士生导师。主要研究方向为人工智能、信息安全、隐私保护。



Ding Qiaolong, born in 2001. Master candidate. His main research interests include deep learning, large language models, and vulnerability detection.
丁桥隆, 2001 年生。硕士研究生。主要研究方向为深度学习、大语言模型、漏洞检测。



Zhang Wenqing, born in 1990. PhD. His main research interests include artificial intelligence and information security.
张文清, 1990 年生。博士。主要研究方向为人工智能、信息安全。



Li Yanbin, born in 1992. PhD, associate professor, master supervisor. His main research interests include information security and privacy protection.
李延斌, 1992 年生。博士, 副研究员, 硕士生导师。主要研究方向为信息安全、隐私保护。



Lin Di, born in 1985. PhD, professor, and PhD supervisor. His main research interests include big data analytics, and information security.
林　迪, 1985 年生。博士, 研究员, 博士生导师。主要研究方向为大数据分析、信息安全。



Xu Hui, born in 1986. PhD, associate professor, master supervisor. His main research interests include artificial intelligence, LLM, and reinforcement learning.
许　辉, 1986 年生。博士, 副研究员, 硕士生导师。主要研究方向为人工智能、大语言模型、强化学习。