基于大语言模型的源代码漏洞检测方法综述

蒋 远 杨子含 苏小红 黄 山 王甜甜
（哈尔滨工业大学计算学部 哈尔滨 150001）

摘 要 随着软件系统规模与复杂度的持续增长，面向源代码的漏洞检测在保障软件安全性与可靠性方面的重要性愈发突出。现有综述多聚焦于传统的静态与动态分析工具，或基于深度学习（Deep Learning，DL）的漏洞检测方法，而针对大语言模型（Large Language Models，LLMs）在该领域的系统性研究仍相对匮乏。近年来，Transformer架构及其衍生的LLMs在自然语言处理和代码理解任务中展现出强大的语义建模与跨域泛化能力，逐步被引入漏洞检测场景，并在多项研究中表现出优于传统DL模型的性能。此外，通用型LLM（如ChatGPT，DeepSeek）借助提示工程、指令微调与上下文学习等机制，可在无需大规模任务特定训练的前提下，直接执行漏洞检测任务。尽管其检测精度仍有提升空间，但其高度的灵活性与迁移能力，为漏洞检测研究开辟了新的范式。本文旨在对基于LLM的源代码漏洞检测方法进行系统性综述，全面梳理该领域的研究进展与代表性工作，深入剖析不同方法在模型设计、检测粒度、性能表现与适用场景等方面的差异与特点。同时，总结当前面临的关键挑战，并探讨潜在解决方案和未来发展方向，为后续基于LLM的智能化源代码漏洞检测研究提供系统框架与参考依据。

关键词 漏洞检测；大语言模型；Transformer；指令微调；提示工程
中图法分类号 TP309 **DOI号** 10.11897/SP.J.1016.2026.01385

A Survey of Source Code Vulnerability Detection Methods Based on Large Language Models

JIANG Yuan YANG Zi-Han SU Xiao-Hong HUANG Shan WANG Tian-Tian
（School of Computing, Harbin Institute of Technology, Harbin 150001）

Abstract As software systems continue to grow in scale and complexity, source code vulnerability detection has become increasingly critical for ensuring software security and reliability. While traditional approaches, including static analysis tools, dynamic testing, and deep learning (DL)-based methods, have been extensively explored, the systematic understanding of Large Language Model (LLM)-based solutions for this task remains insufficiently addressed. Recent developments in the LLM ecosystem, particularly those leveraging Transformer architectures, have led to a new generation of methods that are reshaping intelligent software engineering. These models, pre-trained on large-scale code and natural language corpora, exhibit strong semantic modeling and cross-domain generalization capabilities. As a result, they can better capture code syntax, control/data flow dependencies, and vulnerability semantics than conventional DL-based methods. Empirical evidence shows that LLM-based approaches frequently outperform traditional models on benchmark datasets, especially when detecting

收稿日期：2025-07-29；在线发布日期：2026-01-13。本课题得到国家自然科学基金青年科学基金项目（No. 62302125）、国家自然科学基金面上项目（No. 62272132）资助。**蒋 远**（通信作者），博士，副研究员，中国计算机学会（CCF）专业会员，主要研究领域为代码漏洞检测、软件安全和大模型安全等。E-mail：jiangyuan@hit.edu.cn。**杨子含**，硕士研究生，主要研究领域为代码漏洞检测、越狱攻击和提示工程等。**苏小红**，博士，教授，中国计算机学会（CCF）高级会员，主要研究领域为智能软件工程、软件漏洞检测、智能化软件开发与测试。**黄 山**，本科生，中国计算机学会（CCF）学生会员，主要研究领域为漏洞检测、对抗攻击、模型安全。**王甜甜**，博士，副教授，中国计算机学会（CCF）会员，主要研究领域为基于模型的系统工程、程序修复。

vulnerabilities that span across functions, modules, or even files. Furthermore, general-purpose LLMs, such as ChatGPT, DeepSeek, and Gemini, can be seamlessly adapted to vulnerability detection without requiring task-specific retraining. Leveraging mechanisms such as prompt engineering, instruction tuning, and in-context learning, these models can interpret customized vulnerability descriptions and identify relevant security flaws in previously unseen code. Such capabilities significantly lower the deployment threshold, enabling plug-and-play intelligent analysis pipelines suitable for real-world adoption. Despite current limitations in accuracy, particularly in handling obfuscated or adversarial code, their flexibility, transferability, and ability to operate under zero- or few-shot settings have opened up promising new research avenues. This paper presents the first systematic and structured survey of LLM-based source code vulnerability detection methods. We begin by introducing the motivations and background behind the transition from conventional methods to LLM-driven techniques. Then, we establish a comprehensive taxonomy, which classifies existing techniques along model capacity and core methodology: small-scale Transformer-based detection methods and large-scale general-purpose LLMs. The former are further analyzed in terms of pre-training strategies, tokenization schemes, detection granularity, objective functions, and input length constraints. For large-scale LLMs, we summarize four representative categories: (1) fine-tuning-based detection methods, (2) prompt-optimized detection methods, (3) hybrid approaches combining static analysis with LLMs, and (4) multi-agent frameworks driven by LLM coordination. In addition to reviewing representative methods, we conduct a comparative analysis of their performance on standard benchmarks, including comparisons between LLMs and traditional static analysis tools, LLMs and DL-based models, as well as among different LLMs themselves. These results collectively highlight the performance advantages and practical potential of LLM-based approaches in vulnerability detection. We also identify key research challenges, including prompt sensitivity, robustness against adversarial transformations, limitations in inter-procedural and cross-project vulnerability coverage, and risks arising from data contamination due to overlapping training and evaluation corpora. We further discuss auxiliary techniques, including retrieval-augmented generation, LLM-GNN joint learning, context expansion, and multi-agent collaborative reasoning, which may help alleviate these challenges. This survey offers a timely and comprehensive overview of the current landscape of LLM-based vulnerability detection. It provides a unified conceptual framework, summarizes technical advances and open challenges, and outlines promising directions for future research, aiming to support the development of trustworthy, adaptive, and generalizable vulnerability detection systems powered by large language models.

Keywords vulnerability detection; large language model; Transformer; instruction tuning; prompt engineering

1 引 言

软件漏洞(Software Vulnerabilities,SVs)是指在软件设计、实现或部署过程中存在的安全缺陷,这些漏洞可能被攻击者利用,导致敏感信息泄露、系统功能失效,并造成严重的经济损失。随着软件系统规模与复杂度的持续增长,漏洞数量呈爆发式上升(如图1所示)。源代码作为软件实现的核心载体,如何准确高效地识别其中的潜在漏洞,已成为保障软件系统安全性与稳定性的关键。

传统的漏洞检测方法主要包括静态分析和动态测试等技术,其中静态分析通常依赖人工制定的规则,而动态测试则通过运行程序并观察其行为来识别安全缺陷。然而,这些方法在应对复杂或新型漏洞时常常表现出较大的局限性。为此,研究者开始

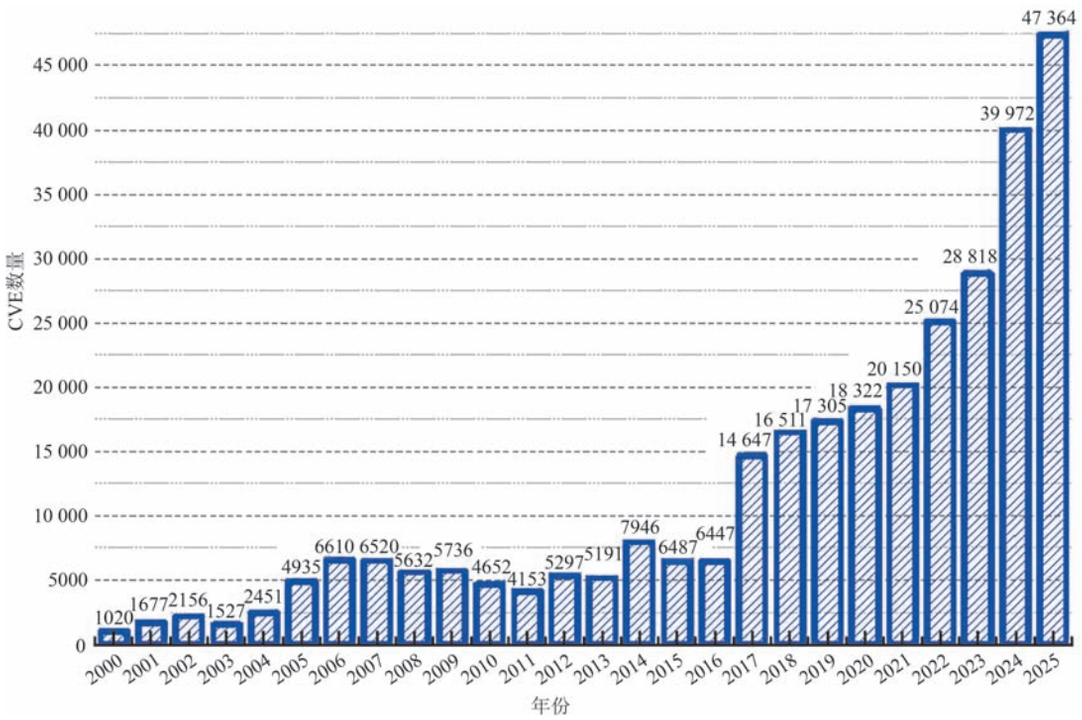

图1　公开漏洞数据库NVD中每年披露的漏洞数量

引入深度学习方法，通过自动学习代码的语义特征与漏洞模式，显著提升了检测的准确性。然而，早期的深度学习模型（如循环神经网络RNN）受限于参数规模与模型结构，难以捕捉程序中的复杂语义和长距离依赖关系，因而无法实现对代码语义的深层建模与逻辑关系的有效推理。

随着Transformer架构在自然语言处理领域的突破性成功，基于该架构的模型（如BERT、CodeBERT、CodeT5等）逐渐应用于漏洞检测，并在多项研究中表现出显著优于传统深度学习方法的性能[1-2]。另外，近年来兴起的超大规模语言模型（如GPT-4/GPT-5、DeepSeek-V3、Claude Sonnet 4.5等），凭借海量预训练语料与百亿级以上参数规模，不仅在自然语言理解与生成方面表现卓越，也为代码分析与漏洞检测带来了全新的技术范式。这些模型可通过指令微调（Instruction Tuning）或上下文学习（In-Context Learning）等机制，在无需针对特定任务进行全量参数重训练的情况下，快速适配多种检测场景，实现“即插即用”的漏洞检测能力。相比早期的深度学习方法，LLMs依托自注意力机制、庞大参数规模与先进的学习范式，在源代码漏洞检测领域实现了关键突破，克服了传统方法难以捕捉长距离依赖和缺乏语义推理能力的缺陷，推动了漏洞检测从“浅层语义理解”迈向“深层语义推理”，为后续方法的发展奠定了新的技术基础。

尽管如此，LLMs在漏洞检测中的应用仍面临多重挑战。对于轻量级Transformer模型，其主要局限包括上下文窗口受限、代码结构信息编码不足以及预训练任务与下游任务目标不匹配等问题；而大规模语言模型则存在知识检索不完备、提示工程鲁棒性不足与计算资源消耗高昂等瓶颈。此外，两类方法均受到一些共性难题制约，如对高质量标注数据的强依赖、模型决策过程的可解释性不足，以及由幻觉引发的可信性等问题。

因此，如何在源代码漏洞检测任务中充分发挥大语言模型的语义理解优势，同时克服其固有的局限性，已成为学术界与工业界亟待解决的关键问题。深入梳理现有研究进展是改进模型设计与指导实际应用的重要前提。然而，现有的综述研究[3-4]大多聚焦于基于传统深度学习模型的检测方法，对基于大语言模型的最新研究尚缺乏系统化的归纳与分析，这使得研究者在理解、比较和拓展LLMs相关检测方法时缺乏全面参考。因此，本文将聚焦于基于LLMs的源代码漏洞检测方法，全面分析不同模型和方法的核心机制与性能差异，并为未来研究提供有益的参考与启示。本文的主要贡献如下：

（1）系统梳理并首次对基于大语言模型的源代码漏洞检测方法进行了分析和归纳，涵盖从轻量级Transformer模型到大规模语言模型的两大技术发

展路径，弥补了现有综述中对该类方法系统性分析的不足。

（2）对现有方法从多个关键维度展开系统性分类与深入比较，例如对轻量级Transformer方法从模型架构、输入粒度与预训练目标等方面进行分析，全面揭示其在漏洞检测、语义建模与可扩展性等方面的优势与局限性。

（3）综合分析并提出当前基于大语言模型的源代码漏洞检测研究所面临的关键挑战，涵盖轻量级Transformer模型的技术难题、大规模语言模型的应用瓶颈及共性问题，并给出了相应的未来研究方向与潜在解决路径。

2 研究框架

随着软件规模与复杂度的持续增长，自动化漏洞检测已成为保障软件安全性的重要手段。早期研究通过神经网络对源代码进行建模，显著提升了漏洞检测的准确性与效率[3,5-7]，相关综述工作多聚焦于此类传统深度学习框架[4]；近年来，Transformer及其衍生模型被引入漏洞检测，并在多项实验中证明其性能超越了传统深度学习方法[1]。对最新基于大语言模型的源代码漏洞检测方法进行系统性的综述与分析，有助于厘清研究脉络、归纳已有成果与挑战，并为后续研究与实际应用提供有力支撑。为此，本节首先形式化定义漏洞检测问题，并在大规模文献调研的基础上构建综述的整体研究框架。

2.1 源代码漏洞检测问题的形式化定义

给定待分析的程序源码片段$P=\{p_1,p_2,\cdots,p_m\}$（如程序的Token序列，抽象语法树的节点序列或程序图的顶点序列），源代码漏洞检测模型f_θ旨在判定该片段中是否存在漏洞以及存在何种类型的漏洞。若漏洞类型集合定义为$V=\{v_1,v_2,\cdots,v_k\}$，则漏洞检测过程可形式化表示为

$$f_\theta(P)\in V\cup\{\text{无漏洞}\} \tag{1}$$

即模型以程序源码为输入，输出结果表示其是否存在漏洞以及存在何种类型的漏洞。对于更细粒度的漏洞检测任务，模型不仅需识别漏洞类型，还应预测漏洞在程序中的具体位置，如漏洞语句、代码行或关联的程序子图等。在实际研究中，为降低任务复杂度，大部分工作将漏洞检测问题简化为二分类任务，即判断输入代码中是否存在漏洞。在这种情况下，源代码漏洞检测可表示为

$$f_\theta(P)=\begin{cases}1, & \text{若存在漏洞}\\ 0, & \text{若不存在漏洞}\end{cases} \tag{2}$$

2.2 研究方法与文献检索策略

本文旨在对基于大语言模型的源代码漏洞检测研究进行系统梳理总结。为全面了解该领域的研究现状，我们通过以下步骤进行文献检索与筛选。

（1）检索关键词与数据库选择

为全面覆盖相关研究，我们选取Vulnerability detection、Vulnerability identification、security vulnerability detection、Vulnerability prediction、vulnerability detection using Transformer，vulnerability detection using LLM等作为核心关键词，并在ACM Digital Library、IEEE Xplore Digital Library、Elsevier ScienceDirect、Springer Link Digital Library、Google Scholar和中国知网等在线数据库中进行检索。我们重点关注2020年以来出现的利用Transformer和大语言模型（如GPT系列、CodeBERT、CodeT5、ChatGPT、StarCoder以及CodeGen等）开展漏洞检测的相关研究（截至2025年12月）。

（2）文献筛选与滚雪球搜索

在初步检索得到的文献集合中，我们通过以下步骤进行筛选：1）分析文献的标题、关键词和摘要，过滤掉与源代码漏洞检测无关的论文；2）对筛选后的文献进行前向及反向滚雪球搜索[8]，即通过已纳入文献的参考文献列表与引用该文献的后续研究文献再次进行检索和补充。

然后，我们对所获取的文献进行了系统性过滤，依据以下标准剔除了不符合研究要求的文献：1）书籍、技术报告、学位论文以及社论等；2）仅在部分章节中提及LLMs，但未在研究方法中使用LLMs的论文；3）同一作者团队发表的重复或高度相似的研究成果；4）篇幅过短或实质性内容不足的论文。

最终，我们获得了一批高质量的与“基于大语言模型的源代码漏洞检测”主题直接相关的研究文献，关于论文数量的统计分布如图2和3所示。在此基础上，我们将这些研究文献进行分类与分析。

由于基于Transformer的方法本质上是深度学习方法的一种分支，本文在第3.3节中还对77篇具有代表性的基于传统深度学习网络的源代码漏洞检测方法进行了简要综述，相关详情见表1。

2.3 研究框架概述

本文的整体研究框架如下。首先，介绍大语言模型的基本架构以及代码预训练模型的相关背景知

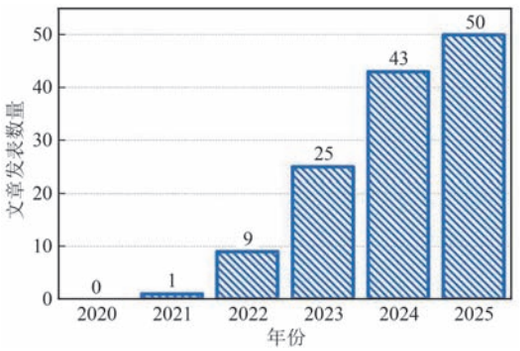

图2　基于大语言模型的源代码漏洞检测文献的年度分布

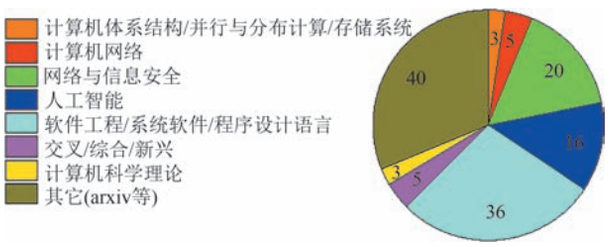

图3　基于大语言模型的源代码漏洞检测文献在不同类型期刊或会议上的发表分布

识。随后，系统梳理传统的源代码漏洞检测技术，包括基于规则和静态分析的方法，以及基于深度学习的检测方法。之后，本文重点综述了当前快速发展的基于大语言模型的源代码漏洞检测方法，围绕其模型设计、训练范式和应用策略展开讨论。接着，介绍该领域常用的评估指标与数据集，为模型性能对比提供参考依据。最后，总结大语言模型在源代码漏洞检测任务中的实际表现，分析其优势与局限以及未来可能面临的挑战，并对未来的研究发展方向进行展望。

为了更清晰地刻画基于大语言模型的源代码漏洞检测方法的演化逻辑与特征差异，本文参考已有研究[9-10]，根据参数规模对现有基于大语言模型的方法进行划分。大语言模型（Large Language Models）一词最早由OpenAI团队在发布GPT-2时提出[11]。GPT-2参数规模最高达1.5B，并在多项下游任务中展现出显著的性能提升。另外，研究表明，模型规模达到该量级附近时，开始具备一定的“涌现能力”，在表现上与轻量级模型形成显著差异。因此，本文以参数规模1.5B为界限，将基于LLM的源代码漏洞检测方法分为如下两类：(1)基于轻量级Transformer的源代码漏洞检测方法：此类方法通常采用参数规模低于1.5B的Transformer模型或其变体。研究者往往针对特定漏洞检测任务或数据集对模型进行预训练或微调，部分方法甚至直接在漏洞数据集上从零开始训练模型。这类方法具有训练高效、部署灵活的优势，适用于资源受限或特定场景的漏洞检测任务，但在跨域泛化与语义建模方面存在一定不足。(2)基于大规模语言模型的源代码漏洞检测方法：此类方法的模型参数规模通常达到数十亿至千亿级别，基于大规模代码与自然语言语料进行预训练，再在下游漏洞检测任务上进行少量微调或零样本适配。部分研究甚至直接利用通用多模态或多任务LLM（如ChatGPT[12]）执行漏洞检测，通过合理的指令设计即可在无需额外微调的情况下实现优异的检测性能。

在后续章节中，第3节将介绍预备知识以及传统漏洞检测方法和基于深度学习的源代码漏洞检测方法。第4节和第5节将分别重点讨论基于轻量级Transformer的源代码漏洞检测方法和基于大规模语言模型的源代码漏洞检测方法。第6节总结常用数据集与评价指标，第7节对基于大语言模型的源代码漏洞检测方法的性能进行综述，以及第8节对未来研究方向进行展望。

3　预备知识

3.1　大语言模型

近年来，大语言模型在自然语言处理和软件工程等领域取得了突破性进展。这一成功不仅得益于超大规模的模型参数、海量多样的训练数据以及强大的计算资源支持[13-15]，也得益于Transformer架构在建模长距离依赖与并行计算方面的显著优势。Transformer作为当前主流LLM的基础架构，其核心组件包括：多头自注意力机制（Multi-Head Self-Attention，MHSA）、前馈网络（Feed-Forward Network，FFN）、残差连接（Residual Connection）[16]、层归一化（Layer Normalization，LN）[17]和位置编码模块。下文将简要介绍各组件的功能与作用机制。

3.1.1　Transformer架构及关键模块

多头自注意力机制（MHSA）：MHSA通过在不同特征子空间并行计算多个注意力头（Heads），以捕获序列中各Token之间的全局依赖关系。具体而言，对于输入序列表示$\boldsymbol{H}^{(l-1)}\in\mathbb{R}^{n\times d_{\text{model}}}$，Transformer首先通过线性变换生成查询（$\boldsymbol{Q}$）、键（$\boldsymbol{K}$）和值（$\boldsymbol{V}$）矩阵，并基于缩放点积注意力计算上下文加权结果。单头注意力计算公式如下所示[18]：

表1 典型基于深度学习的源代码漏洞检测方法特征对比

类型 (Category)	代表性研究 (Representative Studies)	检测模型 (Detection Model)	研究细节 (Research Details)
文件粒度 (File Granularity)	KSEM[42]2015	FCN	基于抽象语法树和编码准则的程序向量表示学习方法
	CODASPY[43]2016	FCN	基于轻量级静态特征与动态特征预测软件漏洞的方法
	ESE[44]2013	CNN	利用基于传统度量标准的故障预测模型进行漏洞检测
	QRS[45]2017	CNN	基于抽象语法树和卷积神经网络的软件缺陷预测框架
	ICTAI[46]2017	GCN	利用控制流图和深度图卷积神经网络自动学习程序语义特征的软件缺陷预测方法
	ICMLA[47]2018	CNN	基于深度特征表示学习的大规模函数级漏洞检测系统
	IST[48]2022	LSTM	一种置信度可解释的、基于深度学习的 Python 源码漏洞检测方法
	Information Sciences[49]2021	GCN/GAT	使用程序的定制中间图表示的软件漏洞分析方法
	JISA[50]2023	BiLSTM，TextCNN	结合代码序列与 AST 结构特征的智能合约漏洞检测方法
	TKDE[51]2023	GRU，DCN	基于自动特征提取与交互的智能合约漏洞检测模型
	NDSS[52]2023	GRU，LSTM	基于深度迁移学习的智能合约多类型漏洞检测框架
	ICBD[53]2023	BiLSTM	基于注意力机制和符号执行的智能合约漏洞检测系统
	IJCAI[54]2019	基于注意力的 CNN	基于注意力机制和代码属性图的细粒度漏洞检测方法
函数粒度 (Function Granularity)	TDSC[55]2019	BiLSTM	基于双向 LSTM 和跨域知识库的函数级漏洞检测方法
	ICECCS[56]2019	GCN	一种针对控制流漏洞的、基于图嵌入的源代码缺陷检测方法
	软件学报[57]2020	BiGRU，BLSTM	基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法
	NeurIPS[58]2019	GNN	通过学习代码语义表示的图神经网络漏洞识别模型
	TIFS[59]2020	GNN	通过学习程序的控制、数据和调用依赖关系的图神经网络模型
	OOPSLA[60]2020	GNN	基于图区间神经网络的源代码分布式表示学习方法
	IST[61]2020	BiGRU	基于程序切片的二进制代码漏洞智能检测系统
	ICICSC[62]2021	GAT	基于注意力图网络的源代码漏洞检测方法
	IST[63]2021	BGNN	基于双向图神经网络的源代码漏洞检测方法
	ICSE[64]2022	CNN	将函数源代码高效转换为图像的源代码漏洞检测方法，兼具可扩展性和准确性
	Computers & Security[66]2022	GGNN，BiLSTM	基于混合语义和双注意力机制的图神经网络漏洞挖掘系统
	Computers & Security[67]2023	GGNN，GRU	基于图嵌入和深度域适应的跨域漏洞检测框架

续表

类型 (Category)	代表性研究 (Representative Studies)	检测模型 (Detection Model)	研究细节 (Research Details)
	Information Sciences[68]2023	BiLSTM，TextCNN	基于特征融合和多模态任务的智能合约漏洞检测方法
	Journal of Systems and Software[69]2023	BiLSTM，GNN	结合序列和图嵌入的深度学习源代码漏洞检测方法
	ICSE[70]2023	GRU，GCN	强调漏洞与良性代码的类别分离特征的图神经网络模型
	ASE[71]2023	GCN，GAT	基于多模态知识融合的函数级漏洞检测方法
	MCSoC[72]2023	GCN	结合图卷积网络和专家模式特征提取的智能合约重入漏洞检测
	IEEE TSE[73]2023	GAT	基于图注意力网络和领域自适应学习实现跨项目漏洞检测
	QRS-C[74]2023	TextCNN	基于 doc2vec 和路径表示的软件漏洞检测框架
	ISSRE[75]2023	BiLSTM	利用预训练模型和基于注意力机制的双向长短期记忆网络检测智能合约重入漏洞
	EAAI[76]2024	GRU	使用图置信学习提取代码结构语义，提升漏洞检测性能
	Computers & Security[77]2024	TextCNN	融合代码语义与结构信息的三元组模型框架
	ICSE[78]2024	GGNN	通过数据流分析启发的图学习框架高效检测代码漏洞
	TOSEM[79]2024	RNN，GAN	结合深度域适应与最大间隔原理的跨项目软件漏洞检测方法
	ASOC[80]2024	BiLSTM	结合操作码和源代码特征的深度学习智能合约漏洞检测方法
片段粒度 (Snippet Granularity)	NDSS[81]2018	BiGRU，BLSTM	使用代码片段表示的深度学习漏洞检测方法
	TDSC[82]2019	BiGRU，BLSTM	基于深度学习和代码注意力机制的多类漏洞检测系统
	MSR[83]2019	CNN	自动从提交信息和代码变更中提取特征以识别缺陷的端到端深度学习框架
	NeurIPS[84]2021	GNN	通过自监督学习同时训练检测器和选择器模型，自动检测和修复代码中的漏洞
	TDSC[85]2021	BiGRU，BLSTM	基于语法语义和向量表示的深度学习 C/C＋＋漏洞检测框架
	ISSRE[87]2021	RGCN	基于切片属性图和三重注意力机制的改进 R-GCN 模型
	Computers & Security[88]2021	BiGRU，TextCNN	基于层次注意力网络的二进制软件漏洞检测方法
	Computers & Security[89]2021	BiGRU	结合BiGRU和注意力机制的软件缺陷检测算法
	Computers & Security[90]2021	GRU，LSTM	结合程序切片和嵌入技术的源代码向量化方法
	Computers & Security[91]2021	BiLSTM	基于代码相似性的源代码漏洞检测方法
	TOSEM[92]2021	GAT	通过图神经网络嵌入代码片段的控制流和数据流信息实现细粒度的漏洞检测
	TOSEM[93]2021	BiLSTM	基于启发式搜索的深度学习漏洞检测解释框架
	JISA[95]2022	Bi-LSTM	基于开源代码学习的深度学习漏洞检测框架
	DSN[96]2022	CNN	通过路径敏感代码片段和多层注意力机制增强语义的漏洞检测方法
	ICSE[97]2023	Bi-LSTM	使用 CWE 树结构的细粒度漏洞检测方法
	Computers & Security[98]2023	GGNN	通过张量表示综合代码图并结合循环门控图神经网络实现多类型漏洞检测

续表

类型 (Category)	代表性研究 (Representative Studies)	检测模型 (Detection Model)	研究细节 (Research Details)
	Computers & Security[99]2023	Bi-LSTM	基于反编译伪代码和 BiLSTM 注意力机制的二进制漏洞检测方法
	Computers and Electrical Engineering[100]2023	Bi-LSTM	基于语义代码结构和自设计神经网络的智能合约漏洞检测
	Computers & Security[101]2024	BGRU	融合多视角特征与层次化增强的多类型漏洞检测框架
	TIFS[102]2024	RNN,GGNN	基于细粒度变量的 C/C++ 代码切片漏洞检测方法
	TIFS[103]2024	GRU	融合上下文关系图和多种嵌入技术的增强图表示学习
	IST[104]2024	LSTM,RGCN	基于关系图卷积神经网络的多语义融合漏洞检测方法
语句粒度 (Statement Granularity)	IJCAI[105]2017	记忆网络	基于记忆网络的端到端数据驱动的缓冲区溢出检测方法
	arXiv[106]2018	记忆网络	研究当前 AI 系统在检测缓冲区溢出漏洞方面的能力极限
	TDSC[107]2021	BiGRU,BLSTM	结合中间代码表示、粒度细化和多文件关联的漏洞定位方法
	IEEE/IFIP[108]2021	边框回归	引入目标检测领域的边界框回归方法来设计细粒度漏洞检测系统
	ESEC/FSE[109]2021	GRU,GCN	基于图卷积网络与可解释 AI 的细粒度漏洞检测方法
	ICSE[110]2022	FS-GNN	基于流敏感图神经网络的语句级内存相关漏洞检测方法
	SANER[112]2022	GGNN+Transformer	结合图神经网络和序列神经网络的集成学习方法
	DSAA[114]2022	GCN	基于多级异构图嵌入的智能合约漏洞细粒度检测方法
	ESEC/FSE[115]2022	GNN	利用异构图嵌入和图神经网络检测智能合约漏洞的方法
	IST[116]2023	GCN,GGNN	基于关系图卷积网络与子图嵌入的语句级软件漏洞检测方法
	IST[117]2023	GCN	有效降低误报率的图卷积跨过程软件漏洞检测方法
	Computers & Security[118]2023	LSTM,GNN	基于切片依赖图和双粒度训练方式的细粒度代码漏洞定位方法
	IEEE TSE[119]2023	GNN	基于元路径和注意力机制的图学习模型

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_{\text{model}}/h}}\right)\mathrm{V} \tag{3}$$

其中，h为注意力头数量，d_{model}为隐藏层维度。多头注意力的总体输出可表示为

$$\text{MHSA}(\boldsymbol{H}^{(l-1)})=\text{Concat}\{\text{Head}_1,\text{Head}_2,\cdots,\text{Head}_h\}\boldsymbol{W}^{O} \tag{4}$$

$\boldsymbol{W}^{O}$为可训练的参数矩阵。通过多头机制，模型能够在不同子空间中并行建模多种语义相关性，从而显著增强了全局特征捕获能力。

在解码阶段(Decoder)，Transformer引入掩码多头自注意力机制，通过在点积注意力计算中加入掩码矩阵$\boldsymbol{M}_{\text{mask}}$，显式约束当前位置仅能关注其自身及之前的序列位置，从而满足自回归生成的因果性要求。其计算公式如下所示：

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_{\text{model}}/h}}+\boldsymbol{M}_{\text{mask}}\right)\boldsymbol{V} \tag{5}$$

其中$\boldsymbol{M}_{\text{mask}}$的上三角部分为$-\infty$，从而确保序列的自回归生成特性。

前馈网络（FFN）：在自注意力层之后，Transformer利用FFN对每个位置的表示独立进行非线性映射，以提升模型的特征抽象能力。FFN由两个线性层和一个中间激活函数（如ReLU）组成，其形式化表示为

$$\text{FFN}(h_i^{(l)})=\max(0,h_i^{(l)}\boldsymbol{W}^{(1)}+b^{(1)})\boldsymbol{W}^{(2)}+b^{(2)} \quad (6)$$

其中，$\boldsymbol{W}^{(1)}\in\mathbb{R}^{d_{\text{model}}\times 4d_{\text{model}}}$和$\boldsymbol{W}^{(2)}\in\mathbb{R}^{4d_{\text{model}}\times d_{\text{model}}}$为可训练参数矩阵。该结构显著增强了Transformer模型的非线性建模能力与表达维度。

残差连接与层归一化：为缓解深层网络训练过程中的梯度消失与梯度爆炸问题，Transformer在每个子层外引入残差连接，并在此基础上进行层归一化（LN）以稳定训练过程。其计算形式为

$$\boldsymbol{H}^{(l)}=\text{LN}(\text{MHSA}(\boldsymbol{H}^{(l-1)})+\boldsymbol{H}^{(l-1)}) \quad (7)$$

$$\boldsymbol{H}^{(l)}=\text{LN}(\text{FFN}(\boldsymbol{H}^{(l)})+\boldsymbol{H}^{(l)}) \quad (8)$$

位置编码：由于自注意力机制本身并未显式建模Token的位置信息，Transformer通过位置编码（Positional Encoding）为输入序列引入位置信号[18]。其基本形式为正弦-余弦函数：

$$p_{pos,2i}=\sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \quad (9)$$

$$p_{pos,2i+1}=\cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \quad (10)$$

在此基础上，研究者提出了多种改进方案，如可学习位置嵌入[19]、相对位置编码[20]、旋转位置编码（RoPE）[21]及ALiBi[22]等，以更好地适配长序列与复杂上下文的建模需求[15,23]。

3.2　代码预训练模型

随着大语言模型在自然语言处理任务中取得突破性进展，软件工程领域也逐步衍生出以代码为核心的模型体系，即代码大语言模型（Code LLMs）。这类模型通过在大规模源代码语料上进行预训练，显著提升了模型在程序理解、代码生成及安全分析等任务中的能力[15]。与通用LLMs类似，代码大语言模型可按照网络结构划分为三类：仅编码器架构、仅解码器架构以及编码器-解码器架构[2,24-26]。当前主流代表如图4所示，其中对应用于漏洞检测任务的模型用虚线框标注。

仅编码器架构：该类模型仅使用Transformer的Encoder模块进行预训练，主要面向代码理解类任务，如类型推断、代码克隆检测和代码搜索。典型代表是CodeBERT[24]，其在多模态代码语料库CodeSearchNet[27]上预训练，通过对齐源代码与自然语言注释，增强模型的语义理解能力。为进一步融入结构特征，GraphCodeBERT[28]在预训练阶段引入数据流（Data Flow）信息，并设计了结构感知目标（如边预测与节点一致性预测），以强化模型的语义建模能力。实验结果表明，融合代码结构与语义关系可显著提升模型在代码理解任务中的性能。

仅解码器架构：此类模型采用自回归（Autoregressive）生成机制，类似于GPT系列模型，主要应用于代码生成、补全及摘要任务。Codex[25]在大规模Python代码上预训练，具备强大的零样本与少样本代码生成能力；StarCoder[26]在多语言语料上训练，强调代码逻辑一致性与补全能力；而近年来发布的GPT-4[29]与GPT-5模型在推理与上下文理解方面进一步提升，可支持更复杂的跨函数逻辑生成。研究表明，这类生成式模型即使在缺乏微调或仅通过少量指令微调的条件下，仍能在漏洞预测与定位任务中保持较高精度与泛化性[30]。

编码器－解码器架构：该架构结合了双向编码器与单向解码器的优势，适用于同时涉及理解与生成的复杂任务。CodeT5[2]通过多任务学习框架，将多种代码任务统一建模，并在微调阶段通过提示指令区分任务类型，从而兼顾生成质量与语义理解能力；其改进版本CodeT5+[31]进一步增强了生成性能。PLBART[32]则采用消噪预训练目标，在自然语言-编程语言平行语料上进行联合建模，为代码翻译、重构与补全任务提供高质量支持。此外，AlphaCode[33]在竞赛级算法问题上展现出卓越的推理与程序合成能力，证明了该架构在复杂代码逻辑建模中的潜力。

近年来，Code LLMs凭借预训练优势与语义理解能力，在程序理解与漏洞检测中展现出了强大潜力，有望突破传统深度学习方法的性能瓶颈，进一步推动智能化漏洞检测体系的构建与完善。

3.3　传统漏洞检测方法

基于程序分析的漏洞检测方法采用不同的程序分析技术（例如动态分析和静态分析）来检测软件中的潜在漏洞。其中，基于动态分析的漏洞检测方法（例如模糊测试[34]和污点分析[35-36]）通过运行程序并监控其执行过程中产生的数据和行为来检测漏洞，准确度较高。然而，动态分析的有效性严重依赖于测试用例的覆盖质量：若输入集合不完整或关键路径未被触发，则潜在漏洞可能无法暴露。此外，程序

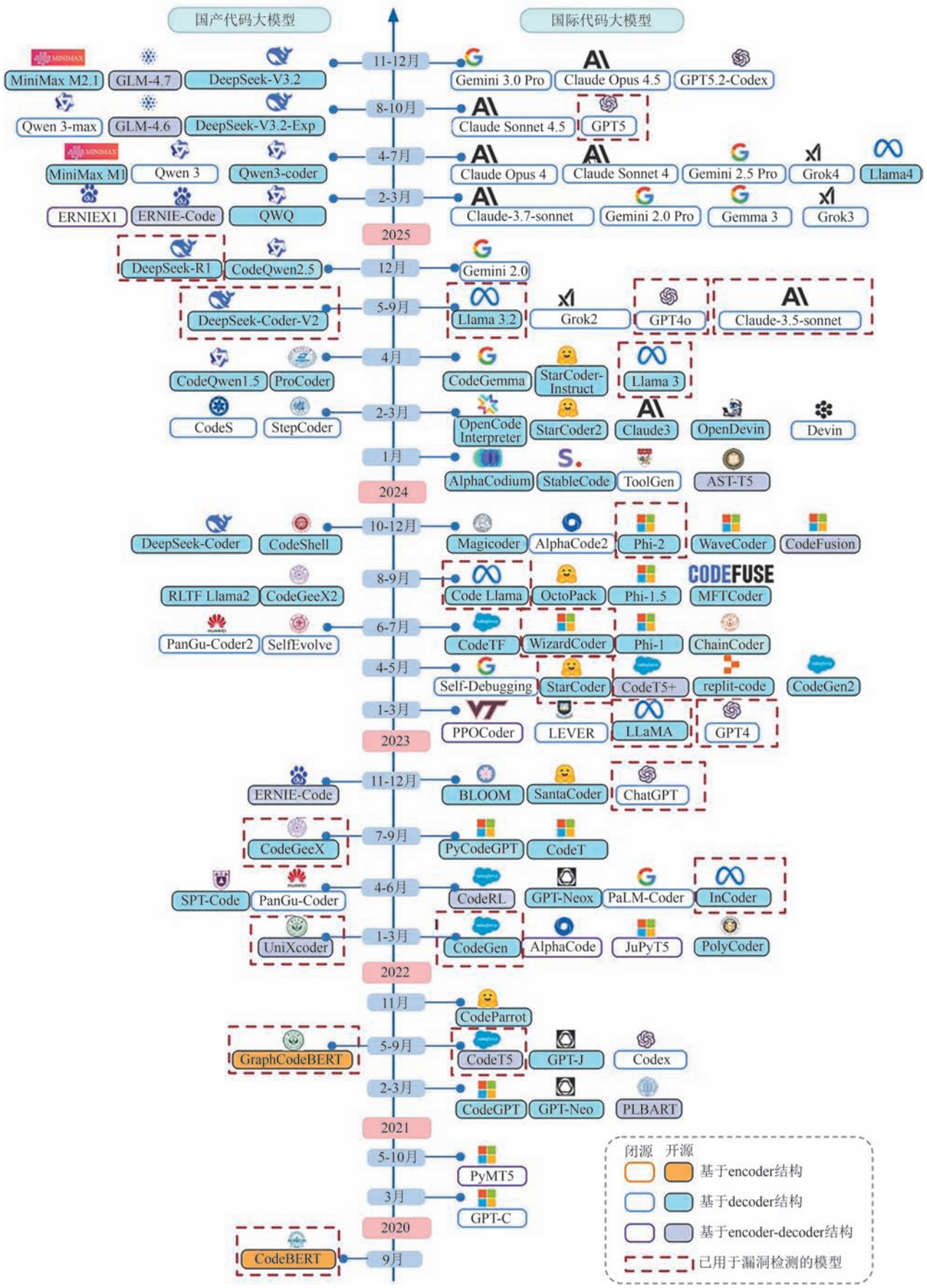

图4 典型大规模代码预训练模型及其在漏洞检测任务中的应用

在运行时难以遍历所有执行路径，导致覆盖率不足，从而造成较高的漏报率。基于静态分析的漏洞检测方法是直接在源代码或中间代码层面进行语法与语义分析来检测漏洞，无需依赖特定的运行环境或执行程序。然而，由于该类方法通常依赖启发式规则[37-38]或模式匹配算法[39-41]，在面对复杂的控制结构与上下文依赖时，往往难以捕捉深层语义关系，仅能针对已知漏洞模式进行识别，因此存在较高的误报率与局限性。

图5展示了当前主流的基于程序分析的漏洞检测工具。相比开源工具，商业工具通常集成更复杂的分析能力（如跨函数数据流分析、上下文建模等），支持更多漏洞类型与编程语言，并在实际应用中表现出更高的检测精度和稳定性。

类型	代表性研究或工具	检测语言	研究机构
商业工具	Checkmarx	20种语言	以色列Checkmarx公司
	Coverity	19种语言	美国Synopsys公司
	Fortify	25种语	美国Macro Focus公司
	CodeSonar	C/C++/Java	美国GrammaTech公司
	Klocwork	C/C++/Java	美国Klocwork公司
	CodeSecure	6种语言	美国Armorize Technologies公司
开源工具	Flawfnder	C/C++	由Linux基金会开源供应链安全总监David Wheeler开发
	CppCheck	C/C++	由瑞典Evidente公司的Daniel Marjamaki等人开发
	FindBugs	Java	美国马里兰大学
	ErrorProne	Java	美国Google公司
	Clang Static Analyzer	C/C++	开源LLVM计划下的子项目
	RATS	C/C++	美国Secure Software Inc公司

图5　基于规则的漏洞检测方法

基于深度学习的源代码漏洞检测方法通过利用神经网络从大规模漏洞数据中自动学习多层次特征表示，从而摆脱了对人工特征设计的依赖，显著降低了特征工程成本，并具备更强的特征抽象能力与泛化性能[5]。目前具有代表性的相关研究如表1所示，现有方法主要从文件级[42-53]、函数级[54-80]、代码段级[81-104]和语句级[105-119]等不同粒度层面开展代码漏洞检测。除检测粒度差异外，深度学习方法在代码表示形式的选择上也展现出显著的差异性。这种差异主要源于研究目标（如漏洞类型）、程序结构复杂度以及建模重点的不同。为了更有效地捕获程序的语法与语义特征，目前常用的代码表示方式可归纳为三大类：基于序列的表示（如Token序列，子词序列）[48,50,69,91,93,95]、基于语法结构的表示（如抽象语法树AST）[42,45,50,97,103]、以及基于程序图的结构化表示（如控制流图CFG、数据流图DFG等）[56-59,62,67,69-70,72,87,92,104,109-110,112-113,116-119]。

其中，基于序列的源代码漏洞检测方法将源代码分词为符号序列，并利用RNN或LSTM等神经网络对其建模，主要关注表层语法特征。基于语法树的源代码漏洞检测方法将代码解析为抽象语法树，保留了标识符、运算符和控制结构之间的嵌套与依赖关系，适用于结构化分析。基于图的源代码漏洞检测方法则通过构建控制流图、数据流图或程序依赖图，并利用图神经网络进行特征传播，以捕捉控制与数据依赖等关键语义信息。

上述基于深度学习的漏洞检测方法多采用RNN、CNN或GNN等架构，通常通过顺序建模或邻接信息聚合来近似构建全局结构。但这类架构在处理长距离依赖和深层语义关系方面仍存在一定局限。尽管已有研究尝试引入代码切片或图结构简化等策略缓解该问题，但模型对复杂上下文的理解能力仍显不足。随着Transformer在自然语言处理与代码理解任务中的广泛应用，其自注意力机制在建模长距离依赖和捕获全局语义特征方面表现出显著优势，从而逐步成为漏洞检测领域的主流方法。基于第2.3节提出的研究框架，本文将现有方法分为两类：基于轻量级Transformer的源代码漏洞检测方法（见第4节）和基于大规模语言模型的源代码漏洞检测方法（见第5节）。

4 基于轻量级Transformer的源代码漏洞检测方法

随着Transformer架构在软件漏洞检测中的广泛应用，基于轻量级Transformer的检测方法因其良好的可训练性与工程实用性，受到了广泛关注。该类模型能够在有限硬件资源和中等规模数据集条件下完成有效训练或微调，并在特定漏洞检测任务上取得较高的检测精度。与超大规模LLM相比，轻量级Transformer模型具有训练成本低、迭代速度快与部署灵活等优势，更适用于快速原型开发、小规模场景定制与工程验证等实际应用需求。

为了便于理解轻量级Transformer模型在漏洞检测任务中的工作机制，本文首先对其整体流程与传统深度学习方法进行了对比分析。如图6(a)所示，传统基于深度学习(如LSTM)的方法通常包含三个核心阶段：首先，对漏洞代码样本进行解析与编码，将其转换为向量表示；其次，将这些向量输入神经网络以提取深层语义特征；最后，利用分类器对特征进行判别，实现漏洞预测与结果输出。相比之下，基于Transformer的源代码漏洞检测流程(见图6(b))通常包括四个核心阶段：(1)预训练数据构建，即收集大规模开源代码及其自然语言描述，形成适用于模型训练的单模态程序语料(Programming Language，PL)或程序与自然语言的双模态语料(Natural Language-Programming Language，NL-PL)；(2)预训练阶段，通过设计特定的预训练任务(如Masked Language Modeling，MLM)对Transformer模型进行预训练，以学习跨语言的通用语义表征能力；(3)任务微调阶段，在具体漏洞检测数据集上采用有监督学习对模型进行适配优化，以提升其在目标任务中的性能；(4)漏洞检测阶段，对待检测函数及其内部语句进行特征编码，分别执行函数级与语句级的漏洞识别，从而实现多粒度检测。

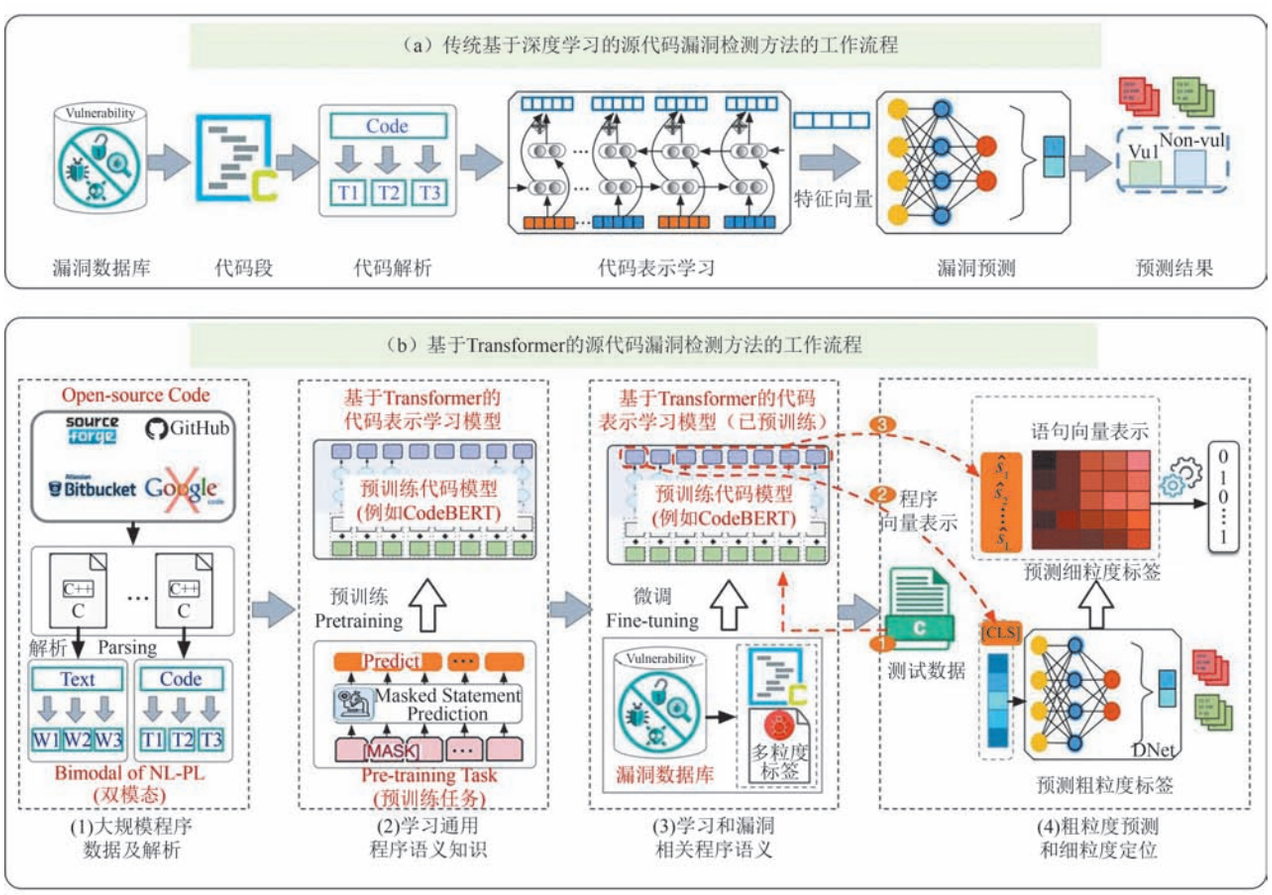

图6 基于Transformer模型的源代码漏洞检测方法(b)与传统基于DL的源代码漏洞检测方法(a)的工作流程对比

本文对近年来基于轻量级Transformer的代表性方法及其关键特征进行了系统性梳理。表2中总结了各方法在预训练模型、Tokenization策略、检测粒度、目标函数以及输入长度等方面的主要差异。

表 2　基于轻量级 Transformer 的源代码漏洞检测方法特征对比

方法 Method	分词 Tokenization	是否 Pre-train Re-Pre-train	预训练模型 Pretrained Model	检测粒度 Detection Granularity	微调损失函数 Fine-tune Loss	最大长度 Max Length
Peculiar (2021)[120]	BPE	No	GraphCodeBERT	Function-level	Cross-entropy loss	512
LineVul (2022)[111]	BPE	No	CodeBERT	Multi-level	Cross-entropy loss	512
LineVD (2022)[113]	BPE	Yes	CodeBERT + GAT	Statement-level	Cross-entropy loss	512
VulBERTa (2022)[121]	BPE	Yes	BERT + CNN	Function-level	Cross-entropy loss	1024
SDBV (2022)[122]	WordPiece	No	HGT	Function-level	Cross-entropy loss	未知
VELVET (2022)[112]	WordPiece	Yes	Transformer + GGNN	Statement-level	Cross-entropy loss	512
Yuan et al. (2023)[123]	WordPiece	No	Transformer Encoder	Function-level	Cross-entropy loss	未知
ASSBert (2023)[124]	BPE	Yes	CodeBERT	Function-level	Cross-entropy loss	512
SVulD (2023)[125]	BPE	No	UniXcoder	Function-level	Contrastive loss	512
EPVD (2023)[126]	BPE	No	CodeBERT + CNN	Path-level	Cross-entropy loss	512
RoBERTa-PFGCN (2023)[127]	BPE	No	RoBERTa + GCN	Function-level	Focal Loss	512
StagedVulBERT (2024)[10]	BPE	Yes	CodeBERT-HLS	Multi-level	Cross-entropy loss	2048
RLFD (2025)[128]	BPE	Yes	CodeBERT-HLS	Fine-level	Customized Loss	2048
DetectVul (2025)[129]	WordPiece	No	BERT	Statement-level	Cross-entropy loss	512

4.1　Tokenization方法：子词切分策略

在漏洞检测任务中，源代码的分词方式对模型性能具有重要影响。传统的深度学习方法通常采用以单词或标识符为基本单元的词级（Word-level）切分策略，常见实现方式为词法分析（Lexical Analysis）。例如，对于代码片段“defunPremulSkImageToPremul(a):”，词法分析会将其切分为[“def”，“unPremulSkImageToPremul”，“(”，“a”，“)”，“:”]。该方法能够较好地保留常见标识符的整体结构，但在面对较长且稀有的标识符时，词表规模会急剧膨胀，从而引发严重的Out-of-Vocabulary问题。

为缓解上述问题，研究者提出了子词级（Subword-level）切分算法，如BPE（Byte Pair En-coding）[130]、WordPiece[131]与Unigram[132]等。以BPE为例，该算法通过频率统计不断合并高频字节对，生成更大的子词单元，从而在压缩词表规模的同时显著降低OOV风险。例如，对于函数名“unPremulSkImageToPremul”，BPE将其切分为[“un”，“Prem”，“ul”，“Sk”，“Image”，“To”，“Prem”，“ul”]。其中，“Image”作为常用词被完整保留，而稀有标识符部分被拆分为可重用的子词单元。这一策略既保留了常见词的语义完整性，又增强了模型对多样化标识符的泛化能力。实验证明，在代码预训练与微调阶段采用BPE子词切分相较于传统词级分词，可获得更优的检测性能[111]。

由于源代码的语法结构复杂多样，直接复用自然语言处理中的分词策略往往难以取得理想效果。因此，当前主流方法通常在大规模代码语料上重新训练专用Tokenizer。例如，LineVul[111]使用CodeBERT模型，并基于CodeSearchNet数据集通过BPE算法构建了专门适用于源代码的Tokenizer。VulBERTa[121]进一步结合编译器前端（Clang）进行词法分析，将代码切分为语法单元后再应用BPE，并在词表中保留C/C++关键字、标点及标准API函数，以确保语法元素的完整性。Chirkova等人[133]提出CodeBPE，在预处理阶段保留如“]);”等标点组合，显著压缩输入序列长度达17%，同时保持下游任务性能。StagedVulBERT[10]则采用分层建模策略，先将程序划分为语句级单元，再在语句内应用BPE切分，并通过Token Transformer和语句编码器提取多粒度语义信息，有效捕捉跨语句依赖。

4.2　预训练模型选择

在基于轻量级Transformer的源代码漏洞检测方法中，预训练模型的选择与适配是影响检测性能的核心环节。根据其设计目标与训练策略的不同，现有研究主要可分为三类：(1)面向通用任务的预训练模型；(2)面向特定漏洞检测任务的专用预训练模型；(3)融合图神经网络的混合模型。下面对三类模型的代表性工作进行系统梳理与分析。

4.2.1　面向通用任务的预训练模型

通用型预训练模型通常在来自GitHub等平台的大规模开源代码语料上进行无监督训练，典型代表包括BERT、CodeBERT、GraphCodeBERT与

UniXcoder等。这类模型通过预训练任务(如Masked Language Modeling,MLM;Replaced Token Detection,RTD;Denoising等)学习代码的语法与语义表示,从而获得跨任务迁移的基础能力。由于其已具备较强的通用表征能力,研究者通常仅需在下游漏洞检测数据集上进行轻量级微调(Fine-tuning),即可实现较优的性能。例如LineVul[111]等方法在CodeBERT的基础上进行针对性微调,从而显著提升了模型在漏洞检测任务上的性能。

面向通用任务的预训练模型主要应用于函数级漏洞检测任务,但也可扩展至更细粒度的语句级检测。LineVul即利用自注意力机制计算每个Token的注意力得分,并在语句级别聚合这些得分以评估该语句的漏洞风险。当函数级预测表明目标函数可能存在漏洞时,模型会进一步在语句级别上定位潜在缺陷,从而实现由粗到细的漏洞定位能力。

4.2.2 面向特定漏洞检测任务的预训练模型

与通用模型不同,面向特定任务的预训练模型旨在提升模型对程序结构特征、语义依赖与漏洞模式的捕捉能力。现有研究主要从模型架构优化与输入特征扩充两方面进行改进。

(1)模型架构的优化。StagedVulBERT[10]是此类方法的代表性工作。该研究在CodeBERT基础上引入分层结构(Hierarchical Layered Structure,HLS),构建了CodeBERT-HLS模型,分别在Token层与语句层提取程序语义特征,并在函数级(粗粒度)与语句级(细粒度)漏洞检测任务上采用双重监督学习机制。相较于原始CodeBERT,StagedVulBERT针对长代码序列与多粒度建模问题设计了更精细的结构与训练目标,显著提升了模型的全局上下文感知与细粒度漏洞定位能力。在此基础上,RLFD[128]进一步结合强化学习思想,将细粒度漏洞定位问题形式化为序列决策过程,并设计基于IoU与排序性能的奖励函数以引导模型训练。实验结果表明,RLFD相较于StagedVulBERT在细粒度漏洞定位任务上取得了更优的性能。除了设计特定于漏洞检测任务的新模型结构外,也有研究借鉴并应用了能够原生处理多模态信息的通用图模型。例如SDBV[122]采用了异构图Transformer(Heterogeneous Graph Transformer,HGT),以联合建模序列信息与图结构信息,实现两类特征的深度融合,从而增强模型的上下文语义理解能力。

(2)输入特征的扩充与融合。在保留Transformer主体结构的基础上,部分方法通过引入多模态或结构化输入信息,进一步增强对漏洞语义特征的建模能力。Peculiar[120]构建了与漏洞触发要素高度相关的数据流关键路径图,并结合GraphCode-BERT对图结构与代码序列进行联合编码,从而提升了对漏洞触发模式的捕捉能力。Yuan等人[123]在Transformer编码器中并行引入多模态输入(包括源代码、中间表示与汇编代码),并提出熵嵌入机制以融合多层编译阶段的语义信息,从而实现更全面的漏洞特征建模。EPVD[126]则通过对函数的控制流图进行多路径拆解,结合卷积神经网络与预训练编码模型,对路径内与路径间的语义关系进行联合建模,从而提升了漏洞检测性能。

虽然上述两类思路在模型改进方式上有所差异,但其共同目标均在于克服原有Transformer或预训练模型对程序复杂结构感知不足的问题,通过合并额外的语法语义信息或更灵活的分层架构,显著提升漏洞检测的准确率与稳定性。随着更多针对特定漏洞检测场景的需求涌现,面向特定任务的预训练模型设计在未来仍将持续受到关注。

4.2.3 通用预训练模型与图神经网络的融合模型

除标准Transformer架构外,近年来研究者也探索了在轻量级Transformer基础上融合图神经网络的混合模型,以进一步提升模型对程序结构与依赖关系的建模能力。Hin等人[113]提出的LineVD模型以CodeBERT生成的Token表示为基础,结合图注意力网络(GAT)提取语句级特征,从而实现细粒度的漏洞检测。此外,也有研究分阶段地利用预训练模型和图网络各自的优势。例如RoBERTa-PFGCN[127]先利用RoBERTa为代码的语义漏洞图(Semantic Vulnerability Graph)节点生成向量表示,再借助图卷积网络(GCN)聚合结构特征进行分类预测。上述混合模型结合了GNN在捕捉局部结构依赖方面的优势与Transformer在建模全局上下文语义方面的能力,在多粒度漏洞检测任务中取得了良好效果,但传统GNN的信息传递范围通常受限于邻域节点,难以充分捕捉远距离依赖特征[112]。为此,Ding等人[112]提出了VELVET框架,通过门控图神经网络与Transformer的集成学习实现局部与全局表征的融合,有效兼顾了程序结构的拓扑依赖与语义关联。该模型在多个数据集上表现出优于单一结构的检测效果,验证了融合GNN与Transformer架构在漏洞识别中的潜力。

4.3 预训练任务设计

是否需要在已有模型基础上重新设计预训练任

务，主要取决于模型结构创新程度及其与下游任务的适配需求。对于直接采用BERT、CodeBERT等通用模型的方法，一般无需额外开发新的预训练任务，常用的掩码语言建模（Masked Language Modeling，MLM）或替换标记检测（Replaced Token Detection，RTD）即可满足训练需求。模型在微调阶段可利用带标签的漏洞数据集进行任务特化，以补充其在漏洞识别领域的特征表示能力。若模型在结构或输入输出形式上进行了显著改进，则需要设计任务相关的预训练目标以匹配新的网络特征。典型代表是StagedVulBERT提出的掩码语句预测（Masked Statement Prediction，MSP）任务[10]。MSP在行级别对语句进行遮掩与预测，以适配分层结构下的长序列建模需求。具体而言，MSP将部分代码行替换为[MASK]或随机Token，并通过LSTM子模块重建完整语句，从而使模型在语句级别学习上下文依赖。与传统的Token级MLM不同，MSP关注更高层次的语义单元，能够同时捕获语句内的深层语义和语句间逻辑关系。MSP与MLM的差异及其在训练流程和粒度上的比较如图7所示。

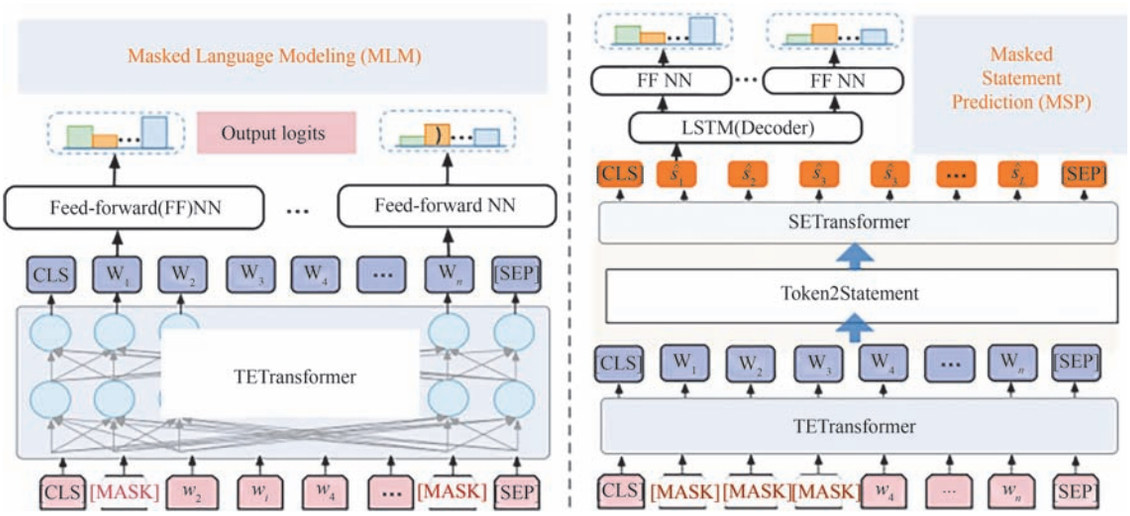

图7　预训练任务MSP（右）和MLM（左）的对比

4.4　检测粒度和微调损失

在基于轻量级Transformer模型的源代码漏洞检测方法中，检测粒度的选择方式与微调阶段损失函数的设计，均对模型性能产生重要影响。

（1）检测粒度设计。现有研究可按检测粒度划分为三类：多粒度、函数级与语句/行级。多粒度方法（如StagedVulBERT[10]、LineVul[111]）在函数与语句两个层面联合建模，通过分层表示与双重监督同时捕获全局语义与局部细节。函数级方法（如VulBERTa[121]、SVulD[125]）以完整函数为基本单元进行判别，强调上下文的完整性与整体语义的一致性，具有较好的可扩展性与稳定性。语句/行级方法（如RLFD[128]）面向细粒度定位，直接在语句或代码行层面识别潜在缺陷，定位精度更高。

（2）微调阶段的损失函数设计。在模型微调阶段，损失函数用于指导参数更新，以最小化预测结果与真实标签之间的差异。最常用的损失形式包括交叉熵损失（Cross-Entropy Loss）与对比损失（Contrastive Loss）。以语句级别的交叉熵损失为例，假设数据集中共有N个函数，每个函数包含n条语句，y_{ij}表示第i个函数中第j条语句的真实标签（有漏洞为1，无漏洞为0），s_{ij}为其输入特征表示，$P_\theta(y_{ij}|s_{ij})$表示参数为θ的模型预测该语句属于漏洞类别的概率。语句级交叉熵损失可定义为

$$L_{\mathrm{CE}}(\theta)=\sum_{i=1}^{N}\sum_{j=1}^{n}-\log P_\theta(y_{ij}|s_{ij}) \tag{11}$$

该损失用于度量模型预测分布与真实分布之间的差异，通过最小化$L_{\mathrm{CE}}(\theta)$，模型可逐步提升对语句层面漏洞特征的区分能力。

当检测粒度扩展至函数级时，每个函数样本仅对应一个标签y_i及特征表示f_i，此时，函数级的交叉熵损失定义为

$$L_{\mathrm{CE-func}}(\theta)=\sum_{i=1}^{N}-\log P_\theta(y_i|f_i) \tag{12}$$

该定义在结构上与语句级形式一致，但以函数整体表示f_i作为输入，以提升函数级漏洞检测性能。

在部分场景中，漏洞函数与其修复版本之间的

差异极小，传统分类损失难以有效学习区分边界。为此，可引入对比损失[125]，其核心思想在于通过约束样本间的距离关系来优化特征空间分布。假设存在函数F，以及对应的正样本函数P（与F语义相近）和负样本函数N（与F无关或差异较大），其嵌入表示分别为E_F、E_P、E_N，则对比损失可定义为

$$L_{\text{contrast}}(\theta)=\sum_{i=1}^{N}\max\left(\left\|E_{F_i}-E_{P_i}\right\|-\left\|E_{F_i}-E_{N_i}\right\|+\varepsilon,0\right) \tag{13}$$

其中，ε为距离阈值。通过最小化$L_{\text{contrast}}(\theta)$，模型将相似函数向量拉近、异质向量拉远，从而提升特征的可分性。例如，SVulD[125]通过三元组结构输入原始、修复与无关函数，借助对比损失提升函数级表示的聚合性与区分性。

交叉熵损失在多分类及二分类任务中适用于语句级或函数级检测场景，具备实现简单与收敛稳定的特点；而对比损失更强调特征空间的结构优化，适合捕获语义相似函数之间的微小差异。在实际应用中，研究者常根据任务粒度与样本特征的差异，结合多种损失函数进行联合优化，以在检测精度、泛化能力与特征可分性之间取得平衡。

4.5 最大检测长度

由于轻量级Transformer架构在序列建模中存在固定的输入长度限制（通常为512或1024个Token），多数预训练模型在处理超长源码文件时面临输入截断问题。在漏洞检测任务中，函数体或文件的Token数量往往远超该阈值，直接截断可能导致关键上下文信息丢失，从而影响检测性能。为此，部分研究提出了针对长序列的改进策略。例如，StagedVulBERT[10]在Token层面对长代码进行分段编码，分别获取局部语义表示；随后，将各分段向量拼接并在语句层次通过第二阶段编码器进行全局融合，以同时捕获局部细节与全局依赖。实验表明，该机制显著提升了模型在真实项目代码中的检测效果，为后续长上下文建模提供了可行思路。

4.6 小 结

轻量级Transformer模型在漏洞检测任务中以较低的计算成本与较高的灵活性展现出良好的实用性，特别适用于中小规模数据集或特定领域的安全检测场景。本文从Tokenization策略、预训练模型选择、再预训练需求、检测粒度与损失函数设计以及输入长度限制等关键方面对该类方法进行了系统梳理与对比分析。结果表明：一方面，基于通用预训练模型的轻量级方法可通过微调快速适配漏洞检测任务，具备良好的迁移能力与开发效率；另一方面，当模型结构或任务特征发生显著变化时，通过再预训练、引入图结构信息（如GNN融合）或采用分层建模策略，能够在性能与泛化性之间实现有效平衡。

5 基于大规模语言模型的源代码漏洞检测方法

大规模语言模型的显著特点是参数规模庞大，通常超过十亿。从零开始训练一个专用于漏洞检测的大规模语言模型，不仅需要海量的高质量语料和算力的支持，还面临任务数据稀缺、标注成本高昂的问题。现有研究通常基于预训练的开源大模型进行微调以适应漏洞检测任务[134-137]，或借助提示工程提升通用大模型在漏洞检测任务上的性能[138-143]，也有研究尝试将大规模语言模型与传统静态分析结合以提升检测效果。此外，近年来基于大模型驱动的多智能体系统在漏洞检测领域逐渐受到关注。因此，本文将目前的研究方法分为四类：基于大模型微调的检测方法、基于提示优化的检测方法、融合静态分析和大模型的检测方法以及基于大模型驱动多智能体系统的检测方法。表3总结了当前代表性方法及其核心特征。

5.1 基于大规模语言模型微调的源代码漏洞检测方法

基于大规模语言模型微调的源代码漏洞检测方法在整体流程上与轻量级Transformer方法类似，但由于大规模语言模型具备更强的语义表征能力和更大的上下文建模空间，其在微调机制、输入处理与优化策略等方面展现出新的特征与挑战。本节将从五个方面展开论述：输入处理策略、模型架构设计、训练方式、损失函数设计及性能优势。图8展示了基于大模型微调的源代码漏洞检测总体流程。

5.1.1 大规模语言模型的输入处理

轻量级Transformer模型通常直接采用基于序列或图的程序表示作为输入，而大规模语言模型对于输入数据的处理主要分为两种情况，常规微调和指令微调。在常规微调中，输入形式与轻量级模型基本一致，通常直接将代码的Token序列或图结构表示输入模型，以适应特定下游任务。与此同时，也有研究针对大规模语言模型的输入特性进行了优化：Shestov等人[134]发现，单个函数的长度通常也有研究针对大规模语言模型的输入特性进行了优化：Shestov等人[134]发现，单个函数的长度通常远小于大规模语言模型的上下文窗口，导致计算资源利用

表 3　基于大规模语言模型的源代码漏洞检测方法特征对比

方法 Method	模型 Model	检测粒度 Detection Granularity	类型 Type	技术细节 Details
Shestov等人[134]	WizardCoder	Function-level	微调	LoRA 微调
Yusuf等人[135]	CodeLlama	Function-level	微调	指令微调
LLMAO[137]	CodeGen	Statement-level	微调	adapters 微调
MSIVD[136]	CodeLlama	Multi-level	微调	多任务指令微调
VulLLM[144]	CodeLlama-13B	Multi-level	微调	多任务指令微调
LLMxCPG[145]	QwQ-32B-Preview	Slice-level	微调	代码属性图、反向切片
ReVD[146]	Qwen2. 5-Coder-7B	Function-level	微调	课程偏好优化
Zhang等人[139]	ChatGPT	Function-level	提示工程	数据流图、API 调用
VUL-GPT[140]	GPT-3. 5-turbo	Function-level	提示工程	上下文学习
GRACE[147]	GPT-4	Function-level	提示工程	上下文学习、图结构
Vul-RAG[148]	GPT-4	Function-level	提示工程	检索增强生成
DLAP[141]	GPT-3. 5-turbo	Function-level	提示工程	检索增强生成、上下文
Mao等人[149]	GPT-3. 5-turbo	Function-level	提示工程	多角色讨论
LOVA[150]	Llama-3. 1-8B	Statement-level	提示工程	自注意力机制
GPTScan[151]	GPT-3. 5-turbo	Function-level	混合	场景属性匹配
INFERROI[152]	GPT-4	Function-level	混合	资源泄露检测
TitanFuzz[153]	GPT-4	Function-level	混合	模糊测试
CodeAgent[154]	GPT-3. 5, GPT-4	Commit-level	多智能体	模拟代码审查
Yildiz等人[155]	GPT-4o, DeepSeek-Coder	Repository-level	多智能体	仓库级上下文
LLM-SmartAudit[156]	GPT-3. 5, GPT-4o	Function-level	多智能体	广度与深度双重分析

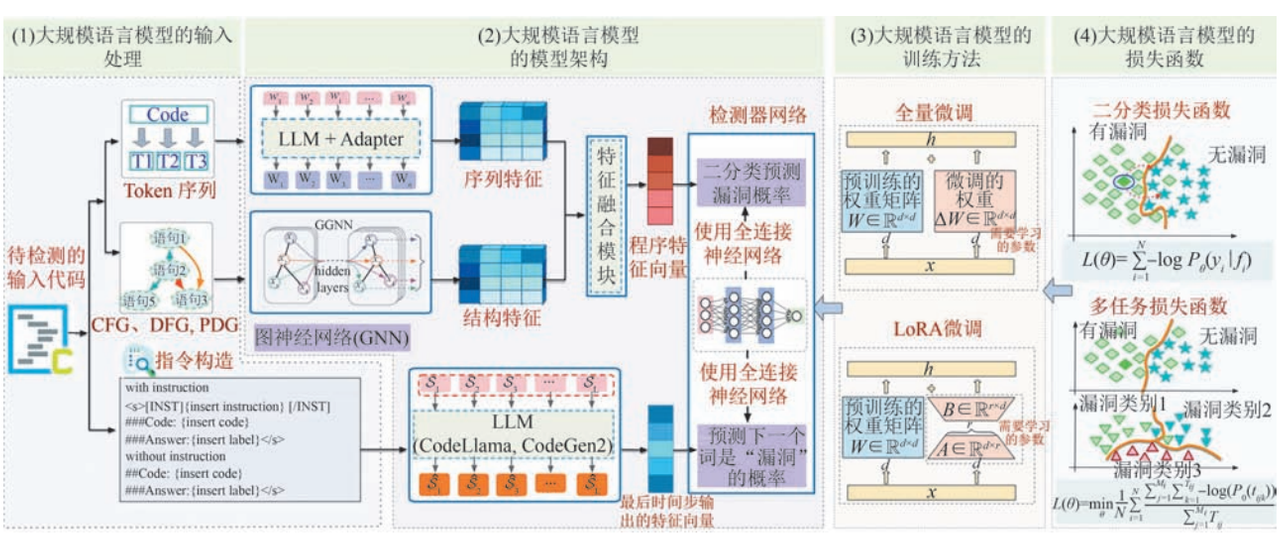

图 8　基于大模型微调的源代码漏洞检测方法的总体流程

不足，因此提出将多个函数样本打包为单个训练输入，从而提升批处理效率。此外，Lekssays等人[145]基于代码属性图(Code Property Graph，CPG)与反向切片技术，从源码中提取与漏洞强相关的语义切片，将这些代码片段作为输入，以强化模型对关键结构的聚焦能力。基于指令微调的方法则需要将漏洞数据构造成{指令(Instruction)、输入(Input)、输出(Output)}的格式，以显式引导模型理解任务目标。Yusuf等人[135]构造了带有自然语言指令和纯代码输入两种微调数据形式，比较不同指令模式下的检测性能；Yang等人[136]进一步将原始数据重组为多轮指令对话形式，将多个漏洞检测与修复子任务融合于统一的指令框架中，提升了模型在漏洞检测以及相关任务上的泛化能力。

5. 1. 2　大规模语言模型的模型架构

与轻量级Transformer类似，大规模语言模型可

通过直接微调或集成其他模块来提升漏洞检测性能，因此本节将基于大规模语言模型的方法划分为两类：直接微调基础开源大模型和在基础开源大模型上集成额外模块。

(1)直接微调基础开源大模型。直接微调方法通常以通用或代码专用的开源大模型为基础，通过有监督学习方式适配漏洞检测任务。常见的基础模型包括通用预训练模型(如LLaMA[157]、Mistral[158])及代码专用模型(如CodeLlama[159]、StarCoder2等)。由于漏洞检测的输入主要为源码，研究者通常倾向选择代码专用模型。例如，Yusuf等人[135]在CodeLlama、CodeT5、LLaMA等五个模型上进行了性能评估，发现CodeLlama在检测准确率方面表现最优，并据此作为后续微调的基础模型。Shestov等人[134]则在二元问答式任务(输出为“YES”或“NO”)中比较了CodeGeeX、WizardCode、CodeGen等模型的检测性能，并综合考虑计算资源与任务适配性，最终选取WizardCode作为基础模型。

(2)结合基础模型与其他模块。另一类研究在基础模型之上引入结构增强模块或轻量参数层，以进一步提升漏洞检测的性能与效率。Yang等人[137]提出基于适配器(Adapter)的框架LLMAO，在CodeGen的Transformer结构中添加双向适配器层，仅更新适配器参数而冻结主模型，从而在保持检测性能的同时显著降低训练开销，提升了大规模语言模型在漏洞检测任务中的实用性。由于轻量级Transformer模型的参数规模较小，适配器结构在此类模型的源代码漏洞检测任务中尚不具备明显的效率优势，因此应用相对较少。

此外，Yang等人[136]将大规模语言模型与图神经网络(GNN)结合，提出在模型顶部增加轻量GNN层，以显式建模程序中的数据流与依赖关系。为增强模型对程序数据流的理解，该方法首先在控制流图上执行可达定义分析(Reaching Definition Analysis)，提取变量传播信息并编码为结构向量，作为GNN的输入。随后，将GNN学得的结构表示与大语言模型的隐层语义向量进行拼接，在前向传播中实现语义与结构信息的联合建模。该融合架构在捕获程序深层语义与数据依赖方面表现突出，显著提升了复杂漏洞的检测准确率。

5.1.3 大规模语言模型的训练方法

大规模语言模型的参数通常达到百亿量级，全量微调不仅耗时长、计算资源消耗巨大，还带来较高的经济成本。因此，其训练策略与轻量级Transformer模型存在显著差异。在微调方式的选择上，低秩适配(Low-Rank Adaptation，LoRA)已成为主流方案[160]。该方法在Transformer架构的各层引入可训练的秩分解矩阵，仅调整其中少量参数而冻结主干网络，从而在保证模型性能的前提下显著降低训练成本。Shestov等人[134]在实践中利用LoRA将训练参数规模从130亿降至2500万，大幅提高了训练效率。在此基础上，Yusuf等人[135]结合梯度检查点(gradient checkpointing)技术，通过在前向传播阶段仅保留部分激活值并在反向传播时动态重计算其余部分，进一步减少了显存占用。Yang等人[136]则在训练过程中引入十折交叉验证，以减轻模型在特定任务上的过拟合风险。此外，Wen等人[146]提出了课程偏好优化(Curriculum Preference Optimization)策略，该策略首先在验证集上评估模型在不同漏洞类型上的识别准确率，然后根据结果动态调整训练样本的选择概率，使模型优先学习识别准确率较低的类别，从而实现对模型弱项的针对性优化。这些方法从不同层面提升了大模型在漏洞检测任务中的训练效率与泛化性能，为大规模预训练模型的高效适配提供了有效途径。

5.1.4 大规模语言模型的损失函数

在损失函数设计方面，漏洞检测任务通常可抽象为下一个词预测任务或分类任务两种形式。在前者中，模型根据输入序列最后一个词的表示预测下一个词的概率分布，通过softmax获取预测标签，并与真实标签计算交叉熵损失。例如，Shestov等人[134]对传统“下一词预测”损失进行了改进，并成功将其应用于漏洞检测任务中的大模型微调。

在分类任务场景中，因大模型通常采用编码器或解码器架构，需在模型输出端添加线性分类层，将程序表示映射为二分类标签(‘0’或‘1’)，并计算交叉熵损失。设函数样本的真实标签为y_i(0表示无漏洞，1表示有漏洞)，输入为f_i，模型预测漏洞概率为$P_\theta(y_i|f_i)$，则函数级交叉熵损失定义为

$$L(\theta)=\sum_{i=1}^{N}-\log P_\theta(y_i|f_i) \tag{14}$$

其中，N为函数样本数，最小化该损失可使模型在函数级别准确判断代码样本是否存在漏洞。为缓解数据集类别不平衡问题，Ding等人[161]在交叉熵中对稀有类(漏洞样本)引入更高权重，以提高模型对少数类的敏感性。

此外，部分研究开始探索多任务指令微调策略，

以同时提升多类漏洞检测任务的表现[136,144]，这一策略在轻量级Transformer模型中较为少见。例如，Yang等人[136]提出的MSIVD方法将每个漏洞样本构造成多轮对话，分别引导模型完成漏洞标签预测、漏洞类别识别和漏洞成因解释等任务，并在CodeLlama-13B Instruct模型上通过联合损失函数实现多任务协同训练。该损失函数定义为

$$L(\theta)=\min_{\theta}\frac{1}{N}\sum_{i=1}^{N}\frac{\sum_{j=1}^{M_i}\sum_{k=1}^{T_{ij}}-\log\left(p_0(t_{ijk})\right)}{\sum_{j=1}^{M_i}T_{ij}} \tag{15}$$

其中，N为任务数量，M_i为第i个任务的样本数，T_{ij}为第i个任务的第j个样本参与损失函数的有效词元数，t_{ijk}是第i个任务的第j个样本的第k个词元。通过多任务联合优化，模型能够同时学习漏洞检测、分类与解释等关联任务之间的潜在联系，从而增强其对漏洞语义和成因的整体理解能力。

5.1.5　大规模语言模型的性能优势

大规模语言模型因其更深的结构和更大的参数规模，在漏洞检测中表现出显著优势。多项研究[134,136]的实验证明，基于StarCoder或CodeLlama等大模型的检测方法在准确率、F1值和泛化性能上均优于CodeBERT、CodeT5等轻量级模型。这一性能差异主要源于大模型更强的语义与结构建模能力，能够捕捉控制流与数据流之间的复杂依赖关系，识别由多语句或跨函数交互引发的漏洞。此外，大模型在预训练阶段接触到更丰富的代码语料，使其具备更强的迁移能力和应对未见样本的鲁棒性。

5.2　基于提示工程的源代码漏洞检测方法

自通用多模态大模型发布以来，其在复杂推理与问答任务中的卓越表现引发了学术界与工业界的广泛关注。近年来，部分研究开始探索利用ChatGPT等商业大模型执行软件漏洞检测任务。Gao等人[30]提出漏洞评估基准VulBench，系统对比了16种主流大规模语言模型及6种基于深度学习或静态分析的源代码漏洞检测方法，结果表明大模型在该领域具有较大潜力。然而，已有多项研究指出，直接调用ChatGPT等商用模型执行漏洞检测仍存在性能不稳定、上下文理解不足等问题[162-167]。受限于商业模型的闭源特性，无法通过微调进行任务定制，目前研究者普遍采用提示工程（Prompt Engineering）来提升其漏洞检测能力。根据提示构造的策略差异，现有方法可分为三类：（1）简单提示增强方法；（2）基于上下文的提示构造方法；（3）基于检索增强生成（Retrieval-Augmented Generation，RAG）的提示构造方法。多数研究并非仅采用单一策略，而是将多种提示优化技术组合使用，以提升模型稳定性与检测准确度。图9展示了基于提示工程的大模型源代码漏洞检测总体流程。

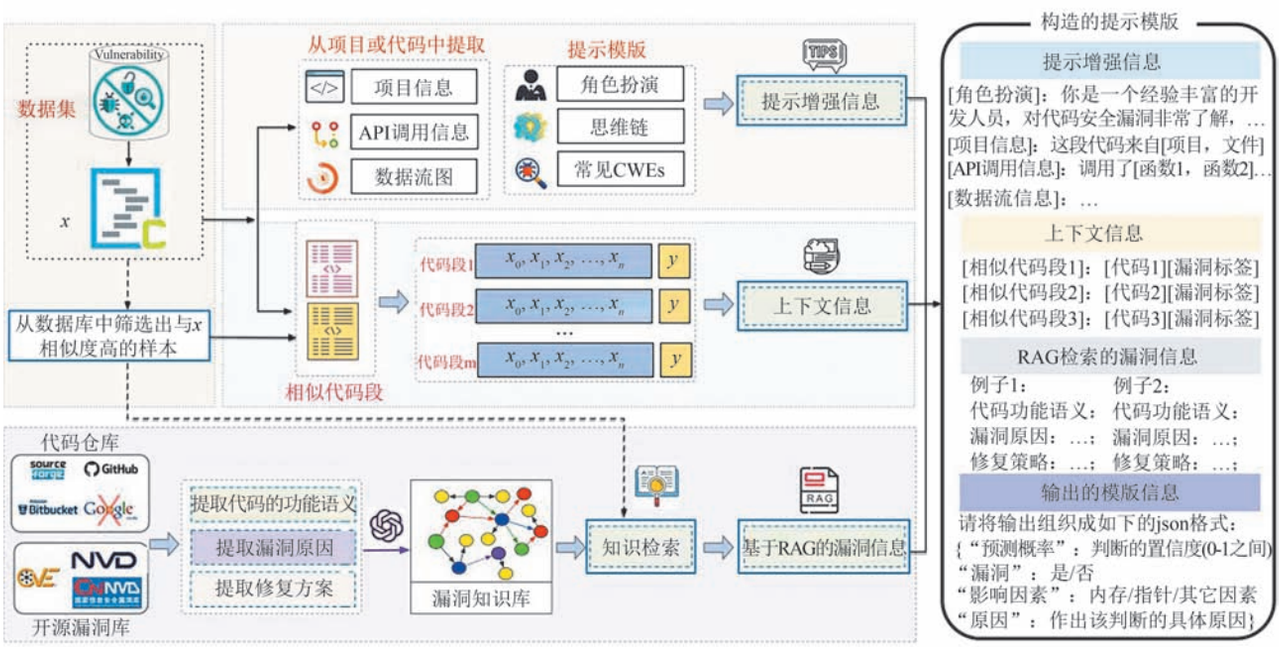

图9　基于大模型提示工程的源代码漏洞检测方法的总体流程

5.2.1 简单提示增强方法

简单提示增强方法通过优化提示语句本身，引导大模型在漏洞检测任务中进行更有效的推理。常见的策略包括角色扮演（Role Play）、思维链推理（Chain-of-Thought，CoT）及提示激励语设计等。

Zhou等人[138]在提示中引入角色扮演机制，向模型提供项目背景、常见漏洞类型及代码语境等辅助信息，从而显著提升了GPT-3.5与GPT-4的检测精度。Bae等人[168]在提示首行加入“若给出更优解决方案将获得奖励”的激励语，并显式要求模型按步骤推理，使GPT-4o与Claude-3.5 Sonnet在漏洞识别任务上均表现出明显性能提升。Zhang等人[139]指出，ChatGPT对代码数据流与API调用顺序的理解能力有限，因此提出在提示中加入从目标代码中提取的数据流图和API调用序列，以增强模型的程序语义感知能力。Mao等人[149]进一步提出多角色共识机制，让大规模语言模型分别扮演开发者与测试人员，以模拟真实代码审查流程。模型在多轮对话中通过角色间讨论达成漏洞判定共识，有效减少了单一模型决策偏差。总体而言，简单提示增强方法通过在提示层面提供语义引导或有限的辅助信息，能够在一定程度上提升检测性能，但其对大模型的推理潜能利用仍较有限，通常作为其他提示策略的辅助模块使用。

5.2.2 基于上下文的提示构造方法

当大模型参数规模超过特定阈值后，其具备较强的上下文学习能力。基于上下文的提示构造方法利用这一特性，通过在提示中引入相似参考代码或漏洞案例，有效提升了模型的识别准确率。

Liu等人[140]提出基于BMF25与TF-IDF的相似样本检索策略，从训练集中选取最相似的代码片段及其漏洞标签，生成包含分析性描述的上下文提示，并与目标代码一并输入GPT-3.5-turbo，以辅助其进行漏洞预测。Lu等人[147]设计了GRACE框架，在上下文信息中同时引入图结构特征以增强语义表达。该方法首先结合语义、语法与词汇相似度，从训练集选出与目标代码最相似的片段，并利用Joern工具生成代码属性图（CPG），以融合控制流图、程序依赖图与抽象语法树信息。随后，将检索到的相似样本及其CPG表征一同加入提示，引导模型在代码结构与语义双重层面进行漏洞判断。

与简单提示增强方法相比，基于上下文的提示构造能够显著提升模型对复杂程序语义和上下文依赖的理解能力。该类方法通常与角色扮演、思维链等提示增强技术联合使用，以进一步提高提示信息的有效性与模型的上下文推理能力。

5.2.3 基于检索增强生成的提示构造方法

检索增强生成（RAG）通过将信息检索技术与大语言模型的生成能力相结合，构建了一种知识增强型生成框架。该框架在生成答案时可动态引用外部知识库的信息，从而兼具可解释性与可定制性，广泛应用于问答系统、智能助手及文档生成等任务中。近年来，研究者开始将RAG框架引入漏洞检测领域，以缓解大模型知识局限和幻觉问题。

该类方法的核心思想是构建领域专属漏洞知识库，并在检测阶段检索与目标代码相关的知识，以增强提示内容的语义充分性和准确性。Du等人[148]提出了一种基于RAG的漏洞检测框架VulRAG，通过引入外部知识有效提升了大规模语言模型的检测性能。该框架首先利用大规模语言模型对公开CVE实例进行解析，自动构建包含函数功能描述、漏洞成因及修复方案等多维信息的知识库；随后，在检测阶段根据目标代码的语义特征检索相关的知识条目，并将其嵌入提示中，引导GPT-4生成推理性分析。该机制显著增强了模型的上下文理解能力与漏洞定位精度。

在此基础上，Yang等人[141]提出了基于深度学习的提示增强框架DLAP。该框架融合上下文提示与思维链推理机制：一方面，利用相似度匹配算法从训练数据库中检索与目标函数最相似的代码片段，作为辅助上下文；另一方面，基于CWE（Common Weakness Enumeration）体系构建思维链模板库，并结合静态分析工具和深度模型自动选择最合适的推理链模板。最终，DLAP将相似代码段与推理模板联合编码为复合提示，显著提升了模型的逻辑推理深度与漏洞识别能力。

总体而言，RAG机制通过结合大语言模型的生成能力与外部知识的可控检索，提升了模型在应对新型和复杂漏洞模式时的泛化与解释能力。未来研究可进一步探索多源知识融合与动态知识更新机制，以构建更完备的漏洞知识增强框架。

5.2.4 基于自注意力机制的源代码漏洞检测方法

自注意力机制作为Transformer架构的核心组件，具备捕获长距离依赖与全局语义关系的能力。近年来，研究者尝试直接利用大语言模型内部的自注意力结构进行漏洞定位，从而提升检测的可解释性。Li等人[150]提出了基于自注意力机制的漏洞定位框架LOVA，该方法通过提示词引导模型聚焦于

特定代码行，并利用注意力权重变化分析潜在漏洞位置。具体而言，LOVA为每个代码样本生成两种输入：一是保持完整结构的基准版本，二是对特定代码行进行显式强调（例如“请重点关注第3行”）的版本。模型分别处理两种输入并输出注意力矩阵，通过计算两者差异得到注意力增量矩阵，以量化各行代码对模型决策的重要性。随后，系统利用Bi-LSTM对各行的注意力变化进行打分，得分高于阈值的行被判定为漏洞行。实验结果表明，该方法在Llama-3.1-8B-Instruct模型上实现了多语言代码场景下的显著性能提升。

5.3　结合大模型与静态检测技术的源代码漏洞检测方法

部分研究并未直接使用大模型进行漏洞检测，而是将其作为静态检测方法的辅助工具，以提升传统方法的性能。Sun等人[151]提出了一种将GPT与静态分析相结合的智能合约漏洞检测方法GPTScan。该方法将每个漏洞类型分解为代码级别的场景和属性，其中场景描述了逻辑漏洞可能发生的情境，而属性则解释了该情境下漏洞代码的具体操作。GPTScan使用GPT将候选函数与预定义场景及属性进行匹配，并进一步识别关键变量与语句，最终由静态分析模块确认漏洞。Wang等人[152]提出了一种基于大规模语言模型的静态资源泄漏检测框架INFERROI。该框架首先利用GPT-4为给定代码片段生成包含资源获取、资源释放以及资源可达性的辅助信息。然后INFERROI基于推断的资源获取和释放API识别潜在的泄露风险路径，并基于推断的可达性验证对不可达资源的泄露风险路径进行剪枝，从而减少误报。实验结果表明，结合了大规模语言模型辅助的INFERROI性能超越了以往先进的静态检测工具。Deng等人[153]提出了结合大规模语言模型与模糊测试的检测框架TitanFuzz，该框架利用大规模语言模型为深度学习代码库的模糊测试生成测试样本，解决了传统模糊测试难以处理张量的难题。

5.4　基于大模型的多智能体源代码漏洞检测方法

为克服提示工程方法依赖单一提示策略的局限，近年来基于多智能体系统（Multi-Agent Systems，MAS）的漏洞检测逐渐成为研究热点。该方法通过引入具备不同角色与职责的智能体（如开发者、审查者、测试者），并利用其交互、讨论与协同决策机制，从多视角分析代码，有效降低单一模型的误判风险，显著提升检测的准确性与鲁棒性。

Yildiz等人[155]评估了基于ReAct（Reasoning and Acting）范式的智能体在代码仓库级漏洞检测任务中的表现。不同于大规模语言模型的静态推理过程，ReAct智能体通过“思考-行动-观察”的循环机制，可按需动态检索调用关系等跨过程上下文信息。实验证明，该机制在处理多跳函数调用导致的复杂漏洞（如空指针解引用）时，较标准大规模语言模型能更准确地刻画漏洞代码与修复版本之间的差异。在智能合约场景中，Wei等人[156]提出了面向审计任务的LLM-SmartAudit框架。该框架模拟专业审计团队的工作流程，设定了项目经理、审计师等四类角色，并实现任务的全流程分工与协同。为兼顾检测的广度与深度，作者设计了“双重分析模式”：一方面利用通用知识对合约进行广泛扫描，另一方面结合预设思维模板对特定漏洞进行深入挖掘。此外，该方法还引入角色互换与共识机制，通过智能体间的交叉验证显著降低误报率，尤其在处理复杂逻辑漏洞时展现出更强的判断能力。此外，Tang等人[154]提出的CodeAgent框架通过模拟真实代码审查流程，结合细粒度的角色划分与协作机制，进一步增强了大模型在漏洞挖掘任务中的实际表现。

总体而言，多智能体驱动的漏洞检测方法通过引入角色分工、协同对话、工具调用与动态推理等机制，有效缓解了单一大语言模型在上下文理解和跨语句或跨函数信息推理方面的能力限制。未来研究可进一步探索智能体间通信效率的优化策略，以及如何将静态分析工具更深度地集成至智能体的闭环推理流程中，从而构建更加精准、高效且具可解释性的自动化安全审计系统。

5.5　小　结

本章系统回顾了基于大规模语言模型的源代码漏洞检测技术研究进展，主要分为四类：一是通过微调预训练模型以提升其在漏洞检测任务上的性能；二是基于提示工程利用模型的上下文学习能力实现即用型检测；三是将大模型的深层语义理解与传统静态分析技术相结合，发挥各自优势；四是利用多智能体系统模拟人类专家在代码审查中的协作与多视角验证过程。总体来看，基于大规模语言模型的方法凭借预训练获得的深层语义表征和自注意力机制所带来的长距离依赖建模能力，能够有效捕捉跨函数、跨文件等复杂漏洞模式，显著增强了检测效果。这些技术优势使得基于大规模语言模型的源代码漏洞检测成为当前研究热点，并为新一代智能化漏洞检测系统的构建提供了重要支撑。

6 大模型时代的源代码漏洞检测评估和数据集

6.1 评估指标

漏洞检测通常可被建模为一个二分类问题，即判断输入代码是否包含漏洞。对于基于大语言模型的源代码漏洞检测方法，其性能评估体系基本沿用传统机器学习模型的评价标准，主要包括准确率（Accuracy）、精确率（Precision）、召回率（Recall）和F1值（F1 Score）四个核心指标。

准确率反映模型整体预测的正确性，其计算公式为

$$\text{Accuracy}=\frac{TP+TN}{TP+TN+FP+FN} \tag{16}$$

精确率衡量模型预测为“有漏洞”的样本中真实为漏洞的比例：

$$\text{Precision}=\frac{TP}{TP+FP} \tag{17}$$

召回率表示模型成功识别出的真实漏洞样本占全部漏洞样本的比例：

$$\text{Recall}=\frac{TP}{TP+FN} \tag{18}$$

F1值是精确率与召回率的调和平均，用于平衡两者的权重关系：

$$\text{F1 Score}=\frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}} \tag{19}$$

其中，真阳性（True Positive，TP）表示真实存在漏洞且被正确识别为漏洞的样本；真阴性（True Negative，TN）表示无漏洞样本被正确判断为安全；假阳性（False Positive，FP）表示安全代码被误报为漏洞；假阴性（False Negative，FN）则表示漏洞代码未被正确识别。

6.2 常用的漏洞数据集

当前漏洞检测领域常用的数据集主要分为人工合成数据集（Synthetic Dataset）和真实项目数据集（Real-world Dataset）。前者依据已知漏洞模式人工构建，后者则源自真实软件项目的漏洞样本。依据构建方式的差异，现有数据集可进一步分为三类：基于规则的人工合成数据集、人工标注数据集以及自动化标注数据集。表4总结了当前具有代表性的数据集及其关键特性。

表4 主流漏洞检测数据集及其特征分析

数据集 DataSet	语言 Language	大小 Size	漏洞/无漏洞 Vul/Non-Vul	类型 Type	相关论文 Relevent papers
Juliet[169]	C/C++	117,220	58,610/58,610	人工合成	[112,144]
SARD	C/C++/Java等	450,000	-	人工合成	[122,139]
SVEN[170]	C/C++	1,606	803/803	手动标注	[145,146]
Devign[58]	C/C++/Java	22,361	10,067/12,294	手动标注	[121,122,126,127,137,140,147,144]
Ponta等人[171]	Java	624	-	手动标注	[134]
Defects4J[172]	Java	357	-	手动标注	[137]
BugsInPy[173]	Python	493	-	手动标注	[137]
BigVul[174]	C/C++	188,636	10,900/177,736	自动标注	[111,113,125,126,10,136,147,150,144]
VulDeePecker[175]	C/C++	61,638	17,725/43,913	自动标注	[121]
SySeVR[85]	C/C++	15,591	14,780/811	自动标注	[139]
Reveal[176]	C/C++	18,169	1,664/16,505	自动标注	[121,122,126,127,147,144]
CrossVul[177]	C/C++/Python等	27,476	13,738/13,738	自动标注	[177]
CVEfixes[178]	C/C++	19,576	8,223/11,347	自动标注	[129,134,135,150,144]
DiverseVul[179]	C/C++	349,437	18,945/330,492	自动标注	[179,144]
PairVul[148]	C/C++	4,314	-	自动标注	[148]
D2A[180]	C/C++	1,295,623	18,653/1,276,970	自动标注	[121,112,127]
PrimeVul[161]	C/C++	235,768	6,968/228,800	自动标注	[161,145,146]

6.2.1 人工合成的数据集

早期研究者基于已知漏洞模式，手工构造标准化测试样本，用于评估漏洞检测工具的准确性和稳定性。典型代表是由NIST（National Institute of Standards and Technology）发布的SATE IV Juliet数据集[169]，其包含约6万个合成测试用例，覆盖多种常见安全漏洞类型，主要以C语言编写。在此基础上，NIST进一步推出了SARD（Software Assurance

Reference Dataset)[①]，扩展支持C/C++、Java和Python等多语言环境，并纳入更广泛的漏洞类型。

尽管人工合成数据集在漏洞的可控性、覆盖范围和复现能力方面具有一定优势，但也存在明显局限。这类数据集通常仅聚焦于特定漏洞模式，难以体现真实项目中的语义复杂性与环境依赖。在实际场景中，漏洞往往不仅由代码错误引发，还与项目上下文、运行环境以及输入数据等因素密切相关。然而，这些上下文因素和环境约束难以在合成场景中准确复现，导致人工数据集与真实漏洞分布存在偏差。

6.2.2　手工标注的数据集

为缩小人工合成数据与真实漏洞分布之间的差距，研究者开始从真实软件项目中提取漏洞实例并进行人工标注。这类数据集通常依赖公开漏洞数据库(如NVD)及对应的版本控制记录，通过人工审查确保标注质量。例如，He等人[170]在SVEN框架中通过人工验证从多源数据集中筛选高质量样本，最终构建了包含1,606个样本、覆盖9类CWE漏洞的C/C++与Python混合数据集。Zhou等人[58]构建的Devign数据集通过多阶段过滤与交叉验证流程，结合安全关键字筛选与专家双轮人工复核(约600工时)，获得了高置信度的函数级漏洞样本。Ponta等人[171]则通过持续监测NVD及50余个项目特定安全追踪网站，历时4年人工收集并验证了205个Java项目的624个漏洞实例。此外，Just等人[172]发布的Defects4J数据集从5个开源Java项目中提取了357个漏洞实例，并对漏洞版本与修复版本的代码差异进行了人工审查，以确保样本的真实性与一致性。在Python领域，Widyasari等人[173]构建的BugsInPy数据集收录了17个项目的493个漏洞样本，同样采用人工复核以保障标注质量。

尽管人工标注数据集因其具备较高的准确性与置信度被视为模型评估的黄金标准，但其构建成本高、效率低，即便是经验丰富的专家也难以快速完成大规模标注。因此，该类数据集更适用于高精度、小规模的评测任务，而非大规模模型训练。

6.2.3　自动标注的数据集

为缓解人工标注成本高、规模受限等问题，研究者提出多种自动化标注机制，通常依托开源漏洞仓库或问题追踪系统自动识别漏洞修复提交，并基于代码差异生成漏洞标签。

典型代表如BigVul数据集[174]，其从CVE Details网站爬取与漏洞相关的GitHub项目信息，通过提交记录筛选出漏洞修复版本，再比较修复前后函数的差异，将修改前的函数标注为“有漏洞”，修改后的函数标注为“无漏洞”。类似地，VulDeePecker[175]、SySeVR[85]、ReVeal[176]、CrossVul[177]、CVEfixes[178]和DiverseVul[179]等数据集也采用了基于提交差异的自动化标注流程。

然而，早期方法多依赖关键字过滤或简单文本匹配来识别漏洞相关提交，容易产生大量误报。为提升标注准确率，D2A数据集[180]引入基于静态分析的差分机制：若工具在修复前版本中检测到漏洞，而在修复后版本中未检测到，则将该提交视为漏洞修复提交。尽管该方法有效缓解误报问题，但仍受限于静态分析工具的本身准确性。

此外，函数级误标是自动标注方法中的另一挑战。实际修复中，开发者常会因多重目的或意图而同时修改多个相关函数，导致非漏洞函数被误标为“有漏洞”。针对这一问题，PrimeVul数据集[161]采用更严格的筛选策略，仅当安全相关提交涉及单一函数的修改时，才将该函数标记为漏洞函数。该策略显著提升了标注质量，人工抽样验证表明其标签准确率可达86%。

自动标注方法在规模和效率上具备明显优势，是当前大模型训练的主要数据来源。然而，如何在保持大规模自动构建能力的同时，进一步提高标注的精度与可信度，仍是值得深入探索的重要方向。

6.3　基于大模型的源代码漏洞检测方法的常用数据集分析

从表4可知，BigVul[174]是当前基于大语言模型的源代码漏洞检测研究中使用最广泛的基准数据集，其优势主要体现在以下四个方面：(1)BigVul源自真实开源项目，样本数量庞大，覆盖的CWE类型丰富，适用于多种检测任务[174]；(2)相较于多数仅提供函数级标签的数据集，BigVul额外提供行级漏洞标注，为研究细粒度检测方法提供了实验支撑；(3)BigVul同时收录漏洞函数及其修复版本，为漏洞修复与补丁生成等任务提供了基础；(4)多个代表性方法(如LineVul[111])均基于BigVul展开实验，使其逐渐成为后续研究的默认评测基准。

除BigVul外，Devign[58]、Reveal[176]、CVE- fixes[178]和D2A[180]等数据集也被广泛采用。这些数据集多

① NIST Software Assurance Reference Dataset(SARD), https://www.nist.gov/itl/ssd/software-quality-group/samate 2026, 1,6

构建自真实项目，样本规模大、漏洞类型多样，具有较高的代表性与通用性。整体来看，基于真实项目自动标注的大规模数据集已成为当前大模型驱动的漏洞检测研究的主流选择。与此相比，人工合成数据集难以反映真实漏洞的语义复杂性，而人工标注数据集虽质量较高，但规模有限、类型覆盖不足，不利于模型学习通用漏洞模式。因此，如何采用自动标注与人工复核相结合的方法构建高质量真实项目数据集，将是未来研究的重要方向。

7 基于LLM的源代码漏洞检测方法的性能评估与对比

虽然LLM在程序理解与生成方面展现出强大的能力，但其在漏洞检测任务中的实际效果仍有待进一步提升。本文参照以往研究思路，从三个方面系统比较LLM的检测能力：与传统静态分析工具（Static Analysis Tools，SATs）的性能差异、与其他深度学习方法的对比表现，以及不同LLM在该任务中的表现差异。

7.1 基于LLM的方法与静态分析工具对比

目前，大量研究表明，LLM在漏洞定位精度和漏洞类型覆盖范围方面，已逐步超越传统静态分析工具，如Checkmarx、FlawFinder、RATS、CodeQL和Semgrep等。图10对比了典型静态工具（Checkmarx、FlawFinder、RATS）与基于LLM的方法（LineVul、SvulD、StagedVulBERT）在大规模漏洞数据集BigVul上的性能。结果显示，表现最优的轻量级Transformer模型StagedVulBERT在F1-Score上相较RATS提升约130%，显著优于传统静态分析工具，表明基于LLM的方法在语义理解与泛化能力方面具有明显优势。这一结论已在多项实证研究中得到进一步验证。例如，Noever等人[181]的实验结果表明，GPT-4可检测的漏洞数量约为Fortify①和Snyk②的四倍。同时，Mohajer等人[182]指出，ChatGPT在检测Null Dereference和Resource Leak等漏洞时，相较Infer③精度更高且表现更稳定。

此外，有研究发现，结合LLM与静态分析工具可进一步提升漏洞检测效果。Li等人[183]的研究表明，将LLM与CodeQL联合使用比单独依赖CodeQL可多识别约35%的漏洞，体现出二者的互补性。这主要是因为传统SATs（如Cppcheck、FlawFinder、Semgrep）依赖预设规则或语法模式，覆盖范围有限，难以发现规则库之外的漏洞[184]，而LLM具备更强的语义理解和上下文建模能力，能够

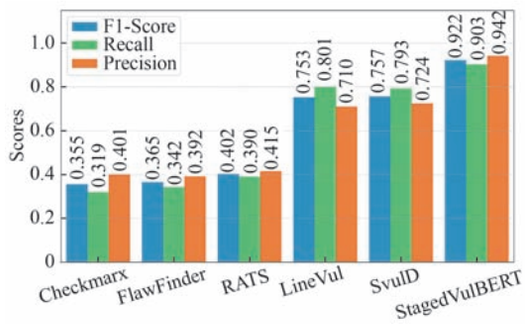

图10 基于大语言模型的源代码漏洞检测方法与静态分析工具的性能对比

在一定程度上弥补这一不足。但与此同时，LLMs的误报率往往高于传统工具。Purba等人[162]发现，尽管ChatGPT能识别更多漏洞类型，但在部分场景下，其误报率显著高于FlawFinder或Checkmarx等工具。类似地，Zhou等人[185]对比了15种SAT与12种主流LLM，指出虽然SAT的平均检出率仅为44.4%，但具备更低的误报率和更高的可解释性。

总体来看，LLM与SATs在漏洞检测中各有优势。LLM依托强大的语义理解能力，擅长发现复杂逻辑漏洞和规则库未定义的边缘漏洞；而SAT的规则驱动机制更具可控性和可解释性，适合高可靠性场景。未来，融合两者优势的混合检测方案，将是值得深入探索的重要方向。

7.2 基于LLM的方法与深度学习方法对比

除了与传统静态分析工具的对比，LLM也常用于与其他深度学习模型（如Devign[58]、IVDetect[109]和Reveal[176]等）进行性能对比。图11展示了典型基于深度学习的源代码漏洞检测方法（Devign、Reveal、IVDetect、AMPLE）与基于大语言模型的方法（如LineVul、SvulD）在BigVul数据集上的性能对比结果。由图可见，基于LLM的方法在F1 Score上较传统深度学习模型平均提升超过200%，表明LLM能够更有效地捕获深层代码语义信息及跨函数的上下文依赖关系。这一趋势已在多项研究中得到验证。例如，Fu等人[111]指出，基于Transformer架构的模型能够更有效地捕获代码的深层语义与语法结构特征，其性能显著优于早期依赖图表示的代码分析方法。Wu等人[120]也在智能合约场景中发现，大模型在准确率与泛化性方面均明显超越传统深度

① Fortify，https://www.fortify.net.cn/ 2026,1,6
② Snyk，https://snyk.io/ 2026,1,6
③ Infer，https://fbinfer.com/ 2026,1,6

学习模型。但LLM并非在所有场景下都优于传统模型。Ni等人[186]指出，LLM对提示设计高度敏感，若提示质量不足，其检测稳定性可能不如序列或图模型。Fu等人[165]也发现，直接使用通用LLM进行漏洞检测往往难以达到预期效果。

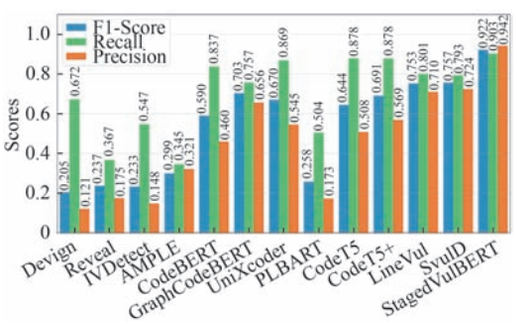

图11　基于大语言模型的源代码漏洞检测方法与传统深度学习方法的性能对比

总体而言，当数据分布、任务场景与模型训练目标高度契合时，LLM在漏洞检测任务中表现尤为出色。然而，其性能仍受到提示设计质量和上下文构造方式的显著影响，在实际应用中仍存在一定的不确定性。同时，相较于传统深度学习模型，LLM在训练与部署阶段的资源消耗和工程复杂度更高，仍需在性能与成本之间权衡优化。

7.3　不同LLM之间的源代码漏洞检测性能对比

近年来，不同LLMs在漏洞检测任务中的性能差异逐渐成为研究热点[30,151]。图12展示了多种LLM（如DeepSeek-Coder、CodeLlama、StarCoder等）在BigVul数据集上微调后的漏洞检测性能对比结果。实验结果表明，不同模型之间的检测效果存在显著差异。例如，DeepSeek-Coder的F1 Score较Mistral提升约30%，说明模型在架构设计与训练策略上的差异会显著影响其漏洞检测能力。除了在上述开源模型间进行漏洞检测性能对比外，部分研究还进一步比较了开源模型与闭源模型的检测性能。例如，Gao等人[30]的研究表明，GPT-3.5和GPT-4在准确率与召回率上显著优于Llama 2和Vicuna 13B等开源模型，尤其在应对复杂逻辑漏洞和非典型安全场景时更具优势。这主要归因于GPT系列在训练规模、数据多样性以及后处理与推理机制等方面的综合优势[186]。然而，并非所有开源模型的表现都逊于GPT系列。针对特定领域或漏洞类型，经过定制化微调的开源模型同样能够取得具有竞争力的结果。例如，Tamberg等人[187]比较了CodeLlama与GPT-4在检测率和误报率方面的性能，结果表明，在针对特定数据集进行指令微调后，CodeLlama在部分漏洞类型上的检测效果可与GPT-4持平，甚至略有超越。

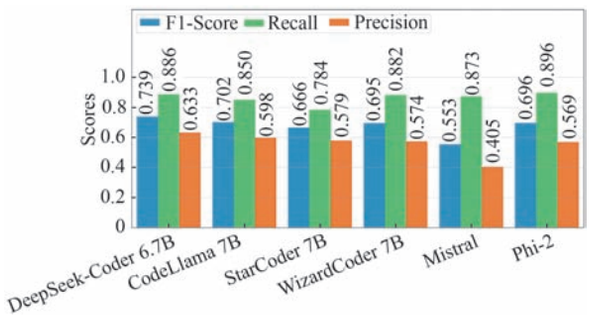

图12　基于微调的大语言模型漏洞检测性能对比

此外，不同商业LLM在漏洞检测任务中的性能差异也受到广泛关注。图13展示了当前主流商业模型在PrimeVul数据集上的漏洞检测性能对比结果。由于调用不同商业模型进行漏洞检测的计算与经济成本较高，本文直接引用了香港中文大学吕荣聪教授团队的实验数据①。结果显示，不同模型在精确率与召回率等关键指标上存在显著差异。例如，Claude-4-Sonnet的F1 Score超过70，而Qwen-2.5由于召回率偏低，其F1 Score仅约为20。这表明模型的结构设计、预训练语料的多样性与覆盖范围，以及训练方法等因素，在程序语义建模和漏洞识别能力方面具有重要影响。

综上，闭源模型（如GPT系列）凭借庞大的参数规模、高质量训练语料和精细化指令对齐，在通用漏洞检测任务中表现出较强的鲁棒性与泛化能力，尤其在复杂逻辑漏洞和跨语言检测场景中优势更为显著。相较之下，开源模型（如Llama、CodeLlama）虽然整体性能略低，但在透明度、可定制性以及特定任务的再训练与优化方面具有独特优势。通过针对漏洞数据集的进一步微调与领域适配，开源LLM在特定任务上的性能有潜力达到甚至超越闭源模型，为构建高效、安全、可控的漏洞检测系统提供了新的研究方向。

7.4　基于LLM的方法与传统方法的训练成本和部署难度对比

基于大语言模型的源代码漏洞检测方法在性能提升的同时，也带来了高昂的训练成本。除了巨大的预训练开销外，即便仅在微调阶段，其计算成本亦不可忽视。例如，在漏洞数据集BigVul上，轻量级Transformer模型LineVul在单张NVIDIA RTX

① Code TREAT，https://code-treat.vercel.app 2026,1,6

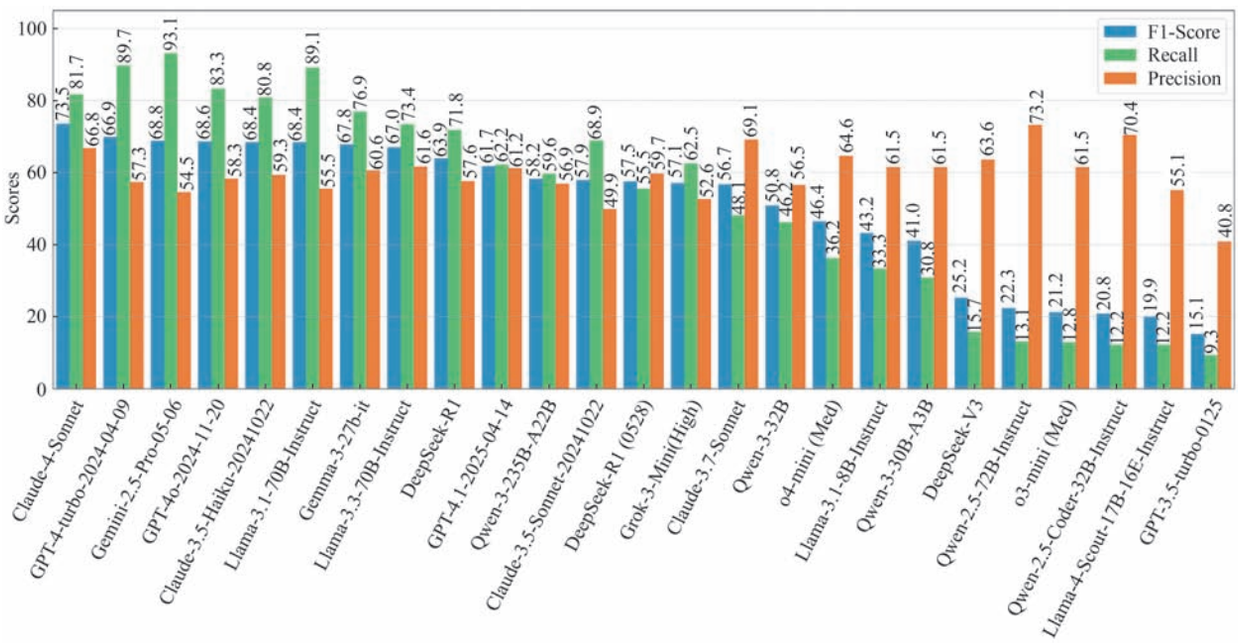

图13 商业大语言模型在漏洞检测任务中的性能对比

3090显卡上微调约7小时[111]，而StagedVulBERT[10]在A6000上的微调时间约为10小时。相比之下，传统深度学习方法在同一数据集上的训练成本明显更低，其中Devign和Reveal在单张3090显卡上的训练时间通常不超过1小时[186]。在漏洞数据集上进行训练或微调之前，LLM通常还需在更大规模的通用语料上完成预训练。例如，StagedVulBERT的预训练过程在五张A100显卡上总计耗时约130小时[10]，若进一步进行复杂的超参数搜索，计算成本将成倍增加。而传统深度学习模型通常无需大规模预训练，因此在整体资源消耗上更具优势。

在部署实践中，传统静态分析工具与LLM面临的挑战各不相同。传统静态分析工具的有效部署高度依赖复杂的环境配置过程，包括编译环境搭建、依赖库安装以及针对不同项目特性的扫描规则定制。相比之下，LLM的部署方式主要包括服务化部署与私有化部署两种模式。服务化部署依赖云端API（如OpenAI、Google、Anthropic等）完成漏洞检测，具备部署简便、资源门槛低等优势，但存在调用成本高、源代码需上传至第三方平台所引发的隐私与合规风险。私有化部署则是在本地运行经过微调的开源模型（如CodeLlama、DeepSeek等），有助于保障数据安全与使用控制权，但对本地算力与运维能力提出了更高要求。该方案需要企业具备GPU集群的运维与调度能力、模型推理优化技术，以及高可用推理服务的构建与维护能力。

7.5 影响LLM漏洞检测性能的因素

LLM在漏洞检测中表现突出，但其性能受多种因素影响。基于7.1-7.3节的分析，本研究将其主要归纳为以下三方面。首先，模型自身特性是最核心的基础因素，包括参数规模、训练数据质量与微调策略等。例如，GPT-4等大规模语言模型凭借参数规模与训练语料的优势，在通用漏洞检测任务中表现优异[30]；而CodeLlama等开源模型通过在安全领域数据集上进行有监督微调，能够在特定任务上达到甚至超越前者的水平[187]。这表明，模型的通用性与专用性可通过不同的训练与微调策略加以平衡。其次，提示工程对模型潜能的发挥具有关键作用。高质量提示有助于引导模型聚焦检测目标，提高准确性；反之，提示设计不当可能导致性能波动，甚至低于经过针对性优化的传统深度学习模型[165,186]。最后，性能评估方法显著影响结论可靠性。模型表现依赖于具体的评估框架与对比基线。与传统静态工具相比，LLM在识别复杂逻辑漏洞方面具备优势，但误报率高、解释性弱的问题仍有待解决[185]。因此，不同场景下难以存在统一最优方案，通常需结合LLM与静态分析工具以实现优势互补[183,188]。

除上述因素外，LLM在漏洞检测中的性能还受到其他多重因素的影响。首先，代码表征质量对模型检测效果具有关键影响。函数或变量命名不清晰、语义不一致将削弱LLM的理解能力[189]；而包含修复信息的注释或被混淆的代码则可能干扰模型判

断，导致误报率上升[190]。其次，模型的静态分析局限性也是一项制约因素。由于LLM无法执行代码，其难以检测依赖运行时信息才能暴露的漏洞，如内存访问错误或环境配置问题[191]。此外，大多数LLM仍受限于输入Token长度，当代码较长时，可能截断关键信息，影响检测的准确性[192]。

8　现实挑战与未来展望

本节从以下三个方面阐述当前面临的主要挑战：基于轻量级Transformer的源代码漏洞检测模型的局限性、基于大规模语言模型的源代码漏洞检测方法的现实挑战，以及两类方法的共性问题，并进一步分析了相应的解决思路与未来研究方向。上述挑战与解决方案的概览如图14所示。

8.1　基于轻量级Transformer的源代码漏洞检测模型的现实挑战与未来展望

8.1.1　上下文窗口限制

轻量级Transformer模型受限于较小的上下文窗口，难以覆盖跨文件的长距离依赖，导致复杂漏洞（如跨文件漏洞）漏报率上升。已有研究表明，上下文范围对漏洞检测性能具有显著影响[10]。以LineVul为例，其输入窗口限制为512个token，被认为是影响性能的关键瓶颈[111]。尽管StagedVulBERT[10]将该长度扩展至2048，以增强模型感知能力，但对超长代码仍需截断，可能造成语义信息丢失。

为缓解窗口限制问题，部分工作引入检索增强生成技术[148,193]，通过在项目中检索相关函数并填充至输入上下文，提高关键信息覆盖率。也有方法通过扩展位置编码等方式直接扩大模型的上下文处理能力[194-195]。除此之外，分层上下文建模提供了另一种高效的解决思路。该方法通过自底向上生成函数级摘要，再将其作为高层上下文输入，实现有限窗口下的全局信息整合。与此同时，将LLM与GNN相结合被视为一种更具前景的结构性突破方向。GNN能够在图结构中显式建模跨函数、跨模块的远程依赖关系，从而为LLM提供超出窗口范围的外部语义补充[58]。通过联合训练或特征对齐，模型可在有限输入长度下获取全局结构信息，从而有效缓解上下文截断导致的性能瓶颈。

8.1.2　代码结构信息编码不足

传统Transformer将源代码视为线性Token序列，难以有效捕捉控制流图（CFG）和程序依赖图（PDG）等关键结构信息[28]。线性化过程破坏了原有的拓扑关系和长距离依赖，限制了模型对程序执行逻辑与数据流的理解。

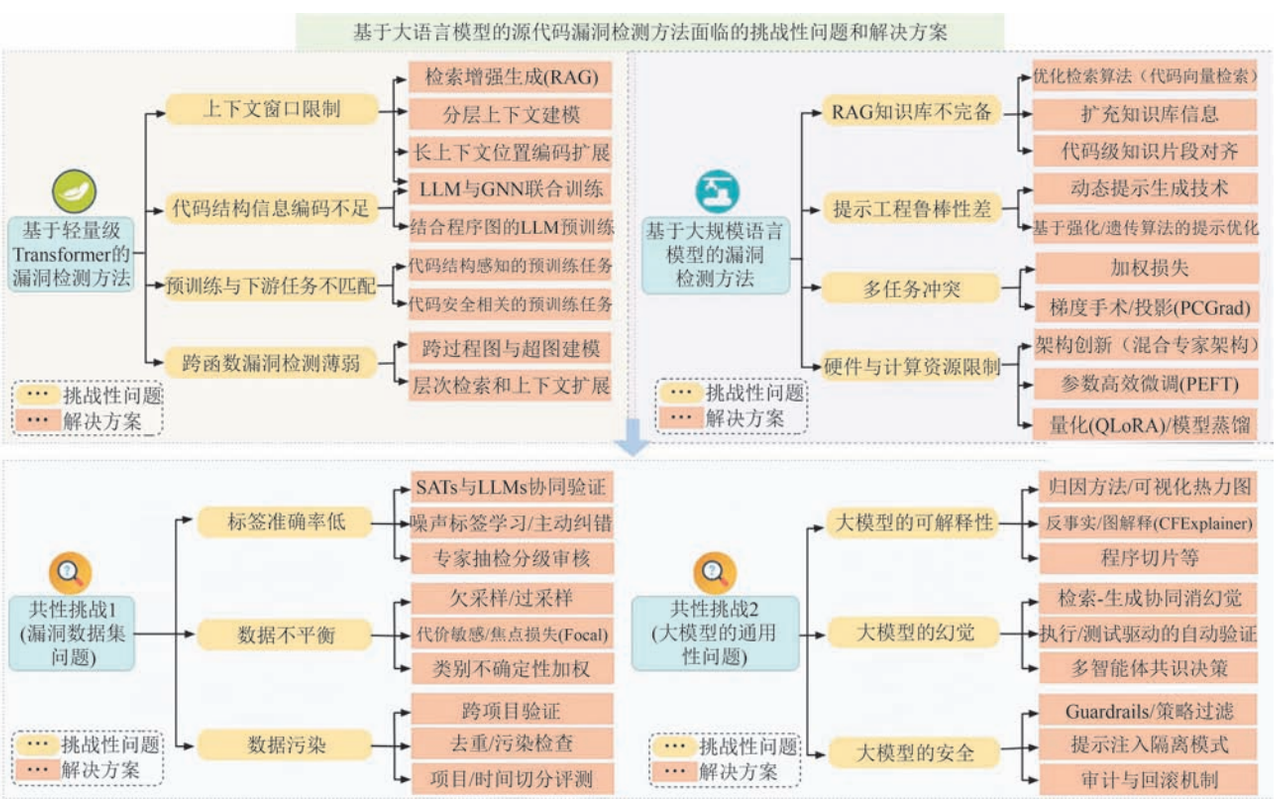

图14　基于大语言模型的源代码漏洞检测方法面临的挑战与潜在解决方向

为缓解该问题，部分研究尝试将GNN提取的结构嵌入与LLM的语义表示进行融合[112-113]，以构建更全面的代码表示。然而，GNN与Transformer的联合训练面临梯度对齐困难，导致优化过程不稳定；此外，现有实验大多仅在漏洞数据集（如BigVul）中的极少部分函数上进行验证，原因在于代码解析工具（如Joern）在大规模生成图结构数据过程中存在准确性不足与处理效率低下的瓶颈[125]。另一方向是探索结构感知预训练。通过在预训练阶段引入程序图信息[2]，设计如AST节点预测、数据流边预测、图文对比学习等任务，帮助模型同时掌握语言模式与结构语义。经此方式训练的LLM，其内部表示自然融合了结构信息，从而在下游漏洞检测任务中能够更有效地建模程序的逻辑依赖关系，进而提升检测的准确性与泛化性。

8.1.3 预训练任务与下游任务不匹配

现有的LLM一般采用通用预训练任务（如掩码语言模型）学习代码的通用表示，但此类任务与漏洞检测，尤其是语句级预测的目标存在明显偏差。StagedVulBERT[10]提出了掩码语句预测（MSP）任务，通过屏蔽整条语句，并引导模型根据上下文语句进行恢复，以学习语句间的逻辑结构与依赖关系。尽管MSP在一定程度上缓解了粒度不一致的问题，但其训练目标仍主要聚焦于结构语义，而非漏洞特征建模，两者在目标层面仍存在偏差。

为进一步缩小这一差距，未来研究可从两个方向设计更具针对性的预训练任务。一是构建结构感知任务，如标识符感知、跨模态对齐或结构边预测等[2,28]，以促使模型超越表层的词法与句法特征，深入建模变量间数据流动和代码块间控制依赖等深层逻辑关系。二是探索融入代码安全语义的任务。例如，通过模拟污点分析训练模型追踪危险数据流，或设计针对特定漏洞模式的对比学习任务，从而直接对齐漏洞检测目标，使模型在预训练阶段便具备识别不安全代码的能力。

8.1.4 跨函数漏洞检测薄弱

现有的大多数研究主要以单个函数为分析单元[111,196]，在识别跨函数甚至跨模块传播的漏洞时存在明显不足。尽管部分方法尝试在模型输入中补充目标函数的上下文信息以缓解该问题，但整体上仍缺乏系统化的跨过程分析机制。

未来研究可从两方面加以推进。一是融合跨过程图（Interprocedural Graph，IPG）与超图（Hypergraph）建模[197]，前者刻画函数间的调用与数据传递关系，后者则捕捉更细粒度的语义依赖（如多变量交互、复杂API调用），两者结合可构建更完整的程序结构表示，支撑跨函数漏洞检测。二是引入基于检索的上下文扩展，通过层次化检索方法，从代码库中高效定位与漏洞模式相关的代码片段，并利用上下文扩展机制对其进行重组与语义对齐，从而为模型提供必要的跨函数信息，使其具备捕捉复杂数据流漏洞的能力。

8.2 基于大规模语言模型的源代码漏洞检测方法的现实挑战与未来展望

8.2.1 提示工程鲁棒性差

在基于大规模语言模型的源代码漏洞检测方法中，提示工程是一项关键技术，但其效果高度依赖人工设计的模板，且对术语与句式极为敏感。轻微改动提示内容便可能导致模型输出大幅波动，暴露出其在鲁棒性方面的不足[138-139]。

为提升鲁棒性，可探索自动化提示生成与优化框架。例如可构建融合强化学习与遗传算法的混合优化系统，前者依据检测性能反馈自适应调整提示策略，后者则探索提示模板的多样性与全局组合最优解。此外，Zhou等人[138]的研究中使用的角色扮演机制也为提示优化提供了新思路。基于该思路，可设计反馈驱动的动态提示生成器，根据目标代码的复杂度特征，自适应调整漏洞描述的粒度层级及推理步骤的分解深度。上述策略不仅能够显著提升提示工程的鲁棒性与泛化性，还可降低人工设计提示模板的专业门槛，为大规模语言模型在安全检测中的推广应用奠定基础。

8.2.2 多任务学习冲突

在基于大规模语言模型的多任务漏洞检测框架中[136]，不同任务对特征的关注点存在差异：漏洞分类更依赖于全局的语义理解，而漏洞定位则更强调细粒度的结构与上下文信息。这种差异导致模型在多任务联合训练过程中容易出现梯度冲突与损失量级失衡问题，进而影响任务间的协同优化。

未来研究可进一步探索高效的多任务协同机制。一方面，可引入动态与自适应的损失加权策略，替代静态或人工设定的任务权重，根据任务难度、收敛速度与梯度分布实现权重的自动调整；另一方面，可采用基于梯度的优化方法，以缓解多任务训练中的梯度冲突问题[198-199]。例如，PCGrad（Projecting Conflicting Gradients）算法可在反向传播过程中检测任务间梯度冲突，并将冲突梯度分量投影至非冲突方向，从而确保每次参数更新不会对其他任务产

生负迁移效应。通过此类优化策略，可在多任务间实现更稳健的参数共享与性能平衡，为构建统一的漏洞检测框架提供关键支撑。

8.2.3　RAG知识库不完备

检索增强生成（RAG）技术在漏洞检测中有效提升了大模型的知识覆盖与推理能力，但其整体性能仍受制于知识库质量与检索机制的双重瓶颈。首先，RAG框架的有效性高度依赖外部漏洞知识库的完备程度。当知识库中缺乏相关漏洞样例或修复语义时，检索模块难以召回高质量参考信息，导致模型检测性能明显下降[148]。其次，检索算法的精度是影响系统表现的关键因素。当前主流RAG方法多基于语义相似度进行向量检索，但代码的词法或语法相似性并不等价于其在控制流或数据流层面的语义一致性，不精确的相似度匹配往往会引入大量无关片段，降低最终生成结果的可靠性。

为解决上述问题，未来研究可从知识库扩展与检索算法优化两方面展开。在知识库层面，应突破仅依赖已知漏洞样本的局限，构建更广义的安全知识体系。可系统整合多源异构数据，包括安全公告、CVE/NVD文本描述、开源项目中的修复提交、开发者讨论及安全论坛内容等，利用自然语言处理与代码挖掘技术自动抽取漏洞模式、修复策略与上下文信息，从而形成结构化、可查询的知识图谱。在检索层面，可采用分阶段检索策略：首先利用高效向量检索快速筛选候选集，再通过深度语义重排序模型进行精细化匹配与打分；同时，代码级知识对齐也是关键环节。通过将文档内容重构为粒度更细的原子知识单元，可显著提升检索精度与召回质量，使RAG在漏洞检测场景中实现更高的上下文利用率与生成稳定性。

8.2.4　硬件与计算资源限制

大规模语言模型的微调过程通常伴随极高的计算资源与存储成本[136-137]。这种资源消耗主要体现在两方面：其一，显存开销极大。以参数量约130亿的模型为例，完成全量微调通常需数百GB显存，必须依赖昂贵的多卡GPU服务器集群；其二，持续算力消耗成本高昂。大规模微调涉及多轮梯度迭代与大批量样本处理，其电力与云端算力成本常成为中小型研究团队难以承受的负担。

针对上述问题，近年来有研究提出了多种参数高效微调（Parameter-Efficient Fine-Tuning，PEFT）技术，如LoRA[160]。该方法通过仅训练低秩参数子集，大幅减少可训练参数量，从而有效降低显存占用。然而，PEFT方法在计算图执行阶段仍需加载完整模型权重，整体计算开销依然较大。为进一步降低资源门槛，可将PEFT与模型压缩技术（如量化与蒸馏）联合使用，以减少推理过程中的冗余计算。此外，混合专家架构（Mixture of Experts，MoE）[200-201]在漏洞检测中展现出良好的潜力。其核心思想是引入多专家子网络，每个子网络专注于特定类型的漏洞（如内存泄漏、命令注入等），并通过门控机制根据输入代码语义动态激活对应专家。例如，当检测涉及指针运算或动态内存分配的代码时，系统会优先路由至“内存安全专家”。这一机制在保持检测精度的同时显著降低了计算冗余，为资源受限的研究团队提供了一条可行的技术路径。

8.3　其他共性挑战性问题与未来展望

8.3.1　漏洞数据集问题

（1）标签准确率低：现有漏洞数据集普遍存在标签噪声问题，典型表现包括将安全代码误标为漏洞（假阳性）、漏标真实漏洞（假阴性）或漏洞类型标注错误等。这一问题主要源于数据构建过程中过度依赖自动化收集手段[196]。自动化方法虽然能够高效汇聚大规模样本，但其对代码上下文的理解有限，难以准确判断漏洞的真实存在性。相比之下，人工标注虽能提升数据质量，却需大量人力与时间投入。尤其对于复杂漏洞，仅凭静态分析往往无法得出确定结论，即便是经验丰富的安全专家，也需在动态调试或实际运行环境中才能判断代码片段的安全性。因此，构建高质量、高置信度的漏洞检测基准仍是当前领域面临的关键挑战之一。

未来的研究可从标注机制创新与数据质量控制两方面入手。一方面，可建立融合多模态验证与分级审核的标注框架：结合静态分析工具（如CodeQL）与LLM的语义验证能力，通过语法清洗、语义过滤与上下文一致性检查提升自动标注精度；同时在人工环节采用“机器预标注-专家复核-共识仲裁”的分级流程，借助可视化交互平台对关键样本进行重点复核，并对存在争议的实例采用多专家投票决策。另一方面，可引入噪声鲁棒学习与主动纠错机制，利用模型自适应发现并修正标签错误，从而进一步提升数据集的一致性与可靠性。

（2）数据不平衡：在多数漏洞数据集中，非漏洞样本数量远超漏洞样本。例如，在广泛使用的BigVul数据集[174]中，漏洞样本比例不足5%。这种严重的类别不平衡使模型训练过程中更倾向于预测主导类别（非漏洞），虽能取得较高的准确率，但对真

实漏洞的识别能力较弱，表现为召回率偏低。

应对解决思路主要包括数据层重采样与算法层损失重构两类方法。数据层面，可通过欠采样与过采样策略调整样本比例，以平衡训练集结构。算法层面，可采用代价敏感学习、焦点损失[202]或基于类别不确定性的自适应加权机制，从损失函数层面对少数类样本赋予更高权重或动态调整关注度，从而提升模型对少数类的识别能力与鲁棒性。

(3) 数据污染：数据污染已成为当前漏洞检测评估中被广泛关注的核心问题。由于大语言模型在预训练阶段通常涵盖海量开源代码、技术文档及安全报告，评测数据中的漏洞样本极有可能已包含于其预训练语料中。若预训练与评测数据存在重叠，模型表现出的高性能可能源于记忆回忆而非真实泛化，进而削弱评估结果的公正性与可信度。

为提升评估的可信度，应引入严格的数据隔离与污染审计机制。首先，可采用跨项目验证，即在一个项目集上训练模型，并在另一个完全独立的项目集上评估其跨代码库泛化能力。其次，应实施重复样本检测与近重复消除，移除与预训练语料在语法或功能上高度相似的样本，降低信息泄露风险。最后，可结合基于时间的分层划分方法，确保测试集中仅包含模型知识截止日期之后公开的漏洞，从而更贴近实际未知漏洞的检测场景，更准确反映模型的泛化推理能力与实用性能。

8.3.2 大模型的通用性问题

(1) 大模型的可解释性不足：在漏洞检测场景下，大语言模型的“黑箱”特性带来了巨大挑战。模型通过高维非线性变换完成代码语义分析，但难以像传统静态分析工具那样提供可追溯的漏洞判定依据或数据流路径解释。这种可解释性缺失使安全工程师难以理解与信任模型的判断，尤其在误报分析和漏洞修复建议生成等关键环节。

现有研究尝试利用注意力机制[150]解释模型决策过程，但注意力权重不等同于因果关系[203]，其解释能力有限。未来可探索更具因果性的解释方法，如使用归因分析方法量化不同输入特征对输出结果的影响，或通过反事实推理与图解释机制刻画模型的决策边界[204]，并结合可视化热力图呈现关键特征的空间分布。此外，也可借鉴程序切片技术，将模型决策过程视为可追踪的“程序执行流”，从输出结果回溯提取最小关联子图，以实现对模型决策行为的结构化解释。以上方法有助于推动漏洞检测模型从“结果导向”向“机制可验证”转变。

(2) 大模型的幻觉问题：LLM在生成过程中可能产生与事实不符的内容，即“幻觉(Hallucination)”现象。这是其概率生成机制的固有缺陷。已有研究表明，大模型在漏洞检测中的性能并不总优于轻量级模型[162-165]，原因在于更复杂的参数空间可能放大错误预测的置信度。即便结合检索增强生成(RAG)技术[148]，在面对新型漏洞或高度嵌套的代码结构时，模型仍容易产生误判。更严重的是，LLM往往无法自我识别幻觉结果，反而可能给出看似合理的解释，进一步加重漏洞验证与修复负担。

为缓解该问题，可从生成约束与验证反馈两方面入手。一方面，通过引入RAG、自检机制、多样性投票等策略，采用多轮生成与结果交叉比对来提升输出可靠性[205-206]；另一方面，可结合执行驱动或测试驱动验证机制，以实际运行结果验证模型预测的正确性。此外，多智能体系统(Multi-Agent Systems)也为幻觉抑制提供了新思路[155-156]：通过设定不同角色(如检测者、审查者、仲裁者)，实现智能体间的质疑与协作，从多视角博弈中达成对漏洞存在与否的高置信共识[154]。

(3) 大模型的安全性隐患：与传统检测工具相比，基于LLM的源代码漏洞检测方法本身也面临新的安全威胁。恶意用户可通过精心设计的代码注释或变量名实施对抗攻击，诱导模型忽略真实漏洞或生成误导性结论[207]。在模型微调阶段，不可信数据源可能引入后门攻击(Backdoor Attack)，导致特定代码模式被错误识别为安全。此外，模型还可能在推理过程中泄露训练数据中的敏感代码片段或商业算法逻辑[208]。

针对这些风险，应构建纵深防御体系以实现事前预防、事中监测与事后补救的全链路防护。首先，在事前预防阶段，可通过指令-输入隔离机制防御对抗攻击，确保系统指令和用户输入的源代码不被攻击者篡改；其次，在事中监测阶段，可部署安全护栏(Guardrails)对模型输入输出进行实时检测与干预，以防异常行为或滥用风险[209]；最后，在事后补救阶段，建立审计与回滚机制，记录模型交互轨迹以便追溯，并在检测异常时快速恢复安全状态。

8.4 拓展性研究展望

除了上述基于大语言模型的源代码漏洞检测方法所面临的挑战及相应的未来展望之外，后续研究还可以进一步拓展代码的表示形式，将研究对象从源代码延伸至更底层的代码结构，例如二进制代码或中间代码表示。其中，基于二进制代码的漏洞检

测方法已成为当前安全研究领域的重要方向之一，具有广泛的应用价值，尤其在处理闭源软件、嵌入式系统和安全审计等场景中表现突出。

目前，大语言模型在二进制漏洞检测领域的研究呈现多样化发展趋势，其中，将大语言模型融入反编译流程已成为重要的研究方向之一。例如，DECLLM[210]和LLM4Decompile[211]致力于提升反编译代码的可执行性，以实现自动化的代码分析；DeGPT[212]则聚焦于增强反编译结果的可读性，从而辅助人工审查。另一重要研究路线是将大语言模型与逆向分析技术相结合。例如LATTE[213]融合了LLM与污点分析，实现了分析流程的自动化；LIFT[214]则通过指令重写提升了动态符号执行的效率。最后，研究者还在积极构建创新的二进制漏洞检测框架与评估基准体系。在检测框架方面，VulBinLLM[215]提出了针对去符号二进制的端到端漏洞检测框架。在基准建设方面，DeBinVul[216]构建了首个二进制漏洞专项数据集，而BINMETRIC[217]则提供了更通用的二进制分析评估基准。

尽管LLM已在二进制漏洞检测中展现出一定潜力，但相比源代码，二进制在语义表达、结构解析和上下文建模方面更具挑战。未来研究可聚焦于构建适用于二进制的预训练机制、设计高效的指令表示方法，并探索面向二进制的多粒度检测框架。

9　总　结

本文针对基于大语言模型的源代码漏洞检测方法进行了系统性综述。首先介绍了研究框架与必要的预备知识，回顾了传统静态分析方法与深度学习方法在漏洞检测中的局限性。随后，系统梳理了基于轻量级Transformer与大规模语言模型这两大技术路线的漏洞检测方法，分别从“Tokenization、预训练模型、预训练方式、检测粒度与长度”以及“微调范式、提示工程、混合分析和多智能体”等多个维度，介绍了各自的研究进展。同时，本文还总结了常用的评估基准和数据集，并通过对比基于大语言模型的方法与传统方法的性能差异，揭示了前者在漏洞检测领域的技术优势与研究定位。本文研究表明，LLM方法凭借深层语义表示和自注意力机制能够覆盖更广泛的漏洞类型，尤其擅长检测复杂逻辑漏洞或静态规则库难以定义的漏洞。然而，正如第8节所述，当前LLM方法在模型架构设计、训练策略、数据质量、可解释性、鲁棒性及模型安全等方面仍面临诸多挑战。未来的研究应围绕这些关键问题，探索系统化的解决思路，以推动大语言模型在漏洞检测领域的持续演进与可靠落地。

作者贡献声明　蒋远、杨子含为共同一作。

致　谢　感谢各位专家提出的宝贵意见，感谢国家自然科学基金青年项目(No. 62302125)和面上项目(No. 62272132)的资助。

参　考　文　献

[1] Steenhoek B, Rahman M M, Jiles R, et al. An empirical study of deep learning models for vulnerability detection//Proceedings of the IEEE/ACM 45th International Conference on Software Engineering. Melbourne, Australia, 2023: 2237-2248

[2] Wang Y, Wang W, Joty S, et al. Codet5: Identifier-aware unified pretrained encoder-decoder models for code understanding and generation. arXiv preprint arXiv:2109.00859, 2021

[3] Ghaffarian S M, Shahriari H R. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. ACM Computing Surveys (CSUR), 2017, 50(4): 1-36

[4] Su Xiaohong, Zheng Weining, Jiang Yuan, et al. Research and progress on learning-based source code vulnerability detection. Chinese Journal of Computers, 2024, 47(2): 337-374 (in Chinese)
(苏小红，郑伟宁，蒋远等. 基于学习的源代码漏洞检测研究与进展. 计算机学报，2024，47(2):337-374)

[5] Lin G, Wen S, Han Q L, et al. Software vulnerability detection using deep neural networks: a survey. Proceedings of the IEEE, 2020, 108 (10):1825-1848

[6] Chakraborty S, Krishna R, Ding Y, et al. Deep learning based vulnerability detection: Are we there yet? IEEE Transactions on Software Engineering, 2022, 48(09):3280-3296

[7] Nong Y, Sharma R, Hamou-Lhadj A, et al. Open science in software engineering: A study on deep learning-based vulnerability detection. IEEE Transactions on Software Engineering, 2023, 49(4):1983-2005

[8] Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering//Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. London, UK, 2014: 1-10

[9] Yin X, Ni C. Multitask-based evaluation of open-source LLM on software vulnerability. arXiv preprint arXiv:2404.02056, 2024

[10] Jiang Y, Zhang Y, Su X, et al. Stagedvulbert: Multi-granular vulnerability detection with a novel pre-trained code model. IEEE Transactions on Software Engineering, 2024, 50(12): 3454-3471

[11] Radford A, Wu J, Child R, et al. Language models are unsupervised multitask learners. San Francisco: OpenAI, OpenAI blog: 2019

[12] Ouyang L, Wu J, Jiang X, et al. Training language models to follow instructions with human feedback//Proceedings of the Advances in Neural Information Processing Systems. New Orleans, USA, 2022: 27730-27744)

[13] Hoffmann J, Borgeaud S, Mensch A, et al. Training compute-optimal large language models. arXiv preprint arXiv: 2203.15556, 2022

[14] Wei J, Tay Y, Bommasani R, et al. Emergent abilities of large language models. arXiv preprint arXiv:2206.07682, 2022

[15] Zhao W X, Zhou K, Li J, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 2023

[16] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, USA, 2016: 770-778

[17] Ba J L, Kiros J R, Hinton G E. Layer normalization. arXiv preprint arXiv:1607.06450, 2016

[18] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need//Advances in Neural Information Processing Systems. Long Beach, USA, 2017: 5998-6008

[19] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018

[20] Shaw P, Uszkoreit J, Vaswani A. Self-attention with relative position representations. arXiv preprint arXiv:1803.02155, 2018

[21] Su J, Ahmed M, Lu Y, et al. Roformer: Enhanced transformer with rotary position embedding. Neurocomputing, 2024, 568: 127063

[22] Press O, Smith N A, Lewis M. Train short, test long: Attention with linear biases enables input length extrapolation. arXiv preprint arXiv:2108.12409, 2021

[23] Lin T, Wang Y, Liu X, et al. A survey of transformers. AI Open, 2022, 3:111-132

[24] Feng Z, Guo D, Tang D, et al. Codebert: A pre-trained model for programming and natural languages//Findings of the Association for Computational Linguistics: EMNLP 2020. Online, 2020: 1536-1547

[25] Chen M, Tworek J, Jun H, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021

[26] Li R, Allal L B, Zi Y, et al. Starcoder: may the source be with you! arXiv preprint arXiv:2305.06161, 2023

[27] Husain H, Wu H H, Gazit T, et al. Codesearchnet challenge: Evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436, 2019

[28] Guo D, Ren S, Lu S, et al. Graphcodebert: Pre-training code representations with data flow. arXiv preprint arXiv: 2009.08366, 2020

[29] Achiam J, Adler S, Agarwal S, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023

[30] Gao Z, Wang H, Zhou Y, et al. How far have we gone in vulnerability detection using large language models. arXiv preprint arXiv: 2311.12420, 2023

[31] Wang Y, Le H, Gotmare A D, et al. Codet5+: Open code large language models for code understanding and generation. arXiv preprint arXiv:2305.07922, 2023

[32] Ahmad W U, Chakraborty S, Ray B, et al. Unified pre-training for program understanding and generation. arXiv preprint arXiv: 2103.06333, 2021

[33] TeamAlphaCode. Alphacode 2 technical report. London: Google DeepMind, technical report: 266058988, 2023

[34] Pewny J, Schuster F, Bernhard L, et al. Leveraging semantic signatures for bug search in binary programs//Proceedings of the 30th An- nual Computer Security Applications Conference. New Orleans, USA, 2014: 406-415

[35] Portokalidis G, Slowinska A, Bos H. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. ACM SIGOPS Operating Systems Review, 2006, 40(4):15-27

[36] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2005: 3-4

[37] Engler D, Chen D Y, Hallem S, et al. Bugs as deviant behavior: A general approach to inferring errors in systems code. ACM SIGOPS Operating Systems Review, 2001, 35(5):57-72

[38] Ferschke O, Gurevych I, Rittberger M. Flawfinder: A modular system for predicting quality flaws in wikipedia//Proceedings of the Conference and Labs of the Evaluation Forum. Rome, Italy, 2012: 1-10

[39] Li Z, Zou D, Xu S, et al. Vulpecker: an automated vulnerability detection system based on code similarity analysis//Proceedings of the 32nd Annual Conference on Computer Security Applications. Los Angeles, USA, 2016: 201-213

[40] Kim S, Woo S, Lee H, et al. Vuddy: A scalable approach for vulnerable code clone discovery//Proceedings of the IEEE Symposium on Security and Privacy. San Jose, USA, 2017: 595-614

[41] Jang J, Agrawal A, Brumley D. Redebug: finding unpatched code clones in entire OS distributions//Proceedings of the IEEE Symposium on Security and Privacy. San Francisco, USA, 2012: 48-62

[42] Peng H, Mou L, Li G, et al. Building program vector representations for deep learning//Proceedings of the International conference on knowledge science, engineering and management. Chongqing, China, 2015: 547-553

[43] Grieco G, Grinblat G L, Uzal L, et al. Toward large-scale vulnerability discovery using machine learning//Proceedings of the 6th ACM Conference on Data and Application Security and Privacy. New Orleans, USA, 2016: 85-96

[44] Shin Y, Williams L. Can traditional fault prediction models be used for vulnerability prediction? Empirical Software Engineering, 2013, 18(1): 25-59

[45] Li J, He P, Zhu J, et al. Software defect prediction via convolutional neural network//Proceedings of the IEEE International Conference on Software Quality, Reliability and Security. Prague, Czech Republic, 2017: 318-328

[46] Phan A V, Le Nguyen M, Bui L T. Convolutional neural

networks over control flow graphs for software defect prediction//Proceedings of the IEEE 29th International Conference on Tools with Artificial In-telligence. Boston,USA, 2017: 45-52

[47] Russell R, Kim L, Hamilton L, et al. Automated vulnerability detection in source code using deep representation learning// Proceedings of the 17th IEEE International Conference on Machine Learning and Applications. Orlando, USA, 2018: 757-762

[48] Wartschinski L, Noller Y, Vogel T, et al. Vudenc: Vulnerability detection with deep learning on a natural codebase for python. Information and Software Technology, 2022, 144:106809

[49] Ghaffarian S M, Shahriari H R. Neural software vulnerability analysis using rich intermediate graph representations of programs. Informa- tion Sciences, 2021, 553:189-207

[50] Zhang H, Zhang W, Feng Y, et al. Svscanner: Detecting smart contract vulnerabilities via deep semantic extraction. Journal of Information Security and Applications, 2023, 75:103484

[51] Li L, Liu Y, Sun G, et al. Smart contract vulnerability detection based on automated feature extraction and feature interaction. IEEE Transactions on Knowledge and Data Engineering, 2023, 36(9):4916-4929

[52] Sendner C, Chen H, Fereidooni H, et al. Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2023

[53] You X, Li H, Wang H, et al. Smartdt: An effective vulnerability detection system of smart contracts based on deep learning// Proceedings of the IEEE International Conference on Big Data. Sorrento, Italy, 2023: 2369-2376

[54] Duan X, Wu J, Ji S, et al. Vulsniper: Focus your attention to shoot fine-grained vulnerabilities//Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence. Macao, China, 2019: 4665-4671

[55] Lin G, Zhang J, Luo W, et al. Software vulnerability discovery via learning multi-domain knowledge bases. IEEE Transactions on De- pendable and Secure Computing, 2019, 18(5):2469-2485

[56] Cheng X, Wang H, Hua J, et al. Static detection of control-flow-related vulnerabilities using graph embedding//Proceedings of the 24th International Conference on Engineering of Complex Computer Systems. Guangzhou, China, 2019: 41-50

[57] Duan X, Wu Jingzheng, Luo Tianyue, et al. Vulnerability mining method based on code property graph and attention-based bidirectional LSTM. Journal of Software, 2020, 31(11): 3404-3420 (in Chinese)
(段旭,吴敬征,罗天悦等.基于代码属性图及注意力双向LSTM的漏洞挖掘方法.软件学报,2020,31(11):3404-3420)

[58] Zhou Y, Liu S, Siow J, et al. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks//Advances in Neural Information Processing Systems. Vancouver, Canada, 2019: 10197-10207

[59] Wang H, Ye G, Tang Z, et al. Combining graph-based learning with automated data collection for code vulnerability detection. IEEE Transactions on Information Forensics and Security, 2020, 16:1943-1958

[60] Wang Y, Wang K, Gao F, et al. Learning semantic program embeddings with graph interval neural network. Proceedings of the ACM on Programming Languages, 2020, 4(OOPSLA):1-27

[61] Tian J, Xing W, Li Z. Bvdetector: A program slice-based binary code vulnerability intelligent detection system. Information and Software Technology, 2020, 123:106289

[62] Li M, Li C, Li S, et al. Acgvd: Vulnerability detection based on comprehensive graph via graph neural network with attention// Proceedings of the International Conference on Information and Communications Security. Chongqing, China, 2021: 243-259

[63] Cao S, Sun X, Bo L, et al. Bgnn4vd: constructing bidirectional graph neural-network for vulnerability detection. Information and Software Technology, 2021, 136:106576

[64] Wu Y, Zou D, Dou S, et al. Vulcnn: An image-inspired scalable vulnerability detection system//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 2365-2376

[65] Mamede C, Pinconschi E, Abreu R. A transformer-based ide plugin for vulnerability detection//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. Rochester, USA, 2022: 1-4

[66] Guo W, Fang Y, Huang C, et al. Hyvuldect: a hybrid semantic vulnerability mining system based on graph neural network. Computers & Security, 2022, 121:102823

[67] Li X, Xin Y, Zhu H, et al. Cross-domain vulnerability detection using graph embedding and domain adaptation. Computers & Security, 2023, 125:103017

[68] Jie W, Chen Q, Wang J, et al. A novel extended multimodal ai frame-work towards vulnerability detection in smart contracts. Information Sciences, 2023, 636:118907

[69] Tang W, Tang M, Ban M, et al. Csgvd: A deep learning approach combining sequence and graph embedding for source code vulnerability detection. Journal of Systems and Software, 2023, 199:111623

[70] Wang W, Nguyen T N, Wang S, et al. Deepvd: Toward class-separation features for neural network vulnerability detection// Proceedings of the IEEE/ACM 45th International Conference on Software Engineering. Melbourne, Australia, 2023: 2249-2261

[71] Ni C, Guo X, Zhu Y, et al. Function-level vulnerability detection through fusing multi-modal knowledge//Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering. Luxembourg, 2023: 1911-1918

[72] Guo L, Huang H, Xue S, et al. Reentrancy vulnerability detection based on graph convolutional networks and expert patterns//Proceedings of the IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip. Singapore, 2023: 312-316

[73] Zhang C, Liu B, Xin Y, et al. Cpvd: Cross project vulnerability detection based on graph attention network and domain adaptation. IEEE Transactions on Software Engineering, 2023, 49(8):4152-4168

[74] Wan T, Lu L, Xu H, et al. Software vulnerability detection via

doc2vec with path representations//Proceedings of the IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion. Chiang Mai, Thailand, 2023: 131-139

[75] Zhu H, Yang K, Wang L, et al. Grabit: A sequential model-based framework for smart contract vulnerability detection//Proceedings of the IEEE 34th International Symposium on Software Reliability Engineering. Florence, Italy, 2023: 568-577

[76] Wang Q, Li Z, Liang H, et al. Graph confident learning for software vulnerability detection. Engineering Applications of Artificial Intelligence, 2024, 133:108296

[77] Sun H, Cui L, Li L, et al. Vdtriplet: Vulnerability detection with graph semantics using triplet model. Computers & Security, 2024, 139:103732

[78] Steenhoek B, Gao H, Le W. Dataflow analysis-inspired deep learning for efficient vulnerability detection//Proceedings of the 46th IEEE/ACM International Conference on Software Engineering. Lisbon, Portugal, 2024: 1-13

[79] Nguyen V, Le T, Tantithamthavorn C, et al. Deep domain adaptation with max-margin principle for cross-project imbalanced software vulnerability detection. ACM Transactions on Software Engineering and Methodology, 2024, 33(6):1-34

[80] Huang H, Guo L, Zhao L, et al. Effective combining source code and opcode for accurate vulnerability detection of smart contracts in edge ai systems. Applied Soft Computing, 2024, 158:111556

[81] Li Z, Zou D, Xu S, et al. Vuldeepecker: A deep learning-based system for vulnerability detection//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2018

[82] Zou D, Wang S, Xu S, et al. μvuldeepecker: A deep learning-based system for multiclass vulnerability detection. IEEE Transactions on Dependable and Secure Computing, 2019, 18(5): 2224-2236

[83] Hoang T, Dam H K, Kamei Y, et al. Deepjit: an end-to-end deep learning framework for just-in-time defect prediction//Proceedings of the IEEE/ACM 16th International Conference on Mining Software Repos- itories. Montreal, Canada, 2019: 34-45

[84] Allamanis M, Jackson-Flux H, Brockschmidt M. Self-supervised bug detection and repair//Advances in Neural Information Processing Systems. Online, 2021: 27865-27876

[85] Li Z, Zou D, Xu S, et al. Sysevr: A framework for using deep learning to detect software vulnerabilities. IEEE Transactions on Dependable and Secure Computing, 2021, 19(4):2244-2258

[86] Duan Ya-Nan. Research on software vulnerability detection based on code property graph and graph convolutional neural network. Harbin Institute of Technology, Harbin, 2020 (in Chinese)
(段亚男. 基于代码属性图和图卷积神经网络的软件漏洞检测方法研究. 哈尔滨工业大学, 哈尔滨, 2020)

[87] Zheng W, Jiang Y, Su X. Vu1spg: Vulnerability detection based on slice property graph representation learning//Proceedings of the IEEE 32nd International Symposium on Software Reliability Engineering. Wuhan, China, 2021: 457-467

[88] Yan H, Luo S, Pan L, et al. Han-bsvd: a hierarchical attention network for binary software vulnerability detection. Computers & Security, 2021, 108:102286

[89] Zhao J, Guo S, Mu D. Doubigru-a: Software defect detection algorithm based on attention mechanism and double bigru. Computers & Security, 2021, 111:102459

[90] Jeon S, Kim H K. Autovas: An automated vulnerability analysis system with a deep learning approach. Computers & Security, 2021, 106: 102308

[91] Sun H, Cui L, Li L, et al. Vdsimilar: Vulnerability detection based on code similarity of vulnerabilities and patches. Computers & Security, 2021, 110:102417

[92] Cheng X, Wang H, Hua J, et al. Deepwukong: Statically detecting software vulnerabilities using deep graph neural network. ACM Transactions on Software Engineering and Methodology, 2021, 30(3):1-33

[93] Zou D, Zhu Y, Xu S, et al. Interpreting deep learning-based vulnera- bility detector predictions based on heuristic searching. ACM Transactions on Software Engineering and Methodology, 2021, 30(2):1-31

[94] Thapa C, Jang S I, Ahmed M E, et al. Transformer-based language models for software vulnerability detection//Proceedings of the 38th Annual Computer Security Applications Conference. Austin, USA, 2022: 481-496

[95] Xu R, Tang Z, Ye G, et al. Detecting code vulnerabilities by learning from large-scale open source repositories. Journal of Information Security and Applications, 2022, 69:103293

[96] Tang Z, Hu Q, Hu Y, et al. Sevuldet: A semantics-enhanced learnable vulnerability detector//Proceedings of the 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Balti- more, USA, 2022: 150-162

[97] Pan S, Bao L, Xia X, et al. Fine-grained commit-level vulnerability type prediction by cwe tree structure//Proceedings of the IEEE/ACM 45th International Conference on Software Engineering. Melbourne, Australia, 2023: 957-969

[98] Wang Y, Jia P, Peng X, et al. Binvuldet: Detecting vulnerability in binary program via decompiled pseudo code and bilstm-attention. Com-puters & Security, 2023, 125: 103023

[99] Fan Y, Wan C, Fu C, et al. Vdotr: Vulnerability detection based on tensor representation of comprehensive code graphs. Computers & Security, 2023, 130:103247

[100] Ren X, Wu Y, Li J, et al. Smart contract vulnerability detection based on a semantic code structure and a self-designed neural network. Computers and Electrical Engineering, 2023, 109: 108766

[101] Kong L, Luo S, Pan L, et al. A multi-type vulnerability detection framework with parallel perspective fusion and hierarchical feature enhancement. Computers & Security, 2024, 140:103787

[102] Wu T, Chen L, Du G, et al. Ultravcs: Ultra-fine-grained variable-based code slicing for automated vulnerability detection. IEEE Transactions on Information Forensics and Security, 2024, 19:3986-4000

[103] Xiao P, Xiao Q, Zhang X, et al. Vulnerability detection based on

enhanced graph representation learning. IEEE Transactions on Informa- tion Forensics and Security, 2024, 19:5120-5135

[104] Xiao W, Hou Z, Wang T, et al. Msgvul: Multi-semantic integration vulnerability detection based on relational graph convolutional neural networks. Information and Software Technology, 2024, 170:107442

[105] Choi M, Jeong S, Oh H, et al. End-to-end prediction of buffer overruns from raw source code via neural memory networks// Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. Melbourne, Australia, 2017: 1546-1553

[106] Sestili C D, Snavely W S, VanHoudnos N M. Towards security defect prediction with ai. arXiv preprint arXiv:1808.09897, 2018

[107] Li Z, Zou D, Xu S, et al. Vuldeelocator: a deep learning-based finegrained vulnerability detector. IEEE Transactions on Dependable and Secure Computing, 2021, 19(4):2821-2837

[108] Tian J, Zhang J, Liu F. Bbreglocator: A vulnerability detection sys- tem based on bounding box regression//Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops. Taipei, China, 2021: 93-100

[109] Li Y, Wang S, Nguyen T N. Vulnerability detection with fine-grained interpretations//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Athens, Greece, 2021: 292-303

[110] Cao S, Sun X, Bo L, et al. Mvd: memory-related vulnerability detection based on flow-sensitive graph neural networks// Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 1456-1468

[111] Fu M, Tantithamthavorn C. Linevul: A transformer-based line-level vulnerability prediction//Proceedings of the 19th International Conference on Mining Software Repositories. Pittsburgh, USA, 2022: 608-620

[112] Ding Y, Suneja S, Zheng Y, et al. Velvet: a novel ensemble learning approach to automatically locate vulnerable statements// Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering. Honolulu, USA, 2022: 959-970

[113] Hin D, Kan A, Chen H, et al. Linevd: statement-level vulnerability detection using graph neural networks//Proceedings of the 19th Interna- tional Conference on Mining Software Repositories. Pittsburgh, USA, 2022: 596-607

[114] Nguyen H H, Nguyen N M, Xie C, et al. Mando: Multi-level heterogeneous graph embeddings for fine-grained detection of smart contract vulnerabilities//Proceedings of the IEEE 9th International Conference on Data Science and Advanced Analytics. Shenzhen, China, 2022: 1-10

[115] Nguyen H H, Nguyen N M, Doan H P, et al. Mando-guru: vulnerability detection for smart contract source code by heterogeneous graph embeddings//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, 2022: 1736-1740

[116] Dong Y, Tang Y, Cheng X, et al. Sedsvd: Statement-level software vulnerability detection based on relational graph convolutional network with subgraph embedding. Information and Software Technology, 2023, 158:107168

[117] Song Z, Wang J, Yang K, et al. Hgivul: Detecting inter-procedural vulnerabilities based on hypergraph convolution. Information and Soft- ware Technology, 2023, 160:107219

[118] Wu B, Zou F, Yi P, et al. Slicedlocator: Code vulnerability locator based on sliced dependence graph. Computers & Security, 2023, 134: 103469

[119] Wen X C, Gao C, Ye J, et al. Meta-path based attentional graph learning model for vulnerability detection. IEEE Transactions on Software Engineering, 2024, 50(3):360-375

[120] Wu H, Zhang Z, Wang S, et al. Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques//Proceedings of the IEEE 32nd International Symposium on Software Reliability Engineering. Wuhan, China, 2021: 378-389

[121] Hanif H, Maffeis S. Vulberta: Simplified source code pre-training for vulnerability detection//Proceedings of the International joint confer- ence on neural networks. Padua, Italy, 2022: 1-8

[122] Yang H, Yang H, Zhang L, et al. Source code vulnerability detection using vulnerability dependency representation graph// Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications. Wuhan, China, 2022: 457-464

[123] Yuan D, Wang X, Li Y, et al. Optimizing smart contract vulnerability detection via multimodality code and entropy embedding. Journal of Systems and Software, 2023, 202:111699

[124] Sun X, Tu L, Zhang J, et al. Assbert: Active and semi-supervised bert for smart contract vulnerability detection. Journal of Information Security and Applications, 2023, 73: 103423

[125] Ni C, Yin X, Yang K, et al. Distinguishing look-alike innocent and vulnerable code by subtle semantic representation learning and explanation//Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. San Francisco, USA, 2023: 1611-1622

[126] Zhang J, Liu Z, Hu X, et al. Vulnerability detection by learning from syntax-based execution paths of code. IEEE Transactions on Software Engineering, 2023, 49(8):4196-4212

[127] Islam N T, Parra G D L T, Manuel D, et al. An unbiased transformer source code learning with semantic vulnerability graph//Proceedings of the IEEE 8th European Symposium on Security and Privacy. Delft, Netherlands, 2023: 144-159

[128] Jiang Y, Qu Z, Treude C, et al. Enhancing fine-grained vulnerability detection with reinforcement learning. IEEE Transactions on Software Engineering, 2025, 51(10):2900-2920

[129] Tran H C, Tran A D, Le K H. Detectvul: A statement-level code vulnerability detection for python. Future Generation Computer Systems, 2025, 163:107504

[130] Sennrich R. Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909, 2015

[131] Schuster M, Nakajima K. Japanese and Korean voice search//Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing. Kyoto, Japan, 2012: 5149-5152

[132] Kudo T. Subword regularization: Improving neural network translation models with multiple subword candidates. arXiv preprint arXiv:1804.10959, 2018

[133] Chirkova N, Troshin S. Codebpe: Investigating subtokenization options for large language model pretraining on source code. arXiv preprint arXiv:2308.00683, 2023

[134] Shestov A, Levichev R, Mussabayev R, et al. Finetuning large language models for vulnerability detection. IEEE Access, 2025, 13: 38889-38900

[135] Yusuf I N B, Jiang L. Your instructions are not always helpful: Assessing the efficacy of instruction fine-tuning for software vulnerability detection. arXiv preprint arXiv:2401.07466, 2024

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

[137] Yang A Z, Le Goues C, Martins R, et al. Large language models for test-free fault localization//Proceedings of the 46th IEEE/ACM International Conference on Software Engineering. Lisbon, Portugal, 2024: 1-12

[138] Zhou X, Zhang T, Lo D. Large language model for vulnerability detection: Emerging results and future directions//Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results. Lisbon, Portugal, 2024: 47-51

[139] Zhang C, Liu H, Zeng J, et al. Prompt-enhanced software vulnerability detection using ChatGPT//Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings. Lisbon, Portugal, 2024: 276-277

[140] Liu Z, Liao Q, Gu W, et al. Software vulnerability detection with gpt and in-context learning//Proceedings of the 8th International Conference on Data Science in Cyberspace. Hefei, China, 2023: 229-236

[141] Yang Y, Zhou X, Mao R, et al. Dlap: A deep learning augmented large language model prompting framework for software vulnerability detection. Journal of Systems and Software, 2025, 219:112234

[142] Zhang Y X, Huang C, Liu R, et al. Vulnerability detection method for smart contracts based on prompt fine-tuning. Journal of Information Network Security, 2025, 25(4): 664-673 (in Chinese)
(张雨轩, 黄诚, 柳蓉等. 结合提示词微调的智能合约漏洞检测方法. 信息网络安全, 2025, 25(4):664-673)

[143] Geng C, Chang S Y, Huang H P. Smart contract vulnerability detection based on prompt engineering in zero-shot scenarios. Journal of Information Countermeasure Technology, 2024, 3(2): 70-81 (in Chinese)
(耿辰, 常舒予, 黄海平. 零样本场景下基于提示工程的智能合约漏洞检测研究. 信息对抗技术, 2024, 3(2):70-81)

[144] Du X, Wen M, Zhu J, et al. Generalization-enhanced code vulnerability detection via multi-task instruction fine-tuning. arXiv preprint arXiv:2406.03718, 2024

[145] Lekssays A, Mouhcine H, Tran K, et al. Llmxcpg: Context-aware vulnerability detection through code property graph-guided large language models//Proceedings of the 34th USENIX Security Symposium. Seattle, USA, 2025: 489-507

[146] Wen X C, Yang Y, Gao C, et al. Boosting vulnerability detection of LLMs via curriculum preference optimization with synthetic reasoning data//Findings of the Association for Computational Linguistics: ACL 2025. Vienna, Austria, 2025: 8935-8949

[147] Lu G, Ju X, Chen X, et al. Grace: Empowering llm-based software vulnerability detection with graph structure and in-context learning. Journal of Systems and Software, 2024, 212: 112031

[148] Du X, Zheng G, Wang K, et al. Vul-rag: Enhancing llm-based vulnerability detection via knowledge-level rag. arXiv preprint arXiv: 2406.11147, 2024

[149] Mao Z, Li J, Jin D, et al. Multi-role consensus through llms discussions for vulnerability detection//Proceedings of the IEEE 24th International Conference on Software Quality, Reliability, and Security Companion. Cambridge, UK, 2024: 1318-1319

[150] Li Y, Li X, Wu H, et al. Attention is all you need for llm-based code vulnerability localization. arXiv preprint arXiv:2410.15288, 2024

[151] Sun Y, Wu D, Xue Y, et al. Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. Lisbon, Portugal, 2024: 1-13

[152] Wang C, Liu J, Peng X, et al. Boosting static resource leak detection via llm-based resource-oriented intention inference. arXiv preprint arXiv:2311.04448, 2023

[153] Deng Y, Xia C S, Peng H, et al. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models//Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. Seattle, USA, 2023: 423-435

[154] Tang X, Kim K, Song Y, et al. Codeagent: Autonomous communicative agents for code review. arXiv preprint arXiv: 2402.02172, 2024

[155] Yildiz A, Teo S G, Lou Y, et al. Benchmarking LLMS and LLM-based agents in practical vulnerability detection for code repositories//Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics. Vienna, Austria, 2025: 30848-30865

[156] Wei Z, Sun J, Sun Y, et al. Advanced smart contract vulnerability detection via LLM-powered multi-agent systems. IEEE Transactions on Software Engineering, 2025, 51(10): 2830-2846

[157] Touvron H, Lavril T, Izacard G, et al. Llama: Open and efficient foundation language models. arXiv preprint arXiv: 2302.13971, 2023

[158] Albert Q J, Sablayrolles A, Mensch A, et al. Mistral 7b. arXiv preprint arXiv:2310.06825, 2023

[159] Roziere B, Gehring J, Gloeckle F, et al. Code llama: Open

foundation models for code. arXiv preprint arXiv: 2308.12950, 2023

[160] Hu E J, Shen Y, Wallis P, et al. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021

[161] Ding Y, Fu Y, Ibrahim O, et al. Vulnerability detection with code lan-guage models: How far are we? arXiv preprint arXiv: 2403.18624, 2024

[162] Purba M D, Ghosh A, Radford B J, et al. Software vulnerability detec- tion using large language models//Proceedings of the IEEE 34th International Symposium on Software Reliability Engineering Workshops. Florence, Italy, 2023: 112-119

[163] Yin X. Pros and cons! evaluating ChatGPT on software vulnerability. arXiv preprint arXiv:2404.03994, 2024

[164] Khare A, Dutta S, Li Z, et al. Understanding the effectiveness of large language models in detecting security vulnerabilities. arXiv preprint arXiv:2311.16169, 2023

[165] Fu M, Tantithamthavorn C K, Nguyen V, et al. ChatGPT for vulnerabil- ity detection, classification, and repair: How far are we?//Proceedings of the 30th Asia-Pacific Software Engineering Conference. Seoul, Republic of Korea, 2023: 632-636

[166] He D, Yu S R, Wang Y F, et al. An empirical study on the effectiveness of large language models for C/C++ code vulnerability detection. Journal of Communications Technology, 2024, 57(5): 519-528 (in Chinese)
(和达，余尚仁，王一凡等. 大语言模型C/C++代码漏洞检测效能的实证研究. 通信技术，2024，57(5):519-528)

[167] Yu L H, Hu S W, Huang L X, et al. A source code security vulnerability detection method using ChatGPT. Computer and Modernization, 2024(4): 88-91, 120 (in Chinese)
(余里辉，胡少文，黄浪鑫等. 一种使用ChatGPT的源代码安全漏洞检测方法. 计算机与现代化，2024(4):88-91，120)

[168] Bae J, Kwon S, Myeong S. Enhancing software code vulnerability detection using GPT-4o and claude-3.5 sonnet: A study on prompt en- gineering techniques. Electronics, 2024, 13(13):2657

[169] Okun V, Delaitre A, Black P E, et al. Report on the static analysis tool exposition (sate) iv. Gaithersburg: National Institute of Standards and Technology, NIST Special Publication: 500, 2013

[170] He J, Vechev M. Large language models for code: Security hardening and adversarial testing//Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. Copenhagen, Denmark, 2023: 1865-1879

[171] Ponta S E, Plate H, Sabetta A, et al. A manually-curated dataset of fixes to vulnerabilities of open-source software//Proceedings of the IEEE/ACM 16th International Conference on Mining Software Repositories. Montreal, Canada, 2019: 383-387

[172] Just R, Jalali D, Ernst M D. Defects4j: A database of existing faults to enable controlled testing studies for java programs//Proceedings of the 2014 international symposium on software testing and analysis. San Jose, USA, 2014: 437-440

[173] Widyasari R, Sim S Q, Lok C, et al. Bugsinpy: a database of existing bugs in python programs to enable controlled testing and debugging studies//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Virtual, USA, 2020: 1556-1560

[174] Fan J, Li Y, Wang S, et al. A c/c++ code vulnerability dataset with code changes and CVE summaries//Proceedings of the 17th International Conference on Mining Software Repositories. Seoul, Republic of Korea, 2020: 508-512

[175] Li Z, Zou D, Xu S, et al. Vuldeepecker: A deep learning-based system for vulnerability detection. arXiv preprint arXiv: 1801.01681, 2018

[176] Chakraborty S, Krishna R, Ding Y, et al. Deep learning based vulnerability detection: Are we there yet? IEEE Transactions on Software Engineering, 2021, 48(9):3280-3296

[177] Nikitopoulos G, Dritsa K, Louridas P, et al. Crossvul: a cross-language vulnerability dataset with commit data//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Athens, Greece, 2021: 1565-1569

[178] Bhandari G, Naseer A, Moonen L. Cvefixes: automated collection of vulnerabilities and their fixes from open-source software//Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering. Athens, Greece, 2021: 30-39

[179] Chen Y, Ding Z, Alowain L, et al. Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection//Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses. Hong Kong, China, 2023: 654-668

[180] Zheng Y, Pujar S, Lewis B, et al. D2a: A dataset built for ai-based vulnerability detection methods using differential analysis//Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice. Madrid, Spain, 2021: 111-120

[181] Noever D. Can large language models find and fix vulnerable software? arXiv preprint arXiv:2308.10345, 2023

[182] Mohajer M M, Aleithan R, Harzevili N S, et al. Effectiveness of ChatGPT for static analysis: How far are we?//Proceedings of the 1st ACM International Conference on AI-Powered Software. Porto de Galinhas, Brazil, 2024: 151-160

[183] Li Z, Dutta S, Naik M. LLM-Assisted Static Analysis for Detecting Security Vulnerabilities. arXiv preprint arXiv: 2405.17238, 2024

[184] Wen X C, Wang X, Chen Y, et al. Vuleval: Towards repository-level evaluation of software vulnerability detection. arXiv preprint arXiv: 2404.15596, 2024

[185] Zhou X, Tran D M, Le-Cong T, et al. Comparison of Static Application Security Testing Tools and Large Language Models for Repo-level Vulnerability Detection. arXiv preprint arXiv: 2407.16235, 2024

[186] Ni C, Shen L, Xu X, et al. Learning-based Models for Vulnerability Detection: An Extensive Study. arXiv preprint arXiv: 2408.07526, 2024

[187] Tamberg K, Bahsi H. Harnessing large language models for software vulnerability detection: a comprehensive benchmarking

study. arXiv preprint arXiv:2405.15614,2024

[188] Bakhshandeh A, Keramatfar A, Norouzi A, et al. Using ChatGPT as a static application security testing tool. arXiv preprint arXiv: 2308.14434,2023

[189] Wang Z, Zhang L, Cao C, et al. How Does Naming Affect LLMs on Code Analysis Tasks? arXiv preprint arXiv: 2307.12488,2024

[190] Fang C, Miao N, Srivastav S, et al. Large language models for code analysis: Do LLMs really do their job?//Proceedings of the 33rd USENIX Security Symposium. Philadelphia, USA, 2024: 829-846

[191] Chen C, Su J, Chen J, et al. When ChatGPT meets smart contract vul- nerability detection: How far are we? ACM Transactions on Software Engineering and Methodology, 2025, 34(4):1-30

[192] Yu J, Liang P, Fu Y, et al. Security code review by LLMs: A deep dive into responses. arXiv preprint arXiv:2401.16310,2024

[193] Yarra R. Llmpatronous: Harnessing the power of llms for vulnerability detection. arXiv preprint arXiv:2504.18423,2025

[194] Chen S, Wong S, Chen L, et al. Extending context window of large language models via positional interpolation. arXiv preprint arXiv:2306.15595,2023

[195] Peng B, Quesnelle J, Fan H, et al. Yarn: Efficient context window extension of large language models. arXiv preprint arXiv: 2309.00071,2023

[196] Zhou X, Cao S, Sun X, et al. Large language model for vulnerability detection and repair: Literature review and the road ahead. ACM Transactions on Software Engineering and Methodology, 2025, 34(5): 1-31

[197] Qiu S, Huang M, Cheng J. Boosting vulnerability detection with inter-function multilateral association insights. arXiv preprint arXiv: 2506.21014,2025

[198] Yu T, Kumar S, Gupta A, et al. Gradient surgery for multi-task learning//Advances in Neural Information Processing Systems. Online, 2020: 5824-5836

[199] Meng F, Xiao Z, Zhang Y, et al. Ri-pcgrad: Optimizing multi-task learning with rescaling and impartial projecting conflict gradients. Applied Intelligence, 2024, 54(22):12009-12019

[200] Yang X, Wang S, Zhou J, et al. One-for-all does not work! enhancing vulnerability detection by mixture-of-experts (moe). Proceedings of the ACM on Software Engineering, 2025, 2(FSE):446-464

[201] Yuan H, Yu L, Huang Z, et al. Mos: Towards effective smart contract vulnerability detection through mixture-of-experts tuning of large language models. arXiv preprint arXiv:2504.12234,2025

[202] Lin T Y, Goyal P, Girshick R, et al. Focal loss for dense object detection//Proceedings of the IEEE international conference on computer vision. Venice, Italy, 2017: 2980-2988

[203] Jain S, Wallace B C. Attention is not explanation. arXiv preprint arXiv:1902.10186,2019

[204] Chu Z, Wan Y, Li Q, et al. Graph neural networks for vulnerability detection: A counterfactual explanation//Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. Vienna, Austria, 2024: 389-401

[205] Zhang W, Zhang J. Hallucination mitigation for retrieval-augmented large language models: a review. Mathematics, 2025, 13(5):856

[206] Zhang Z, Wang C, Wang Y, et al.LLMhallucinations in practical code generation: Phenomena, mechanism, and mitigation. Proceedings of the ACM on Software Engineering, 2025, 2(ISSTA):481-503

[207] Jiang Y, Huang S, Treude C, et al. Shield broken: black-box adversarial attacks on LLM-based vulnerability detectors. IEEE Transactions on Software Engineering, 2025:1-19

[208] Jiang Y, Li Z, Huang S, et al. Effective code membership inference for code completion models via adversarial prompts. arXiv preprint arXiv:2511.15107,2025

[209] Akheel S. Guardrails for large language models: A review of techniques and challenges. Journal of Artificial Intelligence, Machine Learning and Data Science, 2025, 3(1):2504-2512

[210] Wong W K, Wu D, Wang H, et al. Decllm: Llm-augmented recompilable decompilation for enabling programmatic use of decompiled code. Proceedings of the ACM on Software Engineering, 2025, 2 (ISSTA):1841-1864

[211] Tan H, Luo Q, Li J, et al. Llm4decompile: Decompiling binary code with large language models. arXiv preprint arXiv: 2403.05286,2024

[212] Hu P, Liang R, Chen K. Degpt: Optimizing decompiler output with llm//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2024

[213] Liu P, Sun C, Zheng Y, et al. Llm-powered static binary taint analysis. ACM Transactions on Software Engineering and Methodology, 2025, 34(3):1-36

[214] Wang R, Li K, Xu M, et al. Lift: Automating symbolic execution optimization with large language models for ai networks// Proceedings of the 2nd Workshop on Networks for AI Computing. Coimbra, Portugal, 2025: 37-42

[215] Hussain N, Chen H, Tran C, et al. Vulbinllm: Llm-powered vulnerability detection for stripped binaries. arXiv preprint arXiv: 2505.22010,2025

[216] Manuel D, Islam N T, Khoury J, et al. Enhancing reverse engineering: Investigating and benchmarking large language models for vulnerability analysis in decompiled binaries. arXiv preprint arXiv: 2411.04981,2024

[217] Shang X, Chen G, Cheng S, et al. Binmetric: A comprehensive binary analysis benchmark for large language models. arXiv preprint arXiv: 2505.07360,2025

JIANG Yuan, Ph. D., associate researcher. His main research interests include code vulnerability detection, software security, and LLM security.

YANG Zi-Han, M. S. candidate. His main research interests include code vulnerability detection, jailbreak attacks, and prompt engineering.

SU Xiao-Hong, Ph. D., professor. Her main research interests include intelligent software engineering, software vulnerability detection, and intelligent software development and testing.

HUANG Shan, undergraduate student. His main research interests include vulnerability detection, adversarial attacks, and model security.

WANG Tian-Tian, Ph. D., associate professor. Her main research interests include model-based systems engineering and program repair.

Background

Software vulnerabilities are defects in design, implementation, or runtime environments that can be exploited to undermine system security and reliability. As software systems continue to grow in scale and complexity, the need for automated vulnerability detection has become increasingly prominent in modern software engineering. In recent years, deep learning (DL)-based methods have been widely studied, leading to notable improvements in detection effectiveness and scalability.

While several surveys have reviewed DL-based approaches, most focus on conventional model architectures such as CNNs, LSTMs, and GNNs. In contrast, the application of large language models (LLMs) to vulnerability detection has received limited systematic attention. Recently, Transformer-based LLMs have demonstrated strong capabilities in code understanding and have been gradually adapted to vulnerability detection tasks through instruction tuning, prompt engineering, and in-context learning. Empirical studies suggest that LLMs may enable new detection paradigms beyond the capacities of traditional DL models.

To bridge this gap, this paper presents a comprehensive and structured survey of LLM-based vulnerability detection techniques, covering a wide range of models from lightweight Transformer variants to high-capacity general-purpose LLMs. We organize existing methods along key dimensions, including model design, detection granularity, and application scenarios, and provide comparative analyses that highlight differences in architectural choices and performance characteristics. Moreover, we identify major technical challenges and outline future research opportunities to guide the development of next-generation LLM-based detection frameworks.

This work was supported by the National Natural Science Foundation of China under the Young Scientists Fund (No. 62302125) and the General Program (No. 62272132). Both projects focus on advancing deep learning and large language model techniques for software vulnerability detection, aiming to develop practical and high-performance detection solutions. We have published a number of research papers and patents centered on these directions.