计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
2026,35(1):255−262 [doi: 10.15888/j.cnki.csa.010079] [CSTR: 32024.14.csa.010079]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

不同提示语言对 LLM 生成代码安全性的影响①

郭亚楠[1], 马瑞强[1,2], 崔　旭[1], 杨皓然[1]

[1](内蒙古工业大学 数据科学与应用学院, 呼和浩特 010051)
[2](明治大学, 东京 164-8525)
通信作者: 马瑞强, E-mail: marq@imut.edu.cn

摘　要: 近年来, 随着软件开发领域代码规模持续扩大、功能需求与系统架构日趋复杂, 自动化工具已成为提升开发效率的核心支撑, 其中生成式大语言模型 (large language model, LLM) 在软件开发领域得到了广泛应用. 尽管 LLM 能显著缩短开发周期, 但研究表明, LLM 生成的代码常存在安全漏洞, 可能引发系统风险. 已有研究发现 LLM 的输出结果会受提示语言差异的影响, 然而针对不同提示语言生成代码安全性的研究与评价仍较为匮乏. 本文旨在探究输入英文提示语言与中文提示语言对代码安全性的影响, 并对其进行系统性评估. 通过在 LLM 中分别使用英文和中文文本指示代码生成, 分析生成代码的安全性差异. 实验结果表明, 当以英文作为提示语言输入时, LLM 生成的代码中出现的漏洞和 bug 数量相对较少; 而以中文作为提示语言输入时, 生成的代码则更容易出现安全漏洞.

关键词: 大语言模型; 安全编码; 提示语言; 代码生成; 评估

引用格式: 郭亚楠,马瑞强,崔旭,杨皓然.不同提示语言对 LLM 生成代码安全性的影响.计算机系统应用,2026,35(1):255–262. http://www.c-s-a.org.cn/1003-3254/10079.html

Impact of Different Prompt Languages on Security of Code Generated by LLM

GUO Ya-Nan[1], MA Rui-Qiang[1,2], CUI Xu[1], YANG Hao-Ran[1]

[1](College of Intelligent Science and Technology, Inner Mongolia University of Technology, Hohhot 010051, China)
[2](Meiji University, Tokyo 164-8525, Japan)

Abstract: In recent years, with the continuous expansion of code scale and the increasing complexity of functional requirements and system architectures in the field of software development, automated tools have become a core support for improving development efficiency. Among these tools, generative large language models (LLMs) have been widely applied in software development. Although LLMs can significantly shorten the development cycle, studies have shown that the code they generate often contains security vulnerabilities. These vulnerabilities may pose risks to the system.Existing research has found that the outputs of LLMs are influenced by differences in prompt languages. However, research and evaluation on the security of code generated using different prompt languages remain insufficient. This study aims to explore the impact of English and Chinese input prompt languages on code security and to conduct a systematic evaluation of this impact. Specifically, English and Chinese texts are used to instruct code generation in LLMs, and the differences in the security of the generated code are analyzed.The experimental results indicate that when English is used as the prompt language, the number of vulnerabilities and bugs in the code generated by LLMs is relatively smaller. In contrast, when Chinese is used as the prompt language, the generated code is more prone to security vulnerabilities.

Key words: large language model (LLM); secure coding; prompt language; code generation; evaluation

① 基金项目: 内蒙古自治区重点研发与成果转化计划 (2025YFSH0157); 专创融合“Python 语言与数据分析”建设 (ZC2024045)
收稿时间: 2025-03-06; 修改时间: 2025-08-22; 采用时间: 2025-09-22; csa 在线出版时间: 2025-11-26
CNKI 网络首发时间: 2025-11-27

© 中国科学院软件研究所 http://www.c-s-a.org.cn

近年来, 大语言模型 (large language model, LLM) 在学术界引起了广泛关注. LLM 是基于深度学习框架构建的新一代自然语言处理技术, 其核心在于通过大规模无标注文本数据的预训练, 掌握语言的语法规则、语义逻辑与领域知识, 并具备基于上下文生成连贯、符合任务需求的文本内容的能力. 例如, ChatGPT 是由 OpenAI 开发的一种基于 Transformer 架构和无监督预训练技术的对话生成模型. 该模型通过自注意力机制 (self-attention) 捕捉输入序列的上下文信息, 并借助大规模无监督预训练学习语言知识[1]. 即使不具备任何编程经验的用户, 也能够根据自己的需求, 通过向 ChatGPT 输入相应的提示词, 生成符合自己需求的代码片段[2], 从而显著提升开发效率.

在代码生成过程中, 代码安全性是大语言模型应用中不可忽视的关键因素. 代码安全性旨在确保由 LLM 生成的代码在使用和部署过程中不会引入安全漏洞或潜在风险[3], 从而有效防止恶意攻击、错误及漏洞的影响, 保证软件系统的稳定性、可靠性和安全性. 具体而言, 代码安全性涵盖完整性、保密性、可用性以及抵御各类安全威胁的能力. 此外, 在开发早期阶段发现并修复安全问题, 能够降低系统后期维护成本. 因此, 在软件开发过程中, 代码安全性对于保障软件长期稳定运行及用户数据安全具有至关重要的意义.

鉴于 LLM 具备高效生成代码的能力[4], 本文将其作为研究对象. LLM 能够根据自然语言描述的功能需求生成相应的代码, 从而显著减轻软件开发人员的工作负担. 然而, 程序员在使用 LLM 生成代码时, 对其生成代码的安全性存在一定程度的担忧.

1 相关研究现状

自动生成代码工具通常基于预训练模型, 如 OpenAI 的模型, 这些模型通常在大量公开代码数据集上训练. 然而, 这种训练方式可能引发一系列问题, 包括版权合规性争议以及潜在的安全漏洞. 因此, 众多研究人员开始关注利用大语言模型进行软件开发时生成代码的安全性及其实际应用情况[5]. Pearce 等人[6]通过对比实验, 对自动生成代码工具生成的代码片段与程序员手动编写的代码进行了系统性分析与评估, 研究结果表明, 自动生成代码工具生成的代码可能存在安全隐患. Kavian 等人[7]也提出 LLM 在安全领域能力有限, 为此研究提出开源框架 LLMSecGuard, 通过静态代码分析工具与 LLM 的协同提升代码安全性.

Perry 等人[8]通过对比实验发现, 能够使用 AI 助手的开发者更倾向于认为其编写的代码是安全的. 然而, 实验结果表明, 未使用 AI 助手的参与者编写的代码中安全漏洞的数量反而更少. LLM 的普及进一步加剧了这一问题, 因为其驱动的代码生成功能不仅被经验丰富的开发者广泛使用, 甚至也被新手及无编程经验的用户所使用. 针对这一问题, Liu 等人[9]对 ChatGPT 在代码生成任务中的正确性进行了分析, 结果表明, 尽管 ChatGPT 能够为多种编程任务生成功能代码, 但其生成的代码常存在质量缺陷, 包括错误编译、输出错误、可维护性差以及性能低下等问题. Siddiq 等人[10]提出了一个名为 SALLM 的框架, 用于系统性地评估 LLM 生成安全代码的能力. Sandoval 等人[11]的研究指出, 对于 C 语言等低级编程语言, LLM 生成的代码并未显著增加安全错误的发生率. 而对于其他的编程语言, Khoury 等人[12]通过使用 ChatGPT 生成相同程序的 5 种不同的编程语言版本, 并对生成的代码进行安全性评估, 发现经过适当的引导和调整后, ChatGPT 在大多数用例中能够生成相对安全的代码. 此外, Etxaniz 等人[13]针对大语言模型在非英文提示下的性能差异, 提出在 LLM 内部引入一个自翻译过程, 即将翻译后的数据重新输入模型以解决问题. 研究结果表明, 当提示语言为非英文时, LLM 的性能可能表现出显著差异.

尽管已有研究探讨了提示语言对 LLM 生成代码的影响, 但先前的研究尚未充分验证不同提示语言对生成代码安全性的具体影响, 且相关研究主要集中于以英文作为输入语言的情景. 然而, 对于中国用户而言, 中文可能成为主要的提示语言. 因此, 本研究旨在探究提示语言差异对代码安全性的影响, 通过在 ChatGPT、豆包、文心一言中分别使用英文和中文两种命令形式生成相同任务的代码片段来展开系统性研究.

2 实验方法设计

2.1 数据集

本研究所使用的数据集来源于 LeetCode 官方网站提供的在线题库[14]. 在数据收集过程中, 我们依据平台的官方难度标注 (简单、中等、困难) 对题目进行分类, 并分别采集样本. 经统计, 最终获得的数据集包含简单难度题目 602 条、中等难度题目 1 156 条以及困难题目 456 条 (统计截至 2025 年 3 月, 具体题目分布

© 中国科学院软件研究所 http://www.c-s-a.org.cn

见表 1). 每条数据均包含题目名称、题目描述及测试用例, 其中测试用例明确规定了输入参数及其对应的输出结果[15–17].

表 1 LeetCode 官网题目

题目	题解	通过率 (%)	难度
两数之和	24635	53.7	简单
两数相加	13690	43.6	中等
无重复字符的最长子串	15855	39.8	中等
寻找两个正序数组的中位数	7809	42.2	困难

考虑到本研究涉及中文输入与英文输入的对比分析, 我们在数据清洗阶段对爬取的中文题目文本进行了规范化处理, 并通过双语对照翻译工具及人工校对相结合的方式, 确保中英文版本在语义和格式上的一致性. 最终构建的数据集由中文部分与英文部分组成, 可用于后续实验的跨语言对比与分析.

2.2 实验设计

本实验旨在评估 LLM 生成代码的安全性, 重点探究以中文和英文分别作为提示语言输入对所生成代码安全性的影响. 实验的总体流程图如图 1 所示.

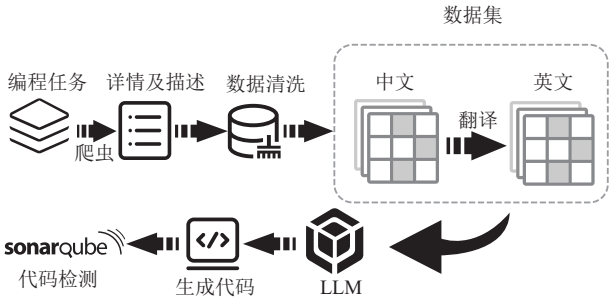

图 1 总体流程图

在本实验中, 首先通过爬取获取每个编程任务的详细信息及描述, 随后对数据进行清洗处理. 经过以上步骤, 得到包含编程任务及其描述的若干题目, 并将中文编程任务翻译为英文, 最终构建由中文和英文两部分构成的数据集. 接下来进入代码生成阶段, 将数据集依次输入到 LLM 中, 使其根据编程任务生成对应的 Python 和 Java 代码. 最后, 对生成的代码段进行检测与评估, 分析不同语言输入对生成代码安全性的影响[18].

2.3 代码生成

本研究中通过 API 调用 LLM 实现代码的批量生成, 以提高实验效率[19]. 在此之前, 数据集已进行分类并存储至相应文件中. 在代码生成阶段, 系统依次读取每个题目的描述, 并通过调用 ChatGPT-3.5、ChatGPT-4o、豆包、文心一言等来生成对应代码. 以 ChatGPT-3.5 为例, 该阶段共生成了 602 段简单难度的代码段、1 156 段中等难度的代码段以及 456 段困难难度的代码段. 生成的代码如图 2 和图 3 所示.

```
def longestCommonPrefix(strs):
    if not strs:
        return ""

    prefix = strs[0]
    for i in range(1, len(strs)):
        while strs[i].find(prefix) != 0:
            prefix = prefix[:-1]
            if not prefix:
                return ""

    return prefix

# 示例测试
strs1 = ["flower","flow","flight"]
strs2 = ["dog","racecar","car"]
print(longestCommonPrefix(strs1)) # 输出: "fl"
print(longestCommonPrefix(strs2)) # 输出: ""
```

图 2 ChatGPT-3.5 生成的 Python 代码片段

```
def longestCommonPrefix(strs):
    if not strs:
        return ""

    prefix = strs[0]
    for i in range(1, len(strs)):
        while strs[i].find(prefix) != 0:
            prefix = prefix[:-1]
            if not prefix:
                return ""

    return prefix

# 示例测试
strs1 = ["flower","flow","flight"]
strs2 = ["dog","racecar","car"]
print(longestCommonPrefix(strs1)) # 输出: "fl"
print(longestCommonPrefix(strs2)) # 输出: ""
```

图 3 ChatGPT-3.5 生成的 Java 代码片段

2.4 安全漏洞分析策略与缓解措施

在本研究中, 为保障静态代码分析工具检测结果的准确性、透明性及公平性, 本研究采用以下策略以降低误报与漏报的潜在影响, 并且确保评估过程的透明度. 具体如表 2 所示.

表 2 SonarQube 参数配置

配置项	具体内容
工具版本	SonarQube 10.4
规则集选择	官方推荐规则集
分析范围	整体代码分析
重复率阈值	≥3%
告警去重策略	基于规则 ID+文件位置去重
漏洞分级标准	默认5级
误报控制	抽样人工验证

© 中国科学院软件研究所 http://www.c-s-a.org.cn

3 评估方法

本研究采用静态代码分析工具对代码进行分析. 静态代码分析工具是一种在不执行代码的情况下, 通过工具或人工方式检查源代码以识别潜在安全漏洞和代码质量问题的技术手段. 静态分析工具能够在代码开发的早期阶段发现安全漏洞和编码错误, 从而降低修复成本. 此外, 该工具能够自动扫描大量代码, 其效率显著高于人工审查. 同时, 静态分析工具提供了一致且全面的检查机制, 有效避免了人工审查中可能出现的疏漏[20].

本研究采用 SonarQube 进行代码分析. 在使用 SonarQube 时, 开发人员首先在集成开发环境 (IDE) 中完成编码, 随后将代码推送至相应的源代码管理 (SCM) 系统中. 在代码分析过程中, 系统会自动触发构建流程, 并执行 SonarScanner 以完成 SonarQube 分析所需的准备工作. 分析完成后, 代码分析报告将被发送至 SonarQube 服务器进行处理, 并将结果存储于 SonarQube 数据库中. 开发人员可通过 Web 端查看检测结果, 并基于分析报告评估代码的安全性及其他相关指标. 具体工作流程如图 4 所示. 本研究主要从代码的可靠性、安全性、可维护性及重复率这 4 个方面展开分析和讨论.

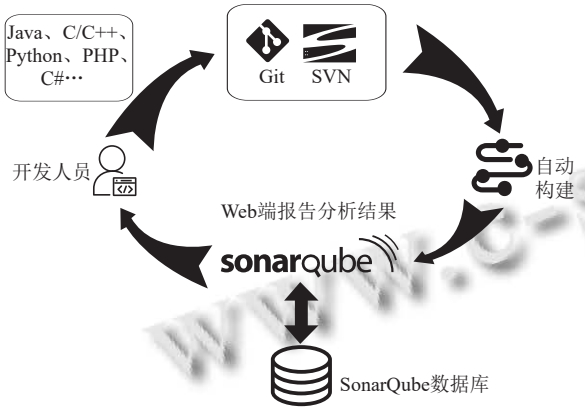

图 4 SonarQube 流程图

3.1 可靠性

可靠性通过代码中 bug 评级的计算方法进行评估, 即根据检测到的 bug 数量确定可靠性等级. 可靠性等级划分为 A–E 这 5 个级别: A 级代表代码中不存在问题, 是代码检测中的最高级别; B 级表示代码中存在一个次要 bug; C 级表示代码中存在一个重要 bug; D 级表示代码中存在一个严重 bug; E 级表示代码中存在一个阻断 bug, 为最低级别. 关于 bug 级别的具体定义如表 3 所示.

表 3 bug 级别定义

级别	详细描述信息
次要 (Minor)	界面、性能缺陷, 建议类问题, 不影响操作功能的执行, 可以优化性能的方案等.
重要 (Major)	功能没有完全实现但是不影响使用, 功能菜单存在缺陷但不会影响系统稳定性.
严重 (Critical)	系统主要功能部分丧失、数据库保存调用错误、功能设计与需求严重不符、模块无法启用等.
阻断 (Blocker)	阻碍开发或测试工作的问题, 造成系统崩溃、死机, 与数据库连接错误, 主要功能丧失等.

3.2 安全性

安全性通过代码中漏洞评级的计算方法进行评估, 即根据检测到的漏洞数量确定安全性等级. 安全性等级同样划分为 A–E 这 5 个级别: A 级表示代码中不存在漏洞, 是代码检测中的最高级别; B 级表示代码中存在一个次要漏洞; C 级表示代码中存在一个重要漏洞; D 级表示代码存在一个严重漏洞; E 级表示代码中存在一个阻断漏洞, 为最低级别. 关于漏洞级别的具体定义如表 4 所示.

表 4 漏洞级别定义

级别	详细描述信息
次要 (Minor)	能够获取一些数据, 但不属于核心数据的操作; 需要用户交互才可以触发的漏洞等.
重要 (Major)	需要在一定条件限制下, 能获取服务器权限、网站权限与核心数据库数据的操作等.
严重 (Critical)	直接获取普通系统权限的漏洞; 严重的逻辑设计缺陷和流程缺陷; 严重的权限绕过类漏洞等.
阻断 (Blocker)	直接获取重要服务器(客户端)的漏洞; 直接导致严重的信息泄露漏洞.

3.3 可维护性

可维护性是指代码在一定时间和成本范围内被修改、理解及扩展的难易程度. 在本研究中, 可维护性被划分为 A–E 这 5 个等级, 分别表示从极佳到极差的可维护性水平. 这一划分基于技术债务比率 (technical debt ratio, *TDR*) 的计算结果. 技术债务是指修复代码中所有问题所需的时间成本, SonarQube 通过计算每个问题的修复时间并累加, 得到整个项目的技术债务总量. 技术债务比率的计算公式如下:

$$TDR = \frac{Tec_{\mathrm{D}}}{Dev_{\mathrm{C}}} \tag{1}$$

© 中国科学院软件研究所 http://www.c-s-a.org.cn

其中, Tec_D 表示修复项目中所有代码所需的成本, Dev_C 表示重新开发该项目的总成本. 可维护性等级划分如下: A 级表示技术债务比率≤0.05; B 级表示 0.05< 技术债务比率≤0.1; C 级表示 0.1<技术债务比率≤0.2; D 级表示 0.2<技术债务比率≤0.5; E 级表示 0.5<技术债务比率.

3.4 重复率

重复率 (duplication rate, *DR*) 是用于量化代码重复程度的指标, 代码重复是指在代码库中存在相同或高度相似的代码片段. 重复率的计算公式如下:

$$DR = \frac{Dup_L}{L} \times 100\% \tag{2}$$

其中, Dup_L 表示重复行数, L 表示总行数.

4 实验结果

本实验从可靠性、安全性、可维护性及重复率这 4 个方面对生成的 Python 代码和 Java 代码进行评估. 这些评估指标旨在全面衡量自动生成代码的质量, 并揭示生成代码在不同编程语言中的表现差异. 具体评估结果如下.

4.1 Python 代码

对生成的 Python 代码进行总体分析的结果如表 5 所示.

表 5 中 A–E 分别代表可靠性、安全性及可维护性的等级, 括号内的数值表示问题代码的数量. 例如, 可靠性为 C (3) 表示代码中存在 3 个重要 bug, 安全性和可维护性的评级方式与之相同. 表 5 显示, ChatGPT 生成的 Python 代码在安全性方面表现优异, 不同难度代码的安全性均达到最高评级, 其他 LLM 略有不同. 然而, 在可靠性方面, 生成代码的表现存在差异, 以 ChatGPT-3.5 为例, 对于简单代码, 中文生成的 Python 代码可靠性略高于英文生成的代码; 而在中等和困难代码中, 英文生成的 Python 代码在可靠性上略优于中文生成的代码. 在可维护性方面, 括号中的数值表示需要维护的代码行数, 需要维护的代码行数越少, 表明代码质量越高. 根据表 5 数据, 英文生成的 Python 代码在可维护性方面普遍优于中文生成的代码. 最后, 在代码重复率方面, 与可靠性结果类似, 中等和困难代码中, 英文生成代码的重复率略低于中文生成的代码. 并且对于该实验, 在多个大语言模型中得到同样的结果.

表 5 Python 代码的总体检测结果

LLM	难易程度	语言	可靠性	安全性	可维护性	重复率 (%)
ChatGPT-3.5	简单 (Easy)	中文	A (2)	A (0)	A (525)	4.9
		英文	C (6)	A (0)	A (311)	6.2
	中等 (Moderate)	中文	E (3)	A (0)	A (1 400)	6.5
		英文	C (3)	A (0)	A (988)	3.3
	困难 (Difficult)	中文	E (3)	A (0)	A (601)	3.6
		英文	C (3)	A (0)	A (470)	3.1
ChatGPT-4o	简单 (Easy)	中文	C (2)	A (0)	A (432)	3.2
		英文	B (1)	A (0)	A (354)	2.8
	中等 (Moderate)	中文	D (2)	A (0)	A (1 342)	5.4
		英文	C (3)	A (0)	A (1 012)	3.5
	困难 (Difficult)	中文	D (3)	A (0)	A (543)	4.3
		英文	C (2)	A (0)	A (485)	4.1
豆包	简单 (Easy)	中文	C (6)	C (5)	A (635)	6.4
		英文	C (2)	C (4)	A (313)	6.2
	中等 (Moderate)	中文	D (3)	C (34)	A (1 688)	6.2
		英文	B (3)	C (30)	A (935)	4.5
	困难 (Difficult)	中文	E (3)	C (22)	A (567)	3.5
		英文	C (3)	C (21)	A (475)	3.1
文心一言	简单 (Easy)	中文	C (2)	C (12)	A (324)	4.4
		英文	C (1)	C (11)	A (307)	4.3
	中等 (Moderate)	中文	B (2)	C (43)	A (1 542)	5.8
		英文	C (1)	C (37)	A (989)	5.4
	困难 (Difficult)	中文	C (3)	C (32)	A (643)	3.8
		英文	D (4)	C (30)	A (433)	3.4

然后对检测出的问题代码进行分类统计. 本实验将问题代码划分为 4 个等级 (阻断、严重、重要、次要), 其严重性根据对软件质量的影响程度确定, 统计单位为问题代码的行数. 问题分类的具体定义如下.

(1) 阻断问题: 可能导致程序崩溃或引发严重错误的代码问题.

(2) 严重问题: 可能严重影响应用程序行为或引发安全漏洞的代码问题, 如内存泄漏、SQL 注入等.

(3) 重要问题: 可能显著影响开发人员工作效率的质量缺陷, 如一段未覆盖的代码等.

(4) 次要问题: 可能轻微影响开发人员工作效率的质量缺陷, 如代码行过长等.

如表 6 所示, 实验结果揭示了两个值得注意的特殊数据点. 首先, 在 ChatGPT-3.5 处理简单代码时, 英文输入生成的代码中阻断问题和严重问题的出现频率较高. 这表明在生成基础代码片段时, 英文输入提示的稳定性与有效性可能不及中文输入提示. 其次, 对于复杂和困难代码, 无论是中文输入提示还是英文输入提

© 中国科学院软件研究所 http://www.c-s-a.org.cn

示, 生成的代码在阻断问题和严重问题方面的表现趋于一致, 表明在复杂代码场景下, 两种语言提示的效果差异较小. 此外, 其他实验结果表明, 英文输入提示生成的代码在整体上问题更少, 尤其是在中等复杂度代码中, 以 ChatGPT-3.5 为例, 如图 5 所示, 英文输入生成的代码问题行数显著低于中文输入生成的代码问题行数, 进一步证明了英文输入在生成中等复杂度代码时的优势.

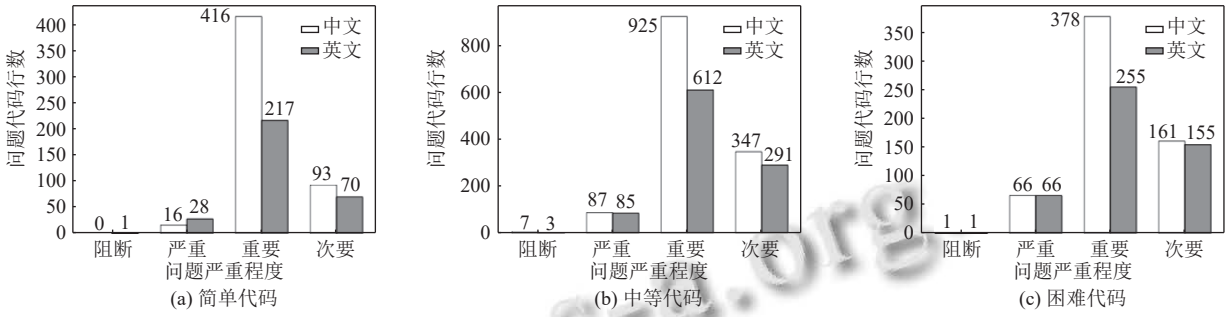

图 5 ChatGPT-3.5 生成不同难度的问题代码对比 (Python)

表 6 问题代码的严重程度 (Python)

LLM	难易程度	语言	阻断	严重	重要	次要
ChatGPT-3.5	简单 (Easy)	中文	0	16	416	93
		英文	1	28	217	70
	中等 (Moderate)	中文	7	87	925	347
		英文	3	85	612	291
	困难 (Difficult)	中文	1	66	378	161
		英文	1	66	255	155
ChatGPT-4o	简单 (Easy)	中文	1	12	413	75
		英文	0	11	243	54
	中等 (Moderate)	中文	5	54	837	324
		英文	2	45	824	267
	困难 (Difficult)	中文	2	55	367	143
		英文	1	50	253	123
豆包	简单 (Easy)	中文	2	24	524	243
		英文	1	20	294	139
	中等 (Moderate)	中文	6	98	945	534
		英文	2	88	723	387
	困难 (Difficult)	中文	4	54	315	418
		英文	2	53	234	249
文心一言	简单 (Easy)	中文	4	31	521	243
		英文	2	29	326	213
	中等 (Moderate)	中文	7	76	834	457
		英文	4	76	655	239
	困难 (Difficult)	中文	3	64	335	142
		英文	1	61	258	118

4.2 Java 代码

对生成的 Java 代码进行总体分析的结果如表 7 所示.

表 7 中 A–E 分别代表可靠性、安全性及可维护性的等级, 括号内的数值表示问题代码的行数. 表 7 中首先展示了几项值得关注的特殊数据, 包括简单代码的安全性及重复率的对比结果, 以及复杂代码的可维护性对比结果. 在这 3 项中, 中文生成的 Java 代码在安全性方面略优于英文生成的代码, 但差距非常微小, 可认为两者表现基本持平. 其次, 在其他场景的对比结果中, 实验数据普遍显示, 英文生成的代码在安全性方面显著优于中文生成的代码. 特别是在复杂代码场景中, 英文输入提示生成的代码在减少潜在漏洞和优化安全性方面表现更为突出, 进一步验证了英文提示在保障代码安全性方面的优势.

在对检测出的问题代码进行统计时, 采用相同的等级分类标准, 具体结果如表 8 所示. 表 8 展示了一些特殊的数据点. 首先, 以 ChatGPT-3.5 为例在处理中等复杂度代码的阻断问题时, 实验结果表明, 英文输入生成的代码在阻断问题方面的代码行数略高于中文输入生成的代码. 其次, 对于困难代码中的严重、重要及次要问题, 英文输入生成的代码问题行数总体上多于中文输入生成的代码, 尽管次要问题的差距较为显著, 但其他类别的问题行数差异较小. 总体而言, 英文输入提示生成的代码在其他场景中通常表现出较少的问题代码. 图 6 进一步表明, 除困难代码中英文输入生成的代码问题行数较多外, 在其他数据点上, 中文输入生成的代码问题行数普遍高于英文输入生成的代码. 这些结果进一步说明, 尽管中文输入在部分复杂代码场景下表现更优, 但在处理某些特定类型的代码问题时, 英文输入生成的代码也展现出其独特的优势.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

表 7 Java 代码的总体检测结果

LLM	难易程度	语言	可靠性	安全性	可维护性	重复率 (%)
ChatGPT-3.5	简单 (Easy)	中文	E (50)	E (23)	B (2600)	12.2
		英文	E (37)	E (24)	B (2200)	12.8
	中等 (Moderate)	中文	E (66)	E (18)	A (4900)	12.2
		英文	E (38)	E (12)	A (4400)	9.9
	困难 (Difficult)	中文	E (14)	E (5)	A (1900)	15.6
		英文	E (7)	E (1)	A (2000)	10.1
ChatGPT-4o	简单 (Easy)	中文	E (43)	E (21)	A (2100)	12.1
		英文	E (32)	E (18)	A (1800)	11.8
	中等 (Moderate)	中文	E (56)	E (13)	A (3400)	10.3
		英文	E (33)	E (12)	A (2400)	9.7
	困难 (Difficult)	中文	E (12)	E (2)	A (2300)	14.8
		英文	E (5)	E (1)	A (1200)	10.2
豆包	简单 (Easy)	中文	E (30)	E (32)	A (2400)	13.1
		英文	C (27)	E (15)	A (1900)	11.2
	中等 (Moderate)	中文	E (68)	E (43)	A (3800)	15.6
		英文	E (33)	E (34)	A (2400)	11.2
	困难 (Difficult)	中文	E (12)	E (21)	A (2100)	13.4
		英文	C (27)	E (17)	A (1700)	9.2
文心一言	简单 (Easy)	中文	E (43)	E (11)	A (3300)	14.8
		英文	E (22)	E (7)	A (2300)	12.4
	中等 (Moderate)	中文	E (63)	E (33)	A (3400)	16.1
		英文	E (41)	E (31)	A (2000)	14.9
	困难 (Difficult)	中文	E (34)	E (26)	A (4100)	12.6
		英文	E (17)	E (7)	A (2400)	9.9

表 8 问题代码的严重程度 (Java)

LLM	难易程度	语言	阻断	严重	重要	次要
ChatGPT-3.5	简单 (Easy)	中文	63	55	1470	1115
		英文	53	49	1194	945
	中等 (Moderate)	中文	35	112	2661	2211
		英文	39	109	2383	1937
	困难 (Difficult)	中文	13	41	993	851
		英文	2	43	1003	938
ChatGPT-4o	简单 (Easy)	中文	35	23	1348	984
		英文	23	11	1084	782
	中等 (Moderate)	中文	27	56	2435	1984
		英文	23	35	2105	1402
	困难 (Difficult)	中文	17	54	1329	730
		英文	11	50	1003	638
豆包	简单 (Easy)	中文	67	25	2313	1749
		英文	54	23	2220	1092
	中等 (Moderate)	中文	34	97	3245	2937
		英文	29	89	2875	1873
	困难 (Difficult)	中文	45	23	2310	2374
		英文	38	10	1974	1784
文心一言	简单 (Easy)	中文	44	38	2201	1843
		英文	27	29	1875	1745
	中等 (Moderate)	中文	78	89	3028	2132
		英文	43	78	2102	1764
	困难 (Difficult)	中文	34	63	1349	973
		英文	18	32	998	437

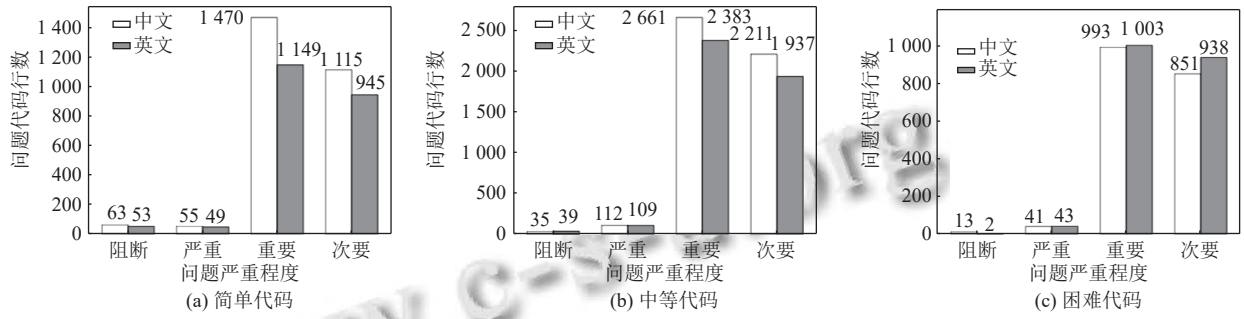

图 6 ChatGPT-3.5 生成不同难度的问题代码对比 (Java)

5 结论

本研究系统评估了大型语言模型 (LLM) 在英文与中文提示下生成代码的安全性表现. 实验结果表明, 当输入提示为英文时, LLM 所生成的代码普遍表现出更高的安全性; 而在中文语境下, 生成代码中潜在的安全漏洞相对增多. 这一差异可能与当前主流 LLM 训练语料中英文编程资源占主导地位有关. 此类资源通常涵盖大量结构化的安全编程范例与最佳实践, 而非英文编程语料在规模与质量上仍显不足, 导致模型在应对非英文提示时缺乏足够的参考依据, 从而影响其生成代码的可靠性.

为提升 LLM 在跨语言场景下的代码生成安全性, 建议从训练数据的多样性与代表性入手, 系统引入更多高质量的非英文编程语料, 尤其应增强安全编程相关示例的覆盖. 此外, 在模型优化过程中应明确纳入跨语言一致性约束, 加强多语言对齐能力, 使其在不同语言输入下均可输出符合同等安全标准的代码.

参考文献

1 张长琳, 仝鑫, 佟晖, 等. 面向网络安全领域的大语言模型

© 中国科学院软件研究所 http://www.c-s-a.org.cn

技术综述. 信息网络安全, 2024, 24(5): 778–793.
2 王树义, 张庆薇. ChatGPT 给科研工作者带来的机遇与挑战. 图书馆论坛, 2023, 43(3): 109–118.
3 Ambati SH. Security and authenticity of AI-generated code [Master's Thesis]. Saskatoon: University of Saskatchewan, 2023.
4 王耀祖, 李擎, 戴张杰, 等. 大语言模型研究现状与趋势. 工程科学学报, 2024, 46(8): 1411–1425.
5 Jiang JY, Wang F, Shen JS, *et al*. A survey on large language models for code generation. ACM Transactions on Software Engineering and Methodology. [2025-02-13]. [doi: 10.1145/3747588]
6 Pearce H, Ahmad B, Tan B, *et al*. Asleep at the keyboard? Assessing the security of GitHub copilot's code contributions. Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2022. 754–768.
7 Kavian A, Pourhashem Kallehbasti MM, Kazemi S, *et al*. LLM security guard for code. Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering. Salerno: ACM, 2024. 600–603.
8 Perry N, Srivastava M, Kumar D, *et al*. Do users write more insecure code with AI assistants? Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. Copenhagen: ACM, 2023. 2785–2799.
9 Liu Y, Le-Cong T, Widyasari R, *et al*. Refining ChatGPT-generated code: Characterizing and mitigating code quality issues. ACM Transactions on Software Engineering and Methodology, 2024, 33(5): 116.
10 Siddiq ML, Santos JCS, Devareddy S, *et al*. Generate and pray: Using SALLMS to evaluate the security of LLM generated code. arXiv:2311.00889, 2023.
11 Sandoval G, Pearce H, Nys T, *et al*. Lost at C: A user study on the security implications of large language model code assistants. Proceedings of the 32nd USENIX Conference on Security Symposium. Anaheim: USENIX Association, 2023. 124.
12 Khoury R, Avila AR, Brunelle J, *et al*. How secure is code generated by ChatGPT? Proceedings of the 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Honolulu: IEEE, 2023. 2445–2451.
13 Etxaniz J, Azkune G, Soroa A, *et al*. Do multilingual language models think better in English? Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Mexico City: Association for Computational Linguistics, 2024. 550–564.
14 LeetCode. LeetCode Platform. https://leetcode.cn/problemset. [2024-08-26].
15 Coignion T, Quinton C, Rouvoy R. A performance study of LLM-generated code on Leetcode. Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering. Salerno: ACM, 2024. 79–89.
16 Wang L, Shi CQ, Du SS, *et al*. Performance review on LLM for solving Leetcode problems. Proceedings of the 4th International Symposium on Artificial Intelligence and Intelligent Manufacturing (AIIM). Chengdu: IEEE, 2024. 1050–1054.
17 Lai U. ChatGPT's code suggestion accuracy evaluation [Master's Thesis]. LAB University of Applied Sciences, 2024.
18 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.
19 Chen YJ, Gao CY, Zhu MJ, *et al*. APIGen: Generative API method recommendation. arXiv:2401.15843, 2024.
20 花子涵, 杨立, 陆俊逸, 等. 代码审查自动化研究综述. 软件学报, 2024, 35(7): 3265–3290. [doi: 10.13328/j.cnki.jos.007112]

(校对责编: 李慧鑫)

© 中国科学院软件研究所 http://www.c-s-a.org.cn