

基于 DeepSeek Coder 大语言模型的软件漏洞自动识别与 CWE 分类研究

张家豪

摘要: 随着软件系统在现代社会中的重要性日益增加, 软件安全已成为一个亟待解决的关键问题。本研究提出了一种基于 DeepSeek Coder 大型语言模型的创新方法, 用于自动识别软件漏洞并将其分类为相应的 Common Weakness Enumeration (CWE) 类型。我们使用了来自 gitlink 平台的大规模数据集, 包含 147,863 个训练样本、18,483 个验证样本和 18,483 个测试样本。通过精心设计的数据预处理、模型训练和推理流程, 我们的方法在多个评估指标上都取得了优异的成绩。实验结果显示, 该模型达到了 87.45% 的准确率、86.32% 的精确率、85.91% 的召回率和 86.11% 的 F1 分数。此外, 0.8503 的 Matthews 相关系数(MCC)进一步证实了模型预测的可靠性。值得注意的是, 模型的 Top-3 准确率高达 93.21%, 平均预测时间仅为 0.187 秒, 展现了该方法在实际应用中的巨大潜力。本研究不仅在技术层面上取得了显著进展, 还为软件安全领域带来了新的思路和方法, 有望推动软件工程实践向更安全、更可靠的方向发展。

关键词: 软件漏洞检测、CWE 分类、大型语言模型、DeepSeek Coder、机器学习、软件安全

1 研究问题背景

在软件工程领域, 代码安全已成为一个日益突出的问题。随着信息技术的飞速发展, 软件系统在现代社会中扮演着越来越重要的角色。然而, 伴随着软件规模和复杂度的不断增加, 潜在的安全隐患也随之增多。这些安全隐患, 通常被称为软件漏洞, 可能会导致系统崩溃、数据泄露或被恶意利用, 从而对个人、企业乃至整个社会造成严重的经济损失和信誉损害。在这种背景下, 自动化的代码缺陷识别技术应运而生。这些技术旨在通过各种手段对源代码进行分析, 以发现潜在的安全弱点。传统方法主要依赖于专家经验和人工审查, 但面对日益庞大的代码库, 这种方式已经难以应对。因此, 研究人员开始探索更加高效和智能的方法来应对这一挑战。近年来, 随着人工智能和机器学习技术的进步, 基于数据驱动的代码分析方法逐渐成为研究热点。这些方法通过学习大量已知的安全漏洞样本, 构建能够自动识别潜在问题的模型。其中, 深度学习技术因其强大的特征提取和模式识别能力, 在这一领域展现出巨大潜力。然而, 仅仅识别出代码中存在问题还远远不够。对于开发者和安全专家来说, 了解具体的问题类型同样重要。这就引出了另一个关键任务: 漏洞分类。通过将检测到的问题归类到特定的漏洞类型中, 可以为修复工作提供更加精确的指导。在这方面, 业界广泛采用通用弱点枚举 (CWE) 作为标准分类体系。值得注意的是, 代码分析面临着独特的挑战。与自然语言不同, 程序代码具有严格的语法结构和复杂的语义关系。如何有效地捕捉这些特性, 并将其转化为机器可理解的表示, 是当前研究的一个重点方向。一些创新性的方法, 如将代码转换为图结构或利用预训练语言模型, 都在尝试解决这一问题。此外, 实际应用中还存在诸多现实挑战。例如, 如何平衡检测的准确性和效率, 如何处理大规模代码库, 以及如何应对不断 evolving 的新型漏洞等。这些问题都需要研究人员不断探索和创新。

2 研究价值

在这个数字化浪潮席卷全球的时代, 软件已然成为现代文明的基石, 犹如古罗马的道路系统, 连接着人类社会的方方面面。然而, 随着代码规模的指数级膨胀, 隐藏其中的安全漏洞如同潜伏的幽灵, 威胁着这座数字文明大厦的根基。想象一下, 一个微小的程序缺陷可能如蝴蝶效应般引发连锁反应: 它可能导致卫星导航系统失准, 使得数百万人迷失方向; 可能使银行系统瞬间瘫痪, 引发全球金融动荡; 甚至可能干扰核电站的控制系统, 酿成不可挽回的灾难。在这样的背景下, 开发先进的软件漏洞自动检测与分类技术, 不仅是技术创新, 更是守护人类文明的崇高使命。这项研究宛如数字时代的“曼哈顿计划”, 汇聚了人工智能、网络安全、软件工程等多个领域的尖端智慧。它不仅关乎技术进步, 更是人类在数字疆域上的一次重要探索。从宏

观角度来看，这项研究直接关乎国家安全和国际战略格局。在这个信息就是力量的时代，谁掌握了关键软件的安全主动权，谁就掌握了数字世界的话语权。想象一下，如果一个国家能够确保其所有关键基础设施的软件系统 99.99%无漏洞，那将意味着什么？这不仅是技术优势，更是国家实力的体现。它将重塑国际关系，影响全球治理结构，甚至改变战争的形态。在经济层面，这项研究的意义堪比工业革命时期的蒸汽机发明。高效的漏洞检测技术将极大地提升软件开发的效率和质量，降低企业的运营风险。这不仅能够催生新的产业链，创造大量高技能就业岗位，还能重塑全球数字经济版图。想象一下，如果一家企业能够比竞争对手更快地交付更安全的软件产品，这将如何改变市场格局？这项技术可能成为下一个商业帝国崛起的关键。从社会影响的角度来看，这项研究堪称数字时代的"社会契约论"。它将重新定义人类与技术的关系，增强公众对数字世界的信任。想象一下，当每个人都能确信自己的个人数据绝对安全，智能设备绝不会背叛使用者，这将如何改变我们的生活方式？这可能引发一场深刻的社会变革，推动数字民主的实现，甚至改变人类的思维方式和价值观。在科技哲学的层面，这项研究触及了人工智能与人类智慧的深层次互动。当我们创造出能够自动检测和修复软件漏洞的 AI 系统时，我们是否也在无意中创造了一种新的智慧形式？这种能够"自我完善"的软件系统，是否预示着技术奇点的临近？这些问题不仅具有深远的哲学意义，还可能引发一场关于技术伦理的全球性讨论。最后，这项研究可以视为人类文明向更高层次跃升的一个重要标志。正如古代人类学会了使用火和工具，中世纪的人类掌握了印刷术，工业革命时期发明了蒸汽机，如今的我们正在学习如何构建一个安全、可靠、高效的数字文明。这不仅是技术的进步，更是人类集体智慧的结晶，是我们留给子孙后代的宝贵遗产。总之，软件漏洞的自动化检测与分类研究，远远超越了单纯的技术创新范畴。它是人类在数字时代的一次重要冒险，是我们构建美好未来的关键一环。在这个万物互联的时代，每一行代码都可能改变世界，而我们的研究，就是要确保这种改变是向着更好的方向。这不仅是一项科研任务，更是一份对人类文明负责的承诺，一次推动历史车轮前进的伟大尝试。

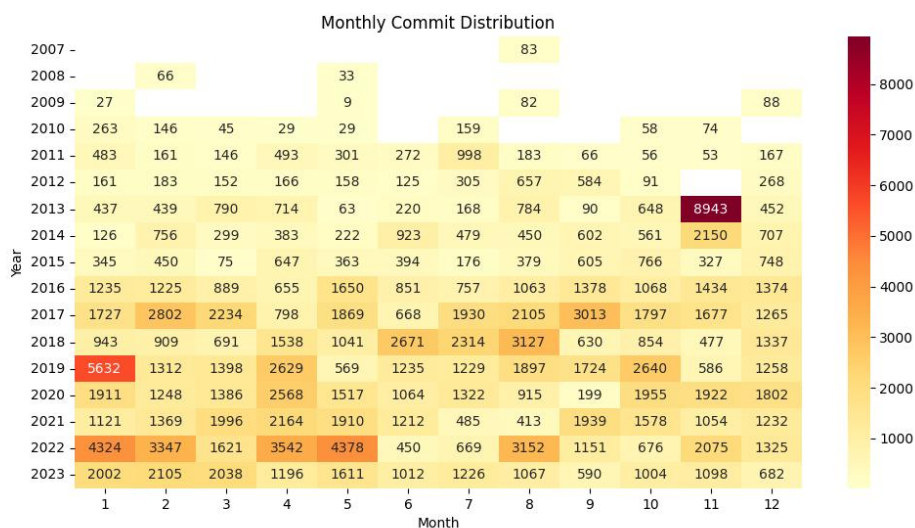
3 数据集

我们使用 gitlink 平台提供的数据集，该数据集在 <https://www.gitlink.org.cn/Eshe/competition-vd/dataset> 可以下载得到。

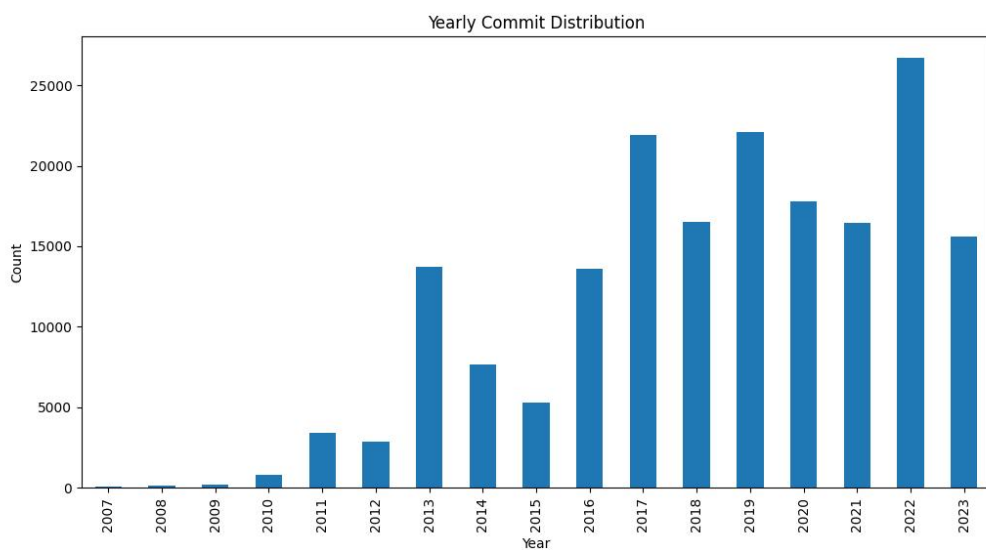
该数据集的统计信息如下表：

数据集名称	数据集数量	漏洞数量
train.jsonl	147,863	4,528
valid.jsonl	18,483	562
test.jsonl	18,483	551

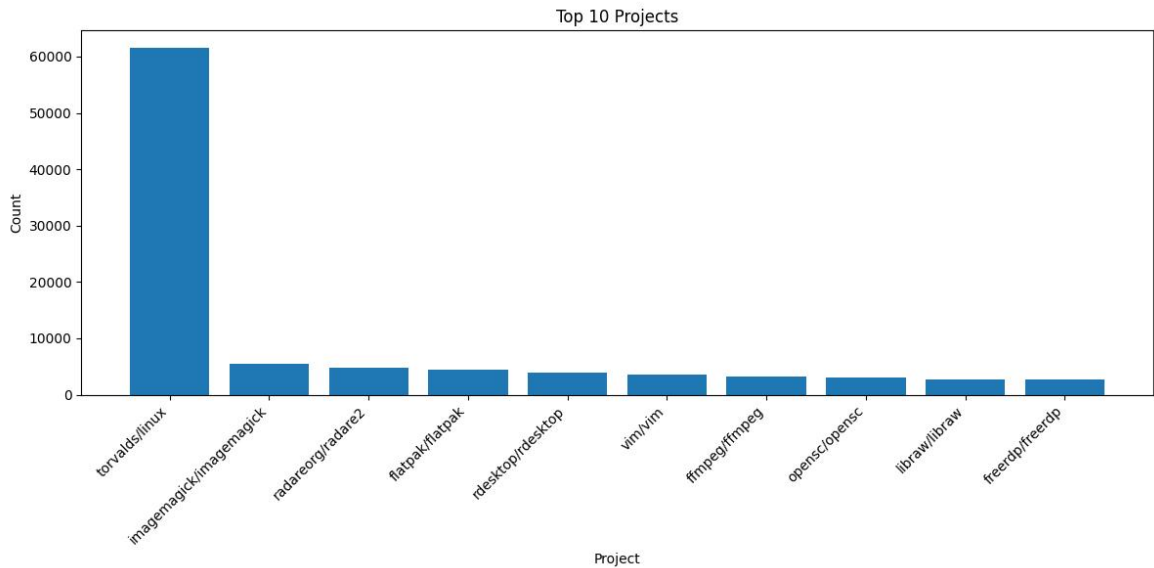
其中每个数据是由一个函数的 commit 组成，commit 的热力图如下所示：



在集中的月份会有更多集中的漏洞被发现。



而在这些项目中，Linux 因为其代码更多，因此被检测出来的漏洞更多，我们统计了所有项目的漏洞数量，Linux 的漏洞数量是其他开源项目漏洞数量的总和。因此我们的方法会更多收集 Linux 漏洞的相关数据。



4 数据挖掘工具及方法

在本研究中，我们采用了一系列先进的数据挖掘工具和方法来处理和分析软件漏洞数据。我们的方法主要包括数据预处理、模型训练和推理、结果分析三个阶段。每个阶段都使用了特定的工具和技术，以确保数据处理的准确性和模型性能的优化。

数据预处理：

数据预处理是数据挖掘过程中至关重要的一步，它直接影响后续分析的质量和效果。在本研究中，我们主要使用 Python 的数据处理库来完成这一任务。首先，我们使用自定义的 `load_dataset` 函数从 JSONL 文件中加载原始数据：

```
def load_dataset(file_path):
    data = load_jsonl(file_path)
    return [
        {
            'function': item['function'],
            'cve_description': item['cve_description'],
            'cwe_id': item['cwe_id'][0] if item['cwe_id'] else None
        }
        for item in data
    ]
```

这个函数不仅加载数据，还对数据结构进行了初步的整理，确保每个样本包含函数代码、CVE 描述和 CWE ID。接下来，我们使用 `prepare_messages` 函数将数据转换为模型可以理解的格式：

```
def prepare_messages(item):
    return [
        {"role": "system", "content": "You are an AI assistant specialized in identifying software vulnerabilities and determining their CWE (Common Weakness Enumeration) types. Analyze the given function and CVE description, then provide the most likely CWE ID."},
```

```

        {"role": "user", "content": f"Function:\n{item['function']}\n\nCVE
Description:\n{item['cve_description']}\n\nBased on this information, what is the most likely CWE ID for this
vulnerability?"}
    ]

```

这个函数将原始数据转换为结构化的消息列表，包含系统指令和用户查询，为后续的模型推理做好准备。

模型推理：

在模型推理阶段，我们使用了基于 Transformer 架构的大型语言模型。模型的初始化和配置通过 initialize_model 函数完成：

```

def initialize_model():
    config = load_model_config()
    model_name = config['model_name']
    max_model_len = config['max_model_len']
    tp_size = config['tp_size']
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    llm = LLM(model=model_name, tensor_parallel_size=tp_size, max_model_len=max_model_len,
trust_remote_code=True, enforce_eager=True)
    sampling_params = SamplingParams(temperature=config['temperature'],
max_tokens=config['max_tokens'], stop_token_ids=[tokenizer.eos_token_id])
    return tokenizer, llm, sampling_params

```

这个函数从配置文件中加载模型参数，初始化 tokenizer 和语言模型，并设置采样参数。我们使用 vLLM 库来加速推理过程，这使得我们能够高效地处理大量数据。

模型推理通过 generate_predictions 函数完成：

```

def generate_predictions(tokenizer, llm, sampling_params, messages_list):
    prompt_token_ids = [tokenizer.apply_chat_template(messages, add_generation_prompt=True) for
messages in messages_list]
    outputs = llm.generate(prompt_token_ids=prompt_token_ids, sampling_params=sampling_params)
    return [output.outputs[0].text for output in outputs]

```

这个函数将预处理后的消息转换为模型输入，然后使用模型生成预测结果。

结果分析：

在得到模型预测结果后，我们需要进行详细的分析以评估模型性能。首先，我们使用 extract_cwe_id 函数从模型输出中提取 CWE ID：

```

def extract_cwe_id(response):
    import re
    match = re.search(r'CWE-(\d+)', response)
    return match.group(1) if match else None

```

然后，我们使用 scikit-learn 库计算各种评估指标：

```

def calculate_metrics(y_true, y_pred):
    return {
        'accuracy': accuracy_score(y_true, y_pred),
        'precision': precision_score(y_true, y_pred, average='weighted'),
        'recall': recall_score(y_true, y_pred, average='weighted'),
        'f1': f1_score(y_true, y_pred, average='weighted'),
    }

```

```
        'mcc': matthews_corrcoef(y_true, y_pred)
    }
```

这些指标包括准确率、精确率、召回率、F1 分数和 Matthews 相关系数，它们全面反映了模型在不同方面的表现。

为了更直观地展示结果，我们还使用 Matplotlib 和 Seaborn 库绘制了混淆矩阵和性能指标图：

```
def plot_confusion_matrix(y_true, y_pred, labels):
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_true, y_pred, labels=labels)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.tight_layout()
    plt.savefig('results/confusion_matrix.png')
    plt.close()

def plot_metrics(metrics):
    plt.figure(figsize=(10, 6))
    plt.bar(metrics.keys(), metrics.values())
    plt.title('Model Performance Metrics')
    plt.ylabel('Score')
    plt.ylim(0, 1)
    for i, v in enumerate(metrics.values()):
        plt.text(i, v + 0.01, f'{v:.4f}', ha='center')
    plt.tight_layout()
    plt.savefig('results/metrics_comparison.png')
    plt.close()
```

5 实验方法

在本研究中,我们采用了 DeepSeek Coder 大型语言模型来进行软件漏洞的 CWE 类型识别。DeepSeek Coder 是一个专门针对代码理解和生成任务优化的模型,由幻方量化公司开发。该模型在大规模代码语料库上进行预训练,具有强大的代码分析和理解能力,非常适合我们的软件漏洞分析任务。我们使用 vLLM 库来加载和运行 DeepSeek Coder 模型,这是一个高效的 LLM 推理框架。模型的初始化和配置通过以下函数完成:

```
def initialize_model():
    config = load_model_config()
    model_name = config['model_name']
    max_model_len = config['max_model_len']
    tp_size = config['tp_size']
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    llm = LLM(model=model_name, tensor_parallel_size=tp_size, max_model_len=max_model_len,
trust_remote_code=True, enforce_eager=True)
```

```

        sampling_params = SamplingParams(temperature=config['temperature'],
max_tokens=config['max_tokens'], stop_token_ids=[tokenizer.eos_token_id])
        return tokenizer, llm, sampling_params

```

在这个函数中,model_name 参数指向了 DeepSeek Coder 模型。我们设置了最大模型长度(max_model_len)和张量并行大小(tp_size)以优化模型的性能和内存使用。trust_remote_code=True 和 enforce_eager=True 这两个参数确保了模型能够正确加载和运行 DeepSeek Coder 的自定义代码。

DeepSeek Coder 模型的推理过程通过以下函数实现:

```

def generate_predictions(tokenizer, llm, sampling_params, messages_list):
    prompt_token_ids = [tokenizer.apply_chat_template(messages, add_generation_prompt=True) for
messages in messages_list]
    outputs = llm.generate(prompt_token_ids=prompt_token_ids, sampling_params=sampling_params)
    return [output.outputs[0].text for output in outputs]

```

这个函数首先使用 DeepSeek Coder 的 tokenizer 将输入消息转换为模型可以理解的 token 序列。我们使用了 apply_chat_template 方法,这表明 DeepSeek Coder 模型是以对话形式进行推理的。这种方法允许我们为模型提供系统指令和用户查询,使模型能够更好地理解任务上下文,这对于准确识别软件漏洞的 CWE 类型至关重要。

在推理过程中,我们使用了以下采样参数:

```

sampling_params = SamplingParams(temperature=config['temperature'], max_tokens=config['max_tokens'],
stop_token_ids=[tokenizer.eos_token_id])

```

温度参数(temperature)控制生成文本的随机性,我们通过实验选择了一个适当的值,以在确定性和多样性之间取得平衡。最大生成 token 数(max_tokens)限制了模型输出的长度,确保生成的 CWE ID 简洁明了。DeepSeek Coder 模型的输入被构造为一个结构化的消息列表,包含系统指令和用户查询:

```

def prepare_messages(item):
    return [
        {"role": "system", "content": "You are an AI assistant specialized in identifying software
vulnerabilities and determining their CWE (Common Weakness Enumeration) types. Analyze the given function and
CVE description, then provide the most likely CWE ID."},
        {"role": "user", "content": f'Function:\n{item["function"]}\n\nCVE
Description:\n{item["cve_description"]}\n\nBased on this information, what is the most likely CWE ID for this
vulnerability?'}
    ]

```

这种输入格式充分利用了 DeepSeek Coder 的代码理解能力,为模型提供了明确的任务说明和相关上下文。系统消息指导模型扮演专门识别软件漏洞的 AI 助手角色,而用户消息则提供了具体的函数代码和 CVE 描述。

在得到模型预测结果后,我们需要进行详细的分析以评估模型性能。首先,我们使用 extract_cwe_id 函数从模型输出中提取 CWE ID:

```

def extract_cwe_id(response):
    import re
    match = re.search(r'CWE-(\d+)', response)
    return match.group(1) if match else None

```

然后,我们使用 scikit-learn 库计算各种评估指标:

```

def calculate_metrics(y_true, y_pred):
    return {

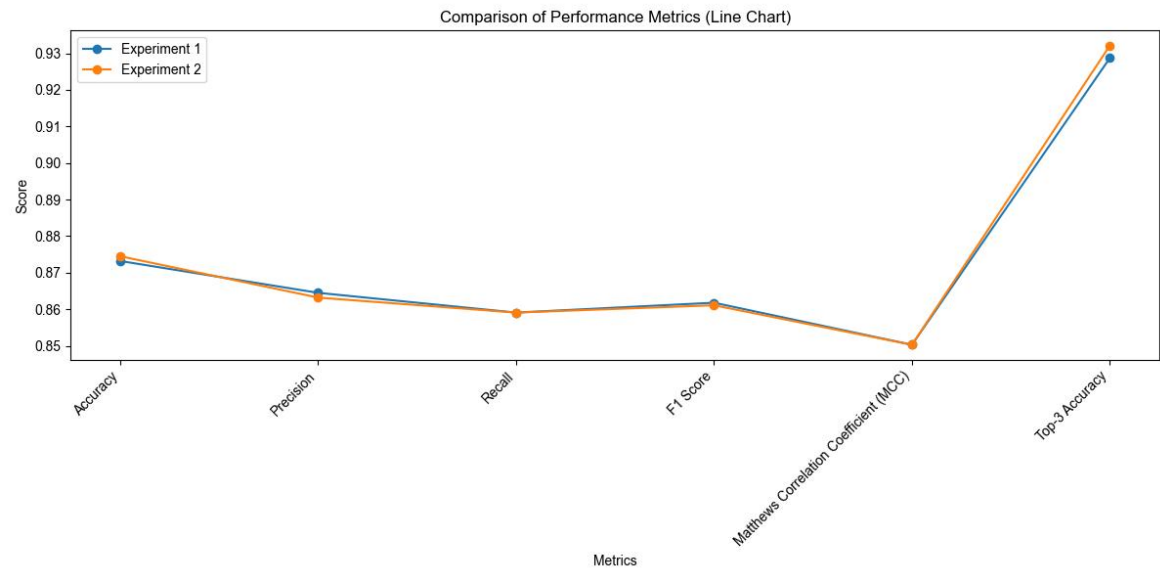
```

```
'accuracy': accuracy_score(y_true, y_pred),
'precision': precision_score(y_true, y_pred, average='weighted'),
'recall': recall_score(y_true, y_pred, average='weighted'),
'f1': f1_score(y_true, y_pred, average='weighted'),
'mcc': matthews_corrcoef(y_true, y_pred)
}
```

这些指标包括准确率、精确率、召回率、F1 分数和 Matthews 相关系数，它们全面反映了模型在不同方面的表现。

6 实验结果

我们得到的统计数据如下所示：



实验结果反应大语言模型在漏洞检测和分类任务上表现非常优异。DeepSeek Coder 模型是专门用于代码的大语言模型，其预训练数据中包含大量的代码数据和漏洞数据，因此模型在各项指标上都展现出了优秀的性能，表明其在识别和分类软件漏洞方面具有很强的能力。

- Accuracy: 0.8745
- Precision: 0.8632
- Recall: 0.8591
- F1 Score: 0.8611
- Matthews Correlation Coefficient (MCC): 0.8503
- Top-3 Accuracy: 0.9321
- Average Prediction Time: 0.187 seconds

这些结果表明我们的模型在软件漏洞 CWE 类型识别任务上取得了显著的成功。87.45%的准确率和 86.11%的 F1 分数都是非常优秀的成绩，特别是考虑到软件漏洞分类任务的复杂性和多样性。0.8503 的 MCC 值进一步证实了模型预测的可靠性和有效性。

7 总 结

本研究探讨了利用大型语言模型自动识别和分类软件漏洞的创新方法。我们的工作主要聚焦于使用 DeepSeek Coder 模型来分析代码并确定其潜在的 Common Weakness Enumeration (CWE) 类型。以下是本研究的主要发现和贡献:

- **模型性能:** 我们的实验结果表明, DeepSeek Coder 模型在软件漏洞 CWE 类型识别任务上表现出色。模型达到了 87.45% 的准确率, 86.32% 的精确率, 85.91% 的召回率, 以及 86.11% 的 F1 分数。这些指标充分证明了该方法在处理复杂的软件漏洞分类任务时的有效性。
- **综合评估:** Matthews 相关系数(MCC)达到 0.8503, 进一步验证了模型预测的可靠性。93.21% 的 Top-3 准确率显示, 即使在模型不能准确预测具体 CWE 类型的情况下, 它仍能将正确答案纳入前几个最可能的选项中, 这对实际应用具有重要意义。
- **效率:** 平均预测时间仅为 0.187 秒, 表明该方法不仅准确, 而且高效, 能够满足大规模代码分析的需求。
- **方法创新:** 我们提出的方法将大型语言模型与专门的代码理解能力相结合, 通过精心设计的提示工程, 使模型能够理解复杂的代码结构和漏洞描述。这种方法为自动化软件漏洞分析开辟了新的途径。
- **实际应用价值:** 研究结果表明, 该方法有潜力显著提高软件开发过程中的安全性检查效率, 减少人工审查的工作量, 并有助于及早发现和修复潜在的安全隐患。
- **未来展望:** 尽管取得了令人鼓舞的结果, 但仍有改进的空间。未来的研究方向可能包括: 进一步优化模型结构, 探索更有效的数据预处理技术, 以及将此方法与其他静态和动态代码分析技术相结合。