一、基于大模型的源码警告识别综述

论文主要搜索关键词（不限于以下关键词）：每一行内的关键词是“或”关系，行与行之间关键词是“并”关系

序号	目的	关键字
1	大模型	1) large language model, 2) LLM, 3) pre-trained model
2	后处理警告操作	1) elimination, 2) reduction, 3) simplification, 4) ranking, 5) classification, 6) reviewing, 7) inspection, 8) detection
3	静态分析	1) static analysis, 2) automated code analysis, 3) source code analysis, 4) automated defects detection, 5) false
4	警告	1) alarm, 2) warning, 3) alert, 4) violations

Ps：论文收集参考文献“Survey of approaches for postprocessing of static analysis alarms”和“Machine Learning for Actionable Warning Identification: A Comprehensive Survey”

验收：由助教统一说明。

二、仓库级源码警告数据集收集技术

现有研究提出了一些自动警告数据集构建方法以收集源码警告数据集，这些方法本质上是通过跟踪警告演变历史进而收集数据集。具体来说，给定同一个项目的多个版本，使用相同的静态分析工具对每个版本进行扫描并得到其扫描结果，通过比对不同版本之间的警告是否相同，如果一个版本中的某个警告 A 存在在后续所有的版本，那么该警告 A 被认为是无效警告，如果一个版本中的某个警告 A 在后续的版本中消失，那么该警告 A 被认为是有效警告。此外，这些方法也开放了对应的警告数据集。然而，这些警告数据集构建领域仍然存在以下问题：第一，现有的警告数据集收集方法由于通过警告匹配策略以跟踪警告演变历史，这些不精确的警告匹配策略是不精确的，进而导致收集到的警告数据集存在很多噪音；

第二，现有的开发语言多种多样，同时源码静态扫描工具也是多种多样的，但是现在的开放的警告数据集仅仅只聚焦与少数几个源码静态扫描工具（如：SpotBugs 和 Infer）和少数开发语言上（Java 和 C/C++）。因此，该技术主要是为了构建一个面向多源异构源码静态扫描工具的警告基准数据集构建。

该研究主要聚焦于 Java 语言的项目和多种开源的源码静态扫描工具产生的警告上，其主要流程如下：

（1）选择 CodeQL、Contrast、Horusec、Insider、SpotBugs、Semgrep、SonarQube、Infer 等 9 个开源的源码静态扫描工具。

开源静态扫描工具	主要的检测语言	源码是否需要编译	链接
CodeQL	Java、C/C++、Python	是	https://codeql.github.com/
Contrast Codesec Scan	Java	否	https://www.contrastsecurity.com/
Horusec	Java	否	https://docs.horusec.io/docs/overview/
Insider	Java	否	https://github.com/insidersec/insider/
SpotBugs	Java	是	https://spotbugs.github.io/
Semgrep	Java、Python	否	https://github.com/semgrep/semgrep
SonarQube	Java	否	https://www.sonarsource.com/products/sonarqube/
Infer	Java、C/C++	是	https://fbinfer.com/

（2）从 apcahe software foundation 中选择 5 个 Java 项目，这些项目需要具备以下属性：第一，超过 1000 个 star 的项目；第二，涉及到的 commit 超过 1000；第三，每个项目的发布版本（每个项目对应的发布版本指得是 github 中每个项目中的 tag）收集时间为 2023/01/01 至今（大家开始做项目的时间）；且每个项目至少要有两个编译的发布版本；第四，每个项目的开发者要超过 10 个。

（3）给定一个被测项目，该项目包括多个 tag（即 tag_1,....., tag_i,....., tag_n），使用每个源码静态扫描工具去扫描所有 tag，然后通过跟踪警告演变历史针对 tag_1 进行源码警告标记。跟踪警告演变历史的具体算法如下：给定警告 A 和警告 B，其中警告 A 来自 tag_i，警告 B 来自 tag_{i+1}，为了去判断 tag_i 中警告 A 是否是一个有效警告，第一，根据警告的属性（如：警告的类型和位置）进行精确匹配，

如果匹配上了，则认为 A 和 B 是相同的警告；如果未匹配上，如果警告 A 和警告 B 的类型相同，则利用模糊匹配对警告 A 和 B 进行对比，即比较警告 A 和警告 B 对应的源代码片段相似度，当相似度达到某个阈值时，A 和 B 则认为是相同的警告，否则，A 和 B 则为不同的警告。

Ps：跟踪警告演变历史的具体算法可参考论文：“Is There a "Golden" Feature Set for Static Warning Identification? An Experimental Evaluation”和“ViolationTracker: Building Precise Histories for Static Analysis Violations”和“Mining Fix Patterns for FindBugs Violations”

（4）通过对警告报告进行预处理（包括警告类型映射）等警告归一化，并将多个源码静态扫描工具的扫描结果进行汇总。在汇总后，如果同一个警告被超过 5 个及其以上的工具报告，则将该警告标记为有效警告。

Ps：按照 CWE 的所有漏洞分类，对不同源码静态扫描工具的类型进行映射，次步骤需要手工处理

（5）最后将所有源码静态扫描工具对应的警告进行汇总，每个警告的属性是所有工具报告的警告属性的并集，下表列出来部分主要的属性，没有考虑到的属性请自行补充。

属性	描述
Category	警告类型，一般来说工具只有一个 category，但是 spotbugs 既有 category 也有 type
Type	警告类别，是 category 的子类，即更细节的划分，如果有些工具没有则该属性可为空
CWE	CWE 对应的编号，比如：CWE-416 表示使用后未释放的漏洞
Message	警告的信息
File_name	警告所在的文件位置
Method_name	警告所在的函数位置
Start_line	警告代码起始行
End_line	警告代码终止行
The length of the warning code	警告代码行的长度

line	
flag	有效警告还是无效警告的标记，用 0/1 表示
Error_trace	产生该警告的错误路径
Fix_suggestion	该警告的修复建议
Patch	该警告的对应的修复代码，一般来说只有有效警告才有该属性
The source code of the warning code line	警告行对应的源码
The source code of the method containing the warning	一个警告所在函数对应的源码

验收：（1）所有源码上传 github，先私有后公开；（2）所有工具类型映射的表格；（3）警告数据集及其对应的属性

三、基于提示工程的智能化源码警告识别实证研究

提示工程（Prompt Engineering）是指设计和优化输入给人工智能模型（如 ChatGPT）的指令，以确保模型生成预期的输出。通过构建精准、详细的提示，能够提升模型的理解和响应能力，从而实现高效的人机交互。目前，提示工程已经在基于大模型的软件工程任务（如：代码摘要和自动缺陷修复）。然而，提示工程在源码警告识别上的性能还未被充分调研。为了解决此问题，该任务旨在系统调研提示工程在源码警告识别的可行性，并比较不同的提示模版在源码警告识别上的优缺点。

该研究的主要流程如下：

（1）源码警告数据集：源码静态扫描工具 CSA/Infer/CppCheck 在 C/C++项目上的警告；源码静态扫描工具 SpotBugs 在 Java 项目上的警告。

Ps：该数据集助教整理好后从 Moodle 提供给大家。

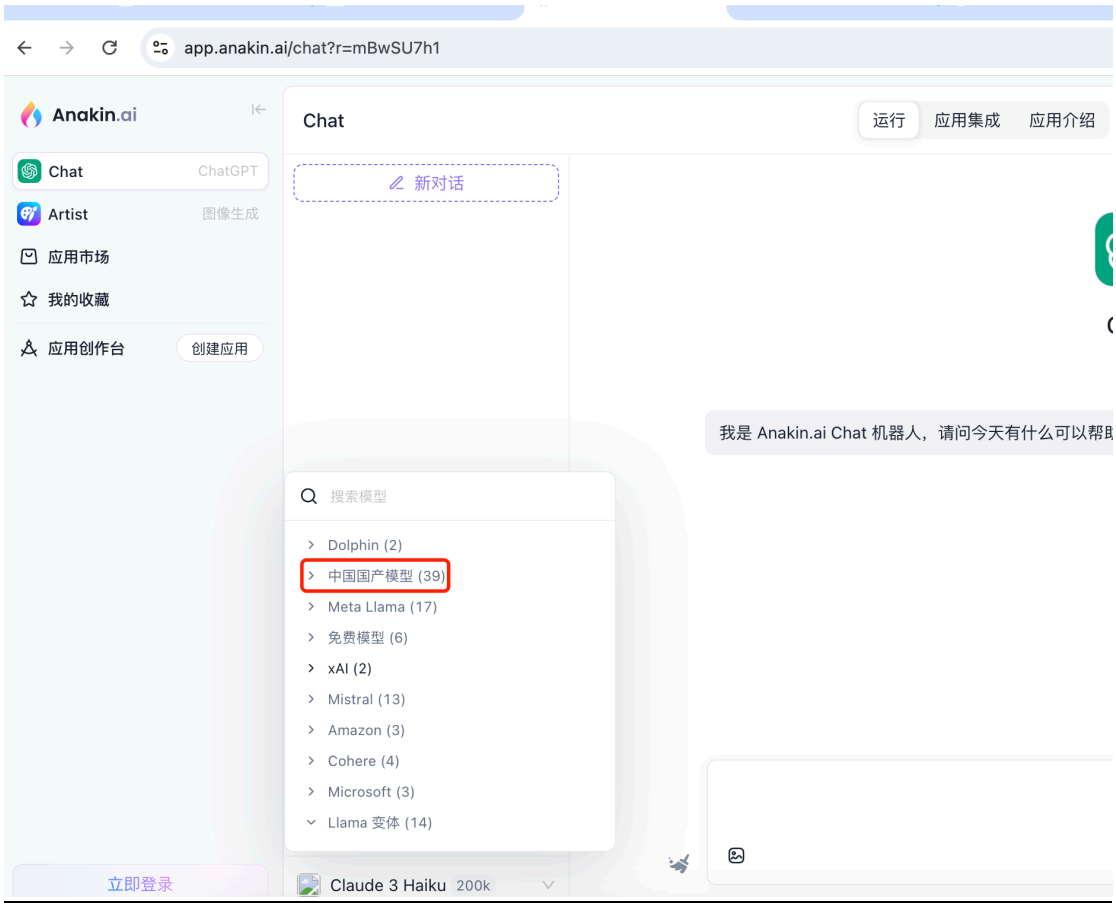
（2）提示模版：zero-shot、one-shot、few-shot、chain-of-thought、general-info、critique、expertise、self-heuristic 等 8 个提示模版。

Ps：不同模版的含义可参见论文“Exploring ChatGPT’s Capabilities on Vulnerability Management”以及论文“Automatic Code Summarization via

ChatGPT: How Far Are We?Source Code Summarization in the Era of Large Language Models”。

（3）大语言模型：codellama（如：7B）、StarChat-β、chatgpt3.5、chatgpt4 等 5 个大模型。

Ps：不同模型参数对应的模型，如 codellama（7B、13B、34B）算一个大语言模型；如果某些模型是付费的（如：chatgpt4 在有限时间段内智能使用 n 次），请寻找新的可交互的大语言模型进行替换。可批量调用 API 的免费的大语言模型主要有国产的大语言模型（链接：https://app.anakin.ai/chat?r=mBwSU7h1、https://bigmodel.cn/pricing）。如果大家经济条件允许，可使用付费 API。



（4）评估准则：precision、recall、f1

（5）研究思路：比较基于深度学习的源码警告识别方法和基于提示工程的源码警告分类性能；比较不同提示模版下的源码警告识别性能；比较不同大语言模型下的源码警告识别性能；比较在不同工具上的源码警告识别性能等等

Ps：可提供更多的且不同粒度的性能评估，如：更多的开发语言，更多的源码静态扫描工具等等

验收：（1）所有源码上传 github，先私有后公开；（2）所有原始实验结果；（3）针对实验结果的可视化，分析和总结