中国科学院软件研究所学术年会 暨重点实验室科技活动周 2025 第十届
学术论文

Write Your Own Code Checker: An Automated Test-Driven Checker Development Approach with LLMs

测试用例驱动的大模型辅助静态代码检查器自动生成

刘珺*，解远远*，燕季薇，黄进豪，严俊，张健

Accepted in 48th International Conference on Software Engineering (ICSE'26 CCF-A)

基础软件与系统重点实验室 · 软件工程技术研究开发中心

联系人：刘珺，燕季薇　联系方式：{liuj2022, yanjw}@ios.ac.cn

研究背景 • *Background*

在现代软件开发中，为了保障代码质量与安全，静态代码检查工具扮演着至关重要的角色。检查工具一般包含多个检查器，每个检查器对应检查一条规则。然而，通用的检查规则往往无法满足特定项目或特定场景的个性化需求，因此“自定义代码检查器”的需求日益增长。

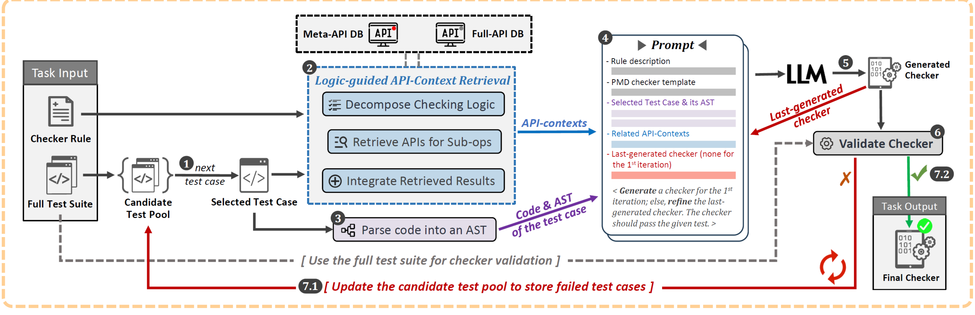
虽然现有的主流检查框架（PMD，SonarQube 等）都提供了定制接口，但是从零开发代码检查器依然是一个困难且要求专家知识的任务，难以实际应用推广。本研究旨在探索利用 LLM 自动编写代码检查器的可能性，期望借此降低开发门槛，填补现有技术空白。

挑战一：生成覆盖充分场景的复杂检查逻辑　挑战二：基于规则描述精准检索检查API知识

方法总览 • *Approach*

AutoChecker 基于“规则描述”和相应的“测试用例集（包括符合/不符合该规则的代码用例）”生成一个相应的代码检查器。两大核心策略如下：

- **测试驱动代码检查器开发（挑战一）**：模仿人类开发者开发过程，对测试用例“逐个击破”，驱动LLM对检查器进行增量式地生成与修复以生成充分检查逻辑。
- **逻辑引导的 API 知识检索（挑战二）**：构建了两个 API 知识库：Meta-API DB 和 Full-API DB。API 检索过程：首先对检查规则进行分解，然后针对每一个得到的“子操作”检索对应的 API 签名或用法片段。



实验评估 • *Evaluation*

- **实验设置**：我们在业界内广泛使用的 Java 静态分析工具 PMD 上实现了AutoChecker，基于分层随抽样样选取了20个官方内置规则（10个简单，10个困难）进行全面评估。

有效性评估：AutoChecker表现显著优于所有基线方法

Method + LLM	#Rule$_{pc}$ (/20)	#Rule$_{pat}$ (/20)	#Rule$_{pat}$ (/20)	#TC$_{pass}$ (/373)	TPR$_{avg}$
NoCaseLLM					*naive baseline without test cases*
+ Llama3.1	0	0	0	0	0.00%
+ Qwen2.5-Coder	5	5	1	40	19.41%
+ GPT-4	7	7	1	62	27.92%
+ DeepSeek-V3🖈	8	8	1	56	28.06%
AllCasesLLM					*naive baseline with all test cases*
+ Llama3.1	0	0	0	0	0.00%
+ Qwen2.5-Coder	4	4	1	17	14.40%
+ GPT-4	5	5	2	36	21.53%
+ DeepSeek-V3🖈	6	6	2	43	24.60%
NoCaseLLM^R					*enhanced baseline with RAG*
+ Llama3.1	2	2	0	16	4.71%
+ Qwen2.5-Coder	9	9	2	60	30.68%
+ GPT-4	10	10	1	108	30.82%
+ DeepSeek-V3🖈	9	9	2	92	32.05%
NoCasesLLM^C					*enhanced baseline with COT*
+ Llama3.1	0	0	0	0	0.00%
+ Qwen2.5-Coder	6	6	1	45	21.18%
+ GPT-4	8	8	1	94	27.26%
+ DeepSeek-V3🖈	9	9	0	66	29.40%
NoCaseLLM^{RC}					*enhanced baseline with RAG + COT*
+ Llama3.1	2	2	0	7	6.25%
+ Qwen2.5-Coder	9	9	1	60	30.49%
+ GPT-4	9	9	1	105	27.74%
+ DeepSeek-V3🖈	11	11	1	101	38.93%
AutoChecker					*our approach*
+ Llama3.1	**3**	**3**	**1**	**22**	**8.11%**
+ Qwen2.5-Coder	**20** ☆	**20** ☆	**4**	**257**	**79.01%**
+ GPT-4🖈	**20** ☆	**20** ☆	**6** ☆	**278** ☆	**82.28%** ☆
+ DeepSeek-V3	**19**	**19**	**4**	**278** ☆	**80.86%**

We keep the result with highest TPR$_{avg}$ across three runs for each method.
#TC$_{pass}$ denotes the number of passed test cases in total; ☆ marks the best result of each metric across all methods; 🖈 is the best LLM (based on TPR$_{avg}$) for each method.

实用性评估：当测试用例充分时，AutoChecker生成的检查器在真实项目中的检查能力与官方检查器相当

Checker Rule	#TC$_{\Delta}$	#Violations on Five Projects	
		official checker	Checker$_{AutoChecker+GPT\text{-}4}$ with TS$_{orig}$	Checker$_{AutoChecker+GPT\text{-}4}$ with TS$_{aug}$
NullAssignment	+5	2,560	1,632 (↓928)✘	2,562 (↑2)
ExcessivePublicCount	+6	389	330 (↓59)	389 (=0)
ExcessiveImports	+0	3,321	3,321 (=0)	3,321 (=0)
AvoidUsingOctalValues	+7	58	0 (↓58)	58 (=0)
MethodNamingConventions	+1	11,562	11,560 (↓2)	11,562 (=0)
AssignmentToNonFinalStatic	+0	8	8 (=0)✘	8 (=0)
StringInstantiation	+0	347	347 (=0)	347 (=0)
InefficientEmptyStringCheck	+2	16	28 (↑12)	16 (=0)

TS$_{orig}$(original test suite) + TC$_{\Delta}$(new test cases) → TS$_{aug}$(augmented test suite);
Checker with TS$_{orig}$/TS$_{aug}$: generated checker based on a rule and its TS$_{orig}$/TS$_{aug}$;
✘ denotes that the checker meets crash during project scan.

实验结果演示网站

要点总结

- ✓ AutoChecker将代码检查器开发的重点从复杂高难度的“编写检查器代码”转变为相对更简单直接的“设计测试用例”
- ✓ 在软件工程领域，LLM+软件分析测试技术为传统的软工任务注入了新活力

Takeaway