UFMG

lab-soft
software engineering laboratory

Detecting and Fixing Code Smells with Large Language Models

Eduardo Figueiredo
Federal University of Minas Gerais (UFMG)

1st Study: Detecting Code Smells with LLMs

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. XX, NO. XX, XX 2025

Beyond Strict Rules: Assessing the Effectiveness of Large Language Models for Code Smell Detection

Saymon Souza, Amanda Santana, Eduardo Figueiredo, Igor Muzetti, João Eduardo Montandon, and Lionel Briand

Saymon Souza (PhD)

***Abstract*—Code smells are symptoms of potential code quality problems that may affect software maintainability, thus increasing development costs and impacting software reliability. Large language models (LLMs) have shown remarkable capabilities to support different software engineering activities, but their use to detect code smells is still underexplored. However, unlike the rigid rules of static analysis tools, LLMs can support flexible and adaptable solutions tailored to different code smells. This paper evaluates the effectiveness of using four LLMs to detect nine code smells in 30 Java projects. For the empirical evaluation, we automatically and manually create ground truths; the latter relies on 76 developers inspecting 268 code smell candidates. Our results indicate that the agreement between LLMs and**

smells [21]. However, to the best of our knowledge, they do not provide strong empirical evidence to indicate, for instance, that LLMs perform better than traditional code smell detection tools.

Moreover, early investigations of LLMs for code smell detection typically employ controlled scenarios and small code samples [22]–[24]. Despite initial results, it is essential to understand the effectiveness of LLMs in detecting code smells in real-world software projects [18]. Such software projects introduce numerous challenges for LLMs, including the need to understand and navigate codebases, adhere to

Goal and Research Questions

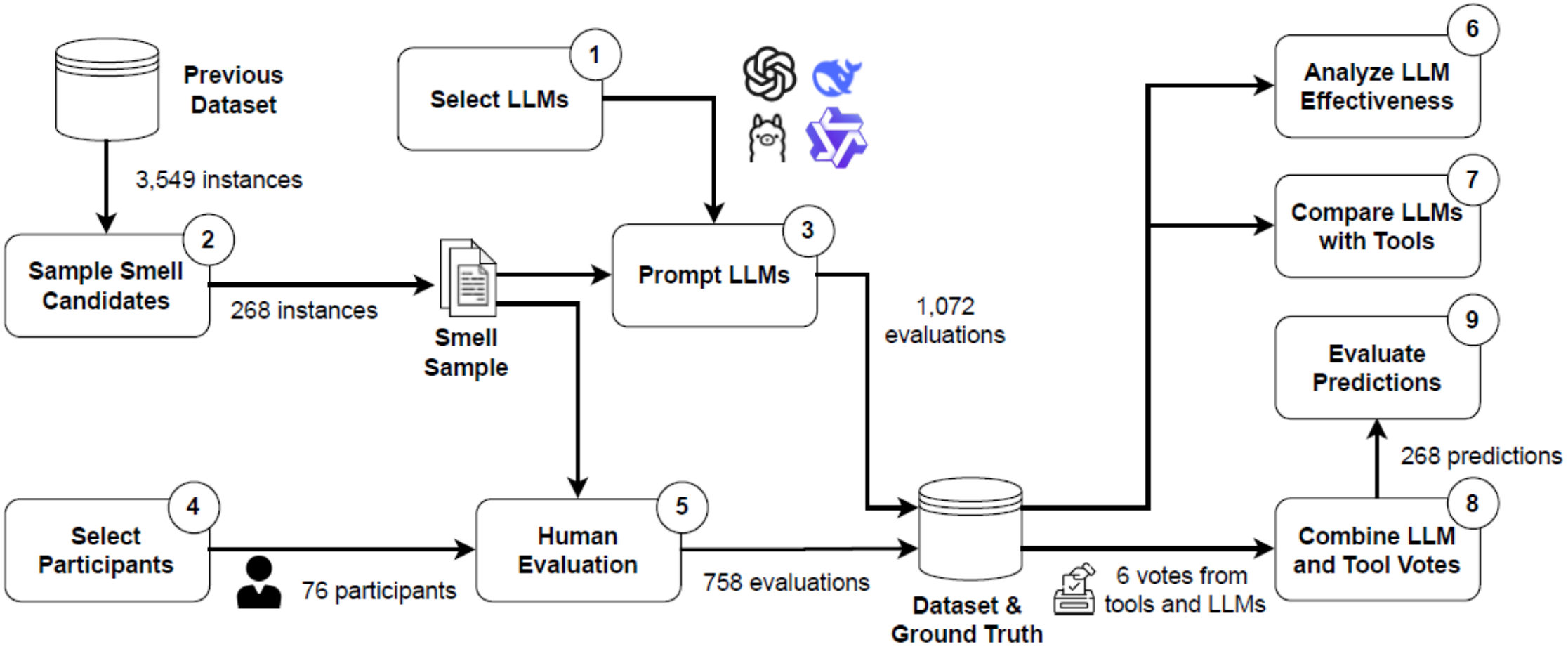
- Our goal is to investigate the effectiveness of **LLMs** in detecting **code smells** in Java projects

- Research Questions
  - **RQ1**: How effective are LLMs in detecting code smells?
  - **RQ2**: How do LLMs compare with static analysis tools?
  - **RQ3**: How effective is combining LLMs and Tools?

Used Dataset

- We analyzed 1,182 classes for code smells from 30 open-source Java projects
- Nine code smells
  - Data Class, Dispersed Coupling, Feature Envy, Intensive Coupling, Long Method, Large Class, Long Parameter List, Refused Bequest, and Shotgun Surgery
- Four LLMs and four static analysis tools
  - GPT-5 mini, Llama-3.3, DeepSeek-R1, and Qwen2.5-Coder
  - JDeodorant, JSpIRIT, Organic, and PMD

Study Steps


Previous Dataset
3,549 instances
1
Select LLMs
2
Sample Smell Candidates
268 instances
Smell Sample
3
Prompt LLMs
1,072 evaluations
4
Select Participants
76 participants
5
Human Evaluation
758 evaluations
Dataset & Ground Truth
6 votes from tools and LLMs
6
Analyze LLM Effectiveness
7
Compare LLMs with Tools
8
Combine LLM and Tool Votes
9
Evaluate Predictions
268 predictions

Summary of Results

	Large Language Models				Static Analysis Tools				Combined Predictions
Smell	DeepSeek	GPT	Llama	Qwen	JDeodorant	JSpIRIT	Organic	PMD	
Data Class	😐	😐	🏅😀	😐			😐	😐	🏅😀
Dispersed Coupling	☹️	☹️	😐	😐		😐	😐		🏅😀
Feature Envy	☹️	☹️	🏅😐	☹️	😐		😐		😐
Intensive Coupling	🏅😀	😐	😐	😐		😐	😐		😐
Large Class	😀	😐	🏅😀	😀	😐	😐			🏅😀
Long Method	😐	🏅😀	😀	😐	😀		😀		🏅😀
Long Parameter List	😐	😐	😐	☹️			🏅😐	☹️	😐
Refused Bequest	☹️	☹️	☹️	☹️		🏅☹️	☹️		☹️
Shotgun Surgery	😐	😐	😐	☹️		🏅😐	😐		🏅😐

Legend: 😀 (F1 $\geq$ 0.80) 😐 (0.51 $\leq$ F1 $<$ 0.80) ☹️ (F1 $\leq$ 0.50) 🏅 Best strategies

Some Findings

- LLMs are **effective for "easily detectable" code smells**, such as Data Class, Large Class, and Long Method

- **Static analysis tools are better** than LLMs in some more complex smells, such as Refused Bequest and Shotgun Surgery

- Combining LLMs and static analysis tools has shown **promising results**

2nd Study: Fixing Code Smells (SANER 2025)

Evaluating the Effectiveness of LLMs in Fixing Maintainability Issues in Real-World Projects

Henrique Nunes
Federal University of Minas Gerais
Belo Horizonte, Brazil
henrique.mg.bh@gmail.com

Eduardo Figueiredo
Federal University of Minas Gerais
Belo Horizonte, Brazil
figueiredo@dcc.ufmg.br

Larissa Rocha
State University of Bahia
Alagoinhas, Brazil
larissabastos@uneb.br

Sarah Nadi
New York University Abu Dhabi
Abu Dhabi, United Arab Emirates
sarah.nadi@nyu.edu

Fischer Ferreira
Federal University of Itajubá
Itajubá, Brazil
fischer.ferreira@unifei.edu.br

Geanderson Esteves
Federal University of Minas Gerais
Belo Horizonte, Brazil
geandersonesteves@gmail.com

Henrique Nunes (PhD)

***Abstract*—Large Language Models (LLMs) have gained attention for addressing coding problems, but their effectiveness in fixing code maintainability remains unclear. This study evaluates LLMs capability to resolve 127 maintainability issues from 10 GitHub repositories. We use zero-shot prompting for Copilot**

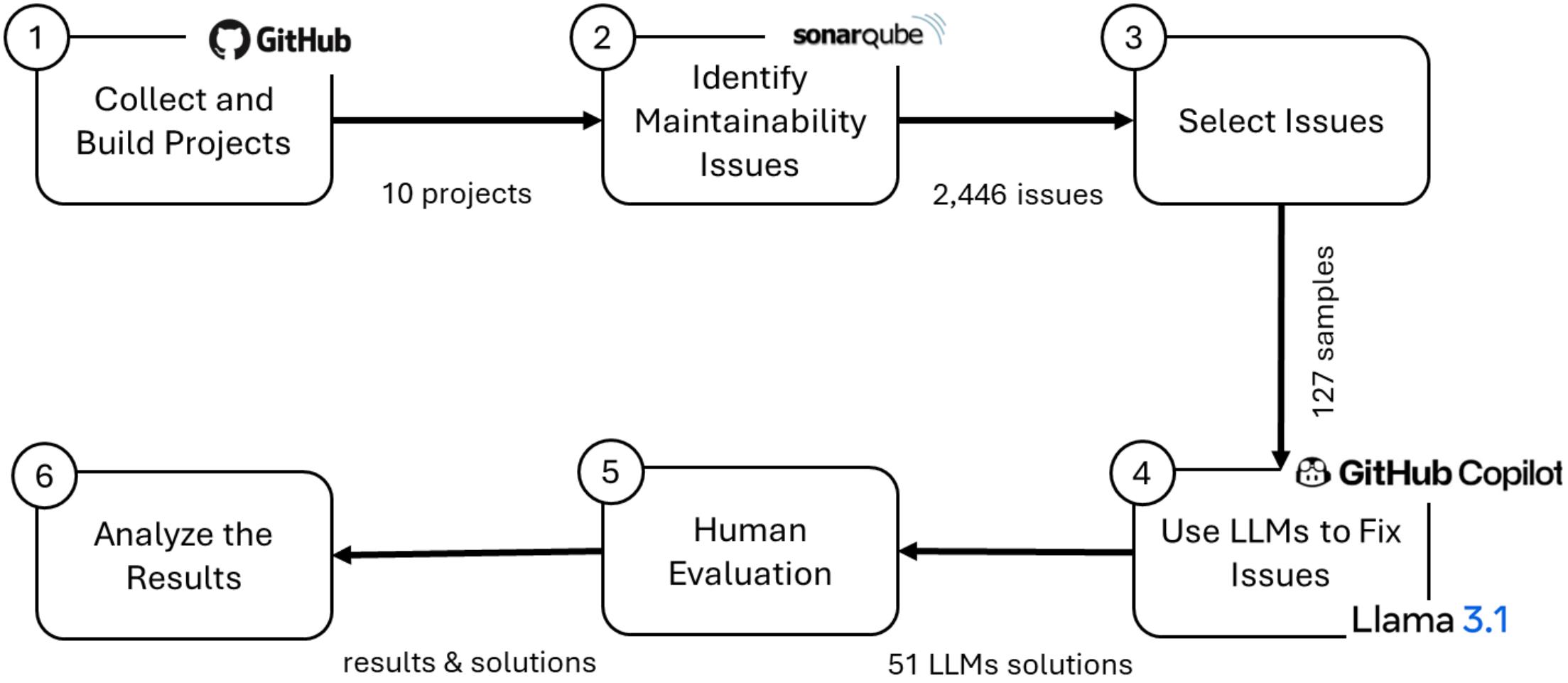
but using LLMs to refactor maintainability issues is underexplored and lacks relevant data [12]. In contrast to traditional automated tools that adhere to rigid rules, LLMs can provide an innovative approach to addressing maintainability issues.

Goal and Research Questions

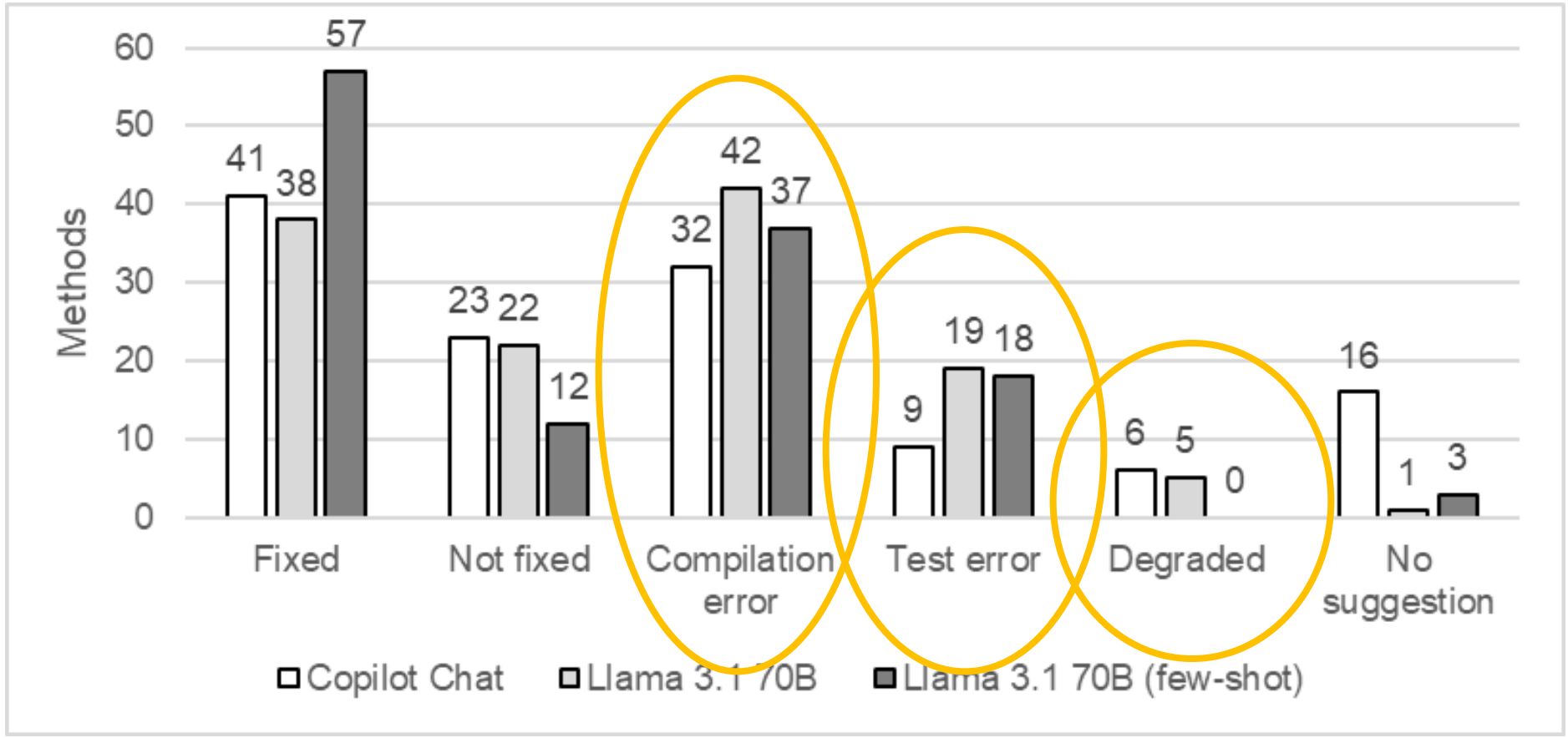
- Goal: Understand the strengths and drawbacks of LLMs in fixing code issues
  - Two LLMs (Copilot and Llama) refactored 127 code issues in 10 open-source Java projects

- Research questions
  1. To what extent can LLMs **fix** maintainability issues?
  2. What are the main **mistakes** made by LLMs when fixing code issues?
  3. To what extent do developers find LLM solutions more **readable**?

Study Steps


1
GitHub
Collect and Build Projects
10 projects
2
sonarqube
Identify Maintainability Issues
2,446 issues
3
Select Issues
127 samples
4
GitHub Copilot
Use LLMs to Fix Issues
Llama 3.1
51 LLMs solutions
5
Human Evaluation
results & solutions
6
Analyze the Results

Some Results about the AI Mistakes



All LLMs make mistakes, such as compilation errors and test failures

Some Findings

- Fixing rate is low
  - Few-shot Llama fixed ~45% of the methods, while zero-shot Copilot Chat and Llama fixed ~330%
  - LLMs also include many compilation and testing errors

- LLM refactoring improve code readability
  - Almost 70% of participants observed readability improvements

Bibliography

- S. Souza, A. Santana, E. Figueiredo, I. Muzetti, J. Montandon, L. Briand. "**Beyond Strict Rules: Assessing the Effectiveness of Large Language Models for Code Smell Detection**". Preprint, 2026.
- H. Nunes, E. Figueiredo, L. Soares, S. Nadi, F. Ferreira, G. Santos. "**Evaluating the Effectiveness of LLMs in Fixing Maintainability Issues in Real-World Projects**". IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2025