Feature Localization based on Large Language Models

FOSD Meeting 2026

Evelyn Rühl

University of Siegen

March 2026

Configuration vs. Code Faults

Addition Variant

- ☒ Calculator
  - ☒ Addition
  - ☐ Subtraction
  - ☐ Multiplication

```
public int calculate(int a, int b) {
    return a + b;
}
```

Multiplication Variant

- ☒ Calculator
  - ☐ Addition
  - ☐ Subtraction
  - ☒ Multiplication

```
public int calculate(int a, int b) {
    return a * b;
}
```

Configuration vs. Code Faults

Addition Variant

- ☒ Calculator
  - ☒ Addition
  - ☐ Subtraction
  - ☐ Multiplication

```
public int calculate(int a, int b) {
    return a + b;
}
```

Multiplication Variant

- ☒ Calculator
  - ☐ Addition
  - ☐ Subtraction
  - ☒ Multiplication

```
public int calculate(int a, int b) {
    return a * b;
}
```

Test Case:

- Input: $a = 2$, $b = 3$
- Expected (Addition variant): 5
- Actual (Multiplication variant): 6

Language Model Selection

- Large Language Models (LLMs) are provided with data extracted from 2 different program variants:
  - configuration of variant A,
  - code of variant A,
  - test suite output of variant B applied on variant A.
- LLMs are tasked to identify faulty lines in the code and related features.
- Evaluation using off-the-shelf LLMs with coding capabilities:

Name	Release Date	Parameter Count	Context Length	Developer	License
Code Llama	Aug 24, 2023	7 B	16K	Meta	Permissive commercial
qwen 2.5-Coder	Nov 12, 2024	0.5 B 7 B	32K 32K	Alibaba Cloud	Apache 2.0
gpt-oss	Aug 5, 2025	20 B	128K	OpenAI	Apache 2.0

Evaluation Dataset

- For the evaluation we extracted 6 configurable systems from the SPL2Go dataset[1].

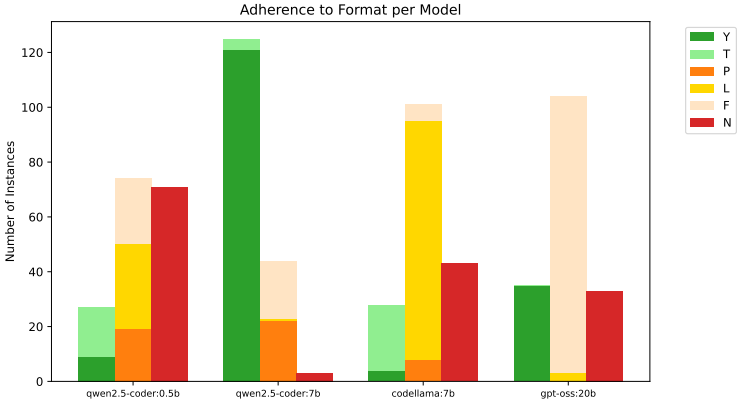
System	#Variants	#LOC	#Features	#Tests
ZipMe	25	3460	13	255.0
GPL	99	1944	27	86.9
Elevator-FH-JML	18	854	6	166.0
ExamDB	8	513	8	133.3
Email-FH-JML	27	439	9	86.0
BankAccountTP	34	143	8	19.8

[1] `http://spl2go.cs.ovgu.de/`

Evaluation of Feature Selection Localization

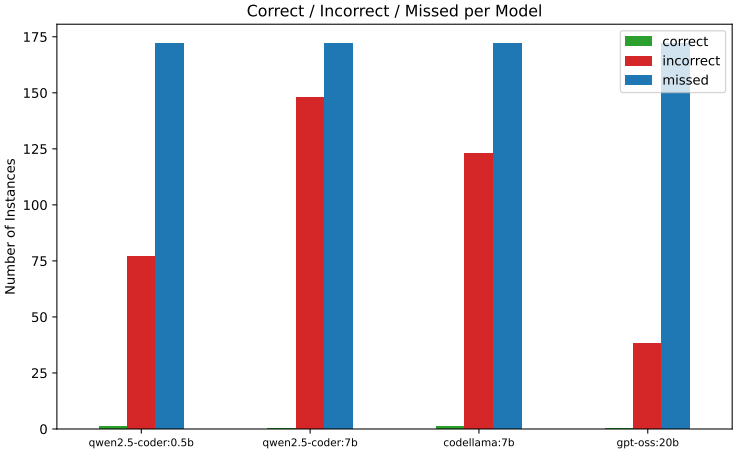
- **(RQ1)** How well do off-the-shelf LLMs adhere to the given output format when generating a response?
- **(RQ2)** How well can off-the-shelf LLMs identify variability-related faults in code using a few-shot approach?
- **(RQ3)** How well can off-the-shelf LLMs provide suitable suggestions for feature selection in a few-shot approach?
- **(RQ4)** How well do off-the-shelf LLMs perform in terms of response time and efficiency?

RQ1: Format Adherence



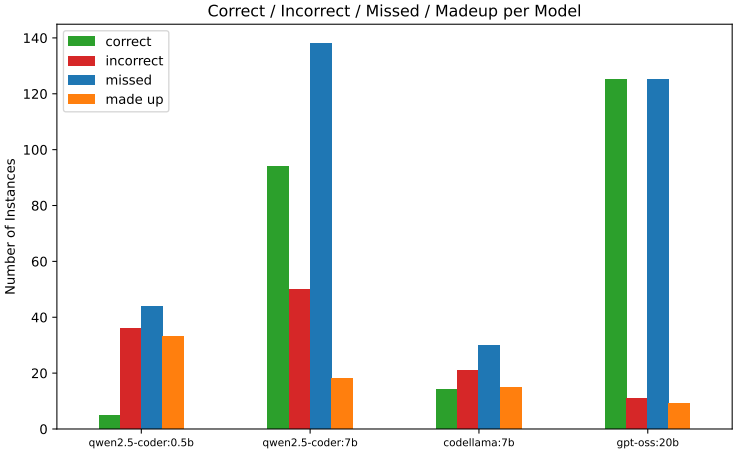
- Overall, LLMs show varying degrees of adherence to the specified output format
- Most responses contain information that can be processed further
- Smaller models tend to struggle more with format adherence

RQ2: Fault Localization



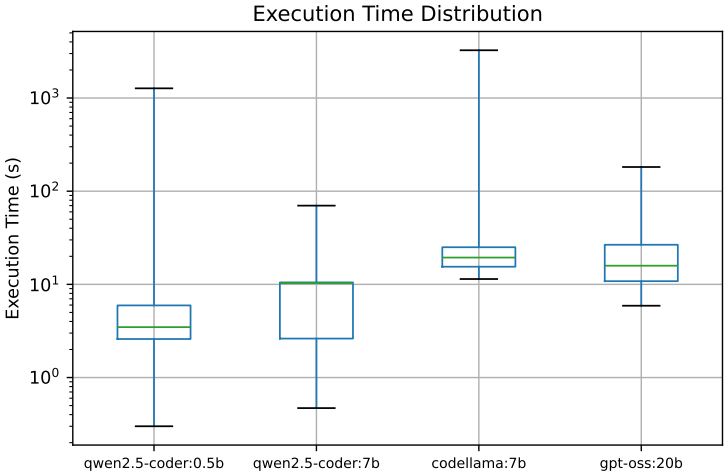
- Code Lines in the LLM response are compared to the difference in the code between program variants
- All models struggle to accurately identify code lines related to variability faults
- Only qwen2.5-coder:0.5b and codellama:7b correctly identified faulty lines in one instance each

RQ3: Feature Localization



- Feature identification shows more promising results compared to Fault Localization
- qwen2.5-coder:7b and gpt-oss:20b identified subset of correct features in most instances
- The suggested feature selection can be used to guide further repair steps

RQ4: Response Time



- Evaluation run on OMNI Cluster[2]
- Most responses are generated within a few seconds
- Longer response times often correlate to faulty responses that repeat similar content multiple times

[2] https://cluster.uni-siegen.de/

Future Work & Next Steps

- Shift focus toward identifying faulty code lines that result from incorrect feature selections
- Investigate which types of information best support LLM-based feature localization

Future Work & Next Steps

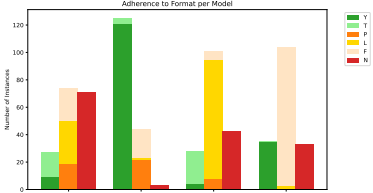
- Shift focus toward identifying faulty code lines that result from incorrect feature selections
- Investigate which types of information best support LLM-based feature localization

	V1	V2	V3	V4
Code A	✓	✓	✓	✓
Code B	-	-	-	-
Config A		✓	✓	✓
Config B		✓	✓	✓
Test Suite A				✓
Test Suite B			✓	✓
Test Suite Output	✓	✓	✓	✓

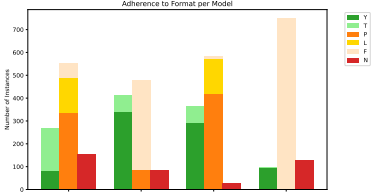
Backup

RQ1: Format Adherence

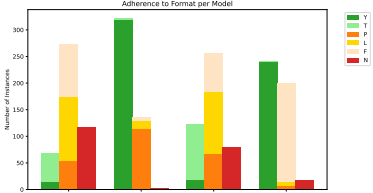
Elevator



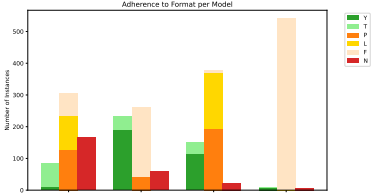
Bank



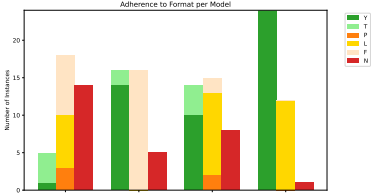
ZIP



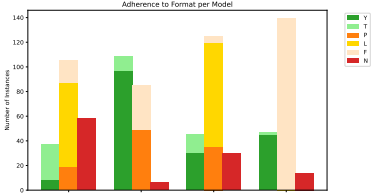
Email



Exam

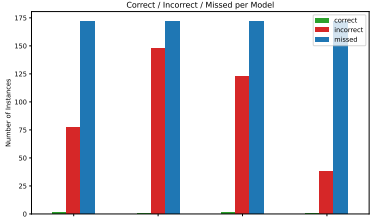


GPL

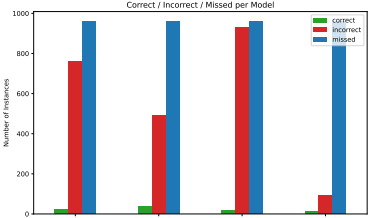


RQ2: Fault Localization

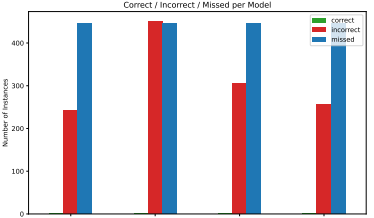
Elevator



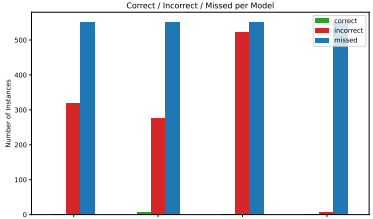
Bank



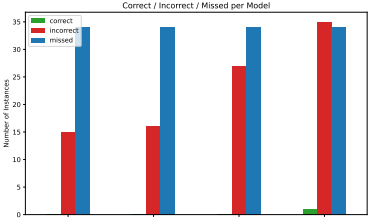
ZIP



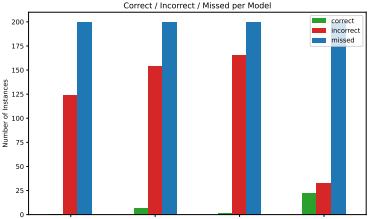
Email



Exam

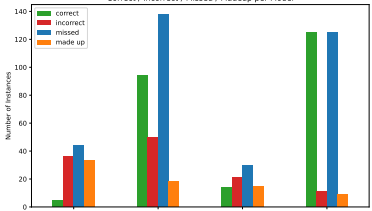


GPL

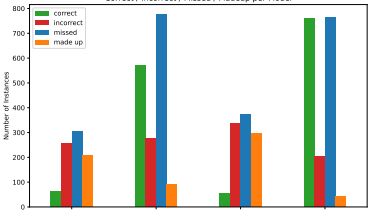


RQ3: Feature Localization

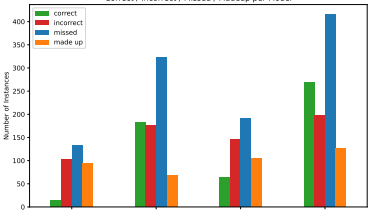
Elevator



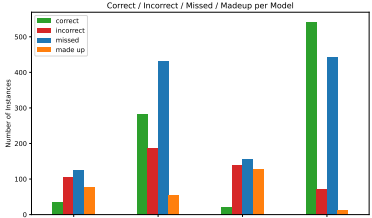
Bank



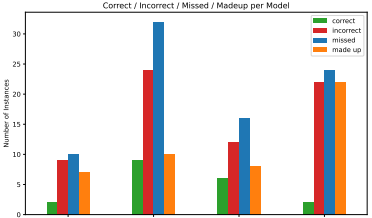
ZIP



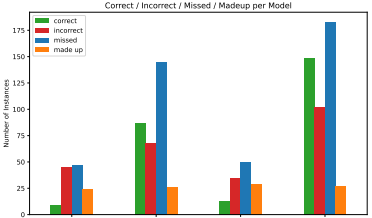
Email



Exam

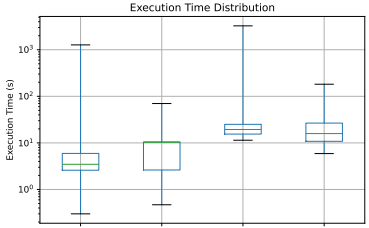


GPL

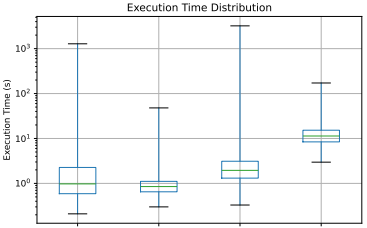


RQ4: Response Time

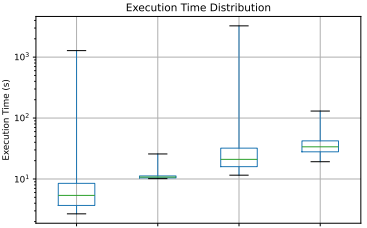
Elevator



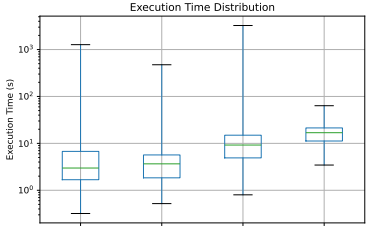
Bank



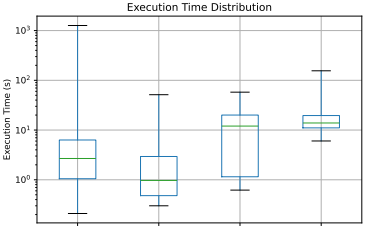
ZIP



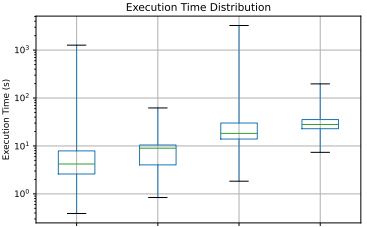
Email



Exam



GPL



Prompting 1/3

```
# Task: Java Feature Localization

You are given an old Java code snippet, a set of new test cases und the set of
    selected features used to run the test cases.

Your job is to identify the **minimal set of line(s)** in the code that correlate
    to the failures.
Additionally, specify the feature(s) that need to be changed.

# Example

Code:
```java
1: public int calculate(int a, int b) {
2:     int c = a;
3:     if (Config.ADDITION_ENABLED) {
4:         c = c + b;
5:     } else {
6:         c = c - b;
7:     }
4:     return c;
5: }
```
```

Prompting 2/3

```
Test cases:
[(5, 3), 2, 8]
[(4, 0), 4, 4]
[(2, 20), -18, 22]

Configuration:
Config.ADDITION_ENABLED = true

Expected answer for example:
Line 3:     if (Config.ADDITION_ENABLED) \{
Line 4:         c = c + b;
Feature(s) to change: ADDITION_ENABLED

# Java Code to Analyze

# Test Cases

# Configuration File
```

Prompting 3/3

```
# Instructions

- Identify and return the **minimum number of specific lines** in the Java code
    that correspond to failing test cases.
- Format each line as: 'Line X: code...'
- Suggest changes in the configuration file in the format 'Feature(s) to change:
    ...'
- Feature names should correspond to those listed in the configuration file
- Only output lines that are the root cause of the failure and the features to be
    changed, if any.
- Do NOT propose code changes or explanations.
- Keep the output short and follow the example format.

# Your Answer:
```