The ability of LLMs to detect and fix logical errors

Table of contents

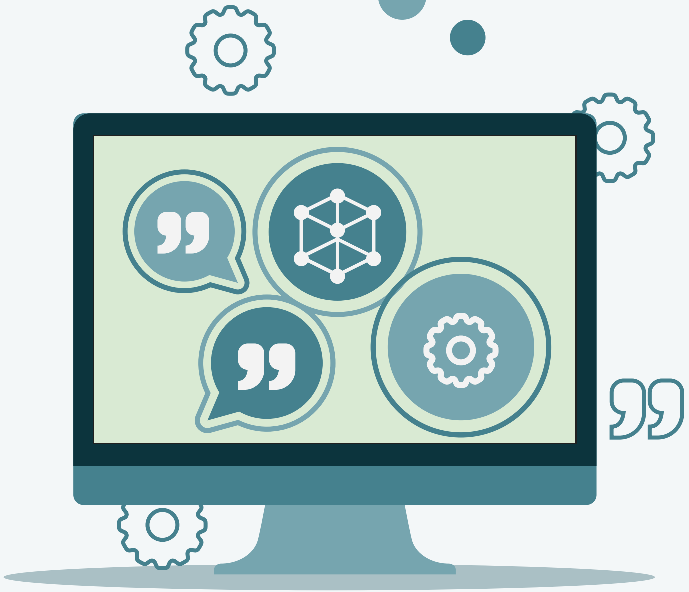
00

Introduction

Can LLM Decoding Logic Errors ?

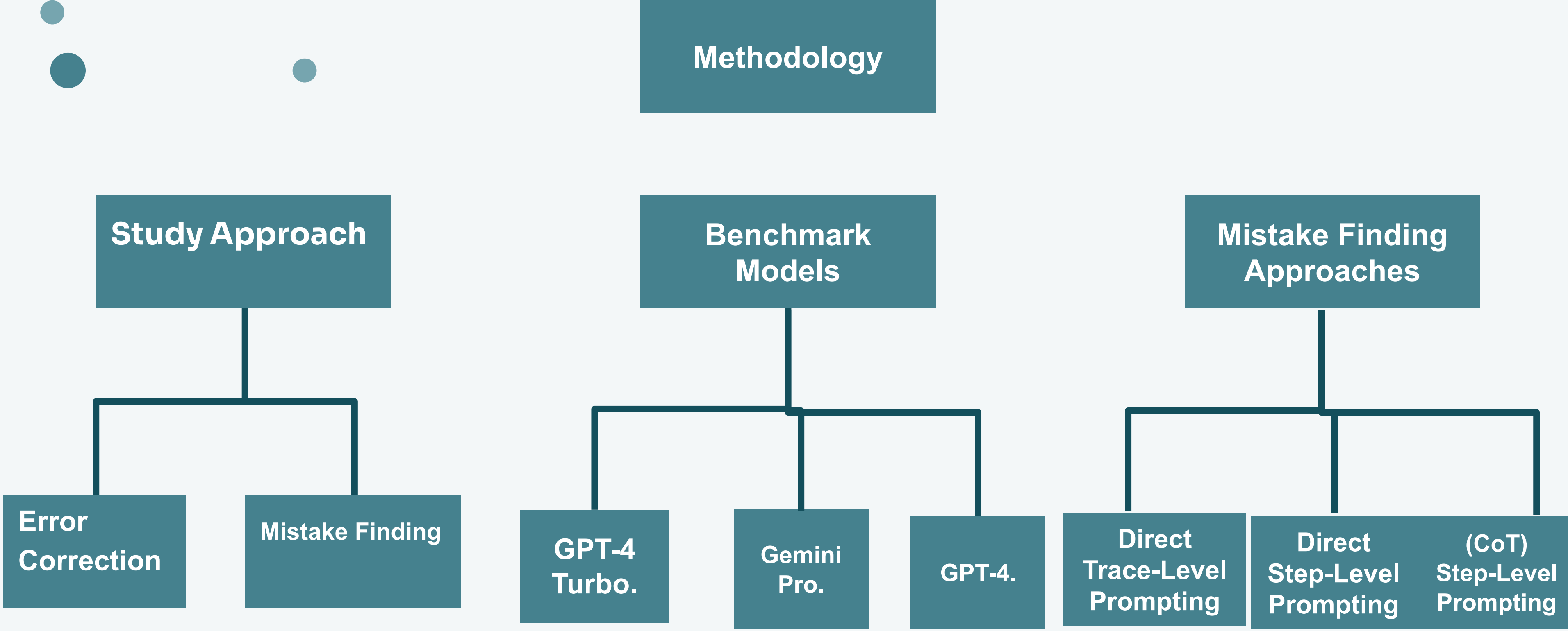
1

LLMs cannot find reasoning errors, but can correct them given the error location



Idea

Large Language Models (LLMs) have achieved impressive results in many natural language processing (NLP) tasks, such as question answering, text generation, and problem-solving. However, a major limitation remains: **LLMs struggle with reasoning accuracy, particularly in detecting logical errors in their own outputs**.


Methodology
Study Approach
Benchmark Models
Mistake Finding Approaches
Error Correction
Mistake Finding
GPT-4 Turbo.
Gemini Pro.
GPT-4.
Direct Trace-Level Prompting
Direct Step-Level Prompting
(CoT) Step-Level Prompting

Dataset

Dataset Name: BIG-Bench Mistake.

Size: 2186 Chain-of-Thought (CoT) traces.

Source: Generated using PaLM 2 Unicorn.

Annotations: Each trace is labeled with the first logical mistake.

Tasks:

. Word sorting

. Tracking shuffled objects

. Logical deduction

. Multistep arithmetic

. Dyck languages

Evaluation Metrics

First: Mistake Finding Accuracy.

Evaluates whether models correctly identify the first logical mistake in a trace.

Performance compared across three prompting strategies (trace-level, step-level, CoT-step-level).

Second: Error Correction Performance.

Measures accuracy improvement when correct mistake locations are provided.

Uses backtracking correction and compares against random resampling.

Results

Mistake Finding Accuracy (Best Model Results)

Model	Direct (trace)	Direct (step)	CoT (step)
Word sorting (11.7)			
GPT-4-Turbo	36.33	33.00	–
GPT-4	35.00	44.33	34.00
GPT-3.5-Turbo	11.33	15.00	15.67
Gemini Pro	10.67	–	–
PaLM 2 Unicorn	11.67	16.33	14.00
Tracking shuffled objects (5.4)			
GPT-4-Turbo	39.33	61.67	–
GPT-4	62.29	65.33	90.67
GPT-3.5-Turbo	10.10	1.67	19.00
Gemini Pro	37.67	–	–
PaLM 2 Unicorn	18.00	28.00	55.67
Logical deduction (8.3)			
GPT-4-Turbo	21.33	75.00	–
GPT-4	40.67	67.67	10.33
GPT-3.5-Turbo	2.00	25.33	9.67
Gemini Pro	8.67	–	–
PaLM 2 Unicorn	6.67	38.00	12.00
Multistep arithmetic (5.0)			
GPT-4-Turbo	38.33	43.33	–
GPT-4	44.00	42.67	41.00
GPT-3.5-Turbo	20.00	26.00	25.33
Gemini Pro	21.67	–	–
PaLM 2 Unicorn	22.00	21.67	23.67
Dyck languages† (24.5)			
GPT-4-Turbo	15.33*	28.67*	–
GPT-4	17.06	44.33*	41.00*
GPT-3.5-Turbo	8.78	5.91	1.86
Gemini Pro	2.00	–	–
PaLM 2 Unicorn	10.98	14.36	17.91

Results

Accuracy Gains After Providing Mistake Location

Held-out task	Trained classifier accuracy$_{mis}$ (Otter)	3-shot prompting accuracy$_{mis}$ (Unicorn)	Difference
Word sorting	**22.33**	11.67	+11.66
Tracking shuffled objects	**37.67**	18.00	+19.67
Logical deduction	6.00	**6.67**	-0.67
Multi-step arithmetic	**26.00**	22.00	+4.00
Dyck languages	**33.57**	10.98	+22.59

Results

Key Observations

GPT-4 performed best overall in mistake detection.

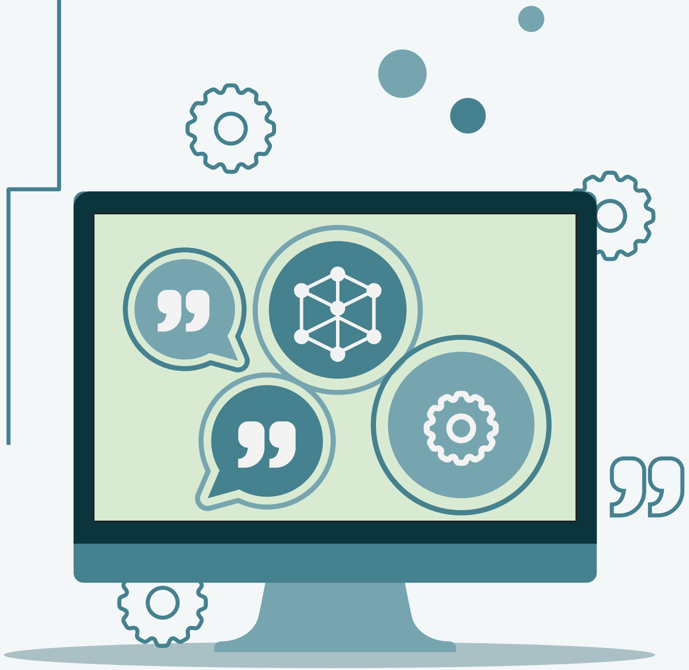
PaLM 2 Unicorn struggled significantly with mistake finding.

Providing explicit mistake locations improved accuracy by up to +43.92%.

Step-level prompting outperformed trace-level prompting.

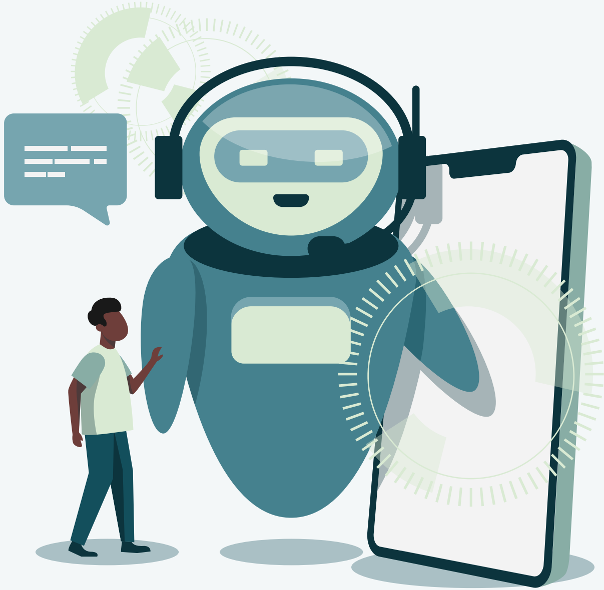
2

Decoding Logic Errors: A Comparative Study on Bug Detection by Students and Large Language Models



Idea

The study was conducted to evaluate whether Large Language Models (LLMs), such as GPT-3 and GPT-4, can effectively detect logic errors in C programs



Study aim to answer

How do students and LLMs compare in identifying logic errors in faulty code?

Which types of logic errors are easiest for students and LLMs to identify?

How many bugs or issues do students and LLMs identify when reviewing faulty and correct code?

Types of logic errors

Code Example 1

```
int LargestValue(int values[], int length) {
    int i, max;

    max = values[0];
    for (i = 1; i < length; i++) {
     if (values[i] > max) {
      max = values[i];
     }
    }

    return max;
}
```

Out of bounds

```
for (i = 0; i < length; i++) {
  if (values[i+1] > max) {
   max = values[i+1];
  }
```

Expression

```
if(values[i]>values[max]
```

Operator

```
if(values[i]<max)
```

The study investigates the ability of Large Language Models (LLMs), specifically GPT-3 and GPT-4, to detect logic errors in C programs and compares their performance to 964 introductory programming students.

2980 total student responses from collected from a C programming lab session

720 LLM responses generated for analysis.

Results

2980 total student responses collected from a C programming lab session, 720 LLM responses generated for analysis.

		Code Example 1				Code Example 2				Code Example 3			
Source		Bug 1	Bug 2	Bug 3	Correct	Bug 1	Bug 2	Bug 3	Correct	Bug 1	Bug 2	Bug 3	Correct
Student	correct	56	91	108	207	71	90	82	222	84	66	69	220
	incorrect	147	146	125	19	184	165	130	6	155	143	162	26
	rate	0.276	0.384	0.464	0.916	0.278	0.353	0.387	0.974	0.351	0.316	0.299	0.894
GPT-3	correct	30	23	8	30	29	27	24	29	29	30	24	12
	incorrect	0	4	22	0	0	0	1	1	0	0	6	17
	rate	1	0.852	0.267	1	1	1	0.96	0.967	1	1	0.800	0.414
GPT-4	correct	29	30	28	19	30	30	28	0	30	30	28	0
	incorrect	0	0	1	11	0	0	1	30	0	0	1	30
	rate	1	1	0.966	0.633	1	1	0.966	0	1	1	0.966	0

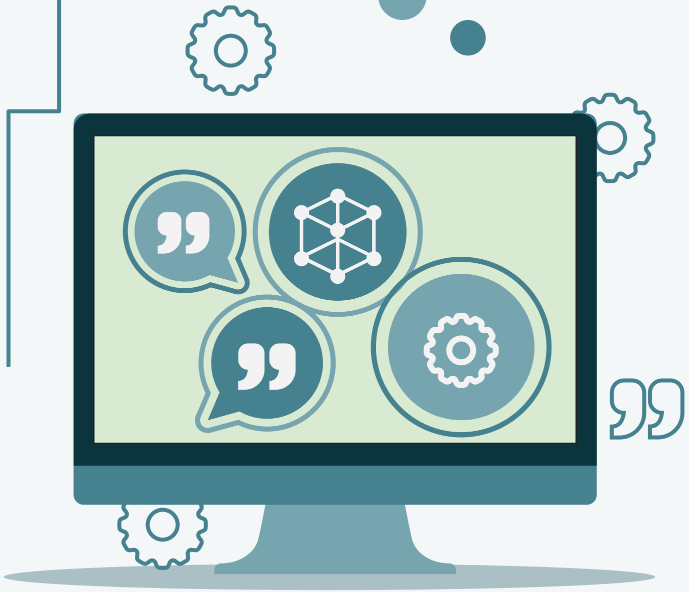
Accuracy

GPT-4	99.2 %
GPT-3	87.3 %
Student	34.5 %

3

Code Linting using Language Models

(June 2024).



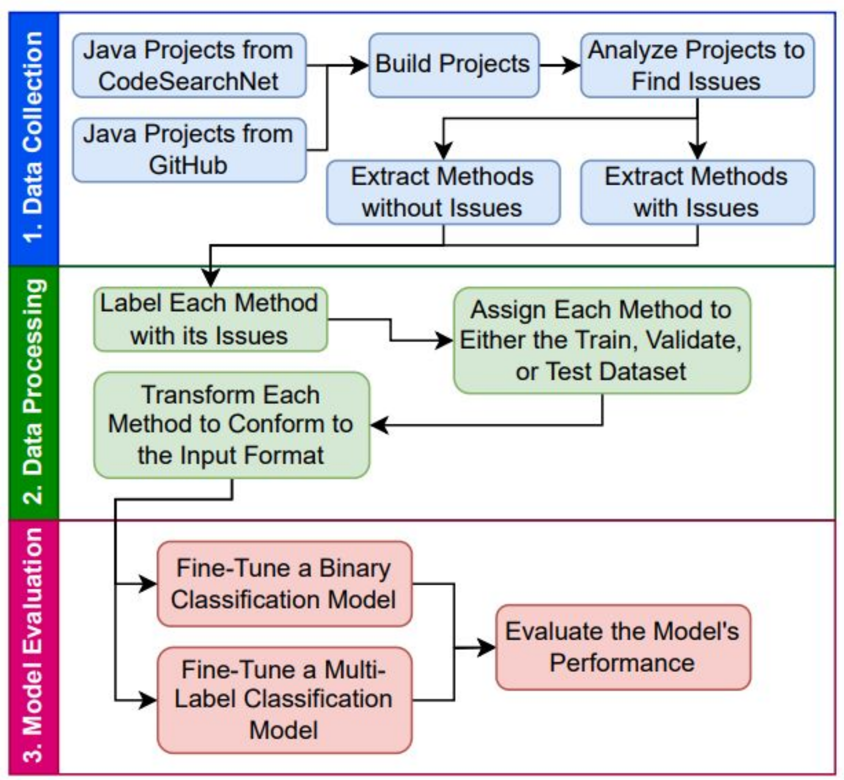
3

A Review of Methodology, Datasets, and Results

3.1 Introduction

- Traditional code linters are **language-specific** and **rule-based**
- They often produce **false positives** and **miss logical errors**
- The study explores **LLM-based linting** as a **versatile, language-independent** alternative
- **Objective:** Detect a wide range of **code issues** efficiently across multiple languages

3.2 Approach


1. Data Collection
Java Projects from CodeSearchNet
Java Projects from GitHub
Build Projects
Analyze Projects to Find Issues
Extract Methods without Issues
Extract Methods with Issues
2. Data Processing
Label Each Method with its Issues
Assign Each Method to Either the Train, Validate, or Test Dataset
Transform Each Method to Conform to the Input Format
3. Model Evaluation
Fine-Tune a Binary Classification Model
Fine-Tune a Multi-Label Classification Model
Evaluate the Model's Performance

3.3 Methodology

- Dataset Collection: Diverse code snippets with labeled issues (multi-language)

- Model Selection & Training:

    a. **Binary Classifier: Detects whether a code snippet has issues**

    GPT-4 and Codex: Codex performed slightly better in precision, while GPT-4 had a better recall, meaning GPT-4 detected more errors, but Codex was more precise in its predictions.

3.3 Methodology

a. **Multi-Label Classifier: Identifies the specific types of issues**

StarCoder: GPT-4 was also tested, but StarCoder showed better performance for classifying multiple types of logical errors at once

- **Training Process: The LLM learns error patterns from the dataset**

3.4 Evaluation Metrics

$$\text{Accuracy} = \frac{TP + TN}{|T|}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1} = 2\frac{Precision \cdot Recall}{Precision + Recall}$$

3.5 Performance Metrics

Binary Classifier Accuracy

84.9%

Traditional code linters 78.2%

Multi-Label Classifier Accuracy

83.6%

Traditional code linters 75.5%

Findings

1. LLM-based linting detects a broad range of issues

2. Outperforms traditional linters in detecting logical errors

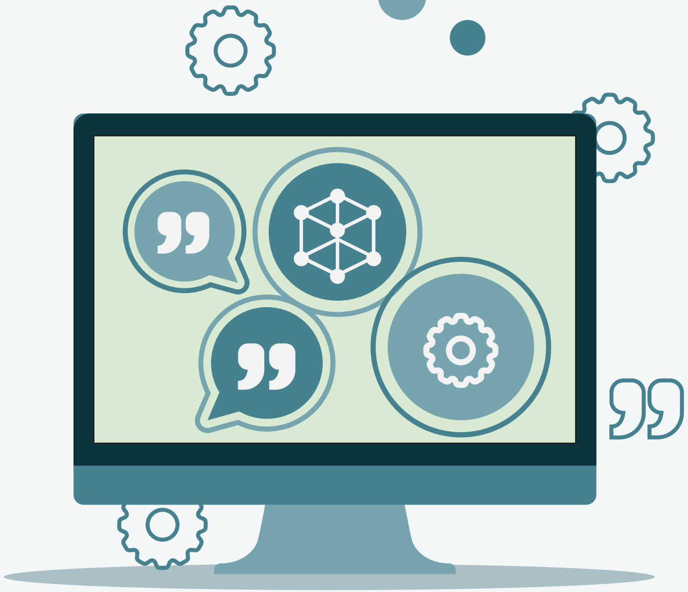
Conclusion & Implications

- **Key Contributions:**
  1. **Language-independent, deep-learning-driven code linting**
  2. **Enhanced detection of logical and semantic errors**

- **Future Work:**
  1. **Improving false positive rates**
  2. **Expanding training datasets**
  3. **Integration into real-world development tools**

4

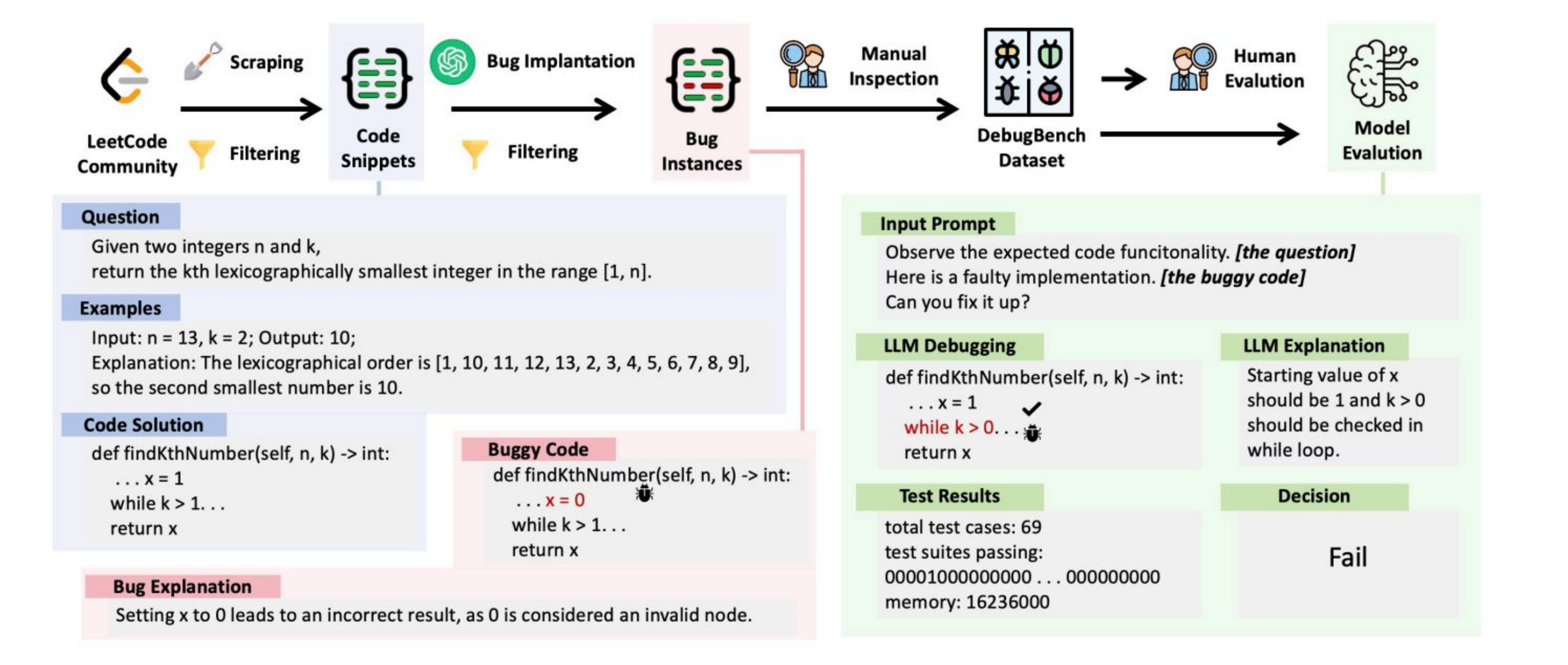
Debug Bench: Evaluating Debugging Capability of Large Language Models



4

Debug Bench: Evaluating Debugging Capability of Large Language Models

"What is DebugBench? Evaluating LLMs in Debugging Logical Errors"



LeetCode Community
Scraping
Filtering
Code Snippets
Bug Implantation
Filtering
Bug Instances
Manual Inspection
DebugBench Dataset
Human Evalution
Model Evalution
Question
Given two integers n and k,
return the kth lexicographically smallest integer in the range [1, n].
Examples
Input: n = 13, k = 2; Output: 10;
Explanation: The lexicographical order is [1, 10, 11, 12, 13, 2, 3, 4, 5, 6, 7, 8, 9],
so the second smallest number is 10.
Code Solution
def findKthNumber(self, n, k) -> int:
. . . x = 1
while k > 1. . .
return x
Buggy Code
def findKthNumber(self, n, k) -> int:
. . . x = 0
while k > 1. . .
return x
Bug Explanation
Setting x to 0 leads to an incorrect result, as 0 is considered an invalid node.
Input Prompt
Observe the expected code funcitonality. [the question]
Here is a faulty implementation. [the buggy code]
Can you fix it up?
LLM Debugging
def findKthNumber(self, n, k) -> int:
. . . x = 1
while k > 0. . .
return x
LLM Explanation
Starting value of x
should be 1 and k > 0
should be checked in
while loop.
Test Results
total test cases: 69
test suites passing:
00001000000000 . . . 000000000
memory: 16236000
Decision
Fail

Dataset

Size: 4,253 code snippets with implanted bugs.

Bug Types

- Syntax (e.g., missing semicolons).
- Reference (e.g., undefined variables).
- Logic (e.g., incorrect loop conditions).
- Multiple (combined errors).

Type	Minor Type	Number
Syntax	misused ==/=	137
	missing colons	129
	unclosed parentheses	133
	illegal separation	68
	illegal indentation	45
	unclosed string	125
	illegal comment	124
Reference	faulty indexing	206
	undefined objects	187
	undefined methods	167
	illegal keywords	124
Logic	condition error	260
	operation error	180
	variable error	100
	other error	50
Multiple	double bugs	750
	triple bugs	750
	quadruple bugs	718

Methodology & Metrics

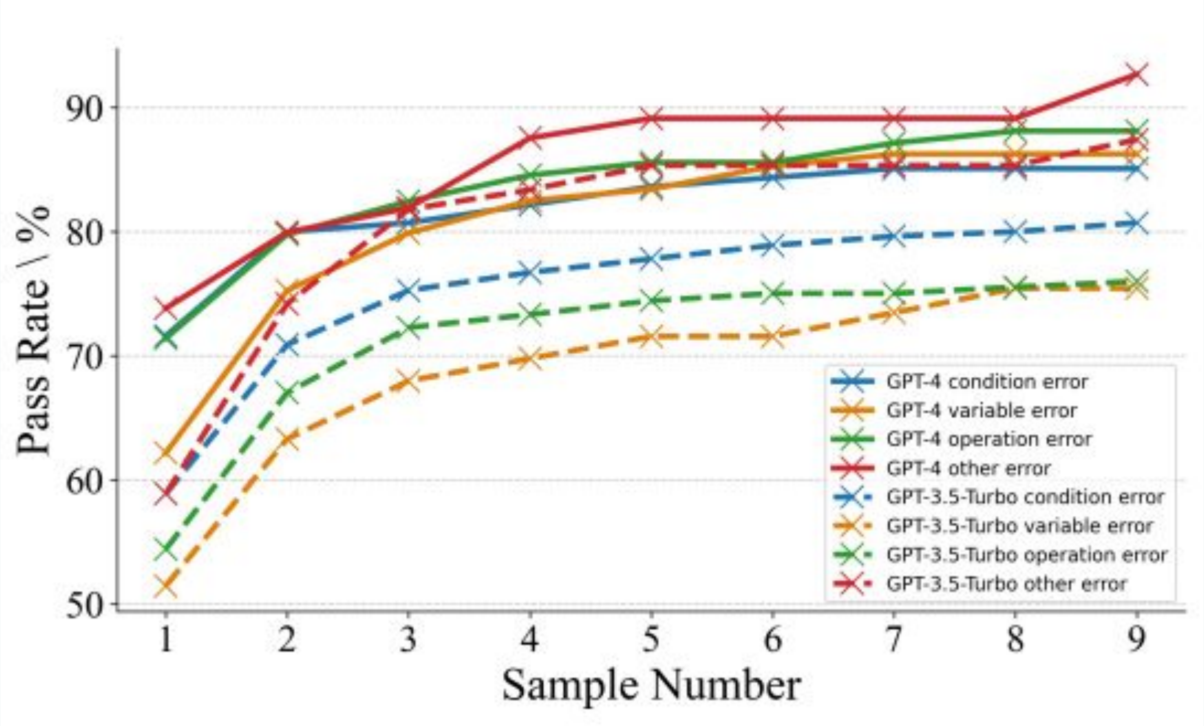
Tested Models

1. Closed-source: GPT-4, GPT-3.5.
2. Open-source: CodeLlama, Llama-3, DeepSeek-Coder, Mixtral.

Evaluation Process

- Models were asked to fix bugs in code snippets.
- Pass Rate: % of bugs fixed correctly.
- Human Baseline: 80% pass rate.

Result



Pass Rate \ %
90
80
70
60
50
1
2
3
4
5
6
7
8
9
Sample Number
GPT-4 condition error
GPT-4 variable error
GPT-4 operation error
GPT-4 other error
GPT-3.5-Turbo condition error
GPT-3.5-Turbo variable error
GPT-3.5-Turbo operation error
GPT-3.5-Turbo other error

Limitations

Synthetic Dataset

Some Bugs were artificially created, not from real-world projects

Open-Source Models

Poor performance due to limited training data

Future Research

Real-World Data

Expand benchmarks with real-world debugging scenarios (e.g., GitHub issues, industry codebases).

Training Enhancements

Curate datasets with debugging workflows to improve LLMs’ ability to reason about errors.

5

Detecting Logical Errors in Programming Assignments Using Code2Seq

5 Detecting Logical Errors in Programming Assignments Using Code2Seq

Idea

Evaluating code manually

Logical errors are harder to detect than syntax errors

Compilers help with syntax errors

Code2Seq

Methodology

- 50% of the dataset was modified with errors for better learning.
- Goal: Test if Code2Seq can detect errors after training.
- Two experiments on Java code with manually added errors.

Dataset

- **9,500 open-source Java projects from GitHub.**
- **Real-world code**
- **No artificially generated code to maintain reliability.**

Dataset

```java
for (Statement statement : method.getBody().
                get().getStatements()) {
    if (statement instanceof ForStmt) {
        ForStmt forStmt = (ForStmt) statement;
        if (forStmt.getCompare().isPresent()) {
            Expression expr = forStmt.getCompare().get();
            if (expr instanceof BinaryExpr) {
                BinaryExpr bExpr = (BinaryExpr) expr;
                if (forStmt.getInitialization().size()==1 &&
                    (bExpr.getRight().isFieldAccessExpr())){
                    list.add(forStmt);
                }
            }
        }
    }
}
```

Dataset

```
eligibleMethod(){
    for(int i = 0;
    i < arr.length; i++){
        //do something
    }
}
```

Evaluation Metrics

- **Using multiple metrics ensures a well-rounded evaluation.**

Accuracy

Precision

Recall

F1-score

Results

- High performance in detecting logical errors.
- Best results in detecting for-loop errors.
- Code2Seq can reduce manual effort in grading.

Limitations

- Might not perform well on complex errors.
- Only detects specific errors like for-loop and if-else mistakes
- Does not cover all types of logical errors.

Conclusion

- **Code2Seq detects logical errors with up to 90% accuracy.**
- **Shows strong potential for AI use in programming education.**
- **Future improvements can expand error detection capabilities.**

References

1. LLMs cannot find reasoning errors, but can correct them given the error location

2. Decoding Logic Errors: A Comparative Study on Bug Detection by Students and Large Language Models

3. LogiCode: an LLM-Driven Framework for Logical Anomaly Detection

4. Debug Bench: Evaluating Debugging Capability of Large Language Models

5. Improving LLM Classification of Logical Errors by Integrating Error Relationship into Prompts

Thanks