
DebugBench: Evaluating Debugging Capability of Large Language Models

Authors: Runchu Tian, Yining Ye, Yujia Qin, Xin Cong,
Yankai Lin, Yinxu Pan, Yesai Wu, Haotian Hui,
Weichuan Liu, Zhiyuan Liu, Maosong Sun

Presenter: Yvonne Zhou

Motivation

- While LLMs are proficient in code generation, debugging capabilities remain under-explored.
- Challenges:
 - Risk of data leakage with commonly used datasets.
 - Small dataset sizes
 - Limited bug type coverage
- Goal: Provide a robust, large-scale benchmark to evaluate LLM debugging performance effectively

Contributions

- Benchmarks to evaluate LLM Debugging capabilities
 - 4,253 instances with diverse bugs.
 - Four major bug categories: Syntax, Reference, Logic, and Multiples, and 18 minor types.
 - Snippet-level code in C++, Java, and Python.
 - Contain human baseline for comparison

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

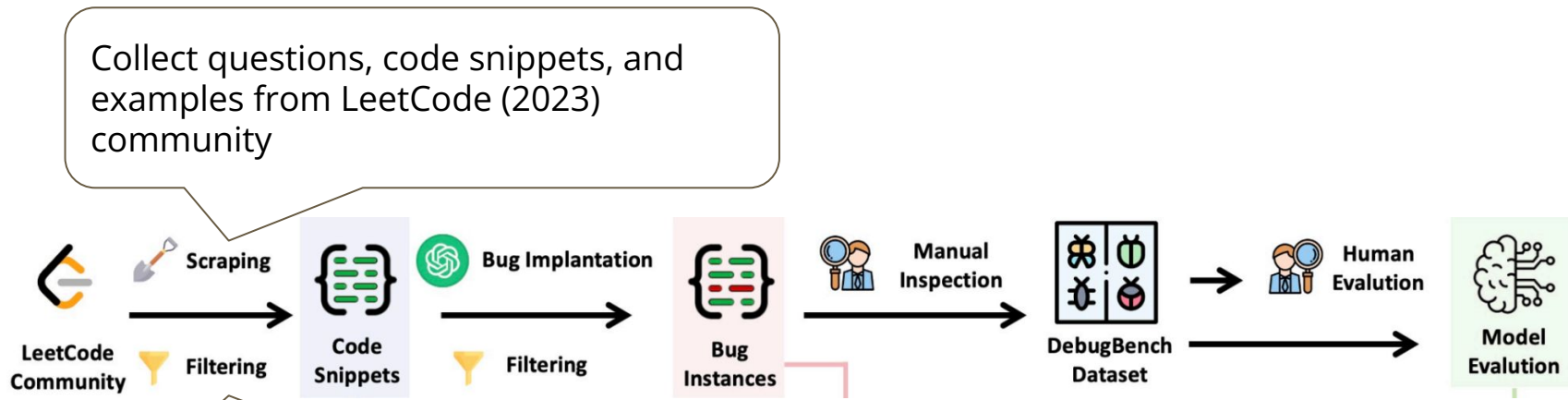
Contribution (cont'd)

Benchmark overcomes challenges faced by prior works:

- Reduce the risk of data leakage: LeetCode data released after July 2022
- Fine-grained evaluation: develop a bug taxonomy based on Barr (2004)'s classification criteria
- Large data scales: prompt GPT-4 to implant bugs into the clean code
- Ensure integrity: automatic filtering and manual inspection

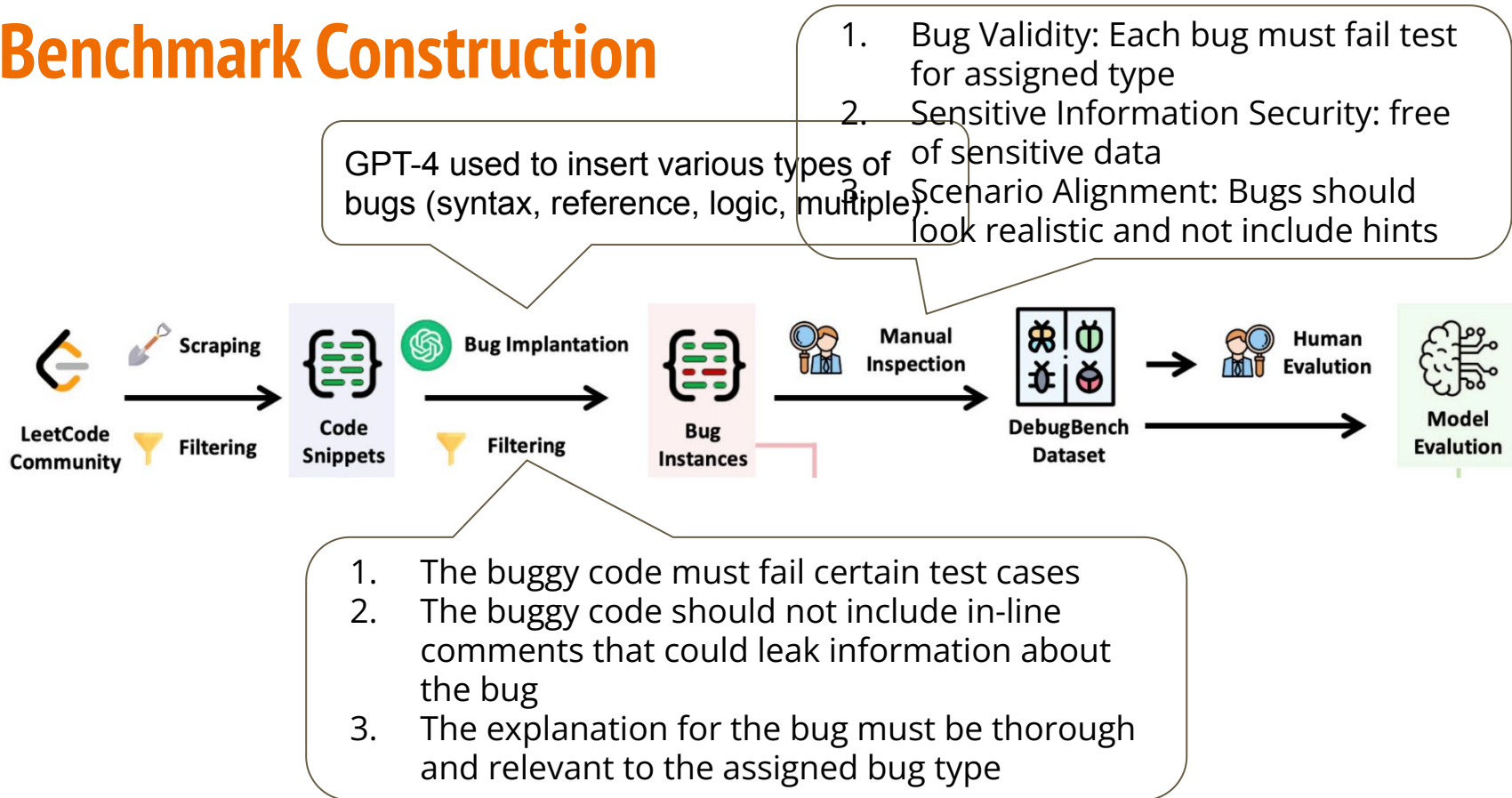
Work	Test Scale	Against Data Leakage	Bug Type Diversity	Model Diversity	Scenario Diversity
Prenner et al. (2022)	40	✗	✗	✗	✗
Sobania et al. (2023)	40	✗	✗	✗	✗
Xia and Zhang (2023a)	60	✗	✗	✓	✓
Zhang et al. (2023)	151	✓	✗	✓	✓
DebugBench	4,253	✓	✓	✓	✓

Benchmark Construction



1. The code solution must be correct
2. The instances must contain necessary information, eg. language, release time, and question id.
3. no earlier than July 2022

Benchmark Construction



Evaluation

- Six Models:
 - Closed-source models: GPT-4, GPT-3.5
 - Four open-source LLMs
- Scenarios: Zero-shot debugging
- Metric: Pass Rate on test suites provide by LeetCode (2023)

$$PR = \sum_{i=0}^n \frac{\bigwedge_{j=0}^m [a_{\theta_i^*}(x_i^j) = y_i^j]}{n} \times 100\%$$

- Contain baseline for comparison: human performance from three programmers with over four years of experience in programming

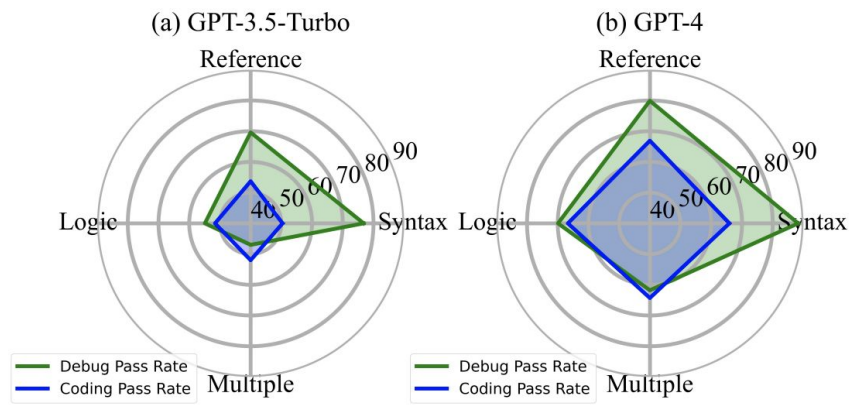
Findings

1. Closed-source models outperform open-source models

Major Category	Minor Type	CodeLlama	Llama-3	DeepSeek	Mixtral	gpt-3.5	gpt-4	human
Syntax	misused ==/=	18.2	58.4	68.6	12.4	70.5	87.9	11/12
	missing colons	23.3	44.2	62.8	25.6	80.9	93.6	12/12
	unclosed parentheses	27.1	51.9	86.5	14.3	81.2	89.6	12/12
	illegal separation	7.4	61.8	77.9	17.6	78.1	89.0	12/12
	illegal indentation	4.4	42.2	77.8	28.9	79.6	87.8	12/12
	unclosed string	28.8	48.0	94.4	9.6	82.0	91.4	12/12
	illegal comment	31.5	41.1	45.2	12.9	67.4	78.0	11/12
Reference	faulty indexing	27.2	53.4	67.5	11.7	72.9	77.1	10/12
	undefined objects	21.9	54.5	68.4	4.3	70.6	81.7	12/12
	undefined methods	15.0	46.7	43.7	6.6	59.3	78.5	11/12
	illegal keywords	58.1	13.5	57.3	18.5	76.1	83.6	11/12
Logic	condition error	13.5	46.5	47.7	22.3	58.5	73.1	10/12
	operation error	8.3	28.3	27.8	3.3	49.5	68.6	10/12
	variable error	10.0	29.0	38.0	10.0	52.3	63.1	9/12
	other error	8.0	40.0	44.0	2.0	61.1	72.2	10/12
Multiple	double bugs	3.3	43.2	46.1	8.4	56.4	70.7	11/12
	triple bugs	6.7	29.3	54.5	5.6	45.5	58.9	9/12
	quadruple bugs	5.0	31.2	49.2	4.5	38.7	55.9	8/12

Findings

1. Closed-source models outperform open-source models
2. Syntax and reference errors are easier than Logic and multiple error for LLM to debug.



Findings

1. Closed-source models outperform open-source models
2. Syntax and reference errors are easier than Logic and multiple error for LLM to debug.
3. Easy-to-debug if it is easy-to-code
 - a. Phi-Coefficient of LLMs' coding and debugging performance

Model	Bug Type	Phi-Coefficient
GPT-4	syntax	0.221
	reference	0.115
	logic	0.353
	multiple	0.273
GPT-3.5-Turbo	syntax	0.148
	reference	0.196
	logic	0.174
	multiple	0.298

Findings

1. Closed-source models outperform open-source models
2. Syntax and reference errors are easier than Logic and multiple error for LLM to debug.
3. Easy-to-debug if it is easy-to-code
 - a. Phi-Coefficient of LLMs' coding and debugging performance
4. Effect of Multiple Sampling: Better debugging at the cost of using more inference tokens
5. Effect of Runtime Feedback: Runtime Feedback is not always useful for debugging LLMs

Limitations

- AI implanted bugs, less realistic than real-world bugs.
- Evaluation test suits from the LeetCode platform only

Scientific Peer Reviewer

— Yize Cheng —

Paper Summary

- Introduces DebugBench, a benchmark designed to assess the debugging capabilities of LLMs
 - Consists of 4253 instances across C++, Python, and Java
 - Consists of 4 Primary bug types and 18 subtypes
- Aims at addressing limitations in previous benchmarks
 - Data Leakage
 - Small Scale
 - No Differentiation among bug types
- Runs experiments to:
 - Compare the debugging capabilities of human, closed-source LLMs, and open-source LLMs.
 - Compare LLM debugging capabilities on different bug types
 - Find correlations between debugging capabilities and code generation capabilities

Technical Correctness - Fixable Major Issue

- **Claim:** *“closed-source models have inferior debugging capabilities compared to humans, and open-source models perform worse in debugging compared to closed-source models”*
 - Human performance: “three programmers, each with over four years of experience in programming”
 - ← **Neither a good representative of the average or optimal human performance**
 - ← **Human can use IDE, with breakpoint features and many more. LLMs just read the code. Why not consider ACI like SWE-agent?**

Noisy estimation — The claim is insufficiently substantiated

Technical Correctness - Fixable Major Issue

- **Claim:** *"closed-source models have inferior debugging capabilities compared to humans, and open-source models perform worse in debugging compared to closed-source models"*
 - Closed-source models: GPT-4, GPT-3.5-Turbo
 - Open-source models: CodeLlama-7b-Instruct, Llama-3-8B-Instruct, DeepSeek-Coder-33B-Instruct, Mixtral-8x7B-Instruct

1.8 Trillion parameters

175 Billion parameters

Imbalance comparison — The claim is insufficiently substantiated

Scientific Contribution

- Provides a new dataset for public use
 - The paper introduces a new dataset that enables further research on language model debugging, giving the community a resource for benchmarking.
- Provides a Valuable Step Forward in an Established Field
 - The paper advances understanding of debugging capabilities across closed- and open-source models, contributing useful insights to the field.

Presentation - Minor Flaws

The final benchmark contains 4253 instances

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

sum=4253

Downloads last month **216**

`</>` Use this dataset ▾ Edit dataset card ⋮

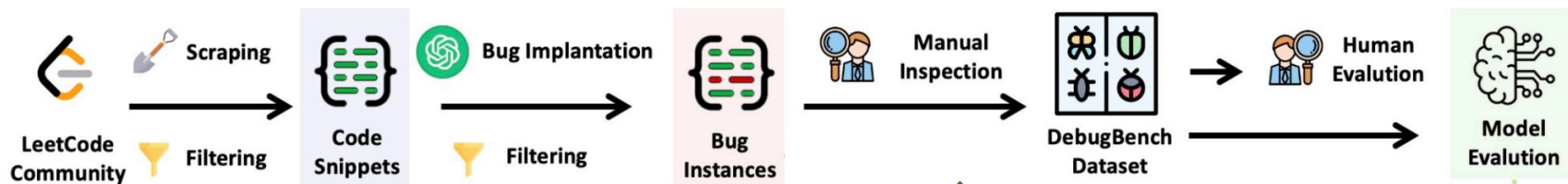
Size of downloaded dataset files:
20.1 MB

Size of the auto-converted Parquet files:
3.06 MB

Number of rows:
4,253

Presentation - Minor Flaws

The final benchmark contains 4253 instances, but...



79.2% of the 3,000 bug-implanted instances pass the filtering process.

92.1% pass here

Criteria	Pass Rate/%
Bug Validity	97.4
Sensitive Information Security	100.0
Scenario Alignment	93.2
All Three criteria	92.1

So how did they ended up with 4253 instances?

Comments - Strengths

- The paper introduces DebugBench, a benchmark specifically designed for evaluating debugging skills in LLMs, addressing an underexplored area in model assessment.
- DebugBench includes 4,253 instances across C++, Java, and Python, with various bug types, offering a thorough testing ground.
- They ran some experiments to compare debugging capabilities of different LLMs and humans, and investigate the role of run-time feedback for debugging and the correlation between debugging and code generation.

Comments - Weaknesses

- Experimental setup does not adequately support the claim regarding comparative debugging performance
 - Imbalanced comparison and limited sample size for human performance
- All instances in the dataset are derived from LeetCode questions, which represent isolated code snippets.
 - Real-world debugging often involves cross-file dependencies, where bugs may emerge from interactions between files rather than isolated code segments.
 - This benchmark thus captures only a simplified debugging scenario.

Comments - Weaknesses

- Although the authors aim to address data leakage concerns, LLMs are evolving, and this approach may not be a fully effective solution in the long term.
- Using LLMs to assess the realism of synthesized bugs is not an accurate approach.
 - LLMs are not specifically optimized for this purpose, and it is unclear whether such considerations were addressed during the LLM alignment process.
 - A more reliable method would involve having highly experienced practitioners in the field examine the bugs. Their expertise could better determine whether these bugs reflect those commonly encountered in real-world scenarios.

Recommended Decision

- Weak Reject (Can be Convinced by a Champion)
- Confidence: Highly Confident.

Archaeologist

— Utkarsh Tyagi —

Previous Work

A Critical Review of Large Language Model on Software Engineering: An Example from ChatGPT and Automated Program Repair

Zhang, Q., Zhang, T., Zhai, J., Fang, C., Yu, B., Sun, W., & Chen, Z. (2023) (<https://arxiv.org/pdf/2310.08879>)

- The paper seeks to review the bug-fixing capabilities of ChatGPT on a clean APR benchmark.
- They introduce EvalGPTFix, a new benchmark with buggy and their corresponding fixed programs from competitive programming problems starting from 2023, after the training cutoff point of ChatGPT.
- Contributions:
 - Overlooked Issue of the data leakage.
 - Clean Benchmark. We construct a new APR benchmark EvalGPTFix
 - Extensive Study. We conduct an in-depth empirical analysis of how ChatGPT is applied to APR.
- Three research questions:
 - The effectiveness of ChatGPT on EvalGPTFix
 - What is the effect of different prompts on the repair performance of ChatGPT?
 - Can dialogues help ChatGPT in improving repair performance?

Previous Work

Results:

RQ1 - The performance of ChatGPT in EvalGPTFix shows that: (1) ChatGPT is effective in fixing different types of bugs, e.g., 96%, 100%, 50% and 71% of CE, TLE, RE and WA bugs are correctly fixed; (2) ChatGPT is able to fix 109 bugs in EvalGPTFix.

RQ2 - The performance under different prompts demonstrates that, ChatGPT can benefit from more advanced prompts with additional information. For example, compared with the basic prompt, 25, 18, and 10 more bugs can be fixed with error information, problem description, and buggy lines.

RQ3 - The performance under a dialogue study demonstrates that, ChatGPT can repair more difficult-to-fix bugs with dynamic execution feedback in an interaction manner, e.g., 9 bugs that have not been fixed in previous prompts are fixed successfully.

►Basic Prompt◀

There's a bug in the program below. Try to fix it and return the complete fix for the code in the form of the markdown code block.

📄 [CODE]

►Exceeded/ Runtime Error Prompt◀

There's a bug in the program below. Try to fix it and return the complete fix for the code in the form of the markdown code block.

📄 [CODE]

The following input triggers a Time Limit Exceeded/ Runtime Error:

📄 [INPUT]

The expected output is:

📄 [EXPECT]

►Dialogue Prompt◀

There's still a Compilation Error/ Time Limit Exceeded Error/ Runtime Error/ Wrong Answer Error in your code triggered by the input:

📄 [INPUT]

The expected output is:

📄 [EXPECT]

The actual output is:

📄 [OUTPUT]

Try to fix it again and return the complete fix for the code.

Previous Work

Work	Test Scale	Against Data Leakage	Bug Type Diversity	Model Diversity	Scenario Diversity
Prenner et al. (2022)	40	✗	✗	✗	✗
Sobania et al. (2023)	40	✗	✗	✗	✗
Xia and Zhang (2023a)	60	✗	✗	✓	✓
Zhang et al. (2023)	151	✓	✗	✓	✓
DebugBench	4,253	✓	✓	✓	✓

Table 1: Limitations of prior studies in LLM debugging. We introduce DebugBench, a new LLM debugging benchmark to overcome these deficiencies.

Subsequent Work

Beyond Correctness: Benchmarking Multi-Dimensional Code Generation For Large Language Models

Zheng, J., Cao, B., Ma, Z., Pan, R., Lin, H., Lu, Y., Han, X. and Sun, L., 2024 (<https://arxiv.org/pdf/2407.11470>)

- In recent years, researchers have proposed numerous benchmarks to evaluate the impressive coding capabilities of large language models (LLMs). However, current benchmarks primarily assess the accuracy of LLM-generated code, while neglecting other critical dimensions that also significantly impact code quality in real-world development.
- Therefore, this paper proposes the RACE benchmark, which comprehensively evaluates the quality of code generated by LLMs across 4 dimensions: Readability, mAintainability, Correctness, and Efficiency.

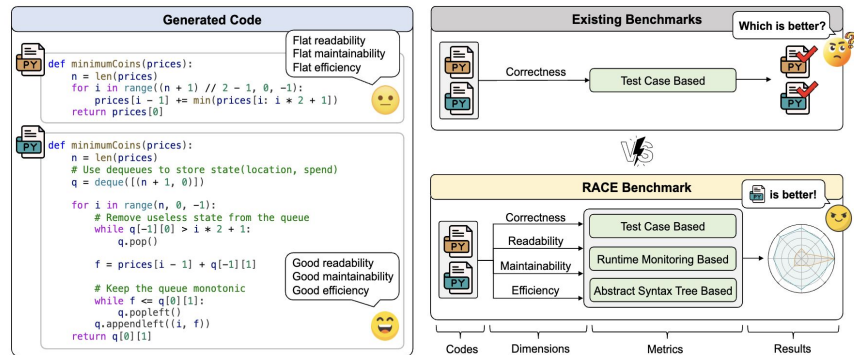


Figure 1: Current benchmarks perform single-dimension evaluations and mostly focus only on code correctness (upper right); our proposed RACE benchmark performs multi-dimensional code evaluations to identify truly high-quality code beyond correctness (lower right).

Subsequent Work

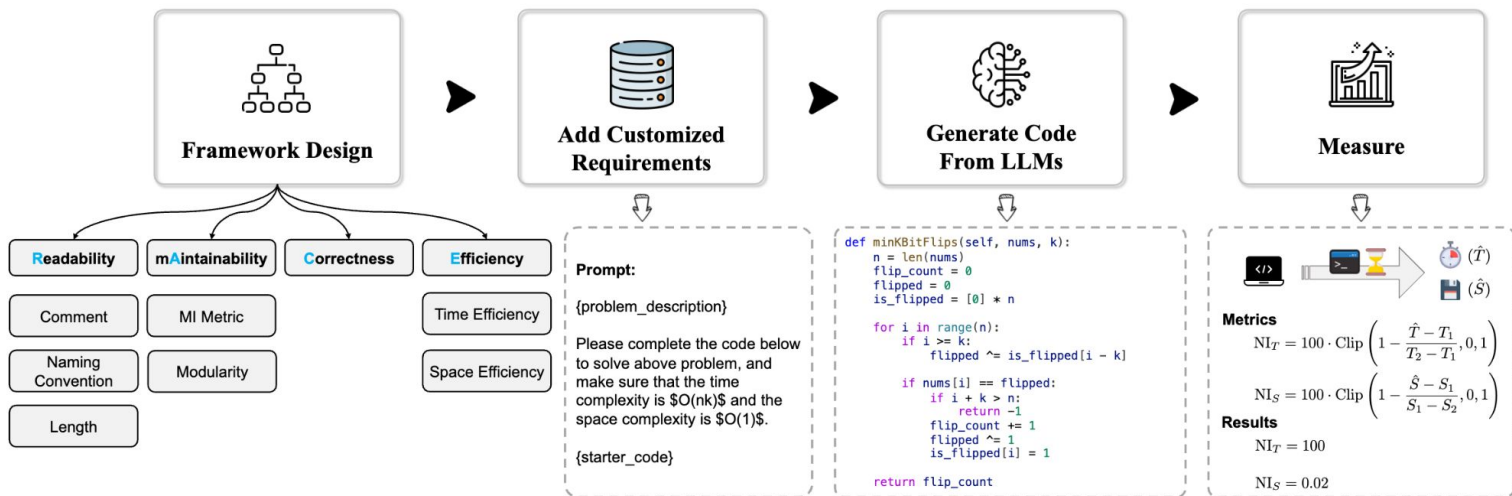


Figure 2: The overall evaluation pipeline in RACE benchmark.

Subsequent Work

- Current code LLMs still have considerable room for improvement in generating correct and user-compliant code across multiple dimensions. For instance, even the most advanced model, o1-mini, achieves only a score of 60.3 in time complexity, with most models below 50.
- Since current benchmarks use correctness as the sole guiding indicator, some LLMs perform well only on correctness but exhibit significant deficiencies in other dimensions. For example, Qwen2.5-Coder-7B-Ins demonstrates comparable levels of code correctness to GPT-4o-mini; however, GPT-4o-mini outperforms it by at least 5 percentage points regarding comments, modularity, and space complexity.

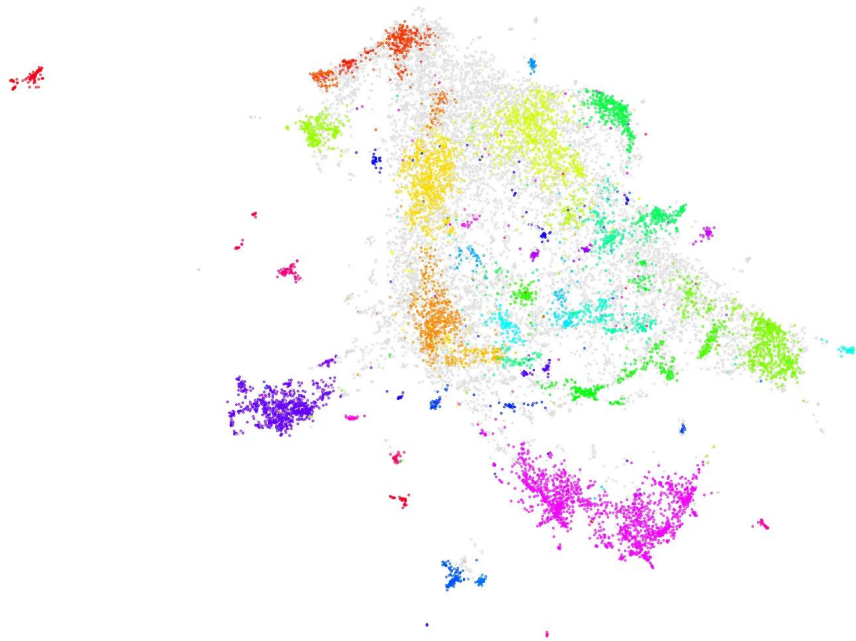
Models	RACE	C	R			M		E	
	Overall	C	RN	RL	RC	MI*	MC	NI _T *	NI _S *
Instruct-Type									
o1-mini-2024-09-12	63.5	70.1	80.7	47.5	77.7	64.4	66.1 [†]	60.3 [†]	40.0 [†]
Claude-3.5-Sonnet	<u>62.3</u>	<u>64.6</u>	74.4	52.0	65.5	75.3	<u>59.8</u>	<u>56.8</u>	49.7
GPT-4o	57.2	59.9	<u>78.6</u>	63.2	70.4	75.1	35.2	44.0	42.0
GPT-4o-mini	52.5	56.4	<u>67.6</u>	55.7	<u>72.9</u>	73.5	23.3	40.3	39.5
GPT-3.5-turbo-0125	43.6	44.7	51.4	46.1	47.5	80.2	18.5	27.5	36.5
CL-7B-Ins	23.2	23.9	17.8	23.4	22.2	71.8	7.2	8.2	8.8
CL-13B-Ins	26.9	24.4	22.9	23.6	29.0	82.1	7.6	10.4	16.1
CL-34B-Ins	24.4	26.0	21.9	17.5	10.7	73.2	8.5	14.4	13.8
DS-Coder-6.7B-Ins	39.8	39.2	45.8	46.6	50.0	79.3	8.2	27.1	30.0
DS-Coder-7B-Ins	38.9	39.9	36.8	46.0	53.7	79.6	8.9	25.1	26.8
DS-Coder-33B-Ins	44.8	44.7	59.0	53.5	54.0	75.7	11.3	35.3	36.1
DS-Coder-V2-16B-Ins	48.2	50.9	41.8	57.7	47.5	78.2	19.8	40.2	47.7
DS-V2.5-236B	57.1	59.0	72.2	<u>66.1</u>	65.8	72.9	33.9	46.4	<u>49.5</u>
CodeQwen1.5-7B-Chat	45.2	46.3	48.8	47.0	62.2	<u>82.3</u>	13.0	30.7	37.7
Qwen2.5-Coder-7B-Ins	49.0	57.1	53.0	51.8	61.3	78.6	17.6	37.0	33.7
Qwen2-72B-Ins	50.1	53.1	73.6	47.6	60.1	79.4	22.8	32.3	39.4
Qwen2.5-72B-Ins	61.3	64.1	77.2	72.1	72.8	76.7	40.4	47.9	49.4
Mixtral-8x22B	42.2	42.0	56.2	47.8	56.1	79.6	9.1	24.7	33.2
Llama3-8B-Ins	35.2	35.6	44.3	23.6	40.0	79.8	8.1	23.5	26.9
Llama3-70B-Ins	47.2	44.4	66.0	47.8	54.2	79.8	25.2	29.2	42.8

Based on the RACE benchmark, the performance results for each LLM in code correctness (C), readability (R), maintainability (M), and efficiency (E). RN, RL, RC, and EC denote the Name Convention, Length, Comments, and Complexity factor. MI denotes the Maintainability Index. MC denotes the Modularity factor. NIT and NIS are metrics for code efficiency. RACE Score represents the overall metrics at the dimension level.

Academic Researcher

Pranav Sivaraman

Embeddings Database of Debug Problems

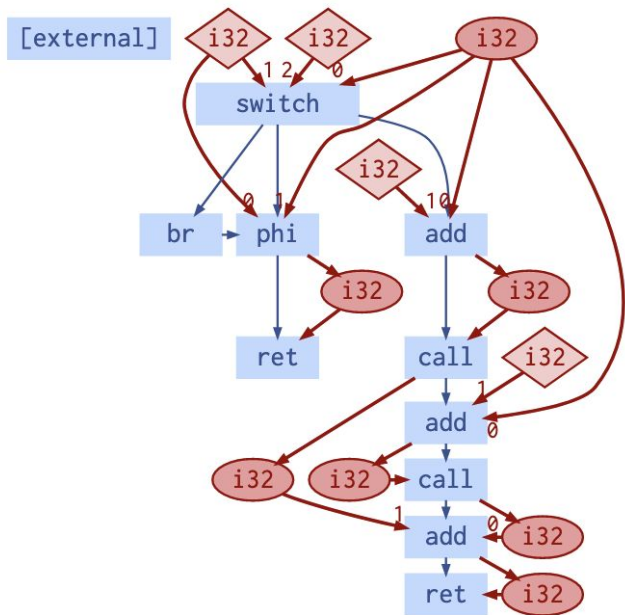


- Create a corpus of debugging samples with embedded representations for each sample.
- Generate the embedding of a new sample and find its nearest neighbors in the corpus.
- New samples can be real world

- Not possible without the dataset. DebugBench has a lot of data points and a taxonomy.

Better Representation for Code

- Instead of text, use alternative representations for code, such as ProGraML.
- Could using a different representation improve performance, especially for open-source models?





Hacker

— Georgios (George) Milis —

Introduction

Supposedly, LLMs are good coders...

But are they good debuggers?

DebugBench puts this to the test!

⚡ EvalPlus Tests ⚡

#	Model	pass@1
1	GPT-4-Turbo (April 2024) ⚡	⚡ 86.6
2	DeepSeek-Coder-V2-Instruct ⚡	⚡ 82.3
3	GPT-4-Turbo (Nov 2023) ⚡	⚡ 81.7
4	GPT-4 (May 2023) ⚡	⚡ 79.3
5	CodeQwen1.5-7B-Chat ⚡	⚡ 78.7
6	claude-3-opus (Mar 2024) ⚡	⚡ 77.4
7	DeepSeek-Coder-33B-instruct ⚡	⚡ 75
8	OpenCodeInterpreter-DS-33B ⚡❤️	⚡ 73.8
9	WizardCoder-33B-V1.1 ⚡	⚡ 73.2
10	Artigenz-Coder-DS-6.7B ⚡	⚡ 72.6

<https://evalplus.github.io/leaderboard.html>

Experiment setup

- Random subsets of DebugBench's buggy Python code
- An LLM
- LeetCode API 🍪 to test the fixes

The Gemini logo features the word "Gemini" in a sans-serif font. The "G" is blue, and the "emini" is a gradient from blue to purple. A purple four-pointed star is positioned above the "i".

Gemini

Buggy code:

```
class Solution:
    def fib(self, n: int) -> int:
        if n == 0:
            return 0
        if n == 1:
            return 1

        Fib = [-1 for _ in range(n + 1)]
        Fib[0] = 0
        Fib[1] = 1

        def compute_fib(n):
            if Fib[n] != -1:
                return Fib[n]
            Fib[n] = compute_fib(n - 1) + compute_fib(n - 2)
            return Fib[n]

        return compute_fib(n)
```

Prompt:

Observe the following faulty code. Your task is to fix up the code and explain on the modification in less than 20 words. You have to write the fixed code again. You should put `<code></code>` and `<exp></exp>` on the boundary of the code and the explanation. Do not write anything else in your response.
`\n{code}`

Fixed code:

```
class Solution:
    def fib(self, n: int) -> int:
        if n == 0:
            return 0
        if n == 1:
            return 1

        Fib = [-1 for _ in range(n + 1)]
        Fib[0] = 0
        Fib[1] = 1

        for i in range(2, n + 1):
            Fib[i] = Fib[i - 1] + Fib[i - 2]

        return Fib[n]
```

Results


Minor Type	CodeLlama	Llama-3	DeepSeek	Mixtral	gpt-3.5	gpt-4
misused ==/=	18.2	58.4	68.6	12.4	70.5	87.9
missing colons	23.3	44.2	62.8	25.6	80.9	93.6
unclosed parentheses	27.1	51.9	86.5	14.3	81.2	89.6
illegal separation	7.4	61.8	77.9	17.6	78.1	89.0
illegal indentation	4.4	42.2	77.8	28.9	79.6	87.8
unclosed string	28.8	48.0	94.4	9.6	82.0	91.4
illegal comment	31.5	41.1	45.2	12.9	67.4	78.0
faulty indexing	27.2	53.4	67.5	11.7	72.9	77.1
undefined objects	21.9	54.5	68.4	4.3	70.6	81.7
undefined methods	15.0	46.7	43.7	6.6	59.3	78.5
illegal keywords	58.1	13.5	57.3	18.5	76.1	83.6
condition error	13.5	46.5	47.7	22.3	58.5	73.1
operation error	8.3	28.3	27.8	3.3	49.5	68.6
variable error	10.0	29.0	38.0	10.0	52.3	63.1
other error	8.0	40.0	44.0	2.0	61.1	72.2
double bugs	3.3	43.2	46.1	8.4	56.4	70.7
triple bugs	6.7	29.3	54.5	5.6	45.5	58.9
quadruple bugs	5.0	31.2	49.2	4.5	38.7	55.9

DebugBench, Figure 6.

Type	Pass rate (%)
misused == or =	60
missing colons	75
unclosed parentheses	100
illegal indentation	100
unclosed string	75
illegal comment	40
Syntax average	74
faulty indexing	70
undefined objects	60
undefined methods	71
illegal keywords	100
Reference average	71
condition error	63
operation error	60
other error	80
Logic average	64
double	63
triple	50
quadruple	75
Multiple average	63



Conclusion

- Gemini's performance seems on par with  models
- Clear difference between syntax/reference VS harder/multiple bugs
- ❖ Authors' claims verified!

Future Work

- Evaluate on the entire DebugBench
- Gemini[★] cheated! Should be tested on LeetCode problems after 11/2023

Industry Practitioner

— Shreya Mishra —

Positives

- **Reduced Debugging Time & Costs**
 - Optimizes LLM debugging capabilities, reducing need for human involvement.
 - Quicker error resolution leads to substantial cost savings and reduced development time.
- **Enhanced Model Performance**
 - Trains models on an extensive, diverse bug dataset.
 - Improved model robustness across code issues, increasing product reliability and customer satisfaction.
- **Better Model Evaluation & Accountability**
 - High-quality testing environment for evaluating debugging capabilities.
 - Facilitates performance standards, boosting transparency and accountability.

Negatives

- **Initial Setup & Training Costs**
 - a. Requires significant initial investment of resources and time for setup, fine-tuning, and integration.
 - b. Could impact short-term budgets.
- **Risk of Overfitting to Synthetic Bugs**
 - a. Diverse bugs, yet synthetically generated.
 - b. Over Reliance on DebugBench may hinder generalization to all real-world code issues.
- **Dependency on External Benchmarks**
 - a. Evaluation relies on DebugBench's external criteria.
 - b. Any benchmark changes could affect long-term model assessment and consistency.

Industry Practitioner

— Sonal Kumar —

The Product

Develop a multi-layered debugging assistant that uses enhanced runtime feedback to provide contextually rich insights based on the type of error.

It will build on the DebugBench findings by tailoring feedback for each error type and dynamically adjusting feedback detail levels.

This system aims to increase debugging accuracy, especially for complex logic errors, where standard runtime feedback is currently inadequate.

Features

1. **Dynamic Error Type Detection:** Use initial code analysis to classify bugs into categories (syntax, reference, logic, etc.) before running diagnostics.
2. **Contextual Runtime Feedback:** Customize feedback based on error type. For example, syntax errors would get traditional stack traces, while logic errors would receive step-by-step logical flow analysis and automated sanity checks on variable states.
3. **Iterative Feedback Loops:** It will support iterative debugging by re-running code with adjusted parameters, gathering insights at each iteration to refine its guidance.
4. **Enhanced Explanations:** Incorporate reasoning steps, especially for logic-based errors, to make complex error insights more accessible.

Pros and Cons

Positive: It would make debugging more intuitive and accurate by tailoring feedback to error types, particularly helping developers navigate complex logic errors.

Negative: Implementing and fine-tuning this multi-layered system could be computationally intensive.

Private Investigator

Zeying Zhu

Third author of the paper -- Yujia Qin

- Recently completed his PhD in Tsinghua University, advised by Zhiyuan Liu (sixth author of the paper)
- Work Experience
 - Seed, ByteDance, 2024.7 - Now
 - Founder of SeqAI Inc., 2024.1 - 2024.7
- Primary Research Areas: LLM/VLM-based agent



Third author of the paper -- Yujia Qin

- Previous Works and recent focus
 - **ML-bench**: Evaluating Large Language Models and Agents for Machine Learning Tasks on Repository-Level Code, submitted to ICLR 2025.
 - **Tool learning**: Explore how to endow large models with higher-order cognitive abilities, allowing them to use complex tools in a manner similar to humans.
 - XAgent, ToolBench, ToolLLM
- Motivation
 - His research interests on LLM agents and the ability of large models to using complex tools. LLM itself as a tool can do tasks such as code debugging and generation. Also, his advisor Zhiyuan Liu, the sixth author of the paper, has a long research history on NLP and LLM.

Private Investigator

Ashish Seth

Advisor-- Zhiyuan Liu

- Currently an Associate Professor of Computer Science and Technology at Tsinghua University
- Academic Journey
 - Dec, 2021 - Now, Tsinghua University, Beijing.
 - Aug, 2011 - Dec, 2013, Postdoc, Tsinghua University, Beijing.
 - Aug, 2006 - Jul, 2011, PhD, Tsinghua University, Beijing.
 - Sep, 2002 - Jul, 2006, Undergraduate, Tsinghua University, Beijing.
- Primary Research Areas:
 - Large Language Model (LLM), Natural Language Processing (NLP), Knowledge Graphs, Representation Learning, and Social Computing.



Advisor-- Zhiyuan Liu

Recent Work:

AI powered tutoring system: An adaptive AI tutoring system powered by large language models, designed to enhance personalized learning through modular processes and memory-based progress tracking.

Multimodal Retrieval Systems: Using multimodal agents to return fusion results of images and texts to answer user questions.

Social Impact Accesser

— Mohammed Afaan
— Ansari

Positive Impact

- **Reduction of Human Effort in Bug Fixing:** By helping LLMs achieve human-level debugging, it could decrease manual debugging workloads, enhancing productivity and potentially improving job satisfaction in software development.
- **Educational Tool:** DebugBench could serve as a valuable educational resource for students and beginner developers, providing a platform to practice and develop real-world debugging skills.
- **Support for Open-Source Development:** The open-source nature of DebugBench encourages collaboration and innovation in the research community, allowing other researchers and developers to contribute to and improve LLM debugging capabilities.

Negative Impact

No major negatives impacts that directly affect the paper's main objective

- **Risk of Generalization Issues:** Focusing primarily on popular programming languages (C++, Java, Python) may lead to limited effectiveness of LLMs in debugging less common languages, restricting their utility in a wide range of development scenarios.
- **Potential for Overfitting:** LLMs trained and tested primarily on injected bugs, like those in DebugBench, may overfit on these specific patterns and perform less effectively on normally occurring, real-world bugs.