# Pull Requests with Bugs: Benchmarking Model Reasoning for Code Reviews

**Laurence Liang**
McGill University
Rootly AI Labs
laurence.liang@mail.mcgill.ca

## Abstract

Leveraging reasoning models for coding tasks is desirable, as reasoning models would leverage capabilities such as multi-hop reasoning to better understand code with missing context and consequently better understand user instructions. However, it is difficult to directly measure a model's reasoning capabilities, and existing coding benchmarks often struggle to evaluate models on realistic tasks. This paper introduces the "Pull Requests with Bugs" benchmark that isolates a multi-hop reasoning task for 60 code review questions sourced from real-world GitHub code. Model evaluations between Claude 3.7 and 4 reasoning models show a significant performance improvement of up to 25% with respect to non-reasoning Llama models. This reinforces this benchmark's ability to measure reasoning capabilities in a real-world coding task, while highlighting that even frontier reasoning models still have a gap to bridge with respect to demonstrating generalizable multi-hop reasoning capabilities.

## 1 Introduction

### 1.1 Reasoning Models

Transformer-based language models are probabilistic algorithms that select the chunk of text (the "token") that has the highest probability of following an existing sequence of text. Vaswani et al. [2023] Wei et al. reported that prompting a model to describe its thought processes in a step-by-step manner resulted in improved model performance on a variety of tasks. These early results encouraged the development of language models that respond with a reasoning block before a final response block. Recent work further demonstrates that adding key words such as "wait" coupled with supervised fine-tuning on reasoning traces shows notable model improvement on math and problem solving tasks. Muennighoff et al. [2025] Similar fine-tuning techniques also show strong performance on coding tasks, reinforcing the usefulness of reasoning traces on a variety of downstream tasks. Yu et al. [2025]

While prompting chain-of-thought reasoning elicits strong model performance on downstream tasks, it becomes relevant to determine whether language models can successfully reason when intermediary steps are not given as contextual data. This type of inference-based reasoning, known as "multi-hop reasoning", is an area of active interest as recent work shows encouraging signs of language models performing multi-hop reasonning in certain cases. [Yang et al., 2025, 2024]

### 1.2 Coding Evaluation

One of the first widely adopted coding evaluations, HumanEval, determined that language models such as GPT-3 were able to successfully write short Python functions based on specific instructions. Chen et al. [2021] Recent developments in coding benchmarks include SWE-Bench and LiveCodeBench's

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

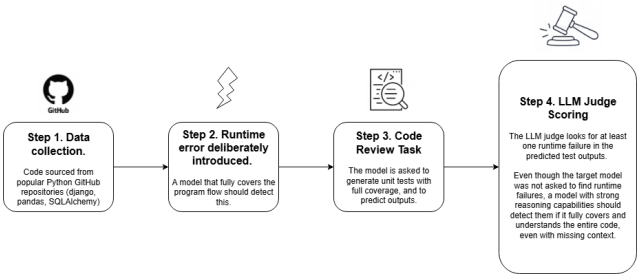


Figure 1: An outline of the Pull Requests with Bugs benchmark.

code execution subtasks that prompt a model to generate runnable programs. Jimenez et al. [2024], Jain et al. [2024] Other recent benchmarks such as CRUX-Eval and LiveCodeBench's unit test subtasks evaluate a model's ability to correctly predict the output of a function in a code snippet when provided with a function input. This latter type of code benchmarking can be considered as a form of multi-hop reasoning, as a model needs to infer intermediary states before predicting a final function output. Gu et al. [2024], Jain et al. [2024]

The conventional approach to benchmarking assumes that benchmarks contain noise, which subsequently justifies that a sufficiently strong model performance (for instance, 95% accuracy) would effectively saturate and "solve" a certain benchmark. However, recent work by Vendrow et al. introduce the concept of "platinum benchmarks", where a benchmark is designed such that a model must necessarily attain a 100% accuracy for the benchmark to be considered as fully solved. This paper's proposed benchmark, "Pull Requests with Bugs," follows the platinum benchmark paradigm, given that a model capable of multi-hop reasoning on a real-world coding task should be able to detect code samples with deliberate errors with perfect accuracy.

## 2 Method

### 2.1 Data Collection

Contrary to many existing coding benchmarks that evaluate models on simplified coding tasks (for example, solving a competitive programming question), this benchmark aims to evaluate models on a real-world scenario that would be used by developers in a production setting. Accordingly, this benchmark asks the target model to serve as a code reviewer for a GitHub pull request, where the model must generate unit tests for every new function in the newly added code, and predict the output of each unit test. This forces the model to perform multi-hop reasoning by following the flow of the program through every function call and code block to determine intermediary states. However, unknown to the model, every code snippet in this benchmark has a deliberate error that is introduced which would cause at least one of the unit tests to throw a runtime error, assuming that the model was successful in covering all of the areas of the newly added code. Figure 1 summarizes this process. Therefore, this benchmark evaluates a model on the following capabilities:

- to identify full test coverage by understanding conditional flow that occur in the code, and to generate unit test to achieve full test coverage
- to identify intermediary states between a unit test's input and its predicted output
- to eventually encounter the deliberate runtime error through multi-hop reasoning

Therefore, given that every question contains a deliberate runtime error, there must exist at least one runtime error for each question. Thus, identifying a runtime error for every question serves as a sufficiency criterion on whether a model is capable of multi-hop reasoning in a production-setting coding task.

This "Pull Requests with Bugs" benchmark uses 60 questions sourced from three popular Python GitHub repositories: django (a web framework), pandas (a tabular data processing library), and SQLAlchemy (a SQL library). All three repositories are widely established Python libraries with more than 10,000 stars. Each question consists of a hypothetical pull request that involves exactly one

Python file. In each question, a deliberate error is introduced by either adding an undeclared variable or adding an indentation error, features that a model capable of multi-hop reasoning should pick up.

### 2.2 Scoring

Each model receives the following system prompt as an instruction:

> You are a GitHub code assistant that reviews pull requests. Given the following added file, generate a unit test for every possible function and flow condition such that you cover every if/else block. Subsequently make your best guess at what the output of each unit test will be. Do not modify or alter the code, just determine what the output of each unit test is.

To simplify the evaluation process in an automated manner, OpenAI's GPT-5 model is used as a language model as a judge with the following evaluation prompt:

> You are an impartial judge.
>
> Given the following model response, you must return either 0 or 1, enclosed in brackets as <answer>0</answer> or <answer>1</answer>.
>
> If the model correctly identified an error any sort in at least one of the unit test responses, return <answer>1</answer>. Otherwise, return <answer>0</answer>.
>
> Model response:
>
> {{Model Response}}

While entrusting a language model to grade a target model's output may raise reliability issues at first glance, the authors of this paper propose that a judge LLM is capable of fairly grading model responses with human-level performance. This is due to the fact that the judge LLM is provided with the ground truth while the target LLM is not. Furthermore, GPT-5 is a highly capable model that is effective at following instructions, as demonstrated by OpenAI's benchmark evaluation on GPT-5. LLMs as a judge have also been used in existing benchmarks, and are considered as a standard way to grade target models. Zheng et al. [2023]

### 2.3 Model Selection

This paper evaluates two Anthropic models, Claude 3.7 Sonnet and Claude 4 Sonnet, as reasoning models. These two models were selected due to existing benchmarks that indicate that Sonnet 3.7 and 4 are both highly capable models with state of the art reasoning capabilities.

This paper also evaluates Qwen3-32B as an open source reasoning model to identify whether comparable reasoning capabilities are also present in open source models. Various Llama models are also evaluated: given that Llama 3 and Llama 4 models are not reasoning models, these results are used to determine to what extent model reasoning is responsible for improvements in benchmark performance.

All of the model evaluation was done using the Groq OpenBench framework Sah [2025], and open source inference was run through Groq as an inference provider.

## 3 Results

Table 1: Claude Model Performance with Different Reasoning Token Lengths

| Model | 0 reasoning tokens | 4,096 reasoning tokens | 8,192 reasoning tokens |
|---|---|---|---|
| Sonnet 4 | 0.683 | 0.783 | **0.900** |
| Sonnet 3.7 | 0.650 | **0.800** | 0.767 |

As shown in Table 1, adding reasoning tokens improves model performance for both Sonnet 3.7 and Sonnet 4. While this evaluation is still quite limited, it is encouraging to show that introducing a reasoning block correlates to improved performance on a coding task that requires multi-hop

reasoning. While improvements in Sonnet 4's performance are proportional to the number of reasoning tokens, Sonnet 3.7 experiences a plateau. These results suggest that different models react differently to increases in reasoning budgets, and that adding reasoning tokens in itself may not be sufficient to achieving a perfect model performance on real-world, reasoning-heavy tasks.

However, no model has yet attained a perfect accuracy. Given that a model with perfectly generalizable multi-hop capabilities should easily identify undeclared variables or erroneous indentations during intermediary states in its reasoning process, these results indicate that further work is needed to develop a model with strong multi-hop reasoning abilities.

Table 2: Open Source Model Comparison

| Model | Accuracy |
|---|---|
| Qwen 3 32B | 0.617 |
| Llama 3.1 8B | 0.367 |
| Llama 4 17B | 0.583 |
| Llama 3.3 70B | 0.517 |
| Llama 4 128B | **0.650** |

Table 2 compares an open-source reasoning model (Qwen 3 32B) to open-source non-reasoning Llama models. The Qwen 3 32B model using default reasoning settings is unable to reproduce the Claude models' strong performance in Table 1. This suggests that not all reasoning models perform equally on this benchmark, and that further work should empirically determine which model components (ie pretraining data, post-training methods, presence of reinforcement learning fine tuning) affect its performance.

The Llama models shown in ascending parameter size in Table 2 indicate that parameter scaling introduces some benefits to model performance, though model accuracy hits a plateau after a certain parameter scale. Furthermore, the best Llama model performances correspond to Sonnet model performances without reasoning tokens. This suggests that reasoning effectively contributes to model performance, when compared to non-reasoning models.

### 3.1 Relevant Applications

This Pull Requests with Bugs benchmark evaluates a model's ability to identify runtime failures during GitHub code reviews. As a result, the immediate application of this benchmark is to evaluate how different models perform in a code review task analogous to what AI-enabled code review tools undergo. Accordingly, this benchmark serves as an indicator of a model's ability to successfully identify software errors ("bugs") in real world code, which thereby serves as a relevant code understanding benchmark.

A secondary application is this benchmark's ability to measure multi-hop reasoning. As shown in the previous section, non-reasoning models struggle to perform well on this benchmarking, reinforcing its ability to measure model reasoning capabilities.

### 3.2 Limitations

As this benchmark is still preliminary, this benchmark's contains limitations that should be addressed in future work. Notably, the number of questions in the benchmark should be increased to enable more comprehensive model evaluations. The number of reasoning and non-reasoning models can also be expanded in future work.

In terms of model scoring, this evaluation uses an LLM-as-a-judge approach to identify signs of runtime errors in the model response. An LLM-as-a-judge has the benefit of being generalizable to different coding scenarios. However, this approach can be expensive for larger question counts, and its current binary scoring evaluation does not identify model failure types. Relevant improvements include: using a comprehensive pattern matching-based scorer that is more deterministic than a LLM-based scorer; a scorer that assigns partial points based on the unit test coverage and the validity of the predicted outputs; and identifying specific failure modes in the model response.

## 4 Conclusion

This paper introduces the “Pull Requests with Bugs" benchmark to evaluate reasoning models' abilities to perform multi-hop reasoning to identify bugs in real-world coding scenarios. Results on Claude Sonnet 3.7 and 4 show that scaling reasoning tokens can contribute to improved model performance, which is significantly better than non-reasoning models that attempted this benchmark using chain of thought generation. Future work should investigate expanding the size and task diversity of similar evaluations, increasing the scale of models being evaluated, and identify model components or training recipes that directly contribute to improved multi-hop reasoning performance on coding tasks.

## Reproducibility Statement

The code and dataset used for this paper are made openly accessible for reproducibility purposes.

The OpenBench fork used to run this benchmark is accessible at this link, while the code used to generate the dataset is accessible at this GitHub repository. The dataset is openly available on Hugging Face at this link.

If the links become invalid at any point in the future, please contact the author of this paper through email (as displayed on page one), who will provide the updated links.

## Acknowledgment

The author wishes to thank Sylvain Kalache and the team at the Rootly AI Labs for their generous support of this research, which enabled the acceptance of this paper at the First Efficient Reasoning Workshop at NeurIPS 2025.

## References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. Cruxeval: A benchmark for code reasoning, understanding and execution, 2024. URL https://arxiv.org/abs/2401.03065.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

Aarush Sah. openbench: Provider-agnostic, open-source evaluation infrastructure for language models, 2025. URL https://openbench.dev.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Joshua Vendrow, Edward Vendrow, Sara Beery, and Aleksander Madry. Do large language model benchmarks test reliability?, 2025. URL https://arxiv.org/abs/2502.03461.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Sohee Yang, Nora Kassner, Elena Gribovskaya, Sebastian Riedel, and Mor Geva. Do large language models perform latent multi-hop reasoning without exploiting shortcuts?, 2024.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. Do large language models latently perform multi-hop reasoning?, 2025. URL https://arxiv.org/abs/2402.16837.

Zhaojian Yu, Yinghao Wu, Yilun Zhao, Arman Cohan, and Xiao-Ping Zhang. Z1: Efficient test-time scaling with code, 2025. URL https://arxiv.org/abs/2504.00810.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.