The GitHub Recent Bugs Dataset for Evaluating LLM-based Debugging Applications

Jae Yong Lee
KAIST
Daejeon, South Korea
jaeyonglee0205@kaist.ac.kr

Sungmin Kang
KAIST
Daejeon, South Korea
sungmin.kang@kaist.ac.kr

Juyeon Yoon
KAIST
Daejeon, South Korea
juyeon.yoon@kaist.ac.kr

Shin Yoo
KAIST
Daejeon, South Korea
shin.yoo@kaist.ac.kr

2024 IEEE Conference on Software Testing, Verification and Validation (ICST) | 979-8-3503-0818-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICST60714.2024.00049

***Abstract*—While Large Language Models (LLMs) have demonstrated strong natural language and code processing capabilities, concern has been raised as to whether existing bug benchmarks are included in their training data. We examine the training data of the open-source LLM StarCoder, and find it likely that data from the widely used Defects4J benchmark was included, raising the possibility of its inclusion in the training data of the GPT model as well. This makes it difficult to tell how well LLM-based results on Defects4J would generalize, as for any results it would be unclear whether a technique's performance is due to LLM generalization or memorization. To remedy this issue and facilitate continued research on LLM-based SE, we present the GitHub Recent Bugs (GHRB) framework, which continuously gathers real-world Java bugs for use in evaluation of LLM-based techniques. To date, we have gathered 89 bugs reported after the GPT-3.5 training data cutoff point of September 2021.**

***Index Terms*—Benchmark, Debugging, Machine Learning**

I. Introduction

A significant portion of software engineering research revolves around the automatic handling of bugs [1]–[4]. As a result, software bug benchmarks using actual bugs from open-source software have been proposed. Examples of such benchmarks include the widely used Defects4J [5], Siemens [6], BugsInPy [7] and BugsJS [8] benchmarks.

However, the permissive licenses that led to these benchmarks also make these repositories a prime target as training data for code-based large language models. Large Language Models (LLMs) have been showing substantial performance gains relative to traditional techniques [2], [9]–[11]. LLMs need a large training dataset [12], and thus a large amount of software data from open-source repositories is gathered [13]. Our own findings show that bugs from the Defects4J benchmark are often included in the training data of the open-source LLM StarCoder [13]; it is reasonable to assume that the closed-source LLMs such as ChatGPT would be similar.

The potential overlap between existing bug benchmarks and LLM training data raises a critical question: are the state-of-the-art results from LLMs due to the strengths of LLM generalization, or simply due to memorization? While such a concern had been voiced in early literature [14], we are unaware of subsequent attempts to build a real-world bug dataset that is not likely to be part of LLM training data.

To this end, we propose the GitHub Recent Bugs (GHRB) framework, which continuously gathers recent Java bugs reported in GitHub. Up to now, we have collected 89 bugs reported after September 2021, which is the cutoff date for many GPT LLMs [15]. Additionally, these bugs were likely *not* used for the training of the open-source StarCoder LLM. Thus, researchers can evaluate LLM-based applications with GHRB without concern about data leakage.

II. Motivation

Under the assumption that LLM developers are likely to gather similar training data, we provide an initial assessment of the degree to which existing benchmark data is included in open-source LLM training data. To do so, we use the 'Data Portraits' tool[1], which allows users to check whether any text was included in the StarCoder training data. Using this tool, we evaluate the widely-used Defects4J v1.0 benchmark. We conservatively assume that if 90% of a test or method is included in StarCoder data, data about that bug was likely included, making it inappropriate for evaluation when using StarCoder. Worryingly, we find 35% of tests and 39% of buggy methods from Defects4J were included in StarCoder training data, using the 90% criterion mentioned above. In combination, 59% of Defects4J bugs were likely compromised. While it is difficult to justify, our finding strongly suggests that OpenAI may have included these projects in its LLM training data in a similar fashion, raising concerns about the validity of LLM-based SE evaluation using Defects4J.

III. Data Collection

This section describes the process used to collect bugs for GitHub Recent Bugs (GHRB), that are recent enough to avoid being included in LLM training data. Specifically, every bug should meet the following requirements:

1. The bug is in the source code: Every bug in the database should exist inside the source code of the project, and not in e.g. the configuration file.

2. The bug is reproducible: Every bug in the database should have at least one test that fails on the buggy version and passes on the fixed version.

3. The bug is isolated: For every bug in the database, the difference between the buggy and fixed versions should be directly related to the bug, and should not include any external changes such as feature additions or refactoring.

[1]https://stack.dataportraits.org/

979-8-3503-0818-1/24/$31.00 ©2024 IEEE
DOI 10.1109/ICST60714.2024.00049

TABLE I
PROJECTS AND NUMBER OF REAL BUGS AVAILABLE IN THE PUBLIC RELEASE OF GHRB (AS OF 19 JANUARY 2024)

Project	Bugs	LoC	Test LoC	# Tests	# Stars	Project	Bugs	LoC	Test LoC	# Tests	# Stars
fastjson	1	43.6k	143.4k	>435	25.4k	jackson-dataformat-xml	1	5.9k	9.4k	>2	541
nacos	6	215.9k	8.7k	2.7k	27.4k	gson	12	9.0k	19.6k	1.3k	22.4k
dubbo	1	146.7k	89.4k	3.3k	39.3k	sslcontext	6	3.7k	7.2k	497	406
rocketmq	19	150.3k	54.7k	1.7k	19.8k	jsoup	5	14.3k	12.5k	1.1k	10.3k
assertj	4	45.9k	161.4k	11.8k	2.4k	openapi-generator	6	9.8k	37.6k	1.8k	17.5k
checkstyle	16	41.7k	238.2k	4.5k	7.8k	seata	2	166.4k	30.5k	1.0k	24.2k
jackson-core	3	30.7k	44.8k	>100	2.2k	retrofit	1	3.9k	6.5k	329	42k
jackson-databind	5	73.1k	71.0k	>28	3.3k	Apktool	1	10.7k	3.4k	202	17.6k
						Total	**89**	**972**	**938k**	**30.8k**	**263k**

Using GitHub Actions, the bug gathering process is automatically triggered at the start of every month. An example result of the process can be found in a recent issue of our repository[2], where the result of the static filtering process (i.e., before running the tests to check if the issue reproduces) is summarized. Using another script, the bugs can be verified and added to the benchmark dataset on a monthly basis. Through this process, even if LLMs are updated with more recent training data, GHRB is capable of autonomously gathering more bugs for the use of evaluation.

IV. DATABASE OF REAL BUGS

While GHRB is a living framework that automatically finds new bugs every month, as of March 11th, 2024, the GHRB dataset consists of 89 bugs from 16 repositories. The chosen repositories vary in size and popularity, but all are primarily written in Java. Table I shows summary statistics of the dataset.

Similarly to Section II, we checked if the *oldest* bug-revealing tests in GHRB were included in the StarCoder training data, finding no overlap using 90% criterion[3], demonstrating that the bugs of GHRB are free from data contamination concerns when evaluating StarCoder-based applications. Since GHRB is based on commits merged after September 2021, GHRB is also 'safe' when used to evaluate when using all OpenAI LLMs except for the GPT-4-preview models, which have a data cutoff date of April 2023. Nonetheless, the number of bugs in GHRB after April 2023 are 24, a number which will grow due to the automatic bug gathering process of GHRB.

The GHRB dataset additionally provides the following:

Metadata. The bug database provides the creation date of the pull requests, the original bug report ID (ID of the pull request), and the URL of the bug report.

Bug revealing tests. The bug database includes of the list of one or more tests that reveal the bug, which fails on the buggy version and passes on the fixed version. For each test, the absolute path and the root cause is available.

Patch information. The bug database includes of the patch information, which is collected by taking the *git diff* between the buggy and fixed version.

[2]https://github.com/coinse/GHRB/issues/6

[3]The maximum overlap observed was 66%: note that non-zero overlap is inevitable as these projects were created before September 2021.

V. INTERFACE OF GHRB

In addition to the collection of bugs described in the previous section, the artifact of GHRB provides the following interfaces to facilitate ease of use.

Interface to Version Control Systems: The interface allow users to access the buggy and fixed versions of included repositories by using a simple flag, without the need of knowledge of the version control system adopted by the contributors. Each bug in the database is mapped to an integer ID, which is in the chronological order of its creation date. In addition, users can filter bugs by their creation date, enhancing compatibility with other LLMs. This creates a layer of abstraction over the version control system of each repository, as mentioned above.

Interface to Build Environments: The interface allows users to compile each target in a simple manner. During runtime, GHRB automatically derives the required build tool (i.e., maven, gradle), the required build tool versions, and the repository's use of a project-specific script for compilation. Overall, this abstraction relieves users from the burden of searching for build environments.

Interface to Testing: The interface allows users to easily test each compiled target. GHRB automatically creates a configuration file when a user checkouts to a specific version of a repository. During testing, the interface searches for the configuration file to derive the failing test information, allowing for efficient bug reproduction via test execution.

VI. CONCLUSION

We introduce GITHUB RECENT BUGS (GHRB), a real-world Java bug-collecting framework and dataset designed to mitigate data leakage concerns when evaluating LLM-based software engineering techniques. Our hope is to present a supplementary evaluation benchmark to the larger and established bug benchmarks, so to allow evaluation of the generality of LLM-based software engineering tools.

ACKNOWLEDGEMENT

This work was supported by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (RS-2023-00208998), and the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (2022-0-00995).

REFERENCES

[1] V. J. M. Manès, H. Han, C. Han, S. K. Cha, M. Egele, E. J. Schwartz, and M. Woo, "The art, science, and engineering of fuzzing: A survey," *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2312–2331, 2021.

[2] S. Kang, J. Yoon, and S. Yoo, "Large language models are few-shot testers: Exploring llm-based general bug reproduction," in *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering*, ser. ICSE 2023, 2023.

[3] M. Soltani, A. Panichella, and A. van Deursen, "Search-based crash reproduction and its impact on debugging," *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1294–1317, 2020.

[4] L. Gazzola, D. Micucci, and L. Mariani, "Automatic software repair: A survey," *IEEE Transactions on Software Engineering*, vol. 45, no. 1, pp. 34–67, 2019.

[5] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ser. ISSTA 2014. New York, NY, USA: Association for Computing Machinery, 2014, pp. 437–440.

[6] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, pp. 405–435, 2005.

[7] R. Widyasari, S. Q. Sim, C. Lok, H. Qi, J. Phan, Q. Tay, C. Tan, F. Wee, J. E. Tan, Y. Yieh, B. K. P. Goh, F. Thung, H. J. Kang, T. Hoang, D. Lo, and E. L. Ouh, "Bugsinpy: a database of existing bugs in python programs to enable controlled testing and debugging studies," *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:226274286

[8] P. Gyimesi, B. Vancsics, A. Stocco, D. Mazinanian, Á. Beszédes, R. Ferenc, and A. Mesbah, "Bugsjs: a benchmark of javascript bugs," *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pp. 90–101, 2019.

[9] N. Jiang, K. Liu, T. Lutellier, and L. Tan, "Impact of code language models on automated program repair," 2023.

[10] Z. Fan, X. Gao, M. Mirchev, A. Roychoudhury, and S. H. Tan, "Automated repair of programs from large language models," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, pp. 1469–1481.

[11] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, "Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models," in *2023 45th International Conference on Software Engineering (ICSE)*, 2023.

[12] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre, "Training compute-optimal large language models," *ArXiv*, vol. abs/2203.15556, 2022.

[13] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, and C. M. et al., "Starcoder: may the source be with you!" *ArXiv*, vol. abs/2305.06161, 2023.

[14] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[15] "Openai official documentation," https://platform.openai.com/docs/model-index-for-researchers/models-referred-to-as-gpt-3-5, accessed: 2023-09-10.