Breaking the Silence: the Threats of Using LLMs in Software Engineering

Sallou, J.; Durieux, T.; Panichella, A.

DOI
10.1145/3639476.3639764

Publication date
2024

Document Version
Final published version

Published in
Proceedings - 2024 ACM/IEEE 46th International Conference on Software Engineering

Breaking the Silence: the Threats of Using LLMs in Software Engineering

June Sallou
TU Delft
The Netherlands
J.Sallou@tudelft.nl

Thomas Durieux
TU Delft
The Netherlands
thomas@durieux.me

Annibale Panichella
TU Delft
The Netherlands
A.Panichella@tudelft.nl

Abstract

Large Language Models (LLMs) have gained considerable traction within the Software Engineering (SE) community, impacting various SE tasks from code completion to test generation, from program repair to code summarization. Despite their promise, researchers must still be careful as numerous intricate factors can influence the outcomes of experiments involving LLMs. This paper initiates an open discussion on potential threats to the validity of LLM-based research including issues such as closed-source models, possible data leakage between LLM training data and research evaluation, and the reproducibility of LLM-based findings. In response, this paper proposes a set of guidelines tailored for SE researchers and Language Model (LM) providers to mitigate these concerns. The implications of the guidelines are illustrated using existing good practices followed by LLM providers and a practical example for SE researchers in the context of test case generation.

CCS Concepts

• **Software and its engineering** → **Empirical software validation**; • **Computing methodologies** → *Machine learning*; • **General and reference** → *Evaluation.*

ACM Reference Format:
June Sallou, Thomas Durieux, and Annibale Panichella. 2024. Breaking the Silence: the Threats of Using LLMs in Software Engineering. In *New Ideas and Emerging Results (ICSE-NIER'24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3639476.3639764

1 Introduction

In recent years, the utilization of Large Language Models (LLMs) has gained substantial traction within the Software Engineering (SE) community. Equipped with language understanding and generation capabilities, these models have impacted various SE research and practice aspects. From code generation to bug detection and natural language interactions with codebases, LLMs have played a pivotal role in recent SE advancements [20, 26, 29, 34, 41].

Despite their promise, researchers must tread cautiously when making claims about the effectiveness of their approaches. The outcomes of experiments involving LLMs can be influenced by numerous intricate factors that can be challenging to discern or control. This intricacy underscores the need to thoroughly examine the validity of research findings when LLMs are involved.

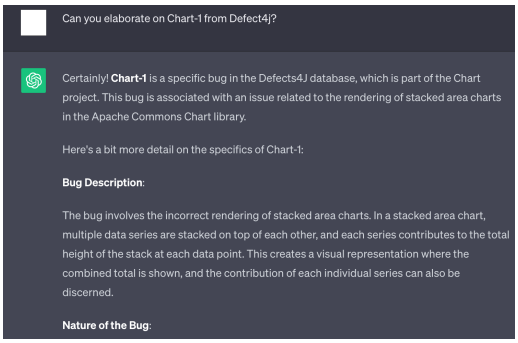
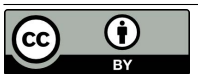


Figure 1: ChatGPT's detailed answers about a specific bug in Defects4J.

For instance, consider the evaluation of LLMs using well-known projects such as those included in Defects4J [24]. Although ChatGPT-3.5, as a general-purpose chatbot, is not fine-tuned for specific SE tasks, it possesses precise knowledge of the bugs within these projects (see Figure 1). Thus, when OpenAI models are employed for tasks like patch generation [42], fault localization [40], or test generation [35] for Defects4J, they have superior performance compared to an unknown code [34]. This foreknowledge arises from the pre-trained process, as ChatGPT has been trained on a large variety of datasets, including scientific papers (though the specifics are not fully disclosed). This raises severe concerns about the threats to both *construct* (*training and evaluating on the same dataset*) and *external* (*do the results hold for unknown projects/code?*) validity.

This paper aims to initiate a community-wide discussion and raise awareness of these issues to facilitate collective progress. Specifically, we focus on three key threats to validity: (1) Using *closed-source* LLMs and their implications w.r.t. data *privacy* and the models' *evolution* unpredictability. (2) The *blurry separation* between training, validation, and test sets and the corresponding potential explicit/implicit data leakage. (3) *Reproducibility* of the published research outcomes over time, due to the *non-stochastic* nature of LLMs answers, the non-transparent releases of *new model versions*, and the lack of complete *traceability*.

While recent papers acknowledge some of these concerns (e.g., [23, 34, 44]), we highlight the necessity for further empirical methodologies—such as code obfuscation, multiple independent prompts or queries, and metadata provision—to alleviate and address these concerns. Therefore, we present an initial set of guidelines aimed at mitigating these threats, specifically targeting SE researchers and Language Model (LM) providers. While we provide a list of actionable suggestions, we emphasize the importance of the wide



ICSE-NIER'24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0500-7/24/04.
https://doi.org/10.1145/3639476.3639764

SE community to follow our initial steps and build a more comprehensive list of threats to validity and research methods to address them, ultimately advancing the field while ensuring the reliability and validity of LLM-based research contributions.

To illustrate the practical application of our guidelines, we demonstrate their implementation in the context of test case generation —a well-established SE task [21, 30]— using ChatGPT 3.5 on two buggy code snippets from Defects4J [24]. We provide a full replication package [13] including (1) the implementation of our guidelines, and (2) the results of our analysis. Finally, we highlight existing good practices from LLMs providers that align with our guidelines.

2 Threats to Validity for LLMs

This section opens the discussion related to LLM-based research within the SE research community.

2.1 Closed-Source Models

A significant portion of the LLMs community heavily relies on closed-source models, as evidenced by the prevalence of ChatGPT-related papers at this year's ICSE conference. This dependence stems from factors such as the models' effectiveness, availability, and cost-effectiveness. To put this into perspective, deploying an open-source LLM model comparable to ChatGPT, like the Falcon 180B model, would incur a monthly cost of $29,495 on AWS.[1]

However, the utilization of closed-source models introduces significant threats to the validity of the research approach.

Model Evolution Unpredictability. One primary concern arises from the lack of control over the evolution of these closed-source models. New models may be released into production [33], and notable changes in the output of OpenAI models have been observed [14]. Such changes can occur during or after the research approach has been presented, potentially making the presented results obsolete. Moreover, this concern is also particularly pronounced for incremental works that use LLM. Indeed, distinguishing whether the improvements claimed in the new contribution are the result of changes to the LLMs' models or due to the novelty of the contribution becomes a complex task.

Privacy Implications. Another significant aspect of concern is privacy. Closed-source models often lack transparency, making it difficult to assess the privacy implications associated with their usage [5] as well as potential copyright infringements.

2.2 Implicit Data Leakage

LLMs are trained on vast textual datasets such as Wikipedia, GitHub, and StackOverflow [15, 36] and derive their understanding of semantics and contextual word relationships from this diverse data. These models contain millions (e.g., BERT) to billions (e.g., ChatGPT-4 and LLaMA) of parameters, which undergo a series of iterative optimizations during *pre-training* that minimize the loss function.

Data leakage due to pre-training. Pre-training in *unsupervised* or *semi-supervised* learning does not tailor models for specific software engineering tasks they will be later evaluated on after parameter re-tuning. Nonetheless, questions can be raised whether LLMs potentially memorize existing code samples used for evaluation, instead of generating new, unseen code [22]. For instance, prior studies [11, 23, 25, 32] highlighted vulnerabilities in code generated by Codex [15], originating from its training set. Siddiq et al. [34] investigated the performances of three LLMs (Codex, ChatGPT3.5 [28], and StarCoder [27]) in generating unit tests for Java programs. They reported remarkable performance discrepancies between the HumanEval [12] (>69% branch coverage) and the SF110 [21] (2% branch coverage) datasets. We remark that the former is available on GitHub [12] while the latter is not (available on `SourceForge` instead).

Additionally, previous studies in *metamorphic testing* [8, 17, 43, 45] have demonstrated how semantic-preserving changes to code snippets can effectively deceive LLMs. These approaches create new data points that differ from the original ones by at least one metamorphic change, increasing the likelihood that LLMs will not recognize the code snippets seen during pre-training.

Data leakage due to fine-tuning. LMs applied to specific SE tasks require parameter tuning via *supervised* learning, adjusting parameters using labeled datasets specific to the task. Despite efforts to separate training, validation, and test sets, ensuring clear distinctions is not guaranteed. In practice, different projects might have common dependencies and use the same pre-defined APIs. For instance, within the Java ecosystem, numerous libraries/tools often rely on common dependencies like Log4j, Apache Commons, Spring, GSon, and others. Consequently, a scenario of data leakage can arise if the LLM is trained on "project A" that employs a specific API, and the resulting model is subsequently used to fix the usage of the same API in another project within the test set.

2.3 Reproducibility

Several concerns can be raised w.r.t the reproducibility of the LLMs outputs. The ability to obtain identical results following the same procedure by external parties is proven to be challenging.

Output Variability. LLMs exhibit variability in their outputs, even when using identical input. Running the same prompt several times may not result in identical output, rendering the usage of LLMs non-deterministic. Examples of such phenomena have been described in the literature, including other application domains, such as the medical domain [19]. We also experimented with an example in software engineering involving a code generation task. In this use case, we demonstrate that running a prompt to generate Python code multiple times results in different responses from the GPT-3.5 model. To avoid data contamination, we get inspiration from the methodology outlined by Chen et al. [14]. We use a coding challenge from the LeetCode [2] platform as the prompt, employing the same prompt twice in two separate sessions[2] on the same day (September 12th, 2023). During these sessions (whose chat links are provided as footnotes), we observe distinct codes generated by GPT-3.5, with varying function operations reasoning, variable names, and initialization values or expressions.

Time-Based Output Drift. Furthermore, there is no assurance that the results will remain consistent over time. As discussed in Section 2.1, many LLMs are closed-source, and there are no established practices akin to regression testing to account for output variability. Running the same prompt at a later time (e.g., days or months) may lead to a drift in the outcome due to potential retraining between sessions, reinforcement learning between sessions, or

[1] Visited on 7-Dec-2023, 8x A100 80GB on https://aws.amazon.com/ec2/instance-types/

[2] first session: https://chat.openai.com/share/a0c7ef5c-74ce-466b-a1d5-5f44e03a626d, second session:https://chat.openai.com/share/6566acff-12eb-470a-a043-3e2294cf6406

adjustments based on user feedback. Chen et al. [14] explored this time-based output drift in terms of accuracy for two versions of GPT models over a three-month interval. They show that, over a range of tasks and application domains, the overall accuracy of outputs decreases, accompanied by a remarkable mismatch in answers. In particular, for code generation, the mismatch is evaluated at 20% for GPT-3.5 and 50% for GPT-4 between March 2023 and June 2023. Moreover, the number of executable outputs drops from 52% to 10% for GPT-3.5, and from 22% to 2% for GPT-4.0 for the same period.

Traceability. Another critical concern associated with the widespread adoption of LLMs is the lack of traceability. Currently, connecting the output of LLMs to specific prompts, along with 'configuration' details such as the version of the used LLM, the date of the query, and other specifications, can be a challenging task.

3 Discussion and Guidelines

In this section, we present initial guidelines and methodologies addressing the mentioned threats. While we provide a list of actionable suggestions as opening steps, we encourage the SE community to work toward establishing standards and expectations at the same level as those commonly applied with traditional AI techniques. We organize the guidelines according to the actors they involve: the LLM providers, and the SE researchers using LLMs.

3.1 Guidelines for LLM Providers

3.1.1 Closed-Source Models We foresee two main strategies to address this category of threats to validity:

Enhance model transparency. LLM providers should prioritize transparency by furnishing comprehensive information about their models. This should encompass details on the model's creation process and the methodology used for data selection during training. Furthermore, providers should share statistics and data-point information to shed light on the model's training dataset. Ideally, an API service could be established, enabling users to verify if a particular data source was included in the model's training or validation datasets. Such a service would not only enhance transparency but could also serve privacy and copyright verification purposes.

Use versioning information. Providers should provide their model version, and they should adopt a versioning nomenclature that distinguishes major revisions from minor updates. This enables users to discern the significance of model version changes.

3.1.2 Data Leakage In light of concerns regarding closed-source models, LLM providers should provide services that allow researchers to verify which projects and sources were considered during pre-training. A positive example is CodeBERT, whose provides do not disclose pre-training code but enable verification of included projects for pre-training through the train split data.[3]

3.1.3 Reproducibility We propose two methods to address this: **Use a fixed random seed.** LLM providers should ensure the inclusion of a settable random seed during the inference of LLMs, render the inference deterministic for each specific case. This practice would help address the variability of output concerning traceability and reproducibility. In the case of closed-source LLMs, a dedicated API could be made accessible, allowing the user to set the seed without requiring access to the entire model. Toward this direction, OpenAPI has recently released a *beta feature* [4] that allows users to set fixed seeds during prompting, although deterministic answer is not fully guaranteed due to different back-end settings.

Use an archiving system. In addition to the Versioning Information, we advocate for the usage of a general archiving system, to ensure that external parties can reproduce the observations made by LLMs. It should be noted that some efforts are already being made in that direction. We can cite the HuggingFace platform [1], which provides pre-trained models for download with information about the model training, file versioning, and datasets. Zenodo [3] is another example of storing and making versioned models and datasets accessible and reusable, which is commonly used in the research community. However, the use of such platforms is not yet a regular and consistent practice. Moreover, a dedicated LLM platform is still missing, as LLM sizes are generally large, posing challenges for downloading or uploading

3.2 Guidelines for SE Researchers

This section outlines guidelines for SE researchers. Along with presenting guidelines, we exhibit their practical applicability through a showcase example, using ChatGPT3.5 to generate JUnit test cases for two buggy programs in the Defect4j dataset [24], `Chart-11` and `Math-5`. The prompts with the collected answers, data analysis, and metadata are available in our replication package [13].

3.2.1 Reproducibility To address the threats to the reproducibility of LLMs-based approaches, we proposed the following guidelines:

Assess output variability. Due to output variability, running the LLMs' inference only once is insufficient to ensure reproducibility. Therefore, we argue in favor of conducting multiple replication runs and using variability metrics during the evaluation. For our showcase example, we queried ChatGPT3.5 ten times over different days using the same prompts (see our replication package) and targeting only the buggy methods (i.e., no the entire classes). We then analyzed the resulting branch coverage and test execution results. For `Math-5`, we report an average branch coverage of 70% for the tested Java method with a large variability (interquartile range or IQR) of 27.5%. We also observe variability in the number of generated tests (between 5 and 7) and number of failing tests (between 1 and 2). For `Chart-11`, the generated tests achieve 71% of branch coverage for the tested Java method, with 20% IQR. ChatGPT also generates between 1 and 5 failing tests for this method.

Provide execution metadata. Associated with the LLMs inference results, we argue that relevant additional data should be made accessible and considered during the evaluation of LLMs. Such information includes, but is not limited to: (i) *Model information:* To reproduce the LLMs' results and evaluation, information concerning the model is necessary (e.g., version, seed, etc.). Furthermore, the model itself should be accessible to enable its use, at least in a black box manner. (ii) *Prompts:* The exact inputs (queries) used for the inference and evaluation of the LLMs. (iii) *Date of LLMs query:* The date is relevant data to share to address the time-based output drift (that can happen because of retraining of models, or reinforcement learning from past human feedback and interactions). (iv) *Output variability metrics and associated assessment package:*

[3] https://huggingface.co/datasets/code_search_net

[4] https://platform.openai.com/docs/guides/text-generation/reproducible-outputs

Information concerning the assessment of output variability would enable the user to understand the risk regarding consistency in using the LLM in question. Providing the package containing the prompts and results used during this evaluation would help to ensure reproducibility. (v) *Scope of reproducibility*: To ensure the trusted and controlled usage of LLMs, information about the scope in which the model has been trained or assessed should be disclosed, including the application domains and studied use cases.

We provide an example of metadata (written using the JSON format) for the showcase of this paper in our replication material.

3.2.2 Data Leakage A few recommendations can be made to tackle the crucial concerns about the potential data leakage:

Assess LLMs on metamorphic data. *Metamorphic testing* is active research for the model robustness [8, 17, 45]. Metamorphic testing generates new data samples (code) by applying metamorphic transformations to the validation or test sets. These new snippets maintain semantic and behavioral equivalence with the original code, yet exhibit structural differences (e.g., distinct Abstract Syntax Trees). Prior studies have shown that `CodeBERT` and `code2vec` are not robust, i.e., they produce different (worse) results when obfuscating the variable names [17], introducing unused variables [45], replacing tokens with plausible alternatives [16], and wrapping-up expressions in identity-lambda functions [8, 9].

Therefore, we advise researchers to complement the analysis of the LLMs performance with new data samples generated with metamorphic testing. The selection of metamorphic transformations should align with the specific task at hand. For example, code obfuscation (for method/class names) should not hinder the ability of LLMs to generate unit tests or patches successfully. Instead, identifier names are crucial for NLP-related tasks (e.g., method name prediction), and other metamorphic transformations should be applied (e.g., wrapping up expressions in identity-lambda functions).

To show the practicability of this guideline, we have applied code metamorphic transformations to the Java methods of `Math-5` and `Chart-11`. In particular, we (1) removed the `javadoc` and (2) renamed the method under test and its input parameters. For renaming, we did not use randomly generated string but opted for synonyms and English words (e.g., changing the method name `reciprocal()` in `complementary()` for `Math-5`) to maintain the naturalness of the transformed code [43]. While we do not observe a significant difference in terms of branch coverage achieved for `Chart-11` (p-value=0.79 according to the Wilcoxon test) between the original and transformed code, we report a `small` negative effect size ($\hat{A}_{12}$=0.63) w.r.t. the number of failing tests (larger for the transformed code). The difference we obtained for `Math-5` on the obfuscated code is much more significant. While ChatGPT constantly generated tests for the original program with a branch coverage of 71%, it struggled to generate any meaningful tests on the transformed code. In particular, ChatGPT always created non-compiling tests with clear examples of *hallucination* [34, 46], i.e., invoking methods/constructors that were never included in the prompts.

Use different sources. As shown by Siddiq et al. [34], LLMs achieve much worse results on projects from SourceForge compared to GitHub. Hence, we recommend researchers gather software projects and data from multiple sources.

Code clone detection. Given the current low transparency of closed-source LLMs, tracing the projects used for pre-training is challenging, if not impossible. However, researchers can use well-established code clone detection techniques [4] to check if the generated code (e.g., test cases) is similar to code seen in online repositories (e.g., manually-written test cases).

Check for common dependencies. To prevent implicit data leakage between training, evaluation, and test sets (for task-specific evaluation), researchers should (1) cross-compare the external dependencies between projects belonging to different sets, (2) use code cloning techniques to assess whether similar code (e.g., API uses) appear between different projects from the different sets.

3.2.3 Closed-Source Models **Perform comparative analysis.** Researchers are encouraged to run experiments using both open-source and closed-source LLMs. This comparative approach can provide additional insights into the strengths and limitations of each. Notably, the open-source LLM community has witnessed an expansion in availability, with models like llama2 [37] and Falcon 180B [7] emerging as viable options. llama2 models, in particular, offer the advantage of running on consumer-grade devices.

Framework Facilitation. To streamline the evaluation process across multiple models, researchers can leverage frameworks such as ONNX [5]. ONNX simplifies the transition between various models, enhancing the efficiency and consistency of experimentation.

4 Conclusion and Future Work

In this article, we have initiated a discussion about the usage of LLMs in SE contributions, along with the challenges and threats to validity they bring. We have identified three primary challenges: the reliance on closed-source LLMs, potential data leakages, and concerns about reproducibility. To mitigate these, we propose an initial set of guidelines. We aim to encourage an ongoing dialogue within the community to navigate these challenges effectively.

It is essential to continue reflecting and staying critical about the usage of LLMs in SE. We must collectively define guidelines and expectations within our community. We believe the conversation should be spread by organizing panels with different experts and stakeholders. We should also monitor the evolution of good practices in the literature and maintain community guidelines.

We emphasize the importance of disseminating evaluation expectations from the SE to the ML community, fostering mutual understanding of evolving practices. It's noteworthy that metrics drawn from Natural Language Processing (NLP) studies (e.g., counting the number of compiling unit tests generated by LLMs as done in [6, 38]) do not adequately reflect the well-established performance metrics for SE tasks that do not have an NLP focus (e.g., see the existing standards for assessing unit test generation tools [10, 18, 31]).

We strongly believe that breaking the silence as a community will enhance the validity and reliability of our LLM-based contributions and the SE community in general. The goal is to ultimately advance the field while ensuring high research quality and high methodological standards (considering different aspects, including data privacy [5], carbon footprint [39], etc.).

[5] https://onnx.ai/

References

[1] 2023. Hugging Face – The AI community building the future. https://huggingface.co [Online; accessed 11. Sept. 2023].

[2] 2023. LeetCode - The World's Leading Online Programming Learning Platform. https://leetcode.com [Online; accessed 12. Sept. 2023].

[3] 2023. Zenodo. https://zenodo.org [Online; accessed 11. Sept. 2023].

[4] Qurat Ul Ain, Wasi Haider Butt, Muhammad Waseem Anwar, Farooque Azam, and Bilal Maqbool. 2019. A systematic review on code clone detection. *IEEE access* 7 (2019), 86121–86144.

[5] Ali Al-Kaswan and Maliheh Izadi. 2023. The (ab)use of Open Source Code to Train Large Language Models. In *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*.

[6] Saranya Alagarsamy, Chakkrit Tantithamthavorn, and Aldeida Aleti. 2023. A3Test: Assertion-Augmented Automated Test Case Generation. *arXiv preprint arXiv:2302.10352* (2023).

[7] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Maitha Alhammadi, Mazzotta Daniele, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. The Falcon Series of Language Models: Towards Open Frontier Models. (2023).

[8] Leonhard Applis, Annibale Panichella, and Ruben Marang. 2023. Searching for Quality: Genetic Algorithms and Metamorphic Testing for Software Engineering ML. In *Proc. of the Genetic and Evolutionary Computation Conference*. 1490–1498.

[9] Leonhard Applis, Annibale Panichella, and Arie van Deursen. 2021. Assessing Robustness of ML-Based Program Analysis Tools using Metamorphic Program Transformations. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1377–1381.

[10] Andrea Arcuri and Lionel Briand. 2014. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability* 24, 3 (2014), 219–250.

[11] Owura Asare, Meiyappan Nagappan, and N Asokan. 2022. Is github's copilot as bad as humans at introducing vulnerabilities in code? *arXiv preprint arXiv:2204.04741* (2022).

[12] Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, et al. 2022. Multi-lingual evaluation of code generation models. *arXiv preprint arXiv:2210.14868* (2022).

[13] Authors. 2023. https://github.com/LLM4SE/obfuscated-ChatGPT-experiments

[14] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. How is ChatGPT's behavior changing over time? *arXiv preprint arXiv:2307.09009* (July 2023). arXiv:2307.09009

[15] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[16] Jürgen Cito, Isil Dillig, Vijayaraghavan Murali, and Satish Chandra. 2022. Counterfactual explanations for models of code. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. 125–134.

[17] Rhys Compton, Eibe Frank, Panos Patros, and Abigail Koay. 2020. Embedding Java Classes with code2vec: Improvements from Variable Obfuscation. In *2020 IEEE/ACM 17th International Conference on Mining Software Repositories (MSR)*. IEEE, New York, NY, USA, 243–253. https://doi.org/10.1145/3379597.3387445

[18] Xavier Devroey, Alessio Gambi, Juan Pablo Galeotti, René Just, Fitsum Kifetew, Annibale Panichella, and Sebastiano Panichella. 2023. JUGE: An infrastructure for benchmarking Java unit test generators. *Software Testing, Verification and Reliability* 33, 3 (2023), e1838.

[19] Richard H. Epstein and Franklin Dexter. 2023. Variability in Large Language Models' Responses to Medical Licensing and Certification Examinations. Comment on "How Does ChatGPT Perform on the United States Medical Licensing Examination? The Implications of Large Language Models for Medical Education and Knowledge Assessment". *JMIR Medical Education* 9, 1 (July 2023), e48305. https://doi.org/10.2196/48305

[20] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated Repair of Programs from Large Language Models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1469–1481. https://doi.org/10.1109/ICSE48619.2023.00128

[21] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: Automatic Test Suite Generation for Object-Oriented Software. In *Proceedings of the 19th ACM SIGSOFT ESEC/FSE* (Szeged, Hungary) *(ESEC/FSE '11)*. ACM, New York, NY, USA, 416–419. https://doi.org/10.1145/2025113.2025179

[22] Huseyin Atahan Inan, Osman Ramadan, Lukas Wutschitz, Daniel Jones, Victor Rühle, James Withers, and Robert Sim. 2021. Training Data Leakage Analysis in Language Models. (February 2021). https://www.microsoft.com/en-us/research/publication/training-data-leakage-analysis-in-language-models/

[23] Kevin Jesse, Toufique Ahmed, Premkumar T Devanbu, and Emily Morgan. 2023. Large Language Models and Simple, Stupid Bugs. *arXiv preprint arXiv:2303.11455* (2023).

[24] René Just, Darioush Jalali, and Michael D. Ernst. 2014. 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* (San Jose, CA, USA) *(ISSTA 2014)*. ACM, NY, USA, 437–440.

[25] Anjan Karmakar, Julian Aron Prenner, Marco D'Ambros, and Romain Robbes. 2022. Codex Hacks HackerRank: Memorization Issues and a Framework for Code Synthesis Evaluation. *arXiv* (Dec. 2022). https://doi.org/10.48550/arXiv.2212.02684 arXiv:2212.02684

[26] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K. Lahiri, and Siddhartha Sen. 2023. CodaMosa: Escaping Coverage Plateaus in Test Generation with Pre-Trained Large Language Models. In *Proc. of the 45th International Conference on Software Engineering (ICSE '23)*. IEEE Press, Melbourne, Victoria, Australia, 919–931. https://doi.org/10.1109/ICSE48619.2023.00085

[27] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474* (2022).

[28] OpenAI. 2023. *OpenAI.* https://openai.com/ Accessed on September 14th, 2023.

[29] Ipek Ozkaya. 2023. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software* 40, 3 (April 2023), 4–8. https://doi.org/10.1109/MS.2023.3248401

[30] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2015. Reformulating branch coverage as a many-objective optimization problem. In *2015 IEEE 8th international conference on software testing, verification and validation (ICST)*. IEEE, 1–10.

[31] Sebastiano Panichella, Alessio Gambi, Fiorella Zampetti, and Vincenzo Riccio. 2021. Sbst tool competition 2021. In *2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 20–27.

[32] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt. 2023. Examining Zero-Shot Vulnerability Repair with Large Language Models. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, Los Alamitos, CA, USA, 2339–2356. https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.10179420

[33] Luiza Pozzobon, Beyza Ermis, Patrick Lewis, and Sara Hooker. 2023. On the Challenges of Using Black-Box APIs for Toxicity Evaluation in Research. arXiv:2304.12397 [cs.CL]

[34] Mohammed Latif Siddiq, Joanna Santos, Ridwanul Hasan Tanvir, Noshin Ulfat, Fahmid Al Rifat, and Vinicius Carvalho Lopes. 2023. Exploring the Effectiveness of Large Language Models in Generating Unit Tests. *arXiv preprint arXiv:2305.00418* (2023).

[35] Yutian Tang, Zhijie Liu, Zhichao Zhou, and Xiapu Luo. 2023. ChatGPT vs SBST: A Comparative Assessment of Unit Test Suite Generation. *arXiv preprint arXiv:2307.00588* (2023).

[36] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[37] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[38] Michele Tufano, Dawn Drain, Alexey Svyatkovskiy, Shao Kun Deng, and Neel Sundaresan. 2020. Unit test case generation with transformers and focal context. *arXiv preprint arXiv:2009.05617* (2020).

[39] Roberto Verdecchia, June Sallou, and Luís Cruz. 2023. A systematic review of Green AI. *WIREs Data Mining and Knowledge Discovery* 13, 4 (2023), e1507. https://doi.org/10.1002/widm.1507

[40] Yonghao Wu, Zheng Li, Jie M Zhang, Mike Papadakis, Mark Harman, and Yong Liu. 2023. Large Language Models in Fault Localisation. *arXiv preprint arXiv:2308.15276* (2023).

[41] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE 2023). Association for Computing Machinery.*

[42] Chunqiu Steven Xia and Lingming Zhang. 2023. Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT. *arXiv preprint arXiv:2304.00385* (2023).

[43] Zhou Yang, Jieke Shi, Junda He, and David Lo. 2022. Natural attack for pre-trained models of code. In *Proceedings of the 44th International Conference on Software Engineering*. 1482–1493.

[44] Wentao Ye, Mingfeng Ou, Tianyi Li, Xuetao Ma, Yifan Yanggong, Sai Wu, Jie Fu, Gang Chen, Junbo Zhao, et al. 2023. Assessing Hidden Risks of LLMs: An Empirical Study on Robustness, Consistency, and Credibility. *arXiv preprint arXiv:2305.10235* (2023).

[45] Noam Yefet, Uri Alon, and Eran Yahav. 2020. Adversarial examples for models of code. *Proc. of the ACM on Programming Languages* 4, OOPSLA (2020), 1–30.

[46] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. *arXiv preprint arXiv:2309.01219* (2023).