Beyond Accuracy and Robustness Metrics for Large Language Models for Code

Daniel Rodriguez-Cardenas
dhrodriguezcar@wm.edu
William and Mary
Williamsburg, Virginia, USA

ABSTRACT

In recent years, Large Language Models for code (LLMc) have transformed the landscape of software engineering (SE), demonstrating significant efficacy in tasks such as code completion, summarization, review, tracing, translation, test case generation, clone detection, and bug fixing. Notably, GitHub Copilot [31] and Google's CodeBot [21] exemplify how LLMc contributes to substantial time and effort savings in software development. However, despite their widespread use, there is a growing need to thoroughly assess LLMc, as current evaluation processes heavily rely on accuracy and robustness metrics, lacking consensus on additional influential factors in code generation. This gap hinders a holistic understanding of LLMc performance, impacting interpretability, efficiency, bias, fairness, and robustness. The challenges in benchmarking and data maintenance compound this issue, underscoring the necessity for a comprehensive evaluation approach. To address these issues, this dissertation proposes the development of a benchmarking infrastructure, named *HolBench*, aimed at overcoming gaps in evaluating LLMc quality. The goal is to standardize testing scenarios, facilitate meaningful comparisons across LLMc, and provide multi-metric measurements beyond a sole focus on accuracy. This approach aims to decrease the costs associated with advancing LLMc research, enhancing their reliability for adoption in academia and industry.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

deep learning, code generation, interpretability, transformers

ACM Reference Format:
Daniel Rodriguez-Cardenas. 2024. Beyond Accuracy and Robustness Metrics for Large Language Models for Code . In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3639478.3639792

1 PROBLEM AND RESEARCH STATEMENT

In recent years, Large Language Models for code (LLMc) have begun to significantly change the landscape of software engineering (SE). Due to their impressive performance, LLMc have been widely employed to auto-complete [1, 3, 6, 8, 34], summarize [10, 16], review [27, 28], trace [15], translate code [17], generate test cases [19, 30, 35], detect cone clones [23, 33], and fix bugs [5, 7, 22, 24–26, 32, 38]. In particular, they have been recently deployed in large-scale solutions to provide code generative services. For instance, GitHub Copilot [31] aids developers by generating code, including entire functions, leading to substantial time and effort savings. Similarly, Google's CodeBot [21] has been applied in diverse projects, including the Android operating system. In essence, these AI tools, like LLMc, significantly enhance the effectiveness and efficiency of software development.

Therefore, there is an increasing interest in thoroughly assessing these LLMc [4, 12, 13, 36] to establish standardized criteria for evaluating the quality of generated code. However, the current evaluation procedure heavily depends on accuracy metrics [2, 14, 37] and robustness metrics [29], without a unanimous agreement on which additional features or properties influence the code generation process. In other words, we are currently not able to holistically evaluate some of the factors that influence the quality of LLMc across different scenarios [11]. Hence, the problem remains that, when attempting to understand the prediction performance of LLMc, no benchmarks are available to articulate a broad set of desiderata beyond accuracy leading to concerning issues related to interpretability, efficiency, bias, fairness, and robustness. The current state of evaluating LLMc is not unexpected, given the inherent challenges in benchmarking, collecting and cleaning data, and maintaining the necessary tools, metrics, and datasets. These challenges are crucial for supporting the swift progress of research in the field of LLMc.

This research aims to develop a benchmarking infrastructure to overcome challenges for conducting high-impact research related to evaluating the quality of Large Language Models for Code. We will conduct a range of research activities to better understand the current barriers to holistically benchmark LLMc research and the ways in which a future community infrastructure can be used to help researchers address existing key challenges. We envision creating an infrastructure for a **Hol**istic **Bench**mark Evaluation for LLMc (*HolBench*) to standardize testing scenarios to meaningfully compare different LLMc and provide multi-metric measurement beyond a single accuracy metric view. We posit that by holistically benchmarking LLMc, the cost of advancing research topics related to LLMc will be substantially decreased, making models more reliable to be adopted by academia and industry.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0502-1/24/04.
https://doi.org/10.1145/3639478.3639792

2 EXPECTED CONTRIBUTIONS

HolBench is aimed at closing **three key open gaps** that researchers and practitioners face while working on evaluating LLMc:

(1) G_1: **Collecting and benchmarking SE-based data to keep pace with new and enhanced LLMc.** While initial efforts have been made to benchmark the evaluation of LLMc [9, 20], nearly all publicly available datasets and benchmarks are not tailored for automatically curating testbeds. This issue remains since testbed curation requires a platform to centralize and standardize datasets to keep pace with ever-growing/improving LLMc. To address G_1, we suggest the implementation of an automated and ongoing pipeline for curating testbeds, constituting the initial element of our proposed infrastructure. This component has the ability to generate organized and validated artifacts and testbeds, meeting the requirements for comprehensive evaluations.

(2) G_2: **Encompassing the Extensive Range of LLMc capabilities.** Existing benchmarks and datasets have not comprehensively addressed software scenarios (*i.e.,* anticipated use cases for LLMc) essential for ensuring a baseline of quality assurance for the models' potential capabilities. Acknowledging the impracticality of testing LLMc across all conceivable combinations of software properties (*e.g.,* tasks, programming languages, granularity of artifacts), we introduce a second component of our infrastructure. This component aims to enhance the coverage of LLMc capabilities by delineating critical software scenarios. Effective and valuable LLMc should manifest a diverse range of capabilities that developers and researchers anticipate from these models.

(3) G_3: **Holistically evaluating LLMc with more metrics beyond accuracy.** The current evaluation practices for LLMc have predominantly emphasized metrics centered on accuracy, such as F1, Recall, AUC, and BLEU [9]. Utilizing existing benchmarks typically results in the reporting of a percentage score or a distance metric that only partially assesses the performance of an LLMc. Consequently, we advocate for an extension of the evaluation framework to embrace a *multi-metric* approach, aligning with a spectrum of societal considerations and requirements identified through preliminary interviews and surveys with LLMc experts. The third component of our infrastructure will initially encompass multi-metric measurements, including interpretability, efficiency, bias, fairness, and robustness.

Our proposed solution aims to address specific gaps by offering practical and actionable measures to enhance interpretability in a way that is accessible to both practitioners and researchers. We intend to achieve this by providing detailed metrics for evaluating LLMc and generating automatic datasets that ensure a fair and comprehensive assessment of emerging models.

3 PROPOSED PLAN AND EVALUATION

We will develop a novel benchmarking that can holistically evaluate LLMc using a multi-metric approach under different software scenarios. The proposed infrastructure is comprised of three main components. The first component is a pipeline to structure and collect required testbeds to holistically evaluate LLMc. The second component is a combination of software properties that ensembles scenarios required to test a LLMc. The third component is a multi-metric approach that consolidates the holistic evaluation.

(1) **Curation Pipeline:** The initial component of *HolBench* constitutes a benchmarking strategy involving a software architecture solution for the curation process. This component plays a role in addressing the primary problem G_1. In the initial phase, we will filter the most popular repositories on GitHub. We aim to capture the most recent commit changes in order to avoid already seen examples for the most recent models. For example our prototype is based on the latest report from ChatGPT [18], we assume that ChatGPT and other analyzed LLMc were not trained on commits between Jan 2, 2022, and Jan 1, 2023. This assumption ensures that our pipeline effectively prevents *data snooping*, which involves the inappropriate use of data points to assess statistical hypotheses using training samples. Subsequently, we will compile a set of novel methods for each commit. The collection of relevant data points will also include their corresponding documentation, excluding inline comments. The entire data collection process is automated, requiring only a fine-tuning of the query to select pertinent samples.

(2) **Software Scenarios:** The second component of *HolBench* is a strategy for coverage capabilities that integrates various attributes of code datasets to generate software scenarios. This component contributes to addressing the second problem G_2. Since the focus of the evaluation is the LLMc itself and not a scenario-specific configuration, it is essential to manage the generation of software scenarios to standardize and impartially compare different models. Ideally, each LLMc should undergo evaluation using identical software scenarios. Unlike previous benchmarks, which are primarily collections of datasets with a single metric (*i.e.,* accuracy), *HolBench* takes a top-down approach. In this approach, we will explicitly define the required software properties for evaluation based on real use cases. To articulate a scenario, *HolBench* dissects it into the following properties: task, programming language (PL), input and output granularity, and type of input/output. These properties can be expanded based on new research strategies and LLMc characteristics. Ultimately, the software scenarios serve as inputs for capability testing, allowing researchers and practitioners to evaluate LLMc under controlled settings.

(3) **Multi-Metric Approach:** The third facet of *HolBench* is designed to assess LLMc based on a comprehensive set of requirements for each software scenario outlined earlier. This component will contribute to addressing the third problem G_3. These requirements encompass a set of evaluation metrics that go beyond interpretability, including efficiency, bias, fairness, and robustness. The categories of metrics that we define will cover various academic demands and societal considerations. While these metrics are quantitative, it is important to note that some of them are challenging to measure due to their recent emergence or limited exploration in the software engineering field (*e.g.,* fairness, bias, or efficiency).

Unlike the prevailing evaluation system, where benchmarks typically measure a specific metric without considering *how well* a model performs across other domains independent of the specifics of each scenario, *HolBench* takes a holistic approach. In *HolBench*, practitioners can assess multiple metrics per scenario, considering a selected subspace of software scenarios X set of metrics. A comprehensive evaluation should capture the diverse subspaces represented by these combinations.

REFERENCES

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, et al. 2021. Program Synthesis with Large Language Models. arXiv:cs.PL/2108.07732

[2] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, et al. 2022. MultiPL-E: A Scalable and Extensible Approach to Benchmarking Neural Code Generation. http://arxiv.org/abs/2208.08227 arXiv:2208.08227 [cs].

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, et al. 2021. Generation Probabilities are Not Enough: Improving Error Highlighting for AI Code Suggestions. (2021). https://doi.org/10.48550/ARXIV.2107.03374 Publisher: arXiv Version Number: 2.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, et al. 2021. Evaluating Large Language Models Trained on Code. http://arxiv.org/abs/2107.03374 arXiv:2107.03374 [cs].

[5] Zimin Chen, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, et al. 2021. SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair. *IEEE Transactions on Software Engineering* 47, 9 (2021), 1943–1959. https://doi.org/10.1109/TSE.2019.2940179

[6] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaolo, Emad Aghajani, et al. 2022. An Empirical Study on the Usage of Transformer Models for Code Completion. *IEEE Transactions on Software Engineering* 48, 12 (2022), 4818–4837. https://doi.org/10.1109/TSE.2021.3128234

[7] A. Connor, A. Harris, N. Cooper, and D. Poshyvanyk. 2022. Can We Automatically Fix Bugs by Learning Edit Operations?. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE Computer Society, Los Alamitos, CA, USA, 782–792. https://doi.org/10.1109/SANER53432.2022.00096

[8] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, et al. 2021. Measuring Coding Challenge Competence With APPS. *CoRR* abs/2105.09938 (2021). arXiv:2105.09938 https://arxiv.org/abs/2105.09938

[9] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, et al. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. http://arxiv.org/abs/2308.10620 arXiv:2308.10620 [cs].

[10] Alexander LeClair, Aakash Bansal, and Collin McMillan. 2021. Ensemble Models for Neural Source Code Summarization of Subroutines. http://arxiv.org/abs/2107.11423 arXiv:2107.11423 [cs].

[11] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, et al. 2022. Holistic Evaluation of Language Models. http://arxiv.org/abs/2211.09110 arXiv:2211.09110 [cs].

[12] Chao Liu, Xuanlin Bao, Hongyu Zhang, Neng Zhang, Haibo Hu, et al. 2023. Improving ChatGPT Prompt for Code Generation. arXiv:cs.SE/2305.08360

[13] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. arXiv:cs.SE/2305.01210

[14] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, et al. [n. d.]. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. arXiv:2102.04664 [cs] http://arxiv.org/abs/2102.04664

[15] Kevin Moran, David N. Palacio, Carlos Bernal-Cárdenas, Daniel McCrystal, Denys Poshyvanyk, et al. 2020. Improving the Effectiveness of Traceability Link Recovery using Hierarchical Bayesian Networks. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. 873–885.

[16] Kevin Moran, Ali Yachnes, George Purnell, Junayed Mahmud, Michele Tufano, et al. 2022. An Empirical Investigation into the Use of Image Captioning for Automated Software Documentation. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 514–525. https://doi.org/10.1109/SANER53432.2022.00069

[17] Anh Tuan Nguyen and Tien N. Nguyen. 2015. Graph-Based Statistical Language Model for Code. In *ICSE'15*. IEEE Press, 858–868.

[18] OpenAI. 2023. GPT-4 Technical Report. arXiv:cs.CL/2303.08774

[19] Veselin Raychev, Martin T. Vechev, and Eran Yahav. 2014. Code completion with statistical language models. *PLDI* (2014).

[20] Daniel Rodriguez-Cardenas, David N. Palacio, Dipin Khati, Henry Burke, and Denys Poshyvanyk. 2023. Benchmarking Causal Study to Interpret Large Language Models for Source Code. http://arxiv.org/abs/2308.12415 arXiv:2308.12415 [cs].

[21] Doug Rosenberg, Barry Boehm, Matt Stephens, Charles Suscheck, Shobha Rani Dhalipathi, et al. 2020. CodeBots: From Domain Model to Executable Architecture. *Parallel Agile–faster delivery, fewer defects, lower cost* (2020), 27–51.

[22] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On Learning Meaningful Code Changes Via Neural Machine Translation. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 25–36. https://doi.org/10.1109/ICSE.2019.00021

[23] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, et al. 2018. Deep Learning Similarities from Different Representations of Source Code. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. 542–553.

[24] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano di Penta, Martin White, et al. 2018. An Empirical Investigation into Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 832–837. https://doi.org/10.1145/3238147.3240732

[25] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, et al. 2019. Learning How to Mutate Source Code from Bug-Fixes. *ICSME 2019* (2019), 301–312. https://doi.org/10.1109/ICSME.2019.00046

[26] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, et al. 2019. An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. *ACM Transactions on Software Engineering and Methodology* 28, 4 (2019), 1–29. https://doi.org/10.1145/3340544

[27] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, et al. 2022. Using Pre-Trained Models to Boost Code Review Automation. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 2291–2302. https://doi.org/10.1145/3510003.3510621

[28] Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards Automating Code Review Activities. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 163–174. https://doi.org/10.1109/ICSE43902.2021.00027

[29] Shiqi Wang, Zheng Li, Haifeng Qian, Chenghao Yang, Zijian Wang, et al. 2022. ReCode: Robustness Evaluation of Code Generation Models. http://arxiv.org/abs/2212.10264 arXiv:2212.10264 [cs].

[30] Cody Watson, Michele Tufano, Kevin Moran, Gabriele Bavota, and Denys Poshyvanyk. 2020. On Learning Meaningful Assert Statements for Unit Test Cases. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. 1398–1409.

[31] Michel Wermelinger. 2023. Using GitHub Copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 172–178.

[32] Martin White, Michele Tufano, Matías Martínez, Martin Monperrus, and Denys Poshyvanyk. 2019. Sorting and Transforming Program Repair Ingredients via Deep Learning Code Similarities. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 479–490. https://doi.org/10.1109/SANER.2019.8668043

[33] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 87–98.

[34] Martin White, Christopher Vendome, Mario Linares-Vasquez, and Denys Poshyvanyk. 2015. Toward Deep Learning Software Repositories. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 334–345. https://doi.org/10.1109/MSR.2015.38

[35] Robert White and Jens Krinke. 2020. ReAssert: Deep Learning for Assert Generation. http://arxiv.org/abs/2011.09784 arXiv:2011.09784 [cs].

[36] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent J. Hellendoorn. 2022. A Systematic Evaluation of Large Language Models of Code. http://arxiv.org/abs/2202.13169 arXiv:2202.13169 [cs].

[37] Wojciech Zaremba, Greg Brockman, and OpenAI. 2021. OpenAI Codex. https://openai.com/blog/openai-codex/.

[38] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. [n. d.]. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. ([n. d.]).