Journal of Data Efficiency
Volume 1 · Number 1 · 2026

Three Tests, Often One Input Shape: A Static Audit of MBPP Examples

Jack Barker[1] · Nicholas Warren[2] · James Harrison[3]

[1] United States · [2] France · [3] Australia

Original publication: October 24, 2026 | Article 655

Abstract

Three examples can vary their values while repeating the same kind of input. We parsed the ordinary tests for all 974 MBPP tasks and classified calls by a declared coarse argument-shape grammar. Every task had three ordinary calls, but 817 tasks used only one signature; 157 used at least two. Reference-code control flow accompanied a 14.55 percentage-point higher probability of multiple signatures after syntax-size, instruction-length, and setup adjustment. This describes the supplied examples under one static grammar. It does not measure branch coverage, semantic test adequacy, or model pass rates, and identical shapes can still contain behaviorally important value differences.

Exploratory benchmark-metadata draft. Downloaded programs were parsed without execution; human scientific review is pending.

Three Tests, Often One Input Shape: A Static Audit of MBPP Examples

Exploratory benchmark-metadata draft. Downloaded programs were parsed without execution; human scientific review is pending.

Abstract

Three examples can vary their values while repeating the same kind of input. We parsed the ordinary tests for all 974 MBPP tasks and classified calls by a declared coarse argument-shape grammar. Every task had three ordinary calls, but 817 tasks used only one signature; 157 used at least two. Reference-code control flow accompanied a 14.55 percentage-point higher probability of multiple signatures after syntax-size, instruction-length, and setup adjustment. This describes the supplied examples under one static grammar. It does not measure branch coverage, semantic test adequacy, or model pass rates, and identical shapes can still contain behaviorally important value differences.

1. A fixed test count can hide repeated structure

Programming benchmarks often provide a small number of visible examples. Counting those examples is straightforward, but a fixed count does not reveal how much their inputs differ. Three tests may pass three positive integers of different magnitude, or they may include a negative value, zero, and a positive value. They can also repeat the same collection structure while changing every contained value. We study this limited form of input diversity without claiming that it determines whether the tests adequately specify the task.

The Mostly Basic Programming Problems dataset contains 974 tasks intended for short Python program synthesis (Austin et al., 2021). Its public records include a natural-language instruction, reference code, ordinary tests, and additional fields. This combination allows a reproducible static audit of how supplied examples vary. Our research question asks how often the ordinary calls differ under a declared coarse grammar and whether reference-code control flow accompanies that diversity after a small set of descriptive adjustments.

Pairwise-preference training for coding assistants in Tao and Kamata, 2026 motivates making the evidence for a proposed change inspectable. A preference for one code edit does not remove the need for useful tests. Our analysis provides no preference labels and evaluates no assistant. It examines one property of the available benchmark examples that could matter when researchers later interpret code correctness or design feedback, while leaving the causal effect of more diverse examples unmeasured.

2. Test structure is not program correctness

HumanEval-style generation in Chen et al., 2021, rigorous expanded evaluation in Liu et al., 2023, and the evolving tasks of Jain et al., 2024 illustrate different ways to assess generated programs. Their evaluation questions extend beyond the visible shape of a few calls. CodeRL (Le et al., 2022) uses execution-related learning signals, whereas our workflow deliberately executes neither the downloaded reference programs nor their tests. A syntactic audit can therefore be reproduced safely while remaining narrower than a functional evaluation.

Code models such as Li et al., 2023, Guo et al., 2024, Fried et al., 2022, and Guo et al., 2020 differ in their training and representations. We do not infer their weaknesses from a count of input signatures. An assistant could fail on several values with the same shape or succeed despite limited visible diversity.

Direct Preference Optimization (Rafailov et al., 2023) and low-rank adaptation (Hu et al., 2022) likewise describe learning procedures, not guarantees that a benchmark's ordinary examples cover every relevant behavior.

Empirical-software guidance in Kitchenham et al., 2002 supports an explicit operational definition. Repository-mining warnings in Bird et al., 2009, label concerns in Herzig et al., 2013, and code-review observations in Bacchelli and Bird, 2013 all caution against converting a convenient proxy into a complete assessment of engineering quality. We therefore name the measured construct coarse argument-shape diversity. Terms such as test coverage, specification completeness, or semantic breadth are reserved for analyses that actually observe those properties.

3. The signature grammar

We parse reference code and each ordinary test with Python's abstract syntax tree parser. Functions defined at the reference module's top level supply a set of target function names. Within ordinary test syntax, we collect calls whose called name belongs to that set, including calls nested inside another expression. We retain their argument list and keyword names. The procedure is deterministic and does not evaluate any expression or resolve runtime objects.

Literal booleans are identified before numbers. Numeric literals are divided into negative, zero, and positive classes; a directly negated numeric literal is handled accordingly. Strings are separated into empty and nonempty. Lists, tuples, and sets retain their container type and the set of recursively observed child-shape categories. Dictionaries retain child-shape sets for keys and values. Other expressions are flagged by abstract-syntax class rather than interpreted as if their runtime value were known.

This grammar intentionally discards most values and container cardinalities. A list of two positive numbers and a list of five positive numbers therefore share a signature, while an empty list differs from either. Argument order is preserved; keyword names are included in canonical sorted form. The outer number of arguments remains visible. These choices define the measured construct, and a different grammar could produce different diversity counts without either result being a direct measure of test adequacy.

The task outcome equals one when at least two distinct signatures occur among ordinary reference calls. We also preserve the number of distinct signatures itself. Challenge tests are excluded from this calculation and are not silently appended to increase the visible budget. This distinction keeps the question tied to the ordinary examples a benchmark user might first inspect. It does not imply that the excluded tests are unimportant or redundant.

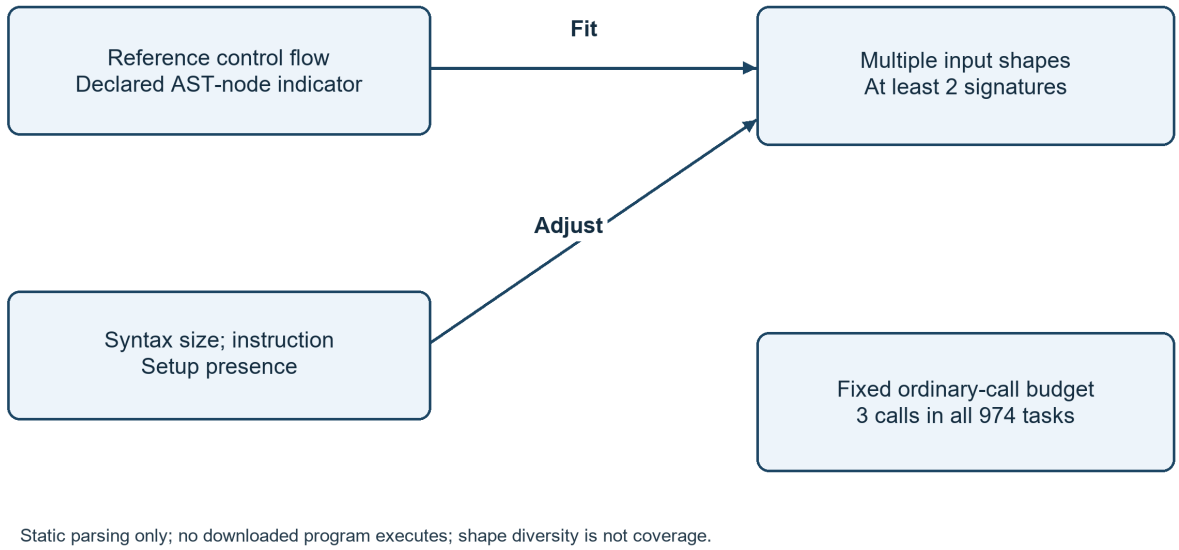


Figure 1. A fixed three-call budget yields a signature-diversity indicator; reference control flow, syntax size, instruction length, and setup presence describe variation in that indicator.

4. Features and sample integrity

All 974 records parse successfully for the required task and test components, and all contain identifiable calls to the reference functions. Every task has exactly three ordinary tests and three collected reference calls. No observation is excluded. Because call count is constant, it is not included as a regression covariate. Treating it as varying information would create a rank-deficient specification and an uninterpretable coefficient alongside the intercept.

Reference control flow is indicated by the presence of an if statement, conditional expression, loop, try statement, match statement, or comprehension node. This includes comprehensions because their iteration structure can be behaviorally relevant even when the source code is short. We separately count all abstract syntax nodes and transform that count logarithmically. Instruction length is log one plus whitespace-delimited word count, and setup presence indicates nonempty test-setup text.

There are 702 tasks with the defined control-flow indicator. The feature does not establish how many branches execute or whether any test reaches them. A program can invoke control flow through a library without displaying these nodes, and a visible branch can be irrelevant to the ordinary examples. Measurement-error cautions in Carroll et al., 2006 therefore apply to both the explanatory feature and the diversity outcome. They are structural proxies with declared limits.

Missing-data principles in Little and Rubin, 2002 motivate keeping exclusion rules visible even when the retained count equals the released count. This full retention does not establish that MBPP is representative of all beginner programming problems or modern coding-assistant requests. The benchmark was assembled for a particular synthesis purpose. Its public task census provides exact descriptive counts under our grammar, while population generalization requires an additional sampling argument.

5. Descriptive regression

We fit an ordinary linear probability model of multiple signatures on control flow, log syntax-node count, log instruction length, and setup presence. HC3 covariance and t-based intervals address heteroskedasticity in the binary response, following the general framework of White, 1980. No feature selection based on significance is performed. All four declared covariates remain in the model, and the constant ordinary-call budget is reported separately.

The interval interpretation concerns hypothetical variation across tasks resembling the retained collection under the model assumptions. It is not uncertainty about the exact count of 157 diverse tasks in this release. Shared authorship, task templates, or semantic families could create dependence that HC3 does not address. The supplied metadata do not provide an adjudicated family hierarchy, so we do not invent one from surface similarity and present it as verified task clustering.

Agreement and reliability concepts in Bland and Altman, 1986 and Koo and Li, 2016 are relevant to future manual validation of the grammar. An independent annotation study could compare whether human reviewers regard our distinctions as useful. We did not conduct that study and do not report inter-rater reliability. The grammar's deterministic reproducibility is a different property from its content validity as a measure of meaningful input variation.

Table 1. Associations with at least two ordinary-call signatures. All tasks have three calls; HC3 intervals accompany the four varying explanatory features.

Model	Term	Estimate (SE)	95% CI	N
M1	Reference control flow	0.1455 (0.0315)	[0.0836, 0.2074]	974
M1	Log syntax nodes	-0.0958 (0.0227)	[-0.1403, -0.0512]	974
M1	Log(1 + instruction words)	0.0553 (0.0525)	[-0.0477, 0.1584]	974
M1	Nonempty test setup	-0.1158 (0.0388)	[-0.1919, -0.0397]	974
M1: fit	R-squared = 0.0307	-	-	974

Model labels: M1 = Multiple input-shape associations.

6. What the static audit finds

The signature counts are one for 817 tasks, two for 127 tasks, and three for thirty tasks. Thus 157 of 974 tasks, or about 16.1 percent, meet the declared multiple-signature definition. The remaining tasks are not automatically poorly tested. Their distinct numeric values, collection lengths, or expected outputs may exercise important behavior that this coarse grammar intentionally ignores. The result describes repeated input structure, not duplicated test text.

After adjustment, the control-flow coefficient is 0.1455, with standard error 0.0315 and interval approximately 0.0836 to 0.2074. Reference syntax size has a negative coefficient of -0.0958, with interval approximately -0.1403 to -0.0512. Instruction length has an uncertain positive association, while nonempty setup accompanies a lower diversity probability. The full model R-squared is about 0.0307. Exact coefficients and covariance calculations are retained in the result file.

The opposing control-flow and syntax-size associations are not contradictory. The first compares the presence of a broad structural feature conditional on code size; the second compares size conditional on that feature and other covariates. They cannot establish that adding a branch causes benchmark authors to

choose more diverse tests. Task semantics can influence both the reference program and the selected examples, and those semantics are not fully represented by four features.

We do not evaluate the regression as a classifier of inadequate tests. Probability calibration in Guo et al., 2017, proper-scoring ideas in Gneiting and Raftery, 2007, ROC analysis in Fawcett, 2006, and imbalance considerations in Saito and Rehmsmeier, 2015 would matter for a deployed screening model. Here the response is a constructed audit indicator, not a verified defect label. Selecting a decision threshold would add an operational claim for which no independent validation outcome has been collected.

7. Where the grammar is deliberately silent

Two inputs with identical signatures may differ at an algorithm's critical boundary. A positive number equal to one can behave differently from a larger positive number; a sorted list can differ from an unsorted list even when both contain positive integers. Conversely, two different signatures may lead through exactly the same execution path. These examples show why static shape diversity cannot substitute for branch coverage, mutation testing, or a semantic analysis of the specification.

Nonliteral expressions create another limit. An abstract-syntax call or name can evaluate to many kinds of object, but our parser records its syntactic category. Evaluating those expressions would require execution or richer static analysis and would change the workflow. We preserve that uncertainty instead of fabricating runtime values. A future extension could use a separately validated abstract interpreter, but should report its unresolved cases and compare against the present deterministic baseline.

Influence diagnostics following Cook, 1977 could locate unusual reference sizes or setup patterns, yet removing those tasks after observing the coefficients would undermine the declared census audit. The guidance in Wasserstein and Lazar, 2016 and concerns in Ioannidis, 2005 similarly support reading the association as exploratory. The substantive result is the explicit distribution of signatures and its limited structural association, rather than a significance-based claim that one category of programming task is intrinsically easier to test.

8. Evidence that can be reused

The grammar also distinguishes diversity from duplication. Three tests with one signature can contain three different function calls, different expected outputs, and different edge-relevant values. They are not counted as copied text, and the manuscript does not accuse their authors of redundancy. The audit asks a deliberately coarser question: whether the types and broad signs or emptiness properties presented to the function change. That distinction is necessary if the result is to inform example design without overstating what a static parser has learned about behavior.

A useful manual validation sample could include tasks from each signature-count group and ask reviewers to identify the behavioral boundaries exercised by ordinary tests. Reviewers should be blinded to the regression outcome and should use an explicit annotation protocol. Comparing those judgments with the deterministic grammar would reveal both false reassurance and missed meaningful differences. Until such evidence exists, a single-signature task is a candidate for closer inspection, not a confirmed specification defect or a reason to discard the benchmark record.

The fixed call budget makes the descriptive contrast unusually clean. Differences in the outcome cannot arise simply because one retained task contains ten ordinary calls while another contains only one. They can still arise from task semantics, author choices, and the grammar's treatment of compound arguments. This combination of a constant budget and an intentionally limited construct is the principal value of the

audit: it makes one hidden dimension of example design visible while leaving broader adequacy questions open.

The processing record follows Gebru et al., 2021 by specifying the grammar, call identification rule, ordinary-test restriction, and constant budget. Wilkinson et al., 2016 motivates retaining the public source file, task-level features, and executable analysis logic. Downloaded program text is treated as data. No reference solution, import statement, test assertion, or challenge test is executed to produce the reported counts and regression.

A useful follow-up would compare the coarse signatures with manually identified boundary cases and actual execution coverage on a reviewed task sample. Another could test whether additional examples selected for shape diversity improve a coding assistant under a fresh evaluation design. Neither result follows automatically from this audit. The present evidence says that a common three-test budget frequently repeats one coarse input structure, while leaving the adequacy of those tests an open substantive question.

References

1. Little, Roderick J. A., and Donald B. Rubin. *Statistical Analysis with Missing Data*. Wiley, 2002. https://doi.org/10.1002/9781119013563.
2. Saito, Takaya and Marc Rehmsmeier. “The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets.” *PLOS ONE*, vol. 10, no. 3, 2015, pp. e0118432. https://doi.org/10.1371/journal.pone.0118432.
3. Wilkinson, Mark D.; Dumontier, Michel; Aalbersberg, IJsbrand Jan; Appleton, Gabrielle; Axton, Myles; Baak, Arie; Blomberg, Niklas; Boiten, Jan-Willem; da Silva Santos, Luiz Bonino; Bourne, Philip E.; Bouwman, Jildau; Brookes, Anthony J.; Clark, Tim; Crosas, Mercè; Dillo, Ingrid; Dumon, Olivier; Edmunds, Scott; Evelo, Chris T.; Finkers, Richard; Gonzalez-Beltran, Alejandra; Gray, Alasdair J.G.; Groth, Paul; Goble, Carole; Grethe, Jeffrey S.; Heringa, Jaap; ’t Hoen, Peter A.C; Hooft, Rob; Kuhn, Tobias; Kok, Ruben; Kok, Joost; Lusher, Scott J.; Martone, Maryann E.; Mons, Albert; Packer, Abel L.; Persson, Bengt; Rocca-Serra, Philippe; Roos, Marco; van Schaik, Rene; Sansone, Susanna-Assunta; Schultes, Erik; Sengstag, Thierry; Slater, Ted; Strawn, George; Swertz, Morris A.; Thompson, Mark; van der Lei, Johan; van Mulligen, Erik; Velterop, Jan; Waagmeester, Andra; Wittenburg, Peter; Wolstencroft, Katherine; Zhao, Jun; Mons, Barend. (2016). The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data. https://doi.org/10.1038/sdata.2016.18.
4. Guo, Chuan, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. “On Calibration of Modern Neural Networks.” *Proceedings of the 34th International Conference on Machine Learning*, PMLR, vol. 70, 2017, pp. 1321-1330. https://proceedings.mlr.press/v70/guo17a.html.
5. Guo, Daya; Zhu, Qihao; Yang, Dejian; Xie, Zhenda; Dong, Kai; Zhang, Wentao; Chen, Guanting; Bi, Xiao; Wu, Y.; Li, Y. K.; Luo, Fuli; Xiong, Yingfei; Liang, Wenfeng. (2024). DeepSeek-Coder: When the Large Language Model Meets Programming -- The Rise of Code Intelligence. arXiv:2401.14196. https://arxiv.org/abs/2401.14196.
6. Rafailov, Rafael; Sharma, Archit; Mitchell, Eric; Ermon, Stefano; Manning, Christopher D.; Finn, Chelsea. (2023). Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290. https://arxiv.org/abs/2305.18290.
7. Jain, Naman; Han, King; Gu, Alex; Li, Wen-Ding; Yan, Fanjia; Zhang, Tianjun; Wang, Sida; Solar-Lezama, Armando; Sen, Koushik; Stoica, Ion. (2024). LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. arXiv:2403.07974. https://arxiv.org/abs/2403.07974.
8. Chen, Mark; Tworek, Jerry; Jun, Heewoo; Yuan, Qiming; Pinto, Henrique Ponde de Oliveira; Kaplan, Jared; Edwards, Harri; Burda, Yuri; Joseph, Nicholas; Brockman, Greg; Ray, Alex; Puri, Raul; Krueger, Gretchen; Petrov, Michael; Khlaaf, Heidy; Sastry, Girish; Mishkin, Pamela; Chan, Brooke; Gray, Scott; Ryder, Nick; Pavlov, Mikhail; Power, Alethea; Kaiser, Lukasz; Bavarian, Mohammad; Winter, Clemens; Tillet, Philippe; Such, Felipe Petroski; Cummings, Dave; Plappert, Matthias; Chantzis, Fotios; Barnes, Elizabeth; Herbert-Voss, Ariel; Guss, William Hebgen; Nichol, Alex; Paino, Alex; Tezak, Nikolas; Tang, Jie; Babuschkin, Igor; Balaji, Suchir; Jain, Shantanu; Saunders, William; Hesse, Christopher; Carr, Andrew N.; Leike, Jan; Achiam, Josh; Misra, Vedant; Morikawa, Evan; Radford, Alec; Knight, Matthew; Brundage, Miles; Murati, Mira; Mayer, Katie; Welinder, Peter; McGrew, Bob; Amodei, Dario; McCandlish, Sam; Sutskever, Ilya; Zaremba, Wojciech. (2021). Evaluating Large Language Models Trained on Code. arXiv:2107.03374. https://arxiv.org/abs/2107.03374.

9. Ioannidis, John P. A.. (2005). Why Most Published Research Findings Are False. PLoS Medicine. https://doi.org/10.1371/journal.pmed.0020124.

10. Cook, R. Dennis. (1977). Detection of Influential Observation in Linear Regression. Technometrics. https://doi.org/10.1080/00401706.1977.10489493.

11. White, Halbert. “A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity.” *Econometrica*, vol. 48, no. 4, 1980, pp. 817-838. https://doi.org/10.2307/1912934.

12. Le, Hung; Wang, Yue; Gotmare, Akhilesh Deepak; Savarese, Silvio; Hoi, Steven C. H.. (2022). CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. arXiv:2207.01780. https://arxiv.org/abs/2207.01780.

13. Fried, Daniel; Aghajanyan, Armen; Lin, Jessy; Wang, Sida; Wallace, Eric; Shi, Freda; Zhong, Ruiqi; Yih, Wen-tau; Zettlemoyer, Luke; Lewis, Mike. (2022). InCoder: A Generative Model for Code Infilling and Synthesis. arXiv:2204.05999. https://arxiv.org/abs/2204.05999.

14. Bland, J. Martin, and Douglas G. Altman. "Statistical Methods for Assessing Agreement between Two Methods of Clinical Measurement." *The Lancet*, vol. 327, no. 8476, 1986, pp. 307-310. https://doi.org/10.1016/S0140-6736(86)90837-8.

15. Austin, Jacob; Odena, Augustus; Nye, Maxwell; Bosma, Maarten; Michalewski, Henryk; Dohan, David; Jiang, Ellen; Cai, Carrie; Terry, Michael; Le, Quoc; Sutton, Charles. (2021). Program Synthesis with Large Language Models. arXiv:2108.07732. https://arxiv.org/abs/2108.07732.

16. Gebru, Timnit, et al. "Datasheets for datasets" *Communications of the ACM*, 2021. https://doi.org/10.1145/3458723.

17. Liu, Jiawei; Xia, Chunqiu Steven; Wang, Yuyao; Zhang, Lingming. (2023). Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. arXiv:2305.01210. https://arxiv.org/abs/2305.01210.

18. Bird, Christian; Rigby, Peter C.; Barr, Earl T.; Hamilton, David J.; German, Daniel M.; Devanbu, Premkumar. (2009). The promises and perils of mining git. https://doi.org/10.1109/MSR.2009.5069475.

19. Herzig, Kim; Just, Sascha; Zeller, Andreas. (2013). It’s not a bug, it’s a feature: How misclassification impacts bug prediction. https://doi.org/10.1109/ICSE.2013.6606585.

20. Koo, Terry K., and Mae Y. Li. "A Guideline of Selecting and Reporting Intraclass Correlation Coefficients for Reliability Research." *Journal of Chiropractic Medicine*, vol. 15, no. 2, 2016, pp. 155-163. https://doi.org/10.1016/j.jcm.2016.02.012.

21. Tao, Hongmo; Kamata, Hiroyuki. (2026). Teaching Coding Assistants What to Change: Fine-Tuning from Pairwise Preferences. Proceedings of the Second Workshop for REsearch on Agent Language Models (REALM 2026), 1363-1374. https://doi.org/10.18653/2026.realm-1.95.

22. Hu, Edward J., et al. “LoRA: Low-Rank Adaptation of Large Language Models.” *International Conference on Learning Representations*, 2022. https://openreview.net/forum?id=nZeVKeeFYf9.

23. Carroll, Raymond J., David Ruppert, Leonard A. Stefanski, and Ciprian M. Crainiceanu. “Measurement Error in Nonlinear Models.” 2006. https://doi.org/10.1201/9781420010138.

24. Guo, Daya; Ren, Shuo; Lu, Shuai; Feng, Zhangyin; Tang, Duyu; Liu, Shujie; Zhou, Long; Duan, Nan; Svyatkovskiy, Alexey; Fu, Shengyu; Tufano, Michele; Deng, Shao Kun; Clement, Colin; Drain, Dawn; Sundaresan, Neel; Yin, Jian; Jiang, Daxin; Zhou, Ming. (2020). GraphCodeBERT: Pre-training Code Representations with Data Flow. arXiv:2009.08366. https://arxiv.org/abs/2009.08366.

25. Kitchenham, Barbara A.; Pfleeger, Shari Lawrence; Pickard, Lesley M.; Jones, Peter W.; Hoaglin, David C.; El Emam, Khaled; Rosenberg, Jarrett. (2002). Preliminary guidelines for empirical research in software engineering. https://doi.org/10.1109/TSE.2002.1027796.

26. Li, Raymond; Allal, Loubna Ben; Zi, Yangtian; Muennighoff, Niklas; Kocetkov, Denis; Mou, Chenghao; Marone, Marc; Akiki, Christopher; Li, Jia; Chim, Jenny; Liu, Qian; Zheltonozhskii, Evgenii; Zhuo, Terry Yue; Wang, Thomas; Dehaene, Olivier; Davaadorj, Mishig; Lamy-Poirier, Joel; Monteiro, João; Shliazhko, Oleh; Gontier, Nicolas; Meade, Nicholas; Zebaze, Armel; Yee, Ming-Ho; Umapathi, Logesh Kumar; Zhu, Jian; Lipkin, Benjamin; Oblokulov, Muhtasham; Wang, Zhiruo; Murthy, Rudra; Stillerman, Jason; Patel, Siva Sankalp; Abulkhanov, Dmitry; Zocca, Marco; Dey, Manan; Zhang, Zhihan; Fahmy, Nour; Bhattacharyya, Urvashi; Yu, Wenhao; Singh, Swayam; Luccioni, Sasha; Villegas, Paulo; Kunakov, Maxim; Zhdanov, Fedor; Romero, Manuel; Lee, Tony; Timor, Nadav; Ding, Jennifer; Schlesinger, Claire; Schoelkopf, Hailey; Ebert, Jan; Dao, Tri; Mishra, Mayank; Gu, Alex; Robinson, Jennifer; Anderson, Carolyn Jane; Dolan-Gavitt, Brendan; Contractor, Danish; Reddy, Siva; Fried, Daniel; Bahdanau, Dzmitry; Jernite, Yacine; Ferrandis, Carlos Muñoz; Hughes, Sean; Wolf, Thomas; Guha, Arjun; von Werra, Leandro; de Vries, Harm. (2023). StarCoder: may the source be with you!. arXiv:2305.06161. https://arxiv.org/abs/2305.06161.

27. Wasserstein, Ronald L.; Lazar, Nicole A.. (2016). The ASA Statement on p -Values: Context, Process, and Purpose. The American Statistician. https://doi.org/10.1080/00031305.2016.1154108.

28. Fawcett, Tom. (2006). An introduction to ROC analysis. Pattern Recognition Letters. https://doi.org/10.1016/j.patrec.2005.10.010.

29. Gneiting, Tilmann; Raftery, Adrian E. (2007). Strictly Proper Scoring Rules, Prediction, and Estimation. Journal of the American Statistical Association. https://doi.org/10.1198/016214506000001437.

30. Bacchelli, Alberto; Bird, Christian. (2013). Expectations, outcomes, and challenges of modern code review. 2013 35th International Conference on Software Engineering (ICSE). https://doi.org/10.1109/icse.2013.6606617.