**Journal of Transfer and Generalization**
Volume 1 · Number 1 · 2026

# How Much Defect Detection Survives a Smaller Test Suite? Exact Retention Probabilities on QuixBugs

Alexander Collins[1] · Blake Reynolds[2] · Evan Anderson[3]

[1] Poland · [2] Hungary · [3] Croatia

Original publication: November 1, 2026 | Article 568

## Abstract

A benchmark test suite can expose a buggy program while a small retained subset misses it. On twenty-six QuixBugs tasks passing a declared reference-execution and oracle-compatibility gate, a uniformly chosen single provided test detected the captured buggy implementation with mean probability 0.690. Keeping three tests raised that probability to 0.927, an average gain of 0.237. These probabilities are calculated exactly from executed test outcomes rather than estimated from repeated random subsets. Five other JSON-backed tasks were excluded by the gate, including numerical-oracle and execution-budget cases. The study concerns test retention under a fixed outcome convention; it neither estimates repair success nor treats a runtime cap as proof of nontermination.

*Bounded execution of reviewed public benchmark functions. No coding assistant was trained or asked to generate a repair.*

# How Much Defect Detection Survives a Smaller Test Suite? Exact Retention Probabilities on QuixBugs

Bounded execution of reviewed public benchmark functions. No coding assistant was trained or asked to generate a repair.


## Abstract

A benchmark test suite can expose a buggy program while a small retained subset misses it. On twenty-six QuixBugs tasks passing a declared reference-execution and oracle-compatibility gate, a uniformly chosen single provided test detected the captured buggy implementation with mean probability 0.690. Keeping three tests raised that probability to 0.927, an average gain of 0.237. These probabilities are calculated exactly from executed test outcomes rather than estimated from repeated random subsets. Five other JSON-backed tasks were excluded by the gate, including numerical-oracle and execution-budget cases. The study concerns test retention under a fixed outcome convention; it neither estimates repair success nor treats a runtime cap as proof of nontermination.


## 1. A passing test is evidence relative to a budget

Program-repair evaluation often summarizes whether a candidate passes a supplied suite. The summary depends on which tests are retained. A small subset can accept the original buggy implementation even when the full suite detects it. That observation does not require generating a new repair: the public buggy and corrected programs already provide a controlled way to ask how much detection evidence a reduced test budget retains.

Our research question is how much uniformly retaining three provided tests rather than one increases the chance of detecting a captured buggy implementation. We separate the execution evidence from the sampling calculation. First, each available JSON case is run on reviewed benchmark functions under a fixed execution policy. Then the resulting pass/fail pattern supplies an exact probability for each retained-test budget. The procedure involves no learned test selector or repeated random-subset approximation.

The preference-learning setting in Tao and Kamata, 2026 makes the quality of evaluation signals relevant: a comparison between coding outputs needs to specify what counts as successful behavior. Our experiment does not reproduce that fine-tuning method or create pairwise training preferences. It examines a more elementary signal, namely whether a reduced subset of existing tests still rejects the original faulty code. Its findings cannot establish how an assistant would respond to richer tests or what it would learn from their outcomes.

## 2. Public repair tasks and the meaning of the oracle

The resource introduced by Lin et al., 2017 contains short algorithmic programs with captured buggy and corrected versions. It was designed as a multilingual repair benchmark. We analyze only its Python functions with line-delimited JSON input/output cases. Broader repair resources such as Jimenez et al., 2023, the agent setting in Yang et al., 2024, and the approach in Xia et al., 2024 involve repository-level context and a different form of evaluation. They are relevant context, not additional tasks included in our sample.

The programming-task collection in Austin et al., 2021 and evaluation issues addressed by Liu et al., 2023 also illustrate that a test count is not a complete description of the evidence tests supply. A case may be

redundant with others, exercise a special boundary, or reject an output that another semantic convention would accept. Our exact retention probabilities account for the observed number of rejecting tests, while leaving semantic adequacy and untested behavior outside the measured endpoint.

Code-model systems in Chen et al., 2021, Le et al., 2022, Fried et al., 2022, Guo et al., 2020, and Li et al., 2023 establish relevant settings in which executable feedback can matter. None is executed here. No model-generated candidate enters the analysis, so an observed detection probability must not be labeled an assistant accuracy or repair pass rate. Keeping that distinction explicit allows a test-budget study to inform evaluation design without borrowing achievements from an unrun model comparison.

## 3. Exact source identities precede execution

The captured official source archive contains forty main benchmark programs. Thirty-one have JSON-backed tasks under the interface used here; the remaining graph-oriented tasks are outside this design. Exact archive paths distinguish `python_programs` from `correct_python_programs`. Suffix matching would be unsafe because the corrected-directory name contains the buggy-directory suffix. We therefore select both files by their complete paths and preserve the archive hash defining the analyzed version.

The functions are parsed and screened before execution. Module contents are limited to function definitions, reviewed imports, and descriptive string literals. The available builtins exclude file access and dynamic evaluation. Reviewed standard helpers for heaps, character alphabets, and counting are supplied directly, and the repository's general test driver is not run. These controls support a bounded experiment on simple functions rather than executing a downloaded project's unrestricted initialization path.

Every call receives a copied version of its JSON arguments so that a mutating implementation cannot alter inputs supplied to a subsequent call. Generator outputs are materialized; tuple and list representations are normalized to a common sequence form. Numeric comparisons use a tolerance of 1e-10. These are evaluation conventions, not discoveries about semantic equivalence. An algorithm with several admissible outputs can require a richer oracle than exact agreement with one recorded output.

A trace counts executed lines in the reviewed benchmark source. Each call has a limit of 100,000 such lines. Hitting the cap is recorded separately from a wrong output or another exception. This operational boundary prevents an unbounded computation from blocking the study, but does not prove that the function would never terminate. Recursion-limit exceptions are likewise recorded as actual execution outcomes under the current interpreter, rather than silently combined with ordinary numerical mismatches.

## 4. Require the corrected reference to pass the declared gate

Before calculating a task's retention probability, its corrected implementation must pass every captured JSON oracle within the same execution policy. This gate avoids using a suite as unquestioned evidence when the declared evaluator does not accept the supplied reference. It also creates a restricted sample. The gate can exclude computationally expensive tasks even if their algorithms are semantically correct, and it can exclude tasks whose numeric outputs satisfy a broader specification while differing from recorded values.

Of the thirty-one JSON-backed tasks, twenty-six pass the gate. The excluded tasks are knapsack, Levenshtein distance, longest common subsequence, possible change, and square root. The first four have at least one corrected call reaching the line limit. Square root has two corrected-output disagreements

under the strict numerical oracle convention. We do not interpret those two disagreements as proof that the corrected algorithm violates every valid square-root specification.

Across all thirty-one tasks, 242 JSON cases are executed on both captured versions. Corrected calls pass 231 cases, reach the line budget in nine, and disagree numerically with two recorded outputs. The eligible twenty-six tasks contribute 198 cases. All their corrected calls pass by construction. The exclusion rule is reported alongside the result so that an apparently favorable test-budget summary is not mistaken for evidence covering every original benchmark task.

Repository-data methods in Bird et al., 2009 and empirical reporting principles in Kitchenham et al., 2002 motivate recording the extracted unit and transformations. Barr et al., 2015 distinguishes an executable oracle from a complete correctness criterion, while Rothermel et al., 2001 addresses ordering tests for regression testing. Our fixed retention probabilities use neither an optimized ordering nor a representative sample of production defects. Their unit is a JSON-backed short task passing the declared evaluator gate, with expected-output conventions that must remain visible.

## 5. From observed failures to an exact probability

For an eligible program, the execution record identifies how many provided cases reject its captured buggy implementation. Rejection includes wrong output, an exception, or exceeding the declared line budget. We consider uniform sampling without replacement from that program's own provided cases. The probability that a retained subset detects the bug is one minus the probability that every selected case belongs to the passing set.

The one-test probability is consequently the observed rejecting fraction. For three tests, the calculation uses the corresponding combinatorial counts of all-passing subsets and all possible subsets. Every eligible task has at least three provided cases, so the same budget is feasible throughout. We calculate the ratio directly rather than simulating many random subsets. There is no Monte Carlo standard error attached to the resulting finite-suite probability.

This exact calculation does not make the test-generation process random. The original suite was curated for its benchmark purpose, and a deployment test selector need not sample uniformly. Our probability belongs to a declared retention policy, conditional on the captured suite and evaluator. It cannot be interpreted as the probability that three arbitrary future tests will detect an arbitrary real-world bug. Such a claim would require an additional model of how tests and defects arise.

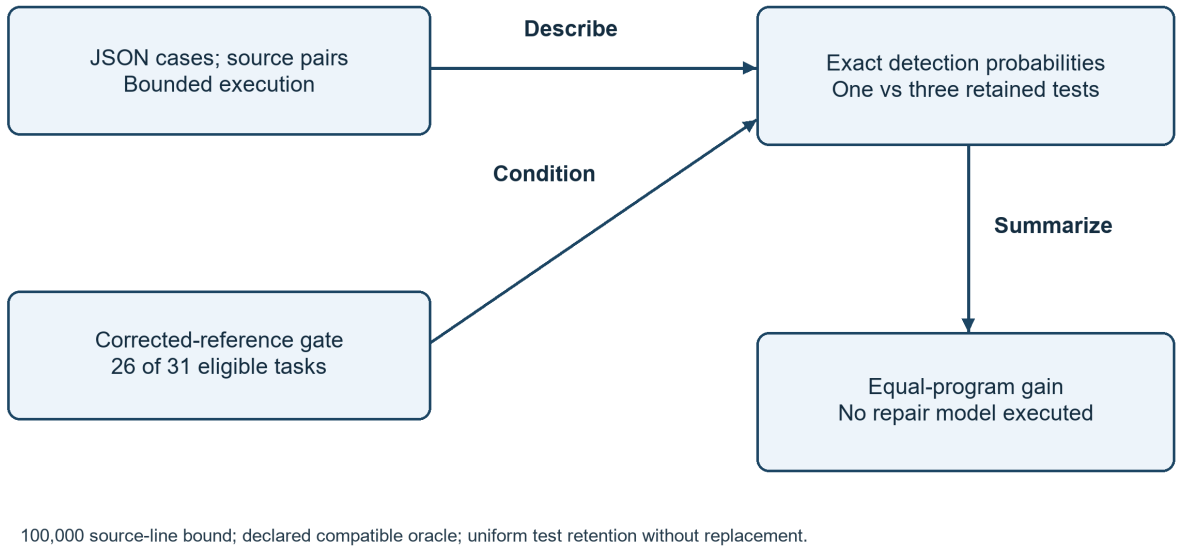



*Figure 1. Reference eligibility and bounded buggy-program outcomes feed exact one-test and three-test detection probabilities.*

## 6. Equal-program summaries and their uncertainty

We give every eligible program equal weight when averaging its one-test probability, three-test probability, and their difference. This prevents a program with more supplied cases from dominating the summary merely because its suite is larger. The equal-program mean asks how a randomly selected task from this fixed eligible set behaves under the declared within-task retention policy. Pooling every test across programs would instead answer a case-weighted question.

An intercept-only ordinary-least-squares regression summarizes the program-level gain. HC3 covariance is used, following the heteroskedasticity considerations in White, 1980, with a normal-reference interval. There are twenty-six observations. The exact finite-corpus mean is already known from the execution table; the interval is meaningful only under a hypothetical interpretation of programs as draws from a broader task population. It is not uncertainty from repeating the same combinatorial calculation.

Dependence perspectives in Cameron and Miller, 2015 caution that programs can share algorithm families or suite-design practices even when their source files differ. No reliable independent family-cluster sample is established here. A small set of clusters invented after seeing the result would not solve that problem. We preserve the simple HC3 summary and describe its interpretation limit instead of claiming a fully population-representative uncertainty calculation.

*Table 1. Equal-program difference between exact three-test and one-test detection probabilities.*

| Model | Term | Estimate (SE) | 95% CI | N |
|---|---|---|---|---|
| M1 | Mean outcome/contrast | 0.2370 (0.0337) | [0.1710, 0.3029] | 26 |
| M1: fit | R-squared = 0.0000 | - | - | 26 |

Model labels: M1 = Mean three-minus-one test detection probability.

## 7. Three tests retain more evidence, with a visible remaining gap

The mean one-test detection probability is 0.68956. With three tests it is 0.92651. The mean gain is 0.23696, with standard error 0.03367 and interval 0.17096 to 0.30295 under the recorded summary model. Expressed as percentage points, the average gain is approximately 23.7. The intercept fit has effectively zero R-squared because it estimates a mean gain rather than explaining variation among programs.

All twenty-six eligible full suites detect their captured buggy implementation. Nevertheless, three-test detection remains below one on average. A reduced suite can therefore accept the original faulty version even in a resource whose complete eligible suite rejects every such version. This finding concerns loss of observed rejecting evidence under retention. It does not imply that every larger suite is adequate, that every added test supplies unique coverage, or that a passing repair is semantically correct.

The execution record distinguishes rejection types rather than presenting all failures as wrong answers. Across the full thirty-one-task execution, captured buggy calls include 105 wrong outputs, twenty-three line-budget events, twenty-two recursion-limit exceptions, eight index errors, four type errors, and nine value errors; seventy-one pass. Those totals describe the broader executed interface, while the probability regression uses only the eligible subset. Mixing those denominators would give an incorrect account of the reported average gain.

Scoring ideas in Gneiting and Raftery, 2007 clarify the role of the endpoint: the object is a probability of rejection under a selected retention rule, not a calibrated confidence that code is correct. A candidate that passes one retained suite still has untested behavior. Measurement issues in Carroll et al., 2006 apply to the evaluator itself, including its declared container normalization, numeric tolerance, and execution limits. The reference gate exposes some such limitations without proving that every remaining oracle is complete.

## 8. Selection and missing tasks constrain generalization

Validation concerns in Roberts et al., 2017 and selection-bias reasoning in Varma and Simon, 2006 become relevant if retention policies or evaluator thresholds are tuned to these same outcomes. The current study fixes one and three tests and the line budget. It does not select a budget maximizing a favorable reported gain. A future policy trained to prioritize tests would need separate tasks for policy development and evaluation, and should retain its own reference-compatibility accounting.

Missing-data principles in Little and Rubin, 2002 also apply to tasks absent from this interface or excluded by the gate. The excluded computationally expensive functions can differ systematically from retained functions in how tests expose faults. Treating their absence as random would be unwarranted. The result therefore remains conditional on twenty-six eligible tasks, not a complete benchmark comparison disguised by a smaller denominator.

Influence reasoning in Cook, 1977 could identify programs with unusually large gains, but they are not deleted here. Guidance in Wasserstein and Lazar, 2016 and concerns in Ioannidis, 2005 support reporting the actual magnitude and eligibility limits. Broader exploration over many budgets would introduce multiplicity issues addressed by Benjamini and Hochberg, 1995 and Benjamini and Yekutieli, 2001. The present exact budget contrast does not eliminate those concerns for a future search over test-selection strategies.

## 9. What this experiment contributes to repair evaluation

The experiment demonstrates that supplied-suite size and supplied-suite adequacy are distinct empirical objects. Under one transparent retention policy, three tests preserve much more of the suite's observed rejection ability than one, while still leaving a detectable evidence gap relative to the complete suite. That conclusion can guide evaluation reporting without assuming any particular repair system. It is especially useful when a study is tempted to describe a small retained suite only by its number of cases.

Documentation following Gebru et al., 2021 records exact source paths, input copying, output normalization, reference eligibility, runtime bounds, rejection categories, and the probability calculation. Wilkinson et al., 2016 motivates preserving both the case execution table and task summaries. Human review remains necessary for oracle semantics and generalization beyond these tasks. The article provides executed evidence about a test-retention decision, while leaving actual assistant improvement, repair correctness, and industrial defect coverage to designs that observe those outcomes directly.

## References


1. 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.
2. Ioannidis, John P. A.. (2005). Why Most Published Research Findings Are False. PLoS Medicine. https://doi.org/10.1371/journal.pmed.0020124.
3. Barr, Earl T.; Harman, Mark; McMinn, Phil; Shahbaz, Muzammil; Yoo, Shin. (2015). The Oracle Problem in Software Testing: A Survey. IEEE Transactions on Software Engineering. https://doi.org/10.1109/tse.2014.2372785.
4. Xia, Chunqiu Steven; Deng, Yinlin; Dunn, Soren; Zhang, Lingming. (2024). Agentless: Demystifying LLM-based Software Engineering Agents. arXiv:2407.01489. https://arxiv.org/abs/2407.01489.
5. 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.
6. 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.
7. Jimenez, Carlos E.; Yang, John; Wettig, Alexander; Yao, Shunyu; Pei, Kexin; Press, Ofir; Narasimhan, Karthik. (2023). SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. arXiv:2310.06770. https://arxiv.org/abs/2310.06770.
8. 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.
9. 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.

10. 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.
11. Varma, Sudhir and Richard Simon. “Bias in error estimation when using cross-validation for model selection.” *BMC Bioinformatics*, vol. 7, no. 1, 2006. https://doi.org/10.1186/1471-2105-7-91.
12. Benjamini, Yoav and Yosef Hochberg. “Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing.” *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 57, no. 1, 1995, pp. 289-300. https://doi.org/10.1111/j.2517-6161.1995.tb02031.x.
13. Lin, Derrick; Koppel, James; Chen, Angela; Solar-Lezama, Armando. (2017). QuixBugs: a multi-lingual program repair benchmark set based on the quixey challenge. Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity. https://doi.org/10.1145/3135932.3135941.
14. Benjamini, Yoav and Daniel Yekutieli. “The control of the false discovery rate in multiple testing under dependency.” *The Annals of Statistics*, vol. 29, no. 4, 2001, pp. 1165-1188. https://doi.org/10.1214/aos/1013699998.
15. Cook, R. Dennis. (1977). Detection of Influential Observation in Linear Regression. Technometrics. https://doi.org/10.1080/00401706.1977.10489493.
16. 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.
17. Gebru, Timnit, et al. "Datasheets for datasets" *Communications of the ACM*, 2021. https://doi.org/10.1145/3458723.
18. Rothermel, G.; Untch, R.H.; Chengyun Chu; Harrold, M.J.. (2001). Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering. https://doi.org/10.1109/32.962562.
19. Carroll, Raymond J., David Ruppert, Leonard A. Stefanski, and Ciprian M. Crainiceanu. “Measurement Error in Nonlinear Models.” 2006. https://doi.org/10.1201/9781420010138.
20. Yang, John; Jimenez, Carlos E.; Wettig, Alexander; Lieret, Kilian; Yao, Shunyu; Narasimhan, Karthik; Press, Ofir. (2024). SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793. https://arxiv.org/abs/2405.15793.
21. Little, Roderick J. A., and Donald B. Rubin. *Statistical Analysis with Missing Data*. Wiley, 2002. https://doi.org/10.1002/9781119013563.
22. 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.
23. 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.
24. Roberts, David R., et al. “Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure.” *Ecography*, vol. 40, no. 8, 2017, pp. 913-929. https://doi.org/10.1111/ecog.02881.
25. 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.
26. 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.
27. Cameron, A. Colin, and Douglas L. Miller. “A Practitioner’s Guide to Cluster-Robust Inference.” *Journal of Human Resources*, vol. 50, no. 2, 2015, pp. 317-372. https://doi.org/10.3368/jhr.50.2.317.
28. 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.

29. 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.

30. 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.