Journal of Audit and Accountability
Volume 1 · Number 1 · 2026

Before a Test Becomes a Preference: Auditing Output Conventions in Public Program Pairs

Jonathan Evans[1] · Charles Reynolds[2] · Gavin Bell[3]

[1] Poland · [2] Hungary · [3] Greece

Original publication: November 7, 2026 | Article 691

Abstract

An executable test still needs a rule for comparing its output with an expected value. Across 31 JSON-backed QuixBugs tasks and 242 cases, replacing literal Python equality with declared container and numeric compatibility changed sixteen test-derived pairwise labels. Changes occurred in three tasks; repository-designated corrected-reference acceptance rose from 213 to 231 cases. The code candidates themselves did not change. This audit shows why a preference-data record should preserve its oracle convention and unresolved ties before being used to teach a coding assistant what to change.

Research draft based on bounded local execution of public benchmark functions. Test-derived labels are exploratory and have not been certified as human training preferences.

Before a Test Becomes a Preference: Auditing Output Conventions in Public Program Pairs

> Research draft based on bounded local execution of public benchmark functions. Test-derived labels are exploratory and have not been certified as human training preferences.

Abstract

An executable test still needs a rule for comparing its output with an expected value. Across 31 JSON-backed QuixBugs tasks and 242 cases, replacing literal Python equality with declared container and numeric compatibility changed sixteen test-derived pairwise labels. Changes occurred in three tasks; repository-designated corrected-reference acceptance rose from 213 to 231 cases. The code candidates themselves did not change. This audit shows why a preference-data record should preserve its oracle convention and unresolved ties before being used to teach a coding assistant what to change.

1. A preference label has an upstream measurement process

Pairwise learning for coding systems often presents a preference as a relation between two candidate programs. When the relation comes from tests, a measurement process sits upstream: the harness must execute each candidate, interpret the returned object, compare it with an oracle, and decide how failures contribute to the pair. If those conventions are implicit, an apparent preference can reflect an interface choice rather than a meaningful difference in the requested program behavior. The code alone does not uniquely determine the label.

Our research question is how much explicit output-comparison conventions can change test-derived pairwise labels for fixed public program pairs. The study executes known buggy and repository-designated corrected implementations from QuixBugs Lin et al., 2017. It compares two declared oracle policies on the same returned values. The broader oracle problem surveyed by Barr et al., 2015 motivates treating output interpretation as a scientific object. We do not assume that a supplied expected value, or a directory named corrected, automatically establishes complete semantic correctness.

Tao and Kamata, 2026 study teaching coding assistants what to change through fine-tuning from pairwise preferences. That setting motivates the audit's practical question: which parts of a preference record need to be preserved so that its meaning can be inspected? We do not reproduce the fine-tuning method or use its data. Our observations are local executions of existing benchmark code. They show how an upstream evaluation convention can change a proposed signal before any model training begins.

The analytical proposition is that comparison-policy sensitivity can alter reference acceptance and pairwise discrimination differently. A convention may accept additional corrected outputs while also accepting some buggy outputs. Therefore, counting added passes is not enough to infer how many pairwise labels change. The audit retains candidate-level acceptance and the resulting pair relation. It reports ties as unresolved comparisons, rather than silently converting both-fail cases into evidence that the programs are equally useful or equally correct.

2. The finite benchmark and its two conventions

The captured archive contains thirty-one tasks with the required JSON-case interface and exact Python source pairs. Nine graph-oriented benchmark tasks without that interface are outside this design. We retain all thirty-one JSON-backed tasks, including tasks whose designated corrected implementation does not pass every available case under our conventions. This differs from using a corrected-reference pass gate to define a clean test-retention cohort. Here, such exclusions would remove precisely the compatibility questions being audited.

The literal policy compares the returned Python value directly with the JSON-decoded expected value using native equality. The compatibility policy recursively materializes generators, treats tuple and list containers as equivalent sequences, and compares numeric values using relative and absolute tolerances of one ten-billionth. Other values use ordinary equality. These policies are deliberately concrete. Literal equality is not asserted to be a fair semantic oracle for every iterable, while compatibility is not asserted to be the uniquely correct interpretation of the task specification.

Each candidate is executed once per case. The literal comparison is recorded before generator materialization, and the compatible comparison is then applied to the same returned object. Both use copied JSON inputs. This construction prevents different candidate runs from introducing a second source of variation between the two oracle policies. A returned generator remains an object under literal equality, while the compatibility policy evaluates its yielded sequence within the same bounded execution record.

The source is screened through an abstract syntax tree before execution. Only the previously reviewed pure functions and explicitly allowed standard helpers are available. File access, operating-system access, dynamic evaluation, and unreviewed imports are rejected. A limit of one hundred thousand executed source lines applies to each call. The tracing mechanism is recorded against the official Python interface Python Software Foundation, 2026. A budget failure is an operational outcome of the harness, not proof that a program cannot terminate under any execution conditions.

3. From candidate acceptance to a pair relation

For a case under a given oracle policy, the pair label is positive one when the designated corrected candidate passes and the buggy candidate fails. It is negative one when the reverse occurs and zero when their binary acceptance agrees. A zero label combines both-pass and both-fail situations unless the underlying acceptance fields are consulted. We retain those fields to avoid interpreting an unresolved failure tie as a positive endorsement of either candidate.

The first outcome is the corrected candidate's acceptance gain under compatibility rather than literal comparison, averaged within a task. The second is the share of that task's cases whose pair label changes. Each task contributes one row to the main regressions, regardless of its number of JSON cases. We separately report case-weighted totals. This distinction matters because tasks have different case counts, and a change affecting a small task can receive more weight under the equal-task estimand than under a pooled-case average.

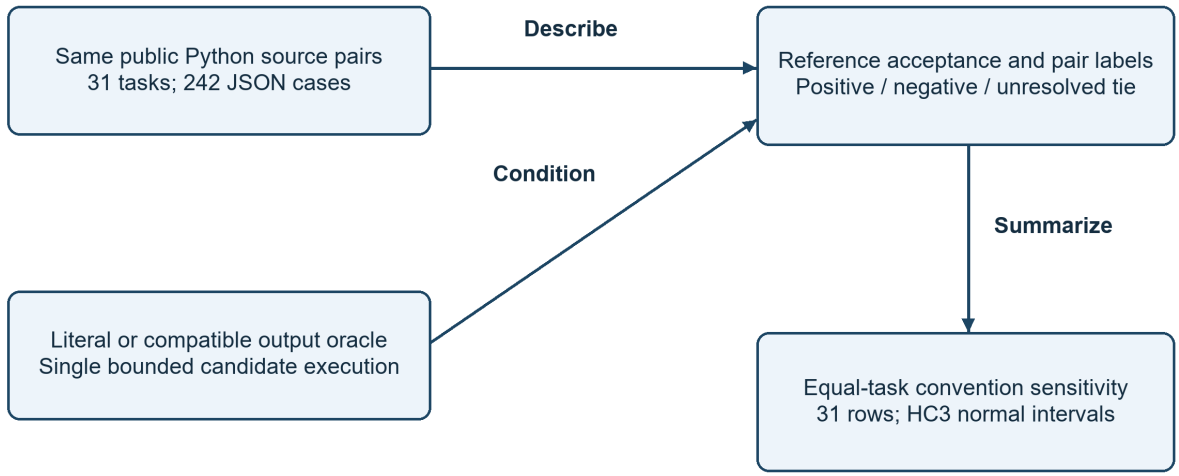

Figure 1. Fixed code executions, two explicit oracles, and auditable pair labels.

Figure 1 holds source programs and test inputs fixed while changing the interpretation of returned values. Its endpoints are reference acceptance and pair-label disagreement. There is no path to fine-tuned model performance because no fine-tuning occurs. Direct preference optimization Rafailov et al., 2023 and instruction learning from human feedback Ouyang et al., 2022 illustrate downstream uses of preference information, but their training objectives do not establish that our mechanically constructed labels represent human judgments. The provenance of the relation remains an upstream requirement.

4. Statistical summaries and their limits

Two intercept-only OLS models summarize the thirty-one task rows. The first estimates the equal-task mean reference-acceptance gain. The second estimates the equal-task mean pair-label-change rate. HC3 standard errors provide conventional heteroskedasticity-adjusted uncertainty, following the robust-covariance tradition of White, 1980. Normal-reference intervals are used. No covariates or fixed effects enter either model, and no case-level independence assumption is invoked to turn the 242 executions into the regression sample size.

These are small finite-corpus summaries, with changes concentrated in three tasks. A symmetric model-based interval can extend below zero even though every observed gain and change rate is nonnegative. Such an endpoint describes the approximation, not a possible negative frequency. The fixed benchmark counts remain directly observed. The general dependence issues discussed by Cameron and Miller, 2015 and structured-validation principles Roberts et al., 2017 caution against treating tasks from one benchmark as a certified random sample of future software work.

The audit does not select a policy by maximizing reported accuracy. Both policies are stated explicitly, including their weaknesses. The model-selection concerns described by Varma and Simon, 2006 nevertheless remain relevant to any later extension: adding conversions only after seeing particular reference failures could turn a compatibility audit into benchmark-specific tuning. Our output therefore retains the actual transformation rules rather than only the final acceptance rate. Any future policy change would require a versioned rerun and an explanation of the new convention.

Testing research also supplies context for the evidence object. EvalPlus Liu et al., 2023 studies evaluation beyond a limited initial test set, while MBPP Austin et al., 2021 and the code-evaluation work associated with Codex Chen et al., 2021 organize other program-generation benchmarks. None of their additional

cases is executed here. The present audit changes how a fixed set of outputs is interpreted, which differs from increasing test coverage or generating more candidate programs. Its scope should not be expanded into a general comparison of those benchmarks.

5. Three tasks account for the observed changes

Across 242 cases, literal comparison accepts the designated corrected implementation in 213 cases, or 0.8801653. Compatibility accepts it in 231 cases, or 0.9545455. The remaining eleven rejections are not erased or relabeled as proof that the benchmark's corrected directory is wrong. They identify unresolved mismatches or operational failures under the declared harness. Repository naming, expected-output semantics, and execution limits need separate investigation before assigning responsibility for any particular rejection.

The changed tasks are flatten, hanoi, and kheapsort. For flatten, corrected-reference acceptance rises from zero of seven to seven of seven, while six case-level pair labels change. For hanoi, acceptance rises from one of eight to eight of eight, with seven label changes. For kheapsort, it rises from zero of four to four of four, with three label changes. The different acceptance and label totals confirm why both candidate outcomes must be preserved when auditing a derived preference.

Table 1. Equal-task sensitivity to the declared oracle convention

Model	Term	Estimate (SE)	95% CI	N
M1	Mean outcome/contrast	0.0927 (0.0527)	[-0.0105, 0.1960]	31
M1: fit	R-squared = 0.0000	-	-	31
M2	Mean outcome/contrast	0.0801 (0.0455)	[-0.0091, 0.1693]	31
M2: fit	R-squared = 0.0000	-	-	31

Model labels: M1 = Corrected-reference acceptance change, equal-program average; M2 = Pairwise oracle-label change, equal-program average.

Table 1 reports intercept-only OLS means across thirty-one tasks, HC3 standard errors, and normal-reference 95% intervals. M1 estimates the task-average corrected-reference acceptance gain; M2 estimates the task-average share of changed pair labels. No controls or fixed effects are included. R-squared is zero for these mean-only models. Intervals are approximate descriptive summaries and do not provide exact bounds for the finite benchmark frequencies.

The equal-task reference gain is 0.0927419, with standard error 0.0526940 and interval from negative 0.0105364 to positive 0.1960203. The equal-task pair-label change rate is 0.0800691, with standard error 0.0455104 and interval from negative 0.0091296 to positive 0.1692678. The corresponding pooled changed-label fraction is sixteen divided by 242, or about 0.0661. These different values arise from weighting, not a disagreement between the execution log and the regression table.

Under literal comparison, 144 cases favor the designated corrected candidate and ninety-eight are ties. Under compatibility, 160 favor it and eighty-two are ties. No observed case favors the buggy candidate under either convention. This does not establish that the resulting positive relations are sufficient training preferences: a binary pass advantage on supplied cases leaves untested behavior, maintainability, requested edit scope, and specification ambiguities unresolved. The relation is an oracle-derived discrimination signal with explicitly limited evidence.

6. What an auditable coding preference should preserve

Real issue-resolution work adds more context than these short functions. SWE-bench Jimenez et al., 2023 and its Lite data release Princeton NLP, 2026 organize repository issues, where a requested change can involve several files and hidden expectations. Mining Git studies Bird et al., 2009 and research on bug-label misclassification Herzig et al., 2013 reinforce the need to preserve how labels were obtained. Code-review research Bacchelli and Bird, 2013 adds another reminder: a useful change is not defined solely by the return value of one standalone function.

Program structure and representation matter at different stages. Cyclomatic complexity McCabe, 1976 describes control-flow properties, while GraphCodeBERT Guo et al., 2020, InCoder Fried et al., 2022, and StarCoder Li et al., 2023 concern learned code representations or generation. None substitutes for an output oracle. A model could represent a program accurately while being trained on an incorrectly interpreted preference label. Conversely, an auditable test-derived relation does not show that a particular representation or learning objective will use it effectively.

The upstream convention also resembles a measurement problem. Measurement-error reasoning Carroll et al., 2006 encourages documenting how an observed variable is constructed rather than treating it as an unmediated truth. Cost-sensitive learning Elkan, 2001 reminds us that different mistakes can carry different consequences, whereas our binary pass rule gives them no such weighting. Scoring-rule theory Gneiting and Raftery, 2007 concerns evaluating specified predictive objects; our pair relation is not a probability forecast or a calibrated preference strength. These distinctions limit which downstream conclusions can be drawn from sixteen changed labels.

A review process could use the audit record to distinguish three kinds of unresolved issue. One concerns representation: a generator and a JSON list can encode the same finite sequence while comparing differently under literal equality. Another concerns numerical tolerance: a small floating discrepancy might or might not be acceptable for the requested function. A third concerns actual output content or operational failure, which container normalization does not repair. Our compatibility policy addresses stated parts of the first two categories, while the execution status and expected-output record preserve evidence needed to inspect the third. It would be inappropriate to collapse these categories into a single claim that one harness is more accurate in every semantic sense.

The source designation also warrants careful language. Calling an implementation corrected identifies its repository path and intended benchmark role. It does not give that implementation authority to redefine every supplied expected value, nor does an incompatible expected value automatically invalidate the implementation. A task-specific specification review would be needed to resolve a genuine conflict. Until then, the relevant outcome is a documented disagreement among code, oracle, and harness convention. Retaining that disagreement makes later preference construction more reviewable than discarding the difficult cases and reporting only the surviving pass rate.

Changes concentrated in three tasks also warrant restraint. Influence diagnostics Cook, 1977 could quantify their leverage on a richer model, but no such model is needed to see the concentration in the saved task table. The contextual guidance of Wasserstein and Lazar, 2016 is preferable to treating the mean-model p-values as a verdict on whether oracle conventions matter. The finite executions already demonstrate that conventions changed labels in identifiable cases. The uncertainty summaries mainly indicate the limitations of extrapolating that frequency beyond this small corpus.

7. Provenance before reuse

Each saved case links a program, source path, expected-output source, raw return type, execution status, line count, acceptance decisions, and pair labels. Dataset documentation Gebru et al., 2021 keeps those links explicit. The production-failure and distributed-tracing topic of Yan, 2026 provides an operational analogue for retaining context around an observed event, although our output-oracle audit measures no distributed system. An isolated pass flag cannot recover the convention that produced it. Technical-debt concerns Sculley et al., 2015 consequently apply to the harness as well as to the candidate code when evaluation labels are reused downstream.

The audit's conclusion is concrete: unchanged code and tests produced different pairwise labels when output conventions changed. Compatibility improved acceptance for three repository-designated corrected implementations and changed sixteen case-level relations, while leaving other failures unresolved. Before such relations are used to teach an assistant what to change, the comparison policy, execution limits, and tie meaning should be reviewable. This study records those ingredients without claiming that execution alone has certified a human preference or a model-training benefit.

References

1. 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.
2. 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.
3. 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.
4. Gebru, Timnit, et al. "Datasheets for datasets" *Communications of the ACM*, 2021. https://doi.org/10.1145/3458723.
5. 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.
6. Princeton NLP. *SWE-bench Lite*. Hugging Face Datasets, public test split, JSON snapshot downloaded 25 Oct. 2026. https://huggingface.co/datasets/princeton-nlp/SWE-bench_Lite.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.

12. 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.

13. 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.

14. Sculley, D., et al. "Hidden Technical Debt in Machine Learning Systems." *Advances in Neural Information Processing Systems*, vol. 28, 2015. https://papers.nips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html

15. Python Software Foundation. (2026). Python 3.12 documentation: sys.settrace. Documentation snapshot captured 7 November 2026. https://docs.python.org/3.12/library/sys.html#sys.settrace.

16. McCabe, T.J.. (1976). A Complexity Measure. IEEE Transactions on Software Engineering. https://doi.org/10.1109/tse.1976.233837.

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

18. 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.

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. Elkan, Charles. “The Foundations of Cost-Sensitive Learning.” *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001, pp. 973-978. https://cseweb.ucsd.edu/~elkan/rescale.pdf.

21. 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.

22. 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.

23. 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.

24. 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.

25. Yan, Biyun. (2026). From Observability to Causality: Detecting Production Failures in Microservices with Distributed Tracing. 2026 9th International Conference on Computer Information Science and Application Technology (CISAT), 1522-1525. https://doi.org/10.1109/CISAT71030.2026.11682113.

26. Ouyang, Long; Wu, Jeff; Jiang, Xu; Almeida, Diogo; Wainwright, Carroll L.; Mishkin, Pamela; Zhang, Chong; Agarwal, Sandhini; Slama, Katarina; Ray, Alex; Schulman, John; Hilton, Jacob; Kelton, Fraser; Miller, Luke; Simens, Maddie; Askell, Amanda; Welinder, Peter; Christiano, Paul; Leike, Jan; Lowe, Ryan. (2022). Training language models to follow instructions with human feedback. arXiv:2203.02155. https://arxiv.org/abs/2203.02155.

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

28. 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.

29. 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.

30. 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.