**Population Variation and Equity**
Volume 1 · Number 1 · 2026

# What Can Three Examples Reject? An Input-Ignoring Probe of MBPP Test Outputs

Jacob Sullivan[1] · Jordan Green[2] · James Harrison[3]

[1] Poland · [2] Hungary · [3] Greece

Original publication: November 3, 2026 | Article 601

## Abstract

Most captured MBPP test triples reject a program that always returns their first expected output. Among 965 eligible tasks, only seven such programs agreed with all three tests. Across the two other cases, agreement was 167/1,930, or 8.65%. Boolean-output tasks showed more other-case agreement than several larger-output classes, but that descriptive pattern does not establish a defect in those tasks. The probe exposes a specific limitation of observed test evidence: agreement with a small suite does not reveal how a program uses its inputs. It does not prove that every input-ignoring program is semantically wrong.

*Executed local stress test on literal assertions. A constant program deliberately receives the first test's expected output; no language model or benchmark reference program is executed.*

# What Can Three Examples Reject? An Input-Ignoring Probe of MBPP Test Outputs

> Executed local stress test on literal assertions. A constant program deliberately receives the first test's expected output; no language model or benchmark reference program is executed.


## Abstract

Most captured MBPP test triples reject a program that always returns their first expected output. Among 965 eligible tasks, only seven such programs agreed with all three tests. Across the two other cases, agreement was 167/1,930, or 8.65%. Boolean-output tasks showed more other-case agreement than several larger-output classes, but that descriptive pattern does not establish a defect in those tasks. The probe exposes a specific limitation of observed test evidence: agreement with a small suite does not reveal how a program uses its inputs. It does not prove that every input-ignoring program is semantically wrong.


## 1. Correct examples and unexamined behavior

A program can agree with several examples while using less information than the task description appears to require. The simplest possibility is to ignore all arguments and return a constant. If the supplied examples share that output, the test suite cannot distinguish the constant from a program that computes the intended function on those examples. The practical question is how frequently this particular ambiguity occurs in a captured public benchmark.

We ask: how often do the three ordinary MBPP tests agree with a program that ignores every input and returns the first test's expected output? This is a deliberately privileged probe. The program is handed one answer rather than being asked to infer an algorithm from a natural-language instruction. That privilege makes the information boundary explicit and creates a reproducible stress condition; it does not represent an ordinary coding assistant's performance.

Tao and Kamata, 2026 investigate learning what to change through pairwise preferences for coding assistants. Test-derived evidence can inform judgments about a candidate change, but its meaning must be specified before treating agreement as a preference label. Our analysis examines the evidence supplied by a small example set. It does not fine-tune a model, compare generated patches, or claim that a human would prefer the constructed constant program.

Importantly, a constant return is not inherently incorrect. Some task domains legitimately require a constant, and examples alone cannot adjudicate the intended function on all admissible inputs. We therefore use complete-suite agreement as the outcome, avoiding labels such as false acceptance or undetected bug. Establishing those stronger claims would require semantic review of each task and a justified counterexample or specification.

## 2. Why output diversity matters separately from input diversity

The source benchmark is MBPP, introduced in work on program synthesis with large language models (Austin et al., 2021). Its ordinary tests provide an accessible set of input-output judgments. Three syntactically different calls can nevertheless have the same expected result. Conversely, similar-looking inputs can require different outputs. Input diversity and output diversity therefore address related but distinct questions about what a finite suite can reject.

Barr et al., 2015 discusses the broader difficulty of judging whether program behavior is correct. In the present setting, the oracle is limited to literal expected values and Python equality. We do not execute the

original task solution to obtain additional outputs. This keeps the probe attached to information already present in the tests, while leaving the completeness and semantic correctness of that information unresolved.

Classical test-data work considers how detecting simple errors can relate to detecting more complicated errors (DeMillo et al., 1978). Our probe is narrower than a mutation-testing study: it does not systematically mutate a correct reference implementation, and it does not claim every constructed program is a faulty mutant. It asks whether one specific input-insensitive behavior is distinguished by the available expected outputs.

Broader code benchmarks and expanded tests make the evidence issue consequential. Code-model evaluation uses execution-based checks to judge generated programs (Chen et al., 2021), while EvalPlus expands testing beyond small original suites (Liu et al., 2023). LiveCodeBench introduces additional concerns about code-evaluation reliability and contamination (Jain et al., 2024). We cite these evaluation settings to motivate transparent test semantics, without importing their reported model performance into the present calculation.

Program complexity is a further independent dimension. Control-flow measures such as those developed by McCabe, 1976 describe source structure; they cannot establish whether a small set of outputs distinguishes an input-ignoring alternative. A complex reference function may still have three equal expected outputs, and a short function may be strongly constrained by different outputs. The present probe therefore measures observed agreement directly rather than using reference complexity as a substitute.

## 3. A strict, inspectable eligibility rule

The captured MBPP file contains 974 tasks. We examine only each task's three ordinary test_list entries, leaving challenge tests outside the experiment. Every eligible entry must parse as one assert statement with one equality comparison. Its left side must be a direct call to a named function, and every argument, keyword value, and expected output must be a supported finite Python literal.

All three calls must name the same function. Expanded keyword dictionaries, arbitrary expressions, external names, nested function calls, and unsupported values are not evaluated to make a case fit. The rule retains 965 tasks and excludes nine. Exclusion reasons and task identifiers are saved before reporting the probe's outcome, making the analysis population reproducible without describing excluded syntax as a failed constant prediction.

Python's abstract-syntax-tree facilities support this restricted parsing and literal evaluation (Python Software Foundation, 2026). Literal evaluation is different from executing a task's reference code or its setup script. The local procedure reads the necessary syntax and values, constructs its own simple function, and never imports or runs benchmark solution code. The measured candidate behavior is therefore known and inspectable by construction.

We allow finite integers and floats, strings, booleans, None, and supported nested list, tuple, set, or dictionary structures. The output comparison uses ordinary Python equality, with no extra numeric tolerance or automatic conversion between container types. Boolean and integer comparisons consequently follow Python's own equality behavior. Those representation conventions are part of the measurement definition, not an implicit universal specification of semantic equivalence (Carroll et al., 2006).

Of the 965 retained tasks, 960 contain three different call syntax trees, four contain two, and one repeats the same call syntax throughout. This is a syntactic count rather than proof of distinct semantic inputs:

different literal syntax can represent equal values. We do not exclude tasks for repeated calls after inspecting outcomes. Keeping them preserves the declared eligibility cohort and exposes a separate suite characteristic.

## 4. Executing the privileged constant probe

For each eligible task, the first expected output becomes the constant. The local candidate accepts arbitrary positional and keyword arguments, ignores them, and returns a deep copy of that value. Deep copying prevents a returned mutable object from being accidentally shared across calls. The candidate is actually invoked on the three parsed argument sets, and each returned value is compared with that case's expected value.

The first case passes by construction and is checked as such. The two remaining cases provide the informative stress outcomes. Their expected values are not used to select a new constant or revise the candidate. No attempt is made to search over other constants, choose a favorable first example, inspect the task description for hints, or generate a program with a language model.

Two task-level quantities summarize the result. Complete-suite agreement equals one when all three calls match their expected outputs. Other-case agreement is the fraction of the two remaining calls that match, taking values zero, one half, or one. The first measures survival of the whole ordinary suite under this probe; the second shows how much of the additional example evidence still agrees after one answer has been supplied.

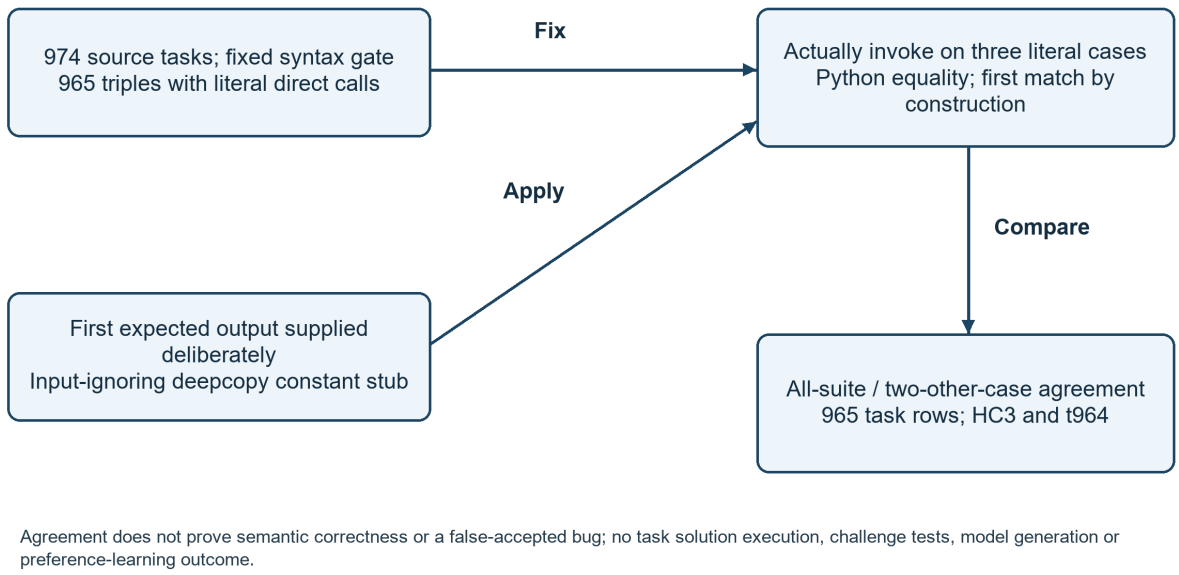



***Figure 1.*** *Literal-assert eligibility defines the cohort; the first expected output fixes an input-ignoring candidate; its actual calls yield two separately reported agreement outcomes. The first example is a declared information privilege.*

Each outcome is regressed on an intercept across 965 task rows. We use HC3 standard errors and t964 confidence intervals, without controls or task-family fixed effects. Because every task supplies two other cases, equal-task other-case agreement also equals the fraction of the 1,930 other cases that agree. This equality would change if tasks contributed different numbers of evaluated cases.

Robust covariance supplies a working summary of between-task variation (White, 1980). Tasks may share templates or authorship histories, however, and no independent task-family cluster labels have been established. Dependence concerns remain relevant even with many rows (Cameron and Miller, 2015). The

reported intervals therefore do not certify population coverage for all coding tasks, and the low event count in complete-suite agreement warrants particular caution about the linear approximation.

***Table 1.*** *Intercept-only agreement regressions, N=965 tasks in each model. Standard errors are HC3 and intervals use t964. Complete-suite agreement is a binary proportion; other-case agreement is the per-task fraction of two additional examples. Fit is intercept-only R-squared, zero up to floating-point precision.*

| Model | Term | Estimate (SE) | 95% CI | N |
|---|---|---|---|---|
| M1 | Mean outcome/contrast | 0.0073 (0.0027) | [0.0019, 0.0126] | 965 |
| M1: fit | R-squared = 0.0000 | - | - | 965 |
| M2 | Mean outcome/contrast | 0.0865 (0.0064) | [0.0740, 0.0991] | 965 |
| M2: fit | R-squared = 0.0000 | - | - | 965 |

Model labels: M1 = All-three agreement of first-test-output constant; M2 = Agreement across two other cases, equal task weight.

## 5. Strong rejection of this probe, with seven surviving triples

Seven tasks agree with the constant on all three ordinary tests. The estimated proportion is 0.007254, with standard error 0.002735 and 95% interval [0.001887, 0.012620]. Thus 958 of the retained tasks reject this specific probe on at least one additional example. The dominant finding is that three captured tests usually distinguish the chosen constant; presenting only the seven survivors would misstate the overall result.

Across the two other cases, 167 of 1,930 comparisons agree, giving 0.086528. The task-level estimate has standard error 0.006397 and interval [0.073975, 0.099082]. A program can agree with one other example while failing the remaining example, so this 8.65% figure is not the proportion passing the complete suite. The denominators and decision rules must stay separate.

Descriptive output classes show heterogeneity. There are 117 tasks whose first expected value is boolean; their other-case agreement is 0.346154. The corresponding proportions are 0.055838 among 394 integer-first tasks and 0.138211 among 123 string-first tasks. These groups describe the type of the first expected value, not necessarily the types of every output in a task, and they are not balanced experimental treatments.

Only one boolean-first task agrees across all three cases. Three integer-first tasks, one list-first task, one string-first task, and one tuple-first task account for the other six survivors. Higher agreement on individual boolean cases therefore does not imply that boolean tasks generally pass the entire constant probe. Pooling those two outcomes into a vague statement about weak tests would lose the main distinction in the evidence.

The first-value type is also not a causal explanation. Boolean outputs have a small set of possible values, but task construction, selected cases, and intended semantics can all affect the observed agreement. We do not fit a post hoc search over type categories or infer that changing a function's output type would alter its test quality. The subgroup descriptions identify questions for further review, not a validated mechanism.

## 6. From test agreement to judgments about code

The meaning of an agreement measure depends on its scale and object (Stevens, 1946). It is useful to distinguish agreement from association or predictive fit, as emphasized in method-comparison work (Bland and Altman, 1986; Lin, 1989). Here the object is literal output equality under supplied examples. Neither the complete-suite indicator nor its confidence interval measures the fraction of admissible unseen inputs on which a program computes the intended function.

Misclassified software labels can alter empirical conclusions (Herzig et al., 2013). Calling all seven surviving probes bugs would introduce exactly that kind of unsupported label. Some may be easily refuted through specification review, but that review has not been completed. The saved task identifiers support such follow-up without turning an unperformed semantic audit into a reported finding.

Repository-scale issue resolution, as in SWE-bench, presents broader context and integration behavior than these small standalone examples (Jimenez et al., 2023). QuixBugs provides another explicit program-and-test setting (Lin et al., 2017). Their differing experimental objects caution against applying the 0.73% figure to repository repair, complex APIs, or all code benchmarks. No program in those resources is executed as part of this probe.

Preference optimization and feedback-based training require an additional connection between judgments and learning outcomes (Rafailov et al., 2023; Ouyang et al., 2022). We measure neither that connection nor a coding assistant's tendency to produce constant functions. Our candidate deliberately receives an expected answer, so its success rate cannot be called model pass@k, an estimate of contamination prevalence, or evidence that fine-tuning has improved or worsened code quality.

Keeping that privilege visible is an information-provenance requirement (Kapoor and Narayanan, 2023). Empirical software-research guidance supports declaring sampling and analysis choices (Kitchenham et al., 2002), while validation principles require matching the evaluation structure to the intended claim (Roberts et al., 2017; Varma and Simon, 2006). We accordingly avoid selecting the first test differently for favorable tasks, replacing failed cases, or broadening eligibility only when doing so increases agreement.

## 7. What a follow-up would need to establish

A stronger semantic investigation would review each surviving task's description, admissible input domain, and reference behavior before constructing independent counterexamples. That process could distinguish a genuinely permissible constant from one that merely survives the supplied examples. It would be a new study with its own case-selection and judgment record, rather than a reinterpretation of the seven observed agreements as already confirmed defects.

Further tests also have costs. Cost-sensitive reasoning distinguishes a resource objective from a classification objective (Elkan, 2001), and the present study measures neither test-generation effort nor runtime savings. Influence analysis matters if a future study uses a small selected set of survivors to estimate a general rate (Cook, 1977). Statistical interpretation should preserve these design limits instead of treating an interval excluding zero as a broad adequacy judgment (Wasserstein and Lazar, 2016).

Saved eligibility decisions, agreement indicators, output types, and model estimates make the probe inspectable (Gebru et al., 2021). A context-preservation issue also motivates the distributed-tracing production-diagnosis topic of Yan, 2026. In this experiment, the relevant context is the declared output convention and the privileged first-output construction, not a chain of service calls. Dependencies in a larger software pipeline could add further ambiguity (Sculley et al., 2015). Most triples reject the constant

probe, seven agree completely, and those seven outcomes still require semantic evidence before being called incorrectly accepted programs.

## References


1. 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.
2. 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.
3. Stevens, S. S.. (1946). On the Theory of Scales of Measurement. Science. https://doi.org/10.1126/science.103.2684.677.
4. Cook, R. Dennis. (1977). Detection of Influential Observation in Linear Regression. Technometrics. https://doi.org/10.1080/00401706.1977.10489493.
5. 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.
6. 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.
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. Carroll, Raymond J., David Ruppert, Leonard A. Stefanski, and Ciprian M. Crainiceanu. “Measurement Error in Nonlinear Models.” 2006. https://doi.org/10.1201/9781420010138.
9. 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
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. Lin, Lawrence I-Kuei. (1989). A Concordance Correlation Coefficient to Evaluate Reproducibility. Biometrics. https://doi.org/10.2307/2532051.
16. 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.

17. 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.

18. 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.

19. 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.

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

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. DeMillo, R.A.; Lipton, R.J.; Sayward, F.G.. (1978). Hints on Test Data Selection: Help for the Practicing Programmer. Computer. https://doi.org/10.1109/c-m.1978.218136.

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

28. Python Software Foundation. (2026). ast: Abstract Syntax Trees. Python3.12 documentation, captured7November2026; capture year is not the language release year. https://docs.python.org/3.12/library/ast.html.

29. 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.

30. Kapoor, Sayash; Narayanan, Arvind. (2023). Leakage and the reproducibility crisis in machine-learning-based science. Patterns. https://doi.org/10.1016/j.patter.2023.100804.