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

Predicting a Captured Coding-Agent Outcome across Repositories from Issue Descriptions

Jordan Wright[1] · Luke Hayes[2] · Benjamin Lewis[3]

[1] Spain · [2] Portugal · [3] Czechia

Original publication: November 1, 2026 | Article 650

Abstract

Repository transfer is a different test from pooling coding tasks. We joined 299 observed SWE-bench Lite identifiers to a captured SWE-agent result release and predicted each repository using models fitted on the others. Issue length and code-fence presence slightly improved Brier loss over the remaining-repository success rate: the equal-repository difference was −0.00350, with a wide working interval spanning zero. Individual repositories moved in both directions. The exercise describes one historical configuration and shows why a pooled result rate gives limited information about a particular omitted repository.

Internal leave-one-repository-out analysis of published result identifiers. No code patches are generated, and the recorded agent configuration is not re-evaluated.

Predicting a Captured Coding-Agent Outcome across Repositories from Issue Descriptions

> Internal leave-one-repository-out analysis of published result identifiers. No code patches are generated, and the recorded agent configuration is not re-evaluated.

Abstract

Repository transfer is a different test from pooling coding tasks. We joined 299 observed SWE-bench Lite identifiers to a captured SWE-agent result release and predicted each repository using models fitted on the others. Issue length and code-fence presence slightly improved Brier loss over the remaining-repository success rate: the equal-repository difference was -0.00350, with a wide working interval spanning zero. Individual repositories moved in both directions. The exercise describes one historical configuration and shows why a pooled result rate gives limited information about a particular omitted repository.

1. A benchmark total does not describe every repository

Software repositories vary in their conventions, dependencies, testing infrastructure, and issue-writing practices. A task collection that contains many examples from one repository may consequently give a pooled summary a different meaning from the average outcome in a typical represented repository. This distinction matters when readers interpret a coding-assistant benchmark as evidence about a new project. The benchmark may supply strong information about its task mix while supplying much less about transport to another repository.

Our research question is whether a simple task-description proxy improves prediction of a captured coding-agent result when the entire repository is omitted from fitting. The proxy uses issue-description length and the presence of a Markdown code fence. These inexpensive features describe the visible problem statement without reading a gold solution. They may correlate with the way tasks are presented, but they are not measures of semantic difficulty, repair quality, or the quality of an assistant's reasoning.

The question is adjacent to preference-based improvement of coding assistants. Tao and Kamata, 2026 study teaching assistants what to change through fine-tuning from pairwise preferences. Here the dependent variable is membership in a published resolved-task list. We do not use preference pairs, run its fine-tuning procedure, or test whether either issue feature improves a trained assistant. The connection is that learning and evaluating what to change require clarity about which outcome labels travel across development contexts.

We examine two propositions. First, a metadata model may improve a proper probabilistic score relative to the success rate estimated from other repositories. Second, its average calibration can conceal opposite errors in individual omitted repositories. The analysis treats these as exploratory questions, not as a contest among many candidate feature sets. The two predictors, leave-out unit, constant comparator, and Brier loss are fixed for the reported calculation.

2. The outcome belongs to a particular evaluation pipeline

SWE-bench Jimenez et al., 2023 places repository-level software tasks into an executable evaluation setting. The Lite snapshot documentation Princeton NLP, 2026 identifies the task records joined here. SWE-agent Yang et al., 2024 supplies a particular agent-computer interface and the captured release of evaluation identifiers. Its result labels are products of that configuration, environment, and evaluation procedure. They cannot be assumed to be timeless attributes of the tasks or outcomes for every current coding assistant.

Program-synthesis benchmarks such as the Codex evaluation Chen et al., 2021 and MBPP Austin et al., 2021 organize other coding settings. EvalPlus Liu et al., 2023 emphasizes the role of stronger testing, while LiveCodeBench Jain et al., 2024 addresses additional benchmark-construction concerns. These works motivate careful interpretation of a reported success label. They do not make the different task populations exchangeable, and their outcome values are not combined with the present repository-task records.

Agentless repair Xia et al., 2024 also provides a distinct system design rather than a substitute label for the captured agent. Our study is therefore a metadata baseline around one existing result file. It is not a new repair method or a current leaderboard comparison. A low predicted success probability could reflect a task presentation associated with the recorded configuration, rather than a stable reason another system would fail.

Empirical software-engineering guidance Kitchenham et al., 2002 motivates stating the unit, context, and evidence source. Repository mining cautions Bird et al., 2009 and bug-label research Herzig et al., 2013 further explain why recorded statuses require interpretation. An observed task outside a resolved list is coded zero for this published outcome, but that zero does not prove the absence of a valid repair. It may include failure modes elsewhere in the execution pipeline.

3. Joining records without turning absence into failure

The public Lite metadata contain 300 unique tasks. The captured result file has several identifier lists, including generated, no-generation, applied, and resolved statuses. We define an observed evaluation identifier as one appearing in the union of these lists. This union contains 299 identifiers, of which 54 occur in the resolved list. The lists are not assumed to be disjoint stages with counts that can simply be added.

One Lite task, sympy__sympy-14817, does not appear in the observed union and is excluded. We do not assign it an unresolved result because absence from the release is not an observed failure. All other matched tasks receive the binary outcome defined by resolved-list membership. The processed table retains the task identifier, repository, two predictors, and matched label, allowing the join and this distinction to be inspected directly.

There are twelve represented repositories. Django contributes 114 of the 299 tasks, or 38.13 percent; SymPy contributes 76. Several repositories contribute only a handful. The task-weighted resolved rate is 54/299, or 18.06 percent. Giving each repository equal weight instead yields 15.24 percent. These two summaries are both correct for their respective denominators, and their difference alone is not evidence that one weighting scheme is biased.

Issue length is the natural logarithm of one plus the count of word-like sequences in the problem statement. The fence indicator equals one when the statement includes three consecutive backticks. Neither feature uses a proposed patch, gold production edits, gold test edits, a repository identifier, or the

hints field. In particular, we avoid assuming that retrospective discussion comments were available when an issue first opened.

A word count is not a model-token count. Subword tokenization Sennrich et al., 2015 motivates keeping that distinction explicit rather than calling the feature context length or inference cost. The fence indicator likewise does not establish the presence of a runnable reproduction script. It can identify documentation formatting as well as substantive code. Both are deliberately coarse descriptors whose empirical usefulness must be assessed rather than presumed.

4. Leave out the repository, then score its tasks

For each repository in turn, we fit a binomial logistic regression on all tasks from the other eleven. The linear predictor contains an intercept, logged issue-word count, and the code-fence indicator. We then obtain probabilities for every task in the omitted repository. Every task is predicted exactly once by a model that did not use outcomes from its own repository. All twelve fitted logistic models converge.

The comparator is the resolved proportion among the same fitting tasks. It supplies one probability to every task in that omitted repository. This is a demanding but transparent comparator for a small metadata model: the proxy must improve over what can already be learned from the other repositories' base rate. No threshold converts probabilities into hard success predictions. Cost-sensitive classification arguments Elkan, 2001 explain why such a threshold would require a separate decision objective.

We use Brier loss, the squared difference between a predicted probability and the binary outcome. Proper scoring rules Gneiting and Raftery, 2007 support evaluating probabilities directly. Calibration research Guo et al., 2017 distinguishes this score from simply comparing the average predicted probability with the observed success fraction. ROC analysis Fawcett, 2006 and precision-recall evaluation Saito and Rehmsmeier, 2015 address ordering and threshold behavior, but neither is substituted for the declared primary loss after observing the results.

Within each repository, we average the proxy's loss minus the comparator's loss. A negative value favors the proxy. We also calculate signed calibration error: mean proxy probability minus the observed resolved fraction. The primary regressions use these twelve repository summaries as observations. This gives a small repository the same primary weight as a large one. Task-weighted results are additionally reported to show how the alternative population changes the summary.

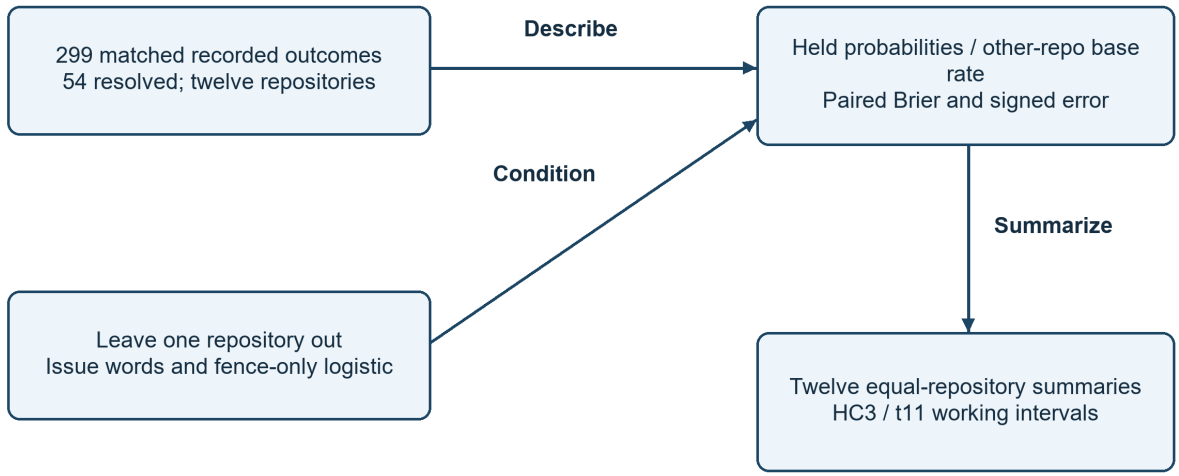


Figure 1. *Each omitted repository receives probabilities from a metadata model and a constant fitted only on the other repositories. Paired task losses are averaged before equal-repository description.*

5. Working uncertainty and dependence

Each reported mean is estimated by intercept-only OLS with HC3 standard errors White, 1980 and t intervals using eleven degrees of freedom. The regression R-squared is effectively zero because the fitted value is a mean rather than a predictive model of repository differences. These models summarize held-out outputs; they are separate from the twelve logistic fitting models that produce task probabilities.

There is an important dependence limitation. The fitting sets overlap heavily across leave-out folds, so the twelve repository loss means are not guaranteed independent. Clustered-inference guidance Cameron and Miller, 2015 makes clear that naming a grouping level does not remove all shared estimation uncertainty. We therefore describe the intervals as limited working summaries. They do not propagate all uncertainty from overlapping training, and they are not honest probability bounds for a future independently sampled repository.

Structured validation principles Roberts et al., 2017 and distribution-shift analysis Ovadia et al., 2019 support leaving out the intended transport unit. That design is more aligned with the question than a random task split, but it does not create an external population of projects. Matching and population-composition reasoning Stuart, 2010 are relevant to the remaining gap: the twelve included repositories are a selected public benchmark cohort, not a representative sample of commercial or scientific software.

Feature selection can also distort a small evaluation. Nested-validation cautions Varma and Simon, 2006 motivate keeping the two-feature model fixed. We do not search many text features and retain the best leave-out score. Measurement-error analysis Carroll et al., 2006 further limits interpretation of logged word count and fences as proxies. Their coefficients could reflect multiple presentation practices, so no causal mechanism is inferred from their predictive contribution.

6. Results across the twelve omitted repositories

The equal-repository mean Brier loss is 0.128151 for the metadata proxy and 0.131655 for the constant. Their paired difference is -0.003505, with standard error 0.003184 and a 95% working interval of [-0.010512, 0.003502]. The proxy has a modest descriptive advantage on this weighting, but the interval does not resolve a directional generalization claim under even the limited working assumptions.

Task weighting yields a larger improvement: proxy loss is 0.146641 compared with 0.153493 for the constant. This does not contradict the equal-repository comparison. It gives greater weight to repositories contributing more tasks, including Django, where the proxy's Brier loss is 0.199649 versus 0.211709. The result illustrates why the analysis population and weighting rule should accompany a benchmark improvement.

Table 1. Equal-repository descriptions of held-out probability performance.

Model	Term	Estimate (SE)	95% CI	N
M1	Mean outcome/contrast	-0.0035 (0.0032)	[-0.0105, 0.0035]	12
M1: fit	R-squared = 0.0000	-	-	12
M2	Mean outcome/contrast	-0.0037 (0.0286)	[-0.0667, 0.0593]	12
M2: fit	R-squared = 0.0000	-	-	12

Model labels: M1 = Equal-repository held-out Brier difference; M2 = Equal-repository held-out signed calibration error.

Table 1 contains twelve observations per model, no controls, HC3 standard errors, and t-based working intervals with eleven residual degrees of freedom. Model 1 measures proxy-minus-constant Brier loss. Model 2 measures signed calibration error. The mean signed error is -0.003702, with interval [-0.066719, 0.059314]. That small average does not establish accurate probabilities for each repository or each individual task.

Repository outcomes show why cancellation matters. Requests has two resolved tasks out of six, and the proxy underpredicts its mean by about 0.1416. Flask has zero resolved tasks out of three, while the proxy overpredicts by about 0.1476. Xarray also has no recorded resolution among five tasks, with mean overprediction around 0.1514. These tiny denominators make strong repository-specific conclusions inappropriate, but they reveal information hidden by an average signed error near zero.

The proxy does not improve every repository's Brier score. Pylint, Pytest, scikit-learn, and Seaborn have higher proxy losses than their constant comparators. For Pylint the values are 0.154031 and 0.139091, respectively. These failures remain in the primary result. Removing them because they oppose the aggregate improvement would change the question from transport performance to performance on a favorable subset.

7. What the baseline contributes

Dataset-level comparisons Demšar, 2006 motivate examining the spread across evaluation units rather than treating a pooled score as exhaustive. Here, however, twelve related repositories and overlapping folds do not support a broad ranking of learning methods. The purpose is to expose the transport question and establish a transparent reference calculation. A future method should be compared on independent releases and larger repository samples before a small improvement is treated as stable.

Influence analysis Cook, 1977 is also relevant because equal-repository weighting increases the leverage of very small projects. That weighting answers a legitimate question about an average included repository,

but it can be noisy. We retain the task-weighted score beside it and avoid selecting whichever produces the more favorable narrative. The ASA statement Wasserstein and Lazar, 2016 reinforces interpreting effect size, uncertainty, and design together rather than using one probability threshold as a quality certificate.

The reproducibility record preserves exact identifier matching, the omitted task, fold definitions, fitted probabilities, and repository summaries. Yan, 2026 addresses production-failure detection with distributed tracing, a different observational setting in which the context attached to a recorded event is also central. A benchmark resolution flag here is a captured agent outcome, not a trace-derived diagnosis or a stable property of a repository. Keeping it bound to configuration and capture time responds to technical-debt concerns Sculley et al., 2015. Subsequent use requires human assessment of whether those labels actually represent the decision being proposed.

Low Brier loss also needs its class-frequency context. A repository with no recorded resolutions can give a low score to conservative probabilities even when the model has learned little about the distinctions between its tasks. This is one reason we compare against a fitting-sample constant rather than presenting the proxy's absolute loss alone. A future evaluation should include enough successes and failures within each repository to examine discrimination and calibration separately, while retaining the proper probability score as its declared objective.

No explanation of individual failures follows from the two-feature baseline. A long problem statement could contain an unusually clear reproduction, extensive irrelevant logs, or a difficult specification. The same word count can arise through any of these processes. Establishing a mechanism would require examination of task content and the agent's execution trace, neither of which is inferred from the sign of a logistic coefficient here.

8. Conclusion

Issue length and code-fence presence offer a small average Brier improvement over other-repository base rates for one captured coding-agent release. The improvement varies across repositories, and a near-zero mean calibration error conceals substantial opposing errors in small groups. The evidence supports explicit repository-held-out reporting, while leaving current coding-assistant performance, preference learning benefits, and generalization to new projects unresolved.

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. Cook, R. Dennis. (1977). Detection of Influential Observation in Linear Regression. Technometrics. https://doi.org/10.1080/00401706.1977.10489493.
3. 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.
4. 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.
5. 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.

6. 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.

7. 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

8. 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.

9. 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.

10. 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.

11. 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.

12. Stuart, Elizabeth A. "Matching Methods for Causal Inference: A Review and a Look Forward." *Statistical Science*, vol. 25, no. 1, 2010, pp. 1-21. https://doi.org/10.1214/09-sts313.

13. Demšar, Janez. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research. https://www.jmlr.org/papers/v7/demsar06a.html.

14. Ovadia, Yaniv, et al. “Can You Trust Your Model’s Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift.” *Advances in Neural Information Processing Systems*, vol. 32, 2019. https://arxiv.org/abs/1906.02530.

15. 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.

16. 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.

17. 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.

18. Sennrich, Rico; Haddow, Barry; Birch, Alexandra. (2015). Neural Machine Translation of Rare Words with Subword Units. https://arxiv.org/abs/1508.07909.

19. 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.

20. 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.

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

22. 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.

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. 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.

25. 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.

26. 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.

27. 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.

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. 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.

30. 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.