Benchmarking and Comparative Evaluation
Volume 1 · Number 1 · 2026

Code Blocks Do Not Identify Broad Fixes: Issue Formatting and Gold-Patch Scope in SWE-bench Lite

Nicholas Wilson[1] · Spencer Coleman[2] · Jason Reed[3]

[1] Denmark · [2] Portugal · [3] Canada

Original publication: October 6, 2026 | Article 566

Abstract

A fenced example in a bug report might clarify a small repair or expose a complicated failure. Across 300 SWE-bench Lite tasks, its presence did not distinguish the scope of the supplied reference patch after adjustment for repository, description length, and available hints. The coefficient for fenced text was -0.247 for log-transformed changed lines, with a 95 percent interval from -0.657 to 0.162; the hunk-count analysis was similarly inconclusive. All reference patches in this snapshot affected one source file. Formatting therefore supplies an observable benchmark feature, but these records do not justify using it as a shortcut for edit scope or assistant difficulty.

Exploratory research draft based on benchmark records. No coding assistant was trained or evaluated in this study.

Code Blocks Do Not Identify Broad Fixes: Issue Formatting and Gold-Patch Scope in SWE-bench Lite

Exploratory research draft based on benchmark records. No coding assistant was trained or evaluated in this study.

Abstract

A fenced example in a bug report might clarify a small repair or expose a complicated failure. Across 300 SWE-bench Lite tasks, its presence did not distinguish the scope of the supplied reference patch after adjustment for repository, description length, and available hints. The coefficient for fenced text was -0.247 for log-transformed changed lines, with a 95 percent interval from -0.657 to 0.162; the hunk-count analysis was similarly inconclusive. All reference patches in this snapshot affected one source file. Formatting therefore supplies an observable benchmark feature, but these records do not justify using it as a shortcut for edit scope or assistant difficulty.

1. The difference between showing code and authorizing an edit

Software issue descriptions routinely mix explanations with snippets, reproduction examples, and expected outputs. A markdown code block gives these materials a visible boundary, yet says little by itself about which repository content should change. A short fenced example may demonstrate a defect in one expression. A much larger unformatted explanation may describe an interaction across several functions. Treating a presentation convention as a measure of repair scope risks confusing how a reporter communicates a problem with the engineering work eventually required to resolve it.

This distinction is central to Teaching Coding Assistants What to Change: Fine-Tuning from Pairwise Preferences (Tao and Kamata, 2026). That study examines assistants' separation of modifiable target code from useful supporting material and investigates training with preferred and dispreferred edits. We ask a narrower observational question about the input side of such tasks: does the occurrence of fenced text in a real issue description correspond to a larger reference edit? Our records contain human-associated benchmark patches, rather than outputs from the preference-trained system. The analysis therefore complements the target paper's task-boundary concern without testing its training intervention.

The empirical setting is SWE-bench Lite, a public subset of repository-level issue-resolution tasks. The broader SWE-bench design connects natural-language issues with repository states, reference changes, and tests (Jimenez et al., 2023). This structure permits an audit of issue and patch characteristics without invoking a language model. It also constrains interpretation: a benchmark's selected reference patch is one recorded resolution, and need not represent every acceptable implementation. The relationship studied here is between observable features of that record, rather than between formatting and the intrinsic complexity of all possible fixes.

2. Why the benchmark unit matters

Repository editing differs from generating a self-contained function. Evaluating Large Language Models Trained on Code and Program Synthesis with Large Language Models establish influential settings for assessing generated programs against task requirements (Chen et al., 2021; Austin et al., 2021). A repair task additionally supplies an existing codebase and an issue narrative whose materials have different roles. Consequently, counts of changed lines or diff hunks describe a repair artifact; they are not interchangeable with functional correctness, natural-language specification quality, or the probability that an assistant will solve the issue.

Interface and workflow research makes the distinction especially relevant. SWE-agent investigates interfaces through which an agent navigates and modifies a repository, while Agentless separates stages of localization and repair within a comparatively simple workflow (Yang et al., 2024; Xia et al., 2024). Both perspectives make localization consequential, but neither implies that markdown fencing identifies the eventual change boundary. Our study isolates a cheaply observable presentation feature before using it in task stratification, training-data selection, or an interface that recommends which examples require additional review.

Code representations provide further reasons to avoid equating visible delimiters with structural information. GraphCodeBERT incorporates data-flow information, whereas InCoder supports infilling and code synthesis (Guo et al., 2020; Fried et al., 2022). Such modeling objectives concern program content and context beyond a single punctuation convention. The present audit extracts no syntax tree or data-flow graph. That deliberate simplification lets us evaluate a transparent feature, while acknowledging the substantial semantic information it omits. A weak relationship would limit the feature's usefulness without demonstrating that structural representations are unhelpful.

Learning objectives and evaluation protocols also answer different questions. CodeRL combines code generation with reinforcement learning, and DeepSeek-Coder develops large-scale programming models (Le et al., 2022; Guo et al., 2024). Their objectives cannot be evaluated by observing the length of an existing gold patch. Likewise, a successful repair may be short or long. We retain changed-line and hunk outcomes as measures of recorded edit scope, explicitly avoiding a quality ranking in which fewer lines automatically signify a better solution.

3. Research question and proposed relationships

The primary research question is whether issues containing at least one triple-backtick delimiter have reference patches with different changed-line counts. We state an exploratory proposition, P1, that fencing is associated with log-transformed edit scope conditional on repository, issue length, and available hints. We do not specify a positive sign: examples can reveal either a local failure or a more extensive interaction. P2 examines hunk count as another dimension of the same recorded repair, using a count model rather than treating it as a continuous normally distributed measurement.

Description length is a necessary adjustment because fenced snippets can themselves make a report longer. Hints are another source of supporting information, although their presence may reflect benchmark construction and issue history. Repository indicators account for average differences among the represented projects. These adjustments make the comparison more explicit, but do not remove unobserved reporter expertise, defect type, codebase architecture, or selection into the Lite subset. The propositions describe conditional associations within an assembled benchmark, with no causal claim that adding a delimiter changes the required repair.

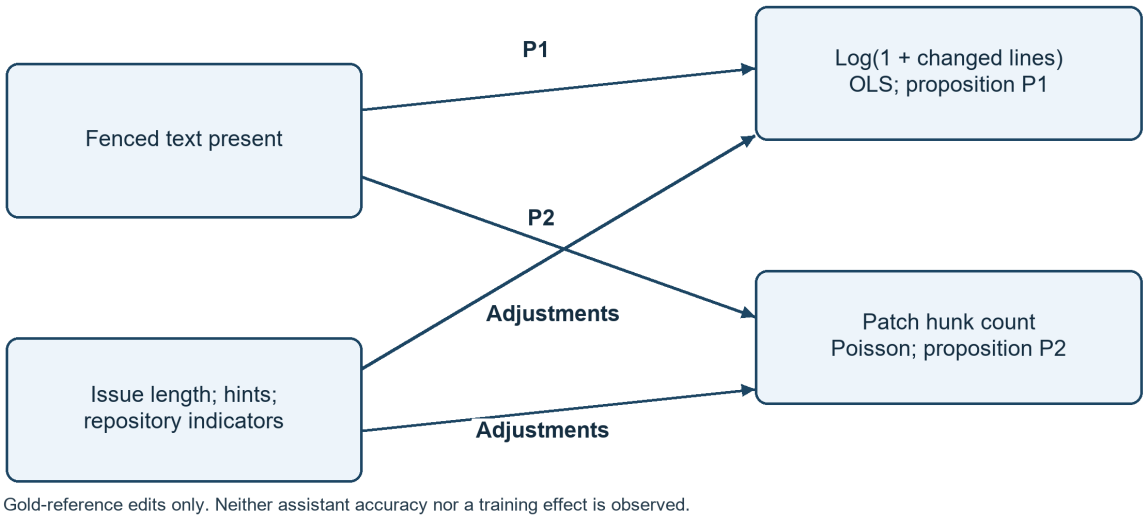

Figure 1. Two exploratory associations between an issue-format indicator and reference-patch scope. P1 uses ordinary least squares for log(1 + changed lines); P2 uses a Poisson model for hunk count. Issue length, hint availability, and repository indicators enter both specifications. The diagram concerns benchmark records and contains no assistant-performance outcome.

4. Public records and reproducible variable construction

We use the 300 test records retained in a previously downloaded SWE-bench Lite snapshot, pinned to dataset revision 6ec7bb89b9342f664a54a6e0a6ea6501d3437cc2. Three saved batches provide 100 records each. Every response was checked for truncated cells, and instance identifiers were verified to be unique. The analysis retains all 300 issues across 12 repositories. Recording the revision, retrieval representation, and extraction decisions follows the broader rationale for documenting dataset provenance and intended use (Gebru et al., 2021). It also allows later benchmark revisions to be distinguished from changes in our calculations.

The input indicator equals one whenever the problem statement contains a triple-backtick sequence. It does not require a correctly closed block, identify a programming language, or distinguish examples from proposed replacements. There are 156 such issues. Description length is the number of word-pattern matches in the original statement, transformed as log(1 + words). Hint availability equals one when the supplied hint field contains non-whitespace text. Empty hints are treated as the observed absence of this field's content, rather than evidence that no contributor ever offered assistance.

The reference patch, stored separately from its test patch, supplies two outcomes. Changed lines count additions and removals after excluding diff-header lines beginning with three plus or minus symbols. A replacement therefore contributes both a removal and an addition. Hunk count is the number of diff-location headers. These are intentionally simple, reproducible definitions, not semantic edit distances. Measurement-error analysis reminds us that an accurately computed proxy can still differ from the construct being discussed (Carroll et al., 2006). Here, the construct is recorded edit extent, with engineering difficulty left unobserved.

All 300 source patches in this snapshot modify exactly one path. File count therefore has no variation and cannot support a regression outcome. We exclude it from modeling and report the restriction as a property of this corpus. Median changed-line count is six. The distinction between a repository record and the development process behind it is familiar in repository mining: version-control histories can be

misinterpreted when their semantics and transformations are ignored (Bird et al., 2009). Our extraction follows the supplied benchmark artifact and does not reconstruct a full development history.

5. Estimation and inferential limits

For P1, we fit ordinary least squares to log(1 + changed lines), including the fence indicator, log-transformed statement length, hint availability, and repository fixed effects. HC3 heteroskedasticity-robust standard errors accompany the estimates. The covariance approach belongs to an established heteroskedasticity-consistent framework (White, 1980). It addresses unequal residual variability under the model's observation assumptions. It does not make repository observations independent or remove omitted-variable bias. The coefficient is reported on its fitted transformed scale to avoid implying a direct arithmetic difference in lines.

P2 uses a Poisson generalized linear model with the same covariates and a log link for the count of patch hunks. Its reported coefficients are log-scale associations. HC3 uncertainty permits variance behavior beyond the simplest Poisson assumption, while the mean specification remains a modeling choice. Repository fixed effects adjust intercept differences, rather than estimating a population distribution of project effects. Multilevel approaches provide another framework for grouped observations (Laird and Ware, 1982; Bates et al., 2015), but fitting such a model would entail additional assumptions not required for this limited descriptive comparison.

The small number of represented repositories deserves particular attention. Cluster-robust procedures depend on their cluster structure and effective information, as discussed by Cameron and Miller (Cameron and Miller, 2015). Our principal table uses HC3 at the issue level and consequently may understate uncertainty if residual dependence remains within projects. We do not interpret the reported intervals as definitive evidence about every Python repository. A future confirmatory study should specify its project-sampling strategy and dependence adjustment before gathering a broader corpus.

The analysis was conducted as an exploratory audit, with both outcomes retained. Multiple-testing procedures such as those of Benjamini and Hochberg and Benjamini and Yekutieli address distinct assumptions about families of comparisons (Benjamini and Hochberg, 1995; Benjamini and Yekutieli, 2001). We do not choose a favorable correction after inspecting coefficients, nor classify the exercise as a preregistered test. The modest collection of descriptive models is presented together so that readers can see the primary null result alongside the secondary hint association. No coefficient is selected as a deployable decision rule.

Table 1. Reference-patch scope models, each using all 300 tasks. Standard errors in parentheses are HC3 estimates; brackets contain 95 percent confidence intervals. Both specifications adjust for repository indicators, statement length, and hint availability. The Poisson column contains log coefficients, and its R2 is not defined as ordinary least-squares R-squared.

Model	Term	Estimate (SE)	95% CI	N
Log changed lines	Fenced text present	-0.2472 (0.2081)	[-0.6568, 0.1625]	300
Log changed lines	Log(1 + issue words)	-0.0378 (0.0699)	[-0.1755, 0.0998]	300
Log changed lines	Hints available	0.0612 (0.1203)	[-0.1756, 0.2980]	300
Hunk count (Poisson log coefficients)	Fenced text present	-0.0723 (0.0974)	[-0.2641, 0.1195]	300
Hunk count (Poisson log coefficients)	Log(1 + issue words)	-0.0188 (0.0354)	[-0.0886, 0.0510]	300
Hunk count (Poisson log coefficients)	Hints available	0.1446 (0.0631)	[0.0204, 0.2689]	300
Log changed lines: fit	R-squared = 0.0787	-	-	300
Hunk count (Poisson log coefficients): fit	Residual deviance = 77.325	-	-	300

6. What the estimates establish

The fencing coefficient in the changed-line model is -0.247, with standard error 0.208 and interval [-0.657, 0.162]. Thus the data do not identify a clear directional difference after adjustment. The model's R-squared is 0.079, indicating limited explanatory power for the included features on this transformed outcome. Converting the fencing coefficient to a ratio would concern the conditional scale of one plus changed lines; it would not turn this observational model into a prediction of how many additional lines a newly formatted issue will require.

The hunk-count coefficient for fencing is -0.072, with standard error 0.097 and interval [-0.264, 0.120]. This outcome provides no strong basis for treating fenced reports as broader multi-region repairs either. Hint availability has a positive secondary coefficient of 0.145 in the hunk model, with interval [0.020, 0.269]. That association is exploratory and potentially affected by case selection. Hints may accompany more involved issues, or may simply be recorded differently across issue histories. The model supplies no intervention that distinguishes those explanations.

No assistant-generated patches appear in either calculation. Correctness-oriented evaluations such as EvalPlus expand tests of generated code, while LiveCodeBench emphasizes the evaluation distribution and contamination concerns (Liu et al., 2023; Jain et al., 2024). These considerations illustrate outcomes a model-performance study would need beyond edit length. A fence indicator could fail to predict patch scope yet still affect an assistant's interpretation of evidence; conversely, a larger patch might be produced correctly. Our findings leave both possibilities open because the necessary generated outputs and execution results are absent.

7. Implications for coding-assistant research

The immediate implication concerns benchmark stratification. Dividing tasks into supposedly easy and difficult groups using the presence of code blocks would impose an interpretation not supported by the observed scope estimates. A more defensible grouping would describe the feature literally and assess additional constructs independently. Empirical software-engineering guidelines emphasize study design and reporting choices, rather than relying on an intuitive proxy alone (Kitchenham et al., 2002). Our explicit definitions let another analyst replace fencing with manually coded fragment roles and test whether that richer annotation is more informative.

The same caution applies to preference data. Direct Preference Optimization offers a learning formulation for preferred versus dispreferred responses, while instruction-following work studies learning from human feedback (Rafailov et al., 2023; Ouyang et al., 2022). Neither means that an input delimiter provides an adequate preference label. A coding example should distinguish the authorized object of modification from demonstrations and constraints. The contribution of the present study is to show how little the simple formatting indicator alone tells us about one observable dimension of the eventual repair, leaving substantive edit authorization to better annotation.

Human review remains a separate activity. Modern code-review research examines expectations and outcomes that extend beyond raw patch size (Bacchelli and Bird, 2013). Issue-label research likewise shows why recorded categories require validation before being reused as ground truth for a different construct (Herzig et al., 2013). Here, code fences are directly observable, but their roles are not labeled. A useful follow-up would have independent reviewers identify reproductions, target snippets, expected outputs, and background documentation, then assess annotation agreement before relating those categories to generated editing errors.

Several design changes would be needed for prediction. A model trained to classify broad edits would require an operational definition of breadth, a held-out evaluation, and an imbalance-aware assessment if broad cases were rare (Saito and Rehmsmeier, 2015). Randomly mixing related project tasks across validation partitions may overstate transportability; structured cross-validation addresses such concerns (Roberts et al., 2017). Selecting features or thresholds on evaluation data introduces another source of optimistic error estimation (Varma and Simon, 2006). We carry out no such classifier selection here, so the fitted associations should not be read as a validated task triage system.

Finally, missing context may be systematic. Empty hint fields do not establish that a development discussion lacked relevant information, and the supplied description need not capture every communication channel. Missing-data theory distinguishes observed absence from assumptions about unobserved values (Little and Rubin, 2002). The current records support a transparent audit of their own contents. They do not support reconstructing the complete problem-solving process, assessing reporter competence, or attributing an assistant's failure to a particular presentation decision.

8. Conclusion

In this fixed SWE-bench Lite snapshot, a visible code-block delimiter does not clearly separate larger from smaller gold-reference edits under either changed-line or hunk-count modeling. The negative point estimates coexist with intervals spanning zero, and the one-file restriction limits the scope variation available for study. Coding-assistant research can use these records to describe issue presentation, but should obtain fragment-role annotations and actual model edits before drawing conclusions about authorization errors. The distinction between supporting code and modifiable code remains substantive even when a simple formatting proxy proves uninformative.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. Carroll, Raymond J., David Ruppert, Leonard A. Stefanski, and Ciprian M. Crainiceanu. “Measurement Error in Nonlinear Models.” 2006. https://doi.org/10.1201/9781420010138.
16. Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. “Fitting Linear Mixed-Effects Models Using lme4.” *Journal of Statistical Software*, vol. 67, no. 1, 2015. https://doi.org/10.18637/jss.v067.i01.
17. 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.
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. Laird, Nan M. and James H. Ware. “Random-Effects Models for Longitudinal Data.” *Biometrics*, vol. 38, no. 4, 1982, pp. 963. https://doi.org/10.2307/2529876.

20. 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.

21. 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.

22. 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.

23. 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.

24. Little, Roderick J. A., and Donald B. Rubin. *Statistical Analysis with Missing Data*. Wiley, 2002. https://doi.org/10.1002/9781119013563.

25. Guo, Daya; Zhu, Qihao; Yang, Dejian; Xie, Zhenda; Dong, Kai; Zhang, Wentao; Chen, Guanting; Bi, Xiao; Wu, Y.; Li, Y. K.; Luo, Fuli; Xiong, Yingfei; Liang, Wenfeng. (2024). DeepSeek-Coder: When the Large Language Model Meets Programming -- The Rise of Code Intelligence. arXiv:2401.14196. https://arxiv.org/abs/2401.14196.

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

27. 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.

28. 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.

29. 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.

30. 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.