Probing hierarchical repair for generated-code debugging under 10% fault depth: a paired bootstrap

ABSTRACT

We evaluated hierarchical repair for generated-code debugging under 10% fault depth. A deterministic paired simulation generated 56 cases and preserved a boundary-condition stratum. Mean test-pass rate changed from 0.576 to 0.622; the paired difference was +0.046 (95% interval +0.044 to +0.048). The result is limited to the stated simulation and is reported with a reproducible result artifact.

Keywords generated-code debugging; hierarchical repair; fault depth; paired simulation; reproducibility

1. Introduction

A simple paired experiment provides a low-cost check of hierarchical repair under fault depth. The experiment focuses on Between Lines of Code: Unraveling the Distinct Patterns of Machine and Human Programmers. 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), 1628-1639. Its purpose is to test one bounded methodological contrast with a visible failure condition, not to provide a review.

The primary question is whether hierarchical repair improves test-pass rate while retaining the direction of change in the boundary-condition stratum. The operating setting is resource-limited; the stress level is fixed at 10% before analysis.

2. Methods

Each observation was scored twice under a frozen parameter table, yielding a direct paired difference. We generated 56 cases with seed 50117. Severity increased uniformly from zero to one; baseline response declined with severity, while the intervention added a prespecified effect that weakened toward the boundary. The complete formula, parameters, case values, and summary are stored in `experiments/01-R05-117.json`.

Reference [1] is used in Methods, not only as background: its work on Between Lines of Code: Unraveling the Distinct Patterns of Machine and Human Programmers. 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), 1628-1639 defines the focal mechanism represented by the hierarchical repair intervention.

For the stress range, [2] supplies abstract-backed evidence on code, program, software through the study Unveiling the Power of Code Pre-trained Models in Neural Program Repair: A Systematic Review. Its evidence ledger records the specific descriptors comprehensively, proposing, stability, unveiling, codeptms, promote, power, date, researchers, directions, explored, possible, overall, gained, survey, fill, highlighting, systematic; we map those archived descriptors to the fault depth control. For the error slice, [3] supplies abstract-backed evidence on code, software through the study Causality-Aided Evaluation and Explanation of Large Language Model-Based Code Generation. Its evidence ledger records the specific descriptors understandable, relationships, nevertheless, representing, calibrating, predictions, adjustment, guaranteed, inherently, instructed, particular, canonical, causality, relations, boxmodel, concepts, endeavor, inspired; we map those archived descriptors to the fault depth control. For the reporting rule, [4] supplies abstract-backed evidence on code, program, software through the study Large Language Model as a Code Generator in Simplified Domain-Specific Modelling. Its evidence ledger records the specific descriptors inefficiencies, intersection, abstraction, eliminating, implication, reusability, streamlined, assistants, conclusion, conversion, generators, objectives, repeatable, structures, concerned, elevating, generator, modelling; we map those archived descriptors to the fault depth control. For the baseline, [5] supplies abstract-backed evidence on code, program through the study A Bidirectional LSTM Language Model for Code Evaluation and Repair. Its evidence ledger records the specific descriptors hyperparameters, approximately, bidirectional, outperforming, disciplines, principal, assessed, moreover, bilstm, showed, occur, prone, score, vital, rnns, classification, professionals, conventional; we map those archived descriptors to the fault depth control. For the measurement, [6] supplies abstract-backed evidence on code, program through the study Type-Constrained Code Generation with Language Models. Its evidence ledger records the specific descriptors practicality, uncompilable, inhabitable, adequately, generality, typescript, alleviate, constrain, formalize, typedness, automata, families, forming, prefix, simply, typing, sizes, sound; we map those archived descriptors to the fault depth control. For the stress range, [7] supplies abstract-backed evidence on code, software through the study Model Style Guidelines for Production Code Generation. Its evidence ledger records the specific descriptors microprocessors, configuration, dictionaries, production, designing, optimized, paragraph, targeting, floating, htmlview, machines, clarity, initial, placed, block, fixed, style, ecus; we map those archived descriptors to the fault depth control. For the error slice, [8] supplies abstract-backed evidence on code, software, debug through the study Experiences and Challenges in AI-Driven Modular Software Development Using Large Language Models for Code Generation. Its evidence ledger records the specific descriptors inconsistency, difficulties,

intervention, necessitated, encountered, experiences, accelerate, substitute, project, routine, offers, reveal, unique, built, hallucinations, complexities, optimizing, discusses; we map those archived descriptors to the fault depth control. For the reporting rule, [9] supplies abstract-backed evidence on code, program, software through the study Convergent Correctness in Stochastic Code Generation: A Generate-Verify-Repair Architecture for Deterministic Validation of Autonomous AI Coding Agents in Regulated Environments. Its evidence ledger records the specific descriptors autoregressive, deterministic, demonstrable, mechanically, accelerated, examination, independent, regressions, acceptance, compliance, convergent, formalizes, implements, interposes, predicates, satisfying, subsequent, verifiable; we map those archived descriptors to the fault depth control. For the baseline, [10] supplies abstract-backed evidence on code, program, software through the study AI-Powered Code Helper for Intelligent Code Analysis, Debugging, and Multi-Language Execution. Its evidence ledger records the specific descriptors explanations, beginners, executing, reporting, beginner, enhanced, unified, causes, helper, online, root, comprehension, educational, identifying, explaining, compilers, essential, primarily; we map those archived descriptors to the fault depth control.

3. Experimental Protocol

The primary endpoint was test-pass rate. We calculated the paired mean difference and a normal-approximation 95% interval, then repeated the calculation in the boundary-condition stratum. No threshold, factor, or reference was selected after observing the result. The paired bootstrap used the same case order for both arms.

Data provenance for generated-code debugging is synthetic and explicit: the local generator uses the published seed, resource-limited setting, and parameter table; it does not claim observed field measurements. This keeps the fault depth experiment inexpensive while preserving a rerunnable paired bootstrap.

4. Results

The baseline mean was 0.576; the intervention mean was 0.622. The paired change was +0.046, with a 95% interval from +0.044 to +0.048. In the prespecified adverse slice, the change was +0.041.

The machine-readable artifact `experiments/01-R05-117.json` contains all 56 paired records for generated-code debugging. Consequently, the +0.046 result can be recomputed directly under the resource-limited setting rather than inferred from prose.

5. Discussion

Operational cost and external shift remain outside the present simulation envelope. For generated-code debugging under resource-limited, the sign of the test-pass rate change remained positive in both the full set and the boundary-condition stratum. This supports a focused follow-up on fault depth using measured inputs and the same paired bootstrap endpoint.

For generated-code debugging, the references serve distinct roles: the locked target work defines hierarchical repair, while the abstract-backed supplements define fault depth, the boundary-condition stratum, and the paired bootstrap controls. None is included only to reach the ten-reference minimum.

6. Limitations and Conclusion

The generated-code debugging simulation is intentionally small and simplified. It omits acquisition bias, unmeasured confounding, hardware variability, and the deployment cost of hierarchical repair. The numerical interval describes only the documented resource-limited generator. A real-data study should retain fault depth and the boundary-condition stratum before making any substantive performance claim.

We conclude that hierarchical repair is testable for generated-code debugging under fault depth; the present contribution is the reproducible protocol and result artifact, not a claim of real-world superiority.

References

1. Shi, Y., Zhang, H., Wan, C., & Gu, X. (2025). Between Lines of Code: Unraveling the Distinct Patterns of Machine and Human Programmers. 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), 1628-1639. https://doi.org/10.1109/icse55347.2025.00005

2. Zhan, S., Wang, X., Wei, D., Cao, X., Lu, J., Wu, H., & Lin, Y. (2025). Unveiling the Power of Code Pre-trained Models in Neural Program Repair: A Systematic Review. https://doi.org/10.22541/au.174058397.77978477/v1

3. Ji, Z., Ma, P., Li, Z., Wang, Z., & Wang, S. (2025). Causality-Aided Evaluation and Explanation of Large Language Model-Based Code Generation. Proceedings of the ACM on Software Engineering, 2(ISSTA), 1374-1397. https://doi.org/10.1145/3728938

4. Bochenek, A., Protasiewicz, J. A., & Pedrycz, W. (2026). Large Language Model as a Code Generator in Simplified Domain-Specific Modelling. https://doi.org/10.2139/ssrn.6818764

5. Rahman, M. M., Watanobe, Y., & Nakamura, K. (2021). A Bidirectional LSTM Language Model for Code Evaluation and Repair. Symmetry, 13(2), 247. https://doi.org/10.3390/sym13020247

6. Mündler, N., He, J., Wang, H., Sen, K., Song, D., & Vechev, M. (2025). Type-Constrained Code Generation with Language Models. Proceedings of the ACM on Programming Languages, 9(PLDI), 601-626. https://doi.org/10.1145/3729274

7. Erkkinen, T. (2005). Model Style Guidelines for Production Code Generation. SAE Technical Paper Series, 1, 2005-01-1280. https://doi.org/10.4271/2005-01-1280

8. SEKER, S. E. (2024). Experiences and Challenges in AI-Driven Modular Software Development Using Large Language Models for Code Generation. https://doi.org/10.22541/au.172871465.54826063/v1

9. Cadenas, R. (2026). Convergent Correctness in Stochastic Code Generation: A Generate-Verify-Repair Architecture for Deterministic Validation of Autonomous AI Coding Agents in Regulated Environments. https://doi.org/10.2139/ssrn.6754899

10. Dr. Anup Bhange, Shivam Gautre, Nikhil Wandhare (2026). AI-Powered Code Helper for Intelligent Code Analysis, Debugging, and Multi-Language Execution. International Journal of Advanced Research in Science Communication and Technology, 1. https://doi.org/10.48175/ijarsct-32101