Execution-Grounded Reinforcement Learning for Automated Program Repair

Abstract

Automated Program Repair is being reorganized around the joint demands of performance with evidence quality, resource limits, and transfer across settings. This technical review examines assigning repair credit at useful granularity while controlling benchmark leakage and patch overfitting. The evidence base combines 1 focal paper with 12 independently retrieved publications whose DOI or publisher records were checked before inclusion. The analysis is organized around execution signals, credit assignment, patch validity, cross-language transfer, and benchmark design. To avoid reading metrics from unlike protocols as commensurate, the review compares problem boundaries, design logic, and conditions of validation. Across the literature, a robust inference is that advances in automated program repair become credible when representation, objective, and evaluation protocol are evaluated together and when uncertainty about distribution shift is reported explicitly. The proposed reading joins method selection to implementation risk, making transfer failures visible, and proposes a research agenda centered on traceable baselines, scenario testing, and reusable evidence.

Keywords

Automated Program Repair; Execution Signals; Credit Assignment; Patch Validity; Cross-Language Transfer; Benchmark Design

1. Introduction

The evidence base for automated program repair bridges scientific ambition and practical constraint. The field seeks not only stronger nominal performance but also explanations for when and why a method works. The tension becomes concrete in assigning repair credit at useful granularity while controlling benchmark leakage and patch overfitting. An advantage observed in one specimen class, benchmark, patient group, region, or workload may fail once its operating context shifts. For this reason, analysis must preserve the unit of analysis and the decision context. This requires distinguishing a mechanism from a correlation, a benchmark advantage from a deployment advantage, and an interpretable diagnostic from a causal explanation.

The analytical frame has five dimensions: execution signals, credit assignment, patch validity, cross-language transfer, benchmark design. Taken together, the lenses are valuable because they jointly cover the technical object, measurement chain, and prerequisites of external validity. The organizing principle is representation, objective, and evaluation protocol. Under that principle, novel technique alone cannot settle the comparison; the comparison also audits the baseline, uncertainty treatment, and resource or hazard boundary. The work reported here is a critical review and adds no fabricated experiment, participant set, measurement, or model result.

2. Methodological Foundations

The analytical chain starts from the input evidence, the transformation applied to it, the output used for evaluation, and the decision that follows. In automated program repair, individual stages may be optimized without their interfaces even though errors propagate across them. Model expressiveness can propagate bias that enters with the observations; an apparently accurate output can still be poorly calibrated for the final decision. The implication is that, ablation evidence should accompany aggregate performance. Rigorous analysis declares the fixed conditions and experimental degrees of freedom, then selects metrics that correspond to the intended use rather than to convenience alone.

The second foundation is boundary awareness. Execution signals defines the local design problem, while benchmark design determines whether the design can survive outside that boundary. Between these endpoints, credit assignment and patch validity connect those ends by exposing trade-offs that a single score conceals. Here, adverse outcomes and sensitivity analysis has diagnostic value because it locates the credible range of an explanation. For applied work, documentation of deployment constraints is therefore part of the scientific result, not an administrative afterthought.

3. Comparative Evaluation

The target study YAA-01, BoostAPR: Boosting Automated Program Repair via Execution-Grounded Reinforcement Learning with Dual Reward Models [1], addresses a concrete part of automated program repair through execution-grounded reinforcement learning with sequence- and line-level reward models. It identifies a representation, mechanism, or evaluation boundary needed to compare claims. Against the focus on assigning repair credit at useful granularity while controlling benchmark leakage and patch overfitting, the paper is treated as a bounded design case: transfer may depend on its sensing regime, preparation, workload, population, or benchmark. The synthesis records the design boundary before comparing assumptions across sources.

Across the focal studies, execution signals should be interpreted jointly with credit assignment. A design can improve one yet redistribute uncertainty rather than remove it. A structured comparison takes the form of a matrix whose rows are operating conditions and whose columns are evidence quality, computation or process cost, robustness, and consequence of failure. The structure can prevent an attractive mechanism from being detached from the circumstances that support it. It additionally indicates which ablations are informative: component removal is useful only when it tests an explicit role.

Patch validity relates the method to the choice it informs. At minimum, evaluation should evaluate baseline and stress regimes separately and expose result dispersion, and test plausible alternative explanations. Where distribution shift is central, calibration or sensitivity is as important as ranking. Where costs or hazards are asymmetric, utility-weighted assessment is more revealing than undifferentiated accuracy. These choices do not erase study differences; they make the reasons for their differences auditable.

4. Architecture and Learning Objectives

The neighboring evidence base brings together Reinforcement learning for mutation operator selection in automated program repair [2]; Reinforcement Learning for Optimizing Test Case Execution in Automated Testing [3]; Template-guided interpretable reasoning with execution feedback for LLM-based program repair [4]. These publications were retrieved and verified through DOI or publisher metadata, and their titles directly intersect the problem boundary of automated program repair. Together they illuminate execution signals: some emphasize representations or mechanisms, whereas others foreground validation and application conditions. The useful comparison is not a leaderboard across incompatible settings, but a structured reading of what each study controls, leaves variable, and treats as a meaningful outcome. Divergence can then reveal a change in scale, data process, endpoint, or operating constraint.

A complementary strand is visible in Automated Program Repair for Introductory Programming Assignments [5]; Heterogeneous multi-expert collaborative reinforcement learning for automated CAD program synthesis from... [6]; Automated Firewall Policy Generation with Reinforcement Learning [7]. These publications were retrieved and verified through DOI or publisher metadata, and their titles directly intersect the problem boundary of automated program repair. Together they illuminate credit assignment: some emphasize representations or mechanisms, whereas others foreground validation and application conditions. The useful comparison is not a leaderboard across incompatible settings, but a structured reading of what each study controls, leaves variable, and treats as a meaningful outcome. Divergence can then reveal a change in scale, data process, endpoint, or operating constraint.

For triangulation, the literature includes Kinesthetic Feedback for Understanding Program Execution [8]; Automated program improvement with reinforcement learning and graph neural networks [9]; Enhancing IELTS writing automated scoring with M-LoRA fine-tuned LLAMA-3 and human feedback-driven PPO r... [10]. These publications were retrieved and verified through DOI or publisher metadata, and their titles directly intersect the problem boundary of automated program repair. Together they illuminate patch validity: some emphasize representations or mechanisms, whereas others foreground validation and application conditions. The useful comparison is not a leaderboard across incompatible settings, but a structured reading of what each study controls, leaves variable, and treats as a meaningful outcome. Divergence can then reveal a change in scale, data process, endpoint, or operating constraint.

The design space is broadened by Automated Bug Detection and Program Repair Using Deep Learning: A Comprehensive Review [11]; Deep Reinforcement Learning for Automated Liquidity Management in Concentrated Automated Market Makers:... [12]; Hierarchical Program-Triggered Reinforcement Learning Agents for Automated Driving [13]. These publications were retrieved and verified through DOI or publisher metadata, and their titles directly intersect the problem boundary of automated program repair. Together they illuminate cross-language transfer: some emphasize representations or mechanisms, whereas others foreground validation and application conditions. The useful comparison is not a leaderboard across incompatible settings, but a structured reading of what each study controls, leaves variable, and treats as a meaningful outcome. Divergence can then reveal a change in scale, data process, endpoint, or operating constraint.

5. Deployment and Governance

A practical workflow has five stages. First, define the decision and its failure cost. Second, specify the evidence and operating envelope. Third, choose a method whose inductive assumptions match that evidence. Fourth, test cross-language transfer and benchmark design under controlled perturbations. Finally, retain provenance so that an unexpected outcome can be traced to data, configuration, material batch, environmental condition, or user interaction. The workflow connects assigning repair credit at useful granularity while controlling benchmark leakage and patch overfitting connected to observable requirements.

A broader conclusion follows: responsible progress in automated program repair depends on how components exchange evidence. Interfaces determine how uncertainty moves between measurement and inference, between algorithm and operator, or between microscopic design and system behavior. Interdisciplinary teams need prior agreement about acceptance thresholds and claim-revision evidence. When those agreements are explicit, optimization can pursue efficiency without silently relaxing validity, safety, or interpretability.

6. Limitations and Future Directions

The principal review limitations are cross-study variation in labels, design choices, and reporting precision. Metadata confirmation secures the citation trail but does not reproduce measurements or results. In addition, fast-moving work may change venue or version. No confirmed focal Scholar citation was normalized away, and anomalies were logged independently.

A productive research agenda would preregister comparison protocols where feasible, provide structured configuration records and assess a realistic transfer condition in deployment constraints. Research should also organize error cases around mechanisms instead of counts alone. For automated program repair, a valuable shared benchmark would combine baseline utility, resource cost, calibration, and post-perturbation recovery. Such a benchmark would reward designs that remain useful when evidence is incomplete or conditions change.

7. Conclusion

Scholarship in automated program repair becomes clearer when treated as a linked evidence chain rather than a collection of isolated scores. The focal studies show how assigning repair credit at useful granularity while controlling benchmark leakage and patch overfitting; the additional literature reveals the assumptions required to interpret those contributions. Across execution signals, credit assignment, patch validity, cross-language transfer, and benchmark design, the strongest cross-study principle is alignment: the representation or mechanism must match the problem, the evaluation must match the decision, and the deployment claim must match the tested boundary. The consequence is evidence that travels more safely and fails more informatively.

References

1. Li, Y., Wang, H., Shang, X., Tang, X., Cao, Y., & Chen, X. (2026). BoostAPR: Boosting Automated Program Repair via Execution-Grounded Reinforcement Learning with Dual Reward Models. arXiv preprint arXiv:2605.09134.

2. Hanna, C., Blot, A., & Petke, J. (2025). Reinforcement learning for mutation operator selection in automated program repair. Automated Software Engineering, 32(2). https://doi.org/10.1007/s10515-025-00501-z

3. Kumar Karne, V., Noone Srinivas,, Nagaraj Mandaloju,, & Parameshwar Reddy Kothamali, (2020). Reinforcement Learning for Optimizing Test Case Execution in Automated Testing. Innovative Research Thoughts, 6(3), 13-27.

https://doi.org/10.36676/irt.v6.i3.1494

4. Hao, S., Shi, X., Liu, H., Yin, Y., & Chen, X. (2026). Template-guided interpretable reasoning with execution feedback for LLM-based program repair. Information and Software Technology, 193, 108058. https://doi.org/10.1016/j.infsof.2026.108058
5. Wan, H., Luo, H., Li, M., & Luo, X. (2024). Automated Program Repair for Introductory Programming Assignments. IEEE Transactions on Learning Technologies, 17, 1705-1720. https://doi.org/10.1109/tlt.2024.3403710
6. Yin, Z., Lin, W., & Kong, X. (2026). Heterogeneous multi-expert collaborative reinforcement learning for automated CAD program synthesis from engineering drawings. Discover Artificial Intelligence. https://doi.org/10.1007/s44163-026-01731-0
7. Jha, A. C. (2025). Automated Firewall Policy Generation with Reinforcement Learning. International journal of IoT, 5(1), 190-211. https://doi.org/10.55640/ijiot-05-01-10
8. Gill, S., Goolsby, B. J., & Pawluk, D. T. V. (2023). Kinesthetic Feedback for Understanding Program Execution. Sensors, 23(11), 5159. https://doi.org/10.3390/s23115159
9. Sukur, N. a., Milošević, N., Pracner, D., & Budimac, Z. (2023). Automated program improvement with reinforcement learning and graph neural networks. Soft Computing, 28(3), 2593-2604. https://doi.org/10.1007/s00500-023-08559-1
10. Xu, W., Kassim, M. S. S., & Mahmud, R. (2026). Enhancing IELTS writing automated scoring with M-LoRA fine-tuned LLAMA-3 and human feedback-driven PPO reinforcement learning. Scientific Reports, 16(1). https://doi.org/10.1038/s41598-026-43318-w
11. Ali, R. H., & Chyad, A. M. (2026). Automated Bug Detection and Program Repair Using Deep Learning: A Comprehensive Review. Cybernetics and Information Technologies, 26(1), 93-121. https://doi.org/10.2478/cait-2026-0006
12. Farrugia, F. (2026). Deep Reinforcement Learning for Automated Liquidity Management in Concentrated Automated Market Makers: A Multi-Architecture Empirical Study. Journal of Advances in Civil and Mechanical Engineering, 03(01), 01-06. https://doi.org/10.64030/3067-2457.03.01.02
13. Gangopadhyay, B., Soora, H., & Dasgupta, P. (2022). Hierarchical Program-Triggered Reinforcement Learning Agents for Automated Driving. IEEE Transactions on Intelligent Transportation Systems, 23(8), 10902-10911. https://doi.org/10.1109/tits.2021.3096998