Evidence Alignment and Transfer Boundaries in Automated Program Repair And Reinforcement Learning For Software Reasoning: BoostAPR Boosting Automated Program and Evo-CuRL Curriculum-Aware Reinforcement Learning

Abstract

Progress in automated program repair and reinforcement learning for software reasoning depends on more than accumulating favorable results. This critical synthesis connects execution-grounded reinforcement learning with sequence- and line-level reward models with curriculum-aware reinforcement learning over code-lineage graphs and asks how measurement choices, boundary conditions, and decision costs shape the interpretation of both. Two target papers are triangulated against 12 locally validated publications. The comparison follows execution signals, credit assignment, patch validity, cross-language transfer, benchmark design and deliberately separates mechanistic interpretation from performance ranking, because the latter can conceal incompatible experimental or operational conditions. Across the evidence base, the decisive issue is alignment: execution signals shapes what is observed, credit assignment shapes how it is compared, and benchmark design governs whether the conclusion can be transferred. Uncertainty is most informative when reported as part of the result rather than treated as a postscript. On this basis, the review proposes an auditable pathway from focal mechanism to application claim, with explicit checkpoints for calibration, external validity, and responsible interpretation.

Keywords

Automated Program Repair And Reinforcement Learning For Software Reasoning; Execution Signals; Credit Assignment; Patch Validity; Cross-Language Transfer; Benchmark Design

1. Introduction

Contemporary investigation of automated program repair and reinforcement learning for software reasoning occupies the boundary between scientific ambition and practical constraint. The objective includes not only stronger nominal performance but also explanations of the mechanisms that support observed behavior. Its practical form appears in comparing execution-grounded reinforcement learning with sequence- and line-level reward models with curriculum-aware reinforcement learning over code-lineage graphs through evidence alignment and transfer boundaries. A result that is compelling in a specific substrate, benchmark, population, spatial unit, or workload may be fragile outside its original boundary. The comparison accordingly needs to 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.

Comparison proceeds across five linked dimensions: execution signals, credit assignment, patch validity, cross-language transfer, benchmark design. They were chosen because they jointly cover what changes, how change is observed, and which conditions must persist. Its governing principle is representation, objective, and evaluation protocol. Under that principle, method novelty must be tested against operating

limits; the comparison also audits the baseline, uncertainty treatment, and resource or hazard boundary. The article is a literature synthesis and is not presented as a new experiment and supplies no synthetic performance claim.

2. Methodological Foundations

A tractable account separates the input evidence, the transformation applied to it, the output used for evaluation, and the decision that follows. In automated program repair and reinforcement learning for software reasoning, individual stages may be optimized without their interfaces even though errors propagate across them. A powerful model can intensify errors embedded in the initial evidence; an apparently accurate output can still be poorly calibrated for the final decision. A direct requirement is that, ablation evidence should accompany aggregate performance. A credible comparison records the constants, perturbations, and uncontrolled factors, then selects metrics that correspond to the intended use rather than to convenience alone.

The next requirement is control of scope. Execution signals defines the local design problem, while benchmark design determines whether the design can survive outside that boundary. The link between them depends on credit assignment and patch validity connect those ends by exposing trade-offs that a single score conceals. This is where negative results and sensitivity evidence maps the conditions under which the explanation remains viable. For applied work, documentation of deployment constraints is therefore part of the scientific result, not an administrative afterthought.

3. Architecture and Learning Objectives

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 and reinforcement learning for software reasoning 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 comparing execution-grounded reinforcement learning with sequence- and line-level reward models with curriculum-aware reinforcement learning over code-lineage graphs through evidence alignment and transfer boundaries, the paper functions as a bounded point of comparison: transfer may depend on its sensing regime, preparation, workload, population, or benchmark. The synthesis maintains the declared scope and triangulates the underlying assumptions.

The target study YAA-03, Evo-CuRL: Curriculum-Aware Reinforcement Learning over Code Lineage Graphs for Software Engineering Reasoning [2], addresses a concrete part of automated program repair and reinforcement learning for software reasoning through curriculum-aware reinforcement learning over code-lineage graphs. It identifies a representation, mechanism, or evaluation boundary needed to compare claims. Against the focus on comparing execution-grounded reinforcement learning with sequence- and line-level reward models with curriculum-aware reinforcement learning over code-lineage graphs through evidence alignment and transfer boundaries, the paper functions as a bounded point of comparison: transfer may depend on its sensing regime, preparation, workload, population, or benchmark. The synthesis maintains the declared scope and triangulates the underlying assumptions.

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. The design space can be represented as 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. The matrix helps determine which ablations are informative: ablation design should target causal attribution instead of numerical

proliferation.

Patch validity connects a method to its decision consequence. At the least, assessment should contrast nominal operation with boundary tests and report more than means, 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 preserve study distinctions; they make the reasons for their differences auditable.

4. Comparative Evaluation

A complementary strand is visible in Reinforcement learning for mutation operator selection in automated program repair [3]; Reinforcement Learning Model Based on Regret for Multi-Agent Conflict Games [4]; Reinforcement Learning for Optimizing Test Case Execution in Automated Testing [5]. These publications were retrieved and verified through DOI or publisher metadata, and their titles directly intersect the problem boundary of automated program repair and reinforcement learning for software reasoning. 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.

For triangulation, the literature includes Stabilizing independent multi-agent reinforcement learning via curriculum-based iterative self-play [6]; Template-guided interpretable reasoning with execution feedback for LLM-based program repair [7]; Multi-hop temporal knowledge graph reasoning with multi-agent reinforcement learning [8]. These publications were retrieved and verified through DOI or publisher metadata, and their titles directly intersect the problem boundary of automated program repair and reinforcement learning for software reasoning. 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.

The design space is broadened by Automated Program Repair for Introductory Programming Assignments [9]; Abmarl: Connecting Agent-Based Simulations with Multi-Agent Reinforcement Learning [10]; Heterogeneous multi-expert collaborative reinforcement learning for automated CAD program synthesis from... [11]. These publications were retrieved and verified through DOI or publisher metadata, and their titles directly intersect the problem boundary of automated program repair and reinforcement learning for software reasoning. 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.

A first comparison set connects Curriculum-Learning-Guided Multi-Agent Deep Reinforcement Learning for N-1 Static Security Prevention an... [12]; Automated Firewall Policy Generation with Reinforcement Learning [13]; Curriculum-assisted multi-agent reinforcement learning for scalable V2X resource allocation [14]. These publications were retrieved and verified through DOI or publisher metadata, and their titles directly intersect the problem boundary of automated program repair and reinforcement learning for software reasoning. 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

For implementation, the review suggests a staged workflow. 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. These stages keep comparing execution-grounded reinforcement learning with sequence- and line-level reward models with curriculum-aware reinforcement learning over code-lineage graphs through evidence alignment and transfer boundaries connected to observable requirements.

A broader conclusion follows: responsible progress in automated program repair and reinforcement learning for software reasoning is determined by coupling as well as local design. Interfaces determine how uncertainty moves between measurement and inference, between algorithm and operator, or between microscopic design and system behavior. A shared protocol should state acceptance conditions and what would overturn a claim. When those agreements are explicit, performance tuning can proceed while preserving visible validity and safety requirements.

6. Limitations and Future Directions

The review remains constrained by divergent terms, methods, and documentation depth. Metadata confirmation secures the citation trail but does not reproduce measurements or results. Publication status can change after metadata collection. The supplied focal strings remain preserved while questionable normalization is shown only as a candidate.

Priority should be given to studies that preregister comparison protocols where feasible, disclose reusable configurations and examine transfer beyond the nominal regime in deployment constraints. Research should also document why failures occur in addition to how often. For automated program repair and reinforcement learning for software reasoning, a valuable shared benchmark would combine standard-condition utility, efficiency, confidence quality, and fault recovery. Such a benchmark would reward designs that remain useful when evidence is incomplete or conditions change.

7. Conclusion

The literature on automated program repair and reinforcement learning for software reasoning forms an evidence network rather than a list of results rather than a collection of isolated scores. The focal studies show how comparing execution-grounded reinforcement learning with sequence- and line-level reward models with curriculum-aware reinforcement learning over code-lineage graphs through evidence alignment and transfer boundaries; 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, assessment must align with action, just as deployment claims align with validation scope. Alignment makes transfer more cautious, replication more feasible, and failure more instructive.

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. Tan, W., Li, Y., & Liang, W. (2026, June). Evo-CuRL: Curriculum-Aware Reinforcement Learning over Code Lineage Graphs for Software Engineering Reasoning. In Proceedings of the 2026 International Conference on Multimedia Retrieval (pp. 1327-1335).
3. 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
4. XIAO, Z., & ZHANG, S. Y. (2009). Reinforcement Learning Model Based on Regret for Multi-Agent Conflict Games. Journal of Software, 19(11), 2957-2967. https://doi.org/10.3724/sp.j.1001.2008.02957
5. 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
6. Akgün, O. (2026). Stabilizing independent multi-agent reinforcement learning via curriculum-based iterative self-play. Neurocomputing, 704, 134819. https://doi.org/10.1016/j.neucom.2026.134819
7. 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
8. Bai, L., Chen, M., & Xiao, Q. (2024). Multi-hop temporal knowledge graph reasoning with multi-agent reinforcement learning. Applied Soft Computing, 160, 111727. https://doi.org/10.1016/j.asoc.2024.111727
9. 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
10. Rusu, E., & Glatt, R. (2021). Abmarl: Connecting Agent-Based Simulations with Multi-Agent Reinforcement Learning. Journal of Open Source Software, 6(64), 3424. https://doi.org/10.21105/joss.03424
11. 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
12. Zhang, X., Li, Z., Quan, X., Cheng, K., & Yu, Y. (2026). Curriculum-Learning-Guided Multi-Agent Deep Reinforcement Learning for N-1 Static Security Prevention and Control. Energy Engineering, 123(9), 1-10. https://doi.org/10.32604/ee.2025.073912
13. 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
14. Morshed, M., & Zaman Chowdhury, M. (2026). Curriculum-assisted multi-agent reinforcement learning for scalable V2X resource allocation. Physical Communication, 76, 103062. https://doi.org/10.1016/j.phycom.2026.103062