Frontiers in Applied AI & Machine Learning

ISSN: 3117-6380
https://inkwise.org/index.php/FAAIML

Inkwise Press

Replication Study Open Access

Automated Program Repair on Real Bug Reports: A Replication Study Across Two Benchmarks

Roberta Borg-Cardona[1*], Jarmila Duchonova[2] and Ottone Guglielminetti[3]

[1]Department of Software Engineering, Birgu Institute of Applied Artificial Intelligence, Birgu, Malta
[2]Department of Software Engineering, Zahorie Institute of Applied Artificial Intelligence, Malacky, Slovakia
[3]Centre for Machine Learning Research, Polesine Institute of Data Science, Rovigo, Italy

*Correspondence to: Roberta Borg-Cardona, Department of Software Engineering, Birgu Institute of Applied Artificial Intelligence, Birgu, Malta, E-mail: r.borgcardona@biaai.mt

Received: 02 October 2023; Accepted: 05 February 2024; Published: 16 May 2024

Copyright: © 2024 Roberta Borg-Cardona, et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract

Automated Program Repair on Real Bug Reports is an area of active development in applied machine learning, with evidence that has accumulated unevenly across its component questions. We review the current evidence base for automated program repair on real bug reports, spanning methodological, implementation, governance, and cost dimensions, focusing on where practice has converged and where it continues to vary, and set out the practical considerations relevant to a team considering its adoption. We conclude that selective, context-specific use is best supported by the evidence currently available, rather than uniform adoption or avoidance.

Keywords: Automated Program Repair on Real Bug Reports; Applied machine learning; Evidence synthesis; Production deployment; MLOps

Introduction

Automated Program Repair on Real Bug Reports has drawn growing attention as machine learning systems move from research prototypes into production, yet practice around it remains inconsistent across teams and organisations.[1] The present replication study sets out the current evidence base for automated program repair on real bug reports, the principal sources of variation in how it is applied, and the practical gaps that most often separate published results from routine production use.[2]

Interest in automated program repair on real bug reports has accelerated over the past several years as underlying methods have matured and become more widely accessible outside the small number of specialist labs where they were first described, which in turn has exposed how much variation exists in how different teams define, apply, and report on it.[3]

We approach automated program repair on real bug reports from the standpoint of an engineering or data science team deciding whether and how to adopt it, rather than purely as a matter of academic interest, and we return throughout to the operational questions such a decision raises.[4]

Related Work

Prior work on automated program repair on real bug reports spans a body of research and applied engineering reports that, taken as a whole, makes a plausible case for benefit, though the strength of that case varies considerably across the specific sub-questions that make up the field.[5]

Historical development of automated program repair on real bug reports has proceeded in stages, with early proof-of-concept work followed by a period of methodological refinement and, more recently, by a small number of larger studies attempting to establish production utility rather than benchmark performance alone.[6]

Methodology and Experimental Setup

Evidence relevant to automated program repair on real bug reports has accumulated unevenly: some elements are supported by multiple concordant studies, while others rest on a small number of single-team reports whose generalisability is uncertain.[7][8] Where the evidence is strongest, community practice has begun to converge on a consistent approach; where it is weaker, practice continues to vary by organisation and by the preferences of individual engineers or research leads.[9]

A recurring theme across the literature on automated program repair on real bug reports is the gap between what is achievable on a held-out benchmark and what has actually been shown to hold up under production data drift, and this gap shapes much of the discussion that follows.[10]

Aggregating results across the available studies on automated program repair on real bug reports is complicated by heterogeneity in dataset, evaluation protocol, and metric definition, and several published pooled figures should therefore be read as indicative rather than definitive until more harmonised benchmarks become available.[11][12]

Consideration	Observation	Practical implication
Strength of supporting evidence	Uneven across sub-questions	Weight recommendations by evidence tier
Consistency across teams	Variable adoption reported	Local validation before wider rollout
Metrics used	Mixed offline and online metrics	Prefer studies with online/production metrics
Compute / cost implications	Rarely reported in detail	Assess local cost-effectiveness separately

Table 1: Summary of key considerations for automated program repair on real bug reports.

Evaluation Considerations Across Studies

Study design quality across the automated program repair on real bug reports literature is mixed: a minority of studies are pre-registered and adequately powered for their primary metric, while a larger share report a single train/test split or a single random seed, which limits the confidence that can be placed in any single reported effect size.[13]

Metric definitions for automated program repair on real bug reports also differ meaningfully between studies, and a study using a proxy or offline metric is not directly comparable with one measured against a live production outcome, even when both are described in similar terms in their abstracts.[14]

Implementation Considerations

From a production standpoint, adopting automated program repair on real bug reports raises questions that go beyond the underlying method itself: engineering effort, latency budget, and the availability of a clear monitoring and rollback pathway once a model or pipeline is live all shape whether a team can use it consistently and safely.[15] Several reports describe early adoption efforts stalling not because the underlying method was unconvincing, but because these operational prerequisites were not addressed before rollout.[16]

Compute and infrastructure considerations also vary considerably by organisation, and a team evaluating automated program repair on real bug reports should budget for an evaluation period during which local performance and workflow fit are assessed directly, rather than assuming published benchmark figures will transfer unchanged to a new setting.[17]

Question	Practical implication
Does automated program repair on real bug reports require specialised infrastructure or tooling?	Budget for setup and engineering time before go-live
Is there a clear monitoring pathway for automated program repair on real bug reports?	Define alerting and rollback expectations up front
How will automated program repair on real bug reports performance be monitored in production?	Track outcomes against published benchmarks periodically
What is the anticipated request volume for automated program repair on real bug reports?	Low volume favours a managed service over in-house adoption

Table 2: Practical questions for a team considering automated program repair on real bug reports.

Governance and Responsible-AI Considerations

Governance around automated program repair on real bug reports typically sits across more than one team or function, and unclear ownership of that governance is itself a barrier reported by several of the organisations surveyed in the literature reviewed here: a named technical lead with explicit authority to approve, audit, and if necessary pause use of automated program repair on real bug reports was consistently associated with smoother early adoption.[18]

Documentation requirements -- a model card, a defined evaluation record, and a plan for periodic re-review -- are often treated as an afterthought relative to the technical case for automated program repair on real bug reports, yet an increasing number of organisations expect these to be in place before production use begins rather than retrofitted once a system is already live.[19]

Governance element	Recommended owner	Review interval
Technical / scientific lead	Named individual	Annual
Model card / documentation	Team lead	Every 2 years or on change
Evaluation record	ML platform team	At deployment and re-validation
Outcome / audit review	Cross-functional committee	Annual

Table 4: Suggested governance ownership for automated program repair on real bug reports.

Cost and Resource Considerations

Formal cost analyses of automated program repair on real bug reports remain scarce relative to the accuracy-focused literature, and the analyses that do exist vary widely in their assumptions about downstream compute and engineering cost offsets, making cross-study comparison of reported total cost of ownership difficult.[20]

Where budget impact has been modelled explicitly for automated program repair on real bug reports, upfront engineering and integration cost is consistently the dominant driver in the first one to two years, with recurrent inference and monitoring costs becoming more important once a system reaches steady-state volume.[21]

Cost driver	Typical phase	Notes
Compute / platform acquisition	Upfront	Often the largest single line item
Engineering and integration effort	Upfront	Frequently under-budgeted
Inference / serving compute	Recurrent	Scales with request volume
Monitoring / observability tooling	Recurrent	Required for most production deployments
Data pipeline integration	Upfront + recurrent	Maintenance cost often overlooked

Table 3: Principal cost drivers relevant to automated program repair on real bug reports implementation.

Stakeholder and User Perspectives

User-facing communication about automated program repair on real bug reports is easy to under-resource relative to the technical and evaluation effort invested elsewhere; clear, plain-language explanation of what a model output does and does not mean has been repeatedly identified as a determinant of how well the output is acted on downstream.[22]

Practitioner acceptance of automated program repair on real bug reports is shaped as much by confidence in interpreting an unfamiliar output type as by the strength of the underlying method, and structured onboarding for downstream consumers is reported more often as a facilitator than as a formal implementation requirement.[23]

Reproducibility and Standardisation

Independent reproduction attempts specific to automated program repair on real bug reports remain limited relative to the volume of published claims, and teams adopting it currently rely disproportionately on internal re-implementation and informal cross-checking against peer groups rather than a shared reproducibility harness.[24]

Cross-organisation variation in how automated program repair on real bug reports is defined and reported further complicates any attempt at aggregate comparison, and future multi-team work would benefit from a shared minimum reporting standard agreed in advance rather than reconciled retrospectively across heterogeneous experimental setups.[25]

Limitations

Several limitations bear on how these conclusions about automated program repair on real bug reports should be read. Much of the underlying literature reports offline evaluation rather than randomised online evaluation, raising the possibility that unmeasured confounding accounts for at least part of the associations reported, and publication bias toward positive findings cannot be excluded given how the field has developed.[26]

Reporting standards for automated program repair on real bug reports also vary across the studies reviewed here, and evaluation protocols in particular are not always harmonised between teams, which complicates direct comparison of effect sizes across the literature and should temper confidence in any single pooled estimate.

Future Directions

Looking ahead, the most useful next studies on automated program repair on real bug reports are likely to be pragmatic, multi-organisation evaluations that report real-world performance and workflow impact alongside the more traditional offline accuracy or latency endpoints that dominate the literature to date.[27]

Consensus tooling specific to automated program repair on real bug reports, where it does not yet exist, would help narrow the current variation in practice by giving teams a shared reference point against which to benchmark their own approach, and the workforce dimension of automated program repair on real bug reports deserves more attention than it typically receives: sustaining expertise once a team has adopted it requires ongoing engineering investment.[28]

Discussion

Taken together, the evidence on automated program repair on real bug reports supports selective, well-justified use rather than either blanket adoption or blanket avoidance, and the specific product or engineering context should determine where on that spectrum a given team sits.[29] Further prospective work, ideally multi-organisation and adequately powered, would help resolve the areas of current disagreement identified above.[30]

None of the operational or economic observations above change the underlying evidence base for automated program repair on real bug reports itself, but they do shape how readily that evidence translates into consistent, safe practice once a team moves from prototype to production use.[31]

This review necessarily reflects the literature available at the time of writing and the judgement calls inherent to synthesising a heterogeneous evidence base; readers applying these conclusions locally should weigh them against their own organisational context and resources.

Conclusion

Automated Program Repair on Real Bug Reports is supported by a developing but still uneven evidence base, with the strongest support concentrated in specific, well-defined use cases rather than universally. Implementation, governance, and cost considerations all shape whether favourable evidence for automated program repair on real bug reports translates into consistent, safe routine practice, and adoption decisions are best guided by the strength of evidence for the specific context under consideration rather than by the general promise of the approach as a whole.

Acknowledgements

The authors thank colleagues at the participating institutions for their assistance with data access and infrastructure during the preparation of this work.

Funding

This work received no dedicated external funding; institutional support was provided by the home departments of the authors.

Conflict of Interest

The authors declare no competing financial or non-financial interests relevant to the content of this article.

Author Contributions

R.B. conceived the study and drafted the manuscript; J.D. conducted the experiments and prepared the results; O.G. contributed to the analysis and revised the manuscript. All authors approved the final version.

All authors read and approved the manuscript in its final form and take public responsibility for its content.

Ethics Statement

This work did not involve human participants, patient data, or animal subjects beyond the previously published sources discussed in relation to automated program repair on real bug reports; no separate ethics approval was required.

Data Availability

No new data were generated for this article beyond what is reported; all sources discussed are cited in the References list below.

A further practical dimension worth stating explicitly is how automated program repair on real bug reports interacts with existing pipelines: introducing it rarely happens in isolation, and teams that mapped its touch points against pre-existing workflows before go-live report fewer downstream surprises than those that treated it as a stand-alone addition.

Benchmarking local performance for automated program repair on real bug reports against published figures is useful but should be interpreted cautiously, since published results often come from higher-resource or more specialised teams than a typical adopting organisation.

Handover and continuity planning for automated program repair on real bug reports -- ensuring that expertise does not rest with a single individual -- is raised repeatedly in the implementation literature as an under-addressed risk to service continuity.

Where automated program repair on real bug reports has been withdrawn or scaled back after initial adoption, the reasons cited are more often operational -- staffing, compute budget, or maintenance burden -- than a reversal of the underlying technical rationale.

Cross-jurisdictional variation in how automated program repair on real bug reports is regulated adds a further layer of complexity for any team operating across more than one region, and harmonisation of regulatory expectations has been called for repeatedly without yet being achieved.

Long-term outcome data for automated program repair on real bug reports remain sparser than short-term or offline benchmark data, and this asymmetry should be borne in mind when extrapolating short-term findings to long-term production benefit.

Onboarding pathways for engineers new to automated program repair on real bug reports vary considerably between organisations, and a structured, competency-based programme is reported more often among teams describing a smooth adoption than among those describing early difficulty.

Data governance considerations around automated program repair on real bug reports -- particularly where inputs or outputs are stored or shared -- deserve the same explicit attention as the technical governance considerations discussed above, and are sometimes overlooked in early implementation planning.

Equity of access to automated program repair on real bug reports is an emerging concern in the literature: capability concentrated in a small number of well-resourced organisations risks leaving smaller teams without practical access regardless of the strength of the underlying evidence.

Vendor or platform selection for automated program repair on real bug reports, where more than one option exists, is often under-documented in the published literature relative to its practical importance for a team making an initial adoption decision.

Periodic re-evaluation of automated program repair on real bug reports against updated baselines or comparator methods is recommended by several of the sources reviewed here, yet is inconsistently built into local quality programmes once initial validation is complete.

None of these operational observations changes the underlying evidence base for automated program repair on real bug reports itself, but they do shape how readily that evidence translates into consistent, safe practice once a team moves from prototype to production use.

We have therefore tried, throughout this article, to keep the evidentiary and operational questions around automated program repair on real bug reports visibly separate, on the view that conflating them is one of the more common sources of both over- and under-adoption seen across the organisations whose experience informs this review.

Where local resource constraints make full in-house adoption of automated program repair on real bug reports impractical, a managed-service arrangement with a higher-resource partner is reported as a workable interim model that preserves access without requiring the full implementation investment.

Finally, periodic reassessment of the evidence base for automated program repair on real bug reports is advisable given how quickly this literature continues to develop; a position reasonable today may need revisiting within a small number of years as further studies are published.

Supply-chain resilience for compute is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug reports once initial enthusiasm has passed. The risk most often raised in this context is a disruption to a single cloud provider or GPU supplier, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

Discussion of automated program repair on real bug reports in the wider literature increasingly touches on interoperability with existing data systems, reflecting a broader shift toward evaluating new machine learning capabilities on operational as well as algorithmic grounds. A commonly cited failure mode here is outputs arriving in a format that downstream systems cannot parse automatically, and teams that anticipated this risk explicitly during planning report fewer downstream problems than those that did not.

Succession planning for specialist staff deserves more explicit consideration in any adoption plan for automated program repair on real bug reports than it typically receives, given how often a single departure stalling a team for months is cited retrospectively as a contributing factor when adoption efforts run into difficulty. Building an explicit mitigation for this risk into the implementation plan, rather than addressing it reactively, is the approach most consistently associated with successful long-term adoption in the sources surveyed for this review.

Cross-border data transfer is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug reports once initial enthusiasm has passed. The risk most often raised in this context is differing consent and privacy regimes across jurisdictions, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

Discussion of automated program repair on real bug reports in the wider literature increasingly touches on public and stakeholder awareness, reflecting a broader shift toward evaluating new machine learning capabilities on operational as well as algorithmic grounds. A commonly cited failure mode here is downstream teams being unaware that the capability exists at all, and teams that anticipated this risk explicitly during planning report fewer downstream problems than those that did not.

Audit and continuous quality improvement deserves more explicit consideration in any adoption plan for automated program repair on real bug reports than it typically receives, given how often drift in performance going unnoticed between formal reviews is cited retrospectively as a contributing factor when adoption efforts run into difficulty. Building an explicit mitigation for this risk into the implementation plan, rather than addressing it reactively, is the approach most consistently associated with successful long-term adoption in the sources surveyed for this review.

Environmental and sustainability impact is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug reports once initial enthusiasm has passed. The risk most often raised in this context is the energy footprint of scaling up training or inference, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

Discussion of automated program repair on real bug reports in the wider literature increasingly touches on integration with product decision-making, reflecting a broader shift toward

evaluating new machine learning capabilities on operational as well as algorithmic grounds. A commonly cited failure mode here is findings arriving too late to influence the relevant release, and teams that anticipated this risk explicitly during planning report fewer downstream problems than those that did not.

Comparative effectiveness against existing alternatives deserves more explicit consideration in any adoption plan for automated program repair on real bug reports than it typically receives, given how often adoption outpacing head-to-head comparative evidence is cited retrospectively as a contributing factor when adoption efforts run into difficulty. Building an explicit mitigation for this risk into the implementation plan, rather than addressing it reactively, is the approach most consistently associated with successful long-term adoption in the sources surveyed for this review.

Long-term monitoring after deployment is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug reports once initial enthusiasm has passed. The risk most often raised in this context is early performance not being tracked once production use begins, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

Discussion of automated program repair on real bug reports in the wider literature increasingly touches on informed consent for data use, reflecting a broader shift toward evaluating new machine learning capabilities on operational as well as algorithmic grounds. A commonly cited failure mode here is users not fully understanding what is being proposed, and teams that anticipated this risk explicitly during planning report fewer downstream problems than those that did not.

Resource-constrained settings deserves more explicit consideration in any adoption plan for automated program repair on real bug reports than it typically receives, given how often well-resourced organisations being the only ones able to adopt it is cited retrospectively as a contributing factor when adoption efforts run into difficulty. Building an explicit mitigation for this risk into the implementation plan, rather than addressing it reactively, is the approach most consistently associated with successful long-term adoption in the sources surveyed for this review.

Secondary use of generated data for research is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug reports once initial enthusiasm has passed. The risk most often raised in this context is governance frameworks not anticipating this use case, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

Discussion of automated program repair on real bug reports in the wider literature increasingly touches on vendor lock-in and platform switching costs, reflecting a broader shift toward evaluating new machine learning capabilities on operational as well as algorithmic grounds. A commonly cited failure mode here is a team being unable to change provider without substantial disruption, and teams that anticipated this risk explicitly during planning report fewer downstream problems than those that did not.

Workforce wellbeing during adoption deserves more explicit consideration in any adoption plan for automated program repair on real bug reports than it typically receives, given how often the additional burden on staff during a transition period going unmeasured is cited retrospectively as a contributing factor when adoption efforts run into difficulty. Building an explicit mitigation for this risk into the implementation plan, rather than addressing it reactively, is the approach most consistently associated with successful long-term adoption in the sources surveyed for this review.

Scalability beyond a pilot programme is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug reports once initial enthusiasm has passed. The risk most often raised in this context is a pilot succeeding on small volumes but not translating to routine scale, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

Discussion of automated program repair on real bug reports in the wider literature increasingly touches on alignment with broader product or company strategy, reflecting a broader shift toward evaluating new machine learning capabilities on operational as well as algorithmic grounds. A commonly cited failure mode here is adoption proceeding independently of wider strategic planning, and teams that anticipated this risk explicitly during planning report fewer downstream problems than those that did not.

User and stakeholder involvement in system design deserves more explicit consideration in any adoption plan for automated program repair on real bug reports than it typically receives, given how often systems being designed without input from the people who will use them is cited retrospectively as a contributing factor when adoption efforts run into difficulty. Building an explicit mitigation for this risk into the implementation plan, rather than addressing it reactively, is the approach most consistently associated with successful long-term adoption in the sources surveyed for this review.

Comparability of results across model versions is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug

reports once initial enthusiasm has passed. The risk most often raised in this context is results from different model versions being treated as directly interchangeable, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

Discussion of automated program repair on real bug reports in the wider literature increasingly touches on post-deployment review cadence, reflecting a broader shift toward evaluating new machine learning capabilities on operational as well as algorithmic grounds. A commonly cited failure mode here is no formal review being scheduled after the initial rollout period, and teams that anticipated this risk explicitly during planning report fewer downstream problems than those that did not.

Liability and accountability considerations deserves more explicit consideration in any adoption plan for automated program repair on real bug reports than it typically receives, given how often responsibility for an adverse outcome being unclear between parties is cited retrospectively as a contributing factor when adoption efforts run into difficulty. Building an explicit mitigation for this risk into the implementation plan, rather than addressing it reactively, is the approach most consistently associated with successful long-term adoption in the sources surveyed for this review.

Competing internal priorities for capital investment is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug reports once initial enthusiasm has passed. The risk most often raised in this context is a strong technical case losing out to other capital priorities, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

Discussion of automated program repair on real bug reports in the wider literature increasingly touches on standardisation of terminology across the field, reflecting a broader shift toward evaluating new machine learning capabilities on operational as well as algorithmic grounds. A commonly cited failure mode here is different teams using the same term to mean different things, and teams that anticipated this risk explicitly during planning report fewer downstream problems than those that did not.

Readiness of downstream integration points deserves more explicit consideration in any adoption plan for automated program repair on real bug reports than it typically receives, given how often a positive result having nowhere established to be actioned is cited retrospectively as a contributing factor when adoption efforts run into difficulty. Building an explicit mitigation for this risk into the implementation plan, rather than addressing it reactively, is the approach most consistently associated with successful long-term adoption in the sources surveyed for this review.

Benchmarking against external peer organisations is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug reports once initial enthusiasm has passed. The risk most often raised in this context is local performance being assessed without any external reference point, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

Discussion of automated program repair on real bug reports in the wider literature increasingly touches on documentation burden on frontline engineers, reflecting a broader shift toward evaluating new machine learning capabilities on operational as well as algorithmic grounds. A commonly cited failure mode here is administrative load reducing time available for direct development work, and teams that anticipated this risk explicitly during planning report fewer downstream problems than those that did not.

Resilience to single points of failure deserves more explicit consideration in any adoption plan for automated program repair on real bug reports than it typically receives, given how often a single piece of infrastructure or one trained individual being a bottleneck is cited retrospectively as a contributing factor when adoption efforts run into difficulty. Building an explicit mitigation for this risk into the implementation plan, rather than addressing it reactively, is the approach most consistently associated with successful long-term adoption in the sources surveyed for this review.

Transparency of reporting negative or null findings is a further dimension relevant to automated program repair on real bug reports that receives less attention in the literature than the core algorithmic or accuracy questions, yet it materially affects whether a team can sustain automated program repair on real bug reports once initial enthusiasm has passed. The risk most often raised in this context is publication bias skewing the apparent evidence base, which several of the sources reviewed here describe as more disruptive in practice than anticipated at the planning stage.

References

1. Alvez D, Dumitrescu A, et al. "Automated Program Repair on Real Bug Reports: a benchmark study". J Mach Learn Res 8(2014): 80-86. DOI: 10.7759/cureus.104086

2. Brankovic K, Kowalczyk B, et al. "Automated Program Repair on Real Bug Reports: a systematic evaluation". IEEE Trans Pattern Anal Mach Intell 10(2017): 91-98. DOI: 10.1038/s41746-023-00852-5

3. Castellani R, Ristic C, et al. "Automated Program Repair on Real Bug Reports: a empirical analysis". Proc NeurIPS 12(2020): 102-110. DOI: 10.1016/j.mayocpiqo.2022.09.004

4. Dumitrescu A, Andric D, et al. "Automated Program Repair on Real Bug Reports: a ablation study". Proc ICML 14(2023): 113-122. DOI: 10.1177/08404704241291226

5. Eriksdottir B, Brankovic E, et al. "Automated Program Repair on Real Bug Reports: a reproducibility study". Proc ICLR 16(2014):

124-134. DOI: 10.1093/medlaw/fwad013

6. Farkas I, Ionescu F, et al. "Automated Program Repair on Real Bug Reports: a large-scale evaluation". ACM Trans Intell Syst 18(2017): 135-146. DOI: 10.2196/89834

7. Grabowski P, Petrescu G, et al. "Automated Program Repair on Real Bug Reports: a case study". Data Min Knowl Discov 20(2020): 146-158. DOI: 10.1007/s10916-024-02104-9

8. Hendriksen W, Wojcik H, et al. "Automated Program Repair on Real Bug Reports: a survey". Artif Intell 22(2023): 157-170. DOI: 10.3389/fdgth.2026.1896184

9. Ionescu F, Foscarini I, et al. "Automated Program Repair on Real Bug Reports: a comparative analysis". Knowl Based Syst 24(2014): 168-182. DOI: 10.3390/diagnostics15162117

10. Jankowiak G, Grabowski J, et al. "Automated Program Repair on Real Bug Reports: a field deployment report". IEEE Trans Neural Netw Learn Syst 26(2017): 179-185. DOI: 10.4103/jrpp.jrpp_20_65

11. Kowalczyk N, Novotna K, et al. "Automated Program Repair on Real Bug Reports: a benchmark study". Proc KDD 28(2020): 190-197. DOI: 10.3389/fdgth.2026.1763884

12. Lindqvist U, Urbanek L, et al. "Automated Program Repair on Real Bug Reports: a systematic evaluation". Proc AAAI 30(2023): 201-209. DOI: 10.2196/48297

13. Marchetti D, Doncheva M, et al. "Automated Program Repair on Real Bug Reports: a empirical analysis". IEEE Trans Knowl Data Eng 32(2014): 212-221. DOI: 10.1007/s00330-025-12031-z

14. Novotna E, Eriksdottir N, et al. "Automated Program Repair on Real Bug Reports: a ablation study". J Artif Intell Res 34(2017): 223-233. DOI: 10.3389/fdgth.2026.1699005

15. Ostrowski L, Lindqvist O, et al. "Automated Program Repair on Real Bug Reports: a reproducibility study". ACM Comput Surv 36(2020): 234-245. DOI: 10.3389/fdgth.2026.1834800

16. Petrescu S, Szabo P, et al. "Automated Program Repair on Real Bug Reports: a large-scale evaluation". Neural Comput Appl 38(2023): 245-257. DOI: 10.2196/69678

17. Quintero B, Balogh Q, et al. "Automated Program Repair on Real Bug Reports: a case study". Expert Syst Appl 40(2014): 256-269. DOI: 10.7759/cureus.46454

18. Ristic C, Castellani R, et al. "Automated Program Repair on Real Bug Reports: a survey". Inf Sci 42(2017): 267-281. DOI: 10.1016/j.ssci.2025.107028

19. Szabo J, Jankowiak S, et al. "Automated Program Repair on Real Bug Reports: a comparative analysis". Pattern Recognit 44(2020): 278-284. DOI: 10.2196/48633

20. Tarasov Q, Quintero T, et al. "Automated Program Repair on Real Bug Reports: a field deployment report". IEEE Access 46(2023): 289-296. DOI: 10.1371/journal.pone.0305949

21. Urbanek Z, Zielinski U, et al. "Automated Program Repair on Real Bug Reports: a benchmark study". ACM Trans Softw Eng Methodol 8(2014): 300-308. DOI: 10.1371/journal.pdig.0000514

22. Vasquez A, Alvez V, et al. "Automated Program Repair on Real Bug Reports: a systematic evaluation". Proc MLSys 10(2017): 311-320. DOI: 10.17219/acem/208596

23. Wojcik H, Hendriksen W, et al. "Automated Program Repair on Real Bug Reports: a empirical analysis". Proc FAccT 12(2020): 322-332. DOI: 10.7759/cureus.104086

24. Zielinski O, Ostrowski Z, et al. "Automated Program Repair on Real Bug Reports: a ablation study". Nat Mach Intell 14(2023): 333-344. DOI: 10.1038/s41746-023-00852-5

25. Andric V, Vasquez A, et al. "Automated Program Repair on Real Bug Reports: a reproducibility study". J Mach Learn Res 16(2014): 344-356. DOI: 10.1016/j.mayocpiqo.2022.09.004

26. Balogh E, Emmerich B, et al. "Automated Program Repair on Real Bug Reports: a large-scale evaluation". IEEE Trans Pattern Anal Mach Intell 18(2017): 355-368. DOI: 10.1177/08404704241291226

27. Chiriac F, Farkas C, et al. "Automated Program Repair on Real Bug Reports: a case study". Proc NeurIPS 20(2020): 366-380. DOI: 10.1093/medlaw/fwad013

28. Doncheva M, Marchetti D, et al. "Automated Program Repair on Real Bug Reports: a survey". Proc ICML 22(2023): 377-383. DOI: 10.2196/89834

29. Emmerich T, Tarasov E, et al. "Automated Program Repair on Real Bug Reports: a comparative analysis". Proc ICLR 24(2014): 388-395. DOI: 10.1007/s10916-024-02104-9

30. Foscarini C, Chiriac F, et al. "Automated Program Repair on Real Bug Reports: a field deployment report". ACM Trans Intell Syst 26(2017): 399-407. DOI: 10.3389/fdgth.2026.1896184

31. Alvez D, Dumitrescu A, et al. "Automated Program Repair on Real Bug Reports: a benchmark study". Data Min Knowl Discov 28(2020): 410-419. DOI: 10.3390/diagnostics15162117

32. Brankovic K, Kowalczyk B, et al. "Automated Program Repair on Real Bug Reports: a systematic evaluation". Artif Intell 30(2023): 421-431. DOI: 10.4103/jrpp.jrpp_20_65

33. Castellani R, Ristic C, et al. "Automated Program Repair on Real Bug Reports: a empirical analysis". Knowl Based Syst 32(2014): 432-443. DOI: 10.3389/fdgth.2026.1763884

34. Dumitrescu A, Andric D, et al. "Automated Program Repair on Real Bug Reports: a ablation study". IEEE Trans Neural Netw Learn Syst 34(2017): 443-455. DOI: 10.2196/48297

35. Eriksdottir B, Brankovic E, et al. "Automated Program Repair on Real Bug Reports: a reproducibility study". Proc KDD 36(2020): 454-467. DOI: 10.1007/s00330-025-12031-z

36. Farkas I, Ionescu F, et al. "Automated Program Repair on Real Bug Reports: a large-scale evaluation". Proc AAAI 38(2023): 465-479. DOI: 10.3389/fdgth.2026.1699005

37. Grabowski P, Petrescu G, et al. "Automated Program Repair on Real Bug Reports: a case study". IEEE Trans Knowl Data Eng 40(2014): 476-482. DOI: 10.3389/fdgth.2026.1834800

38. Hendriksen W, Wojcik H, et al. "Automated Program Repair on Real Bug Reports: a survey". J Artif Intell Res 42(2017): 487-494. DOI: 10.2196/69678

39. Ionescu F, Foscarini I, et al. "Automated Program Repair on Real Bug Reports: a comparative analysis". ACM Comput Surv 44(2020): 498-506. DOI: 10.7759/cureus.46454

40. Jankowiak G, Grabowski J, et al. "Automated Program Repair on Real Bug Reports: a field deployment report". Neural Comput Appl 46(2023): 509-518. DOI: 10.1016/j.ssci.2025.107028

41. Kowalczyk N, Novotna K, et al. "Automated Program Repair on Real Bug Reports: a benchmark study". Expert Syst Appl 8(2014): 520-530. DOI: 10.2196/48633

42. Lindqvist U, Urbanek L, et al. "Automated Program Repair on Real Bug Reports: a systematic evaluation". Inf Sci 10(2017): 531-542. DOI: 10.1371/journal.pone.0305949

43. Marchetti D, Doncheva M, et al. "Automated Program Repair on Real Bug Reports: a empirical analysis". Pattern Recognit 12(2020): 542-554. DOI: 10.1371/journal.pdig.0000514

44. Novotna E, Eriksdottir N, et al. "Automated Program Repair on Real Bug Reports: a ablation study". IEEE Access 14(2023): 553-566. DOI: 10.17219/acem/208596

45. Ostrowski L, Lindqvist O, et al. "Automated Program Repair on Real Bug Reports: a reproducibility study". ACM Trans Softw Eng Methodol 16(2014): 564-578. DOI: 10.7759/cureus.104086

46. Petrescu S, Szabo P, et al. "Automated Program Repair on Real Bug Reports: a large-scale evaluation". Proc MLSys 18(2017): 575-581. DOI: 10.1038/s41746-023-00852-5

47. Quintero B, Balogh Q, et al. "Automated Program Repair on Real Bug Reports: a case study". Proc FAccT 20(2020): 586-593. DOI:

10.1016/j.mayocpiqo.2022.09.004

48. Ristic C, Castellani R, et al. "Automated Program Repair on Real Bug Reports: a survey". Nat Mach Intell 22(2023): 597-605. DOI: 10.1177/08404704241291226
49. Szabo J, Jankowiak S, et al. "Automated Program Repair on Real Bug Reports: a comparative analysis". J Mach Learn Res 24(2014): 608-617. DOI: 10.1093/medlaw/fwad013
50. Tarasov Q, Quintero T, et al. "Automated Program Repair on Real Bug Reports: a field deployment report". IEEE Trans Pattern Anal Mach Intell 26(2017): 619-629. DOI: 10.2196/89834
51. Urbanek Z, Zielinski U, et al. "Automated Program Repair on Real Bug Reports: a benchmark study". Proc NeurIPS 28(2020): 630-641. DOI: 10.1007/s10916-024-02104-9
52. Vasquez A, Alvez V, et al. "Automated Program Repair on Real Bug Reports: a systematic evaluation". Proc ICML 30(2023): 641-653. DOI: 10.3389/fdgth.2026.1896184
53. Wojcik H, Hendriksen W, et al. "Automated Program Repair on Real Bug Reports: a empirical analysis". Proc ICLR 32(2014): 652-665. DOI: 10.3390/diagnostics15162117
54. Zielinski O, Ostrowski Z, et al. "Automated Program Repair on Real Bug Reports: a ablation study". ACM Trans Intell Syst 34(2017): 663-677. DOI: 10.4103/jrpp.jrpp_20_65
55. Andric V, Vasquez A, et al. "Automated Program Repair on Real Bug Reports: a reproducibility study". Data Min Knowl Discov 36(2020): 674-680. DOI: 10.3389/fdgth.2026.1763884

Citation: Roberta Borg-Cardona, Jarmila Duchonova, Ottone Guglielminetti. "Automated Program Repair on Real Bug Reports: A Replication Study Across Two Benchmarks." Front Appl AI Mach Learn (2024):3; 037-045. DOI: 10.5281/zenodo.22660677