Right Fix, Wrong Reason: Repair and Diagnosis in a CHIA RTL Debugging Loop

Samarth Narang
sn2910@nyu.edu
New York University
New York, NY, USA

Brandon Reagen
bjr5@nyu.edu
New York University
New York, NY, USA

Abstract

The tests used to validate an RTL repair do not check the diagnosis supplied with the patch. We present YARROW, a debugging system built on CHIA that constructs and verifies a patch, then uses simulation of that patch to support the diagnosis. During repair, compilation checks reject invalid edits, independent test runs verify candidates, and a fixed budget is divided across fresh repair attempts. Once a patch passes, the agent replays the original and patched designs under identical inputs and identifies the first internal signal whose value differs. A separate comparison between the original design and a golden reference supplies the label for checking the agent's claimed signal, with bounded retries when the claim disagrees. We evaluate three Gemini models on 82 RTL bugs, including 14 from open-source hardware projects. In baseline runs, Gemini 3.1 Pro repairs every measured instance of these 14 bugs but matches the reference-derived signal label in only 64% of them. On the same 14 bugs, YARROW increases the fraction of evaluations achieving both a passing repair and a matching diagnosis from 64% to 86% for Gemini 3.1 Pro and from 25% to 64% for Gemini 2.5 Flash. Flash's repair rate also increases from 50% to 71%.

Keywords

RTL debugging, automated program repair, root-cause diagnosis, LLM agents, CHIA

ACM Reference Format:

Samarth Narang and Brandon Reagen. 2026. Right Fix, Wrong Reason: Repair and Diagnosis in a CHIA RTL Debugging Loop. In *A^3: Agentic Approaches to Architecture Workshop at MICRO 2026; Hackathon Short Paper, Nov 01, 2026, Athens, Greece.* ACM, New York, NY, USA, 5 pages.

1 Introduction

An engineer reviewing an RTL repair needs to understand both what the patch changes and why that change addresses the failure. A debugging agent can propose edits, run simulations, and use test results to revise its patch, but those tests do not check the diagnosis it supplies. The patch may pass while the accompanying report identifies a signal that does not match the observed failure. Evaluating the repair therefore leaves part of the debugging result unchecked.

We study this gap through a diagnosis target obtained from simulation. Given a faulty design and a compatible golden reference, we run both under identical inputs and record the signals whose values first differ. We then check whether the signal named by the agent belongs to that set. This provides a reproducible way to evaluate one part of the diagnosis without using another language model as a judge. In our baseline experiments, Gemini 3.1 Pro repairs every measured instance of 14 bugs from open-source hardware projects, yet its claimed signal matches this target in only 64% of those repairs.

YARROW (Yet Another RTL Root-cause Oracle Workflow) addresses patch construction and diagnosis in separate stages of a CHIA [3] debugging loop. During repair, the agent works within a fixed budget divided across independent repair attempts, each starting from the original design. An edit tool rolls back changes that break compilation, while a separate verification step rebuilds and tests candidate patches. These controls help the agent search for a passing repair without spending the entire budget on one sequence of unsuccessful edits.

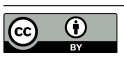
Once a patch passes, the agent uses self-replay to compare the original and patched designs under the same inputs. This comparison shows where the submitted repair first changes the observed execution and gives the agent evidence for its diagnosis. A separate check compares the claimed signal with the label derived from the golden reference and can request a bounded revision. The two comparisons serve distinct purposes: self-replay supplies evidence about the agent's own repair, while the reference comparison supplies a common target for evaluation.

We characterize repair and diagnosis across three Gemini models and 82 bugs drawn from synthetic mutations, existing benchmarks, and open-source hardware projects. We then evaluate YARROW on the 14 bugs from those projects, reporting both repair success and the fraction of evaluations that produce a passing patch with a matching diagnosis. For Gemini 2.5 Flash, YARROW raises repair success from 50% to 71% and this joint success rate from 25% to 64%. For Gemini 3.1 Pro, repair success remains 100% while joint success rises from 64% to 86%. The loop, benchmark, results, and scorer are available in the project artifact [9]; full transcripts are available from the authors on request.

2 Benchmark

We evaluate 82 RTL bugs spanning synthetic cache-controller mutations, an existing repair benchmark, and open-source hardware projects. All 82 bugs support the baseline study; YARROW is evaluated on the 14 open-source project bugs.

The synthetic collection contains 44 faulty versions (29 single-fault and 15 composed) of a roughly 400-line, two-way set-associative write-back cache controller in SystemVerilog. A test suite spanning eight cache configurations detects every faulty version.



This work is licensed under a Creative Commons Attribution 4.0 International License.
A^3 Workshop at MICRO 2026; Hackathon Short Paper, Athens, Greece
© 2026 Copyright held by the owner/author(s).

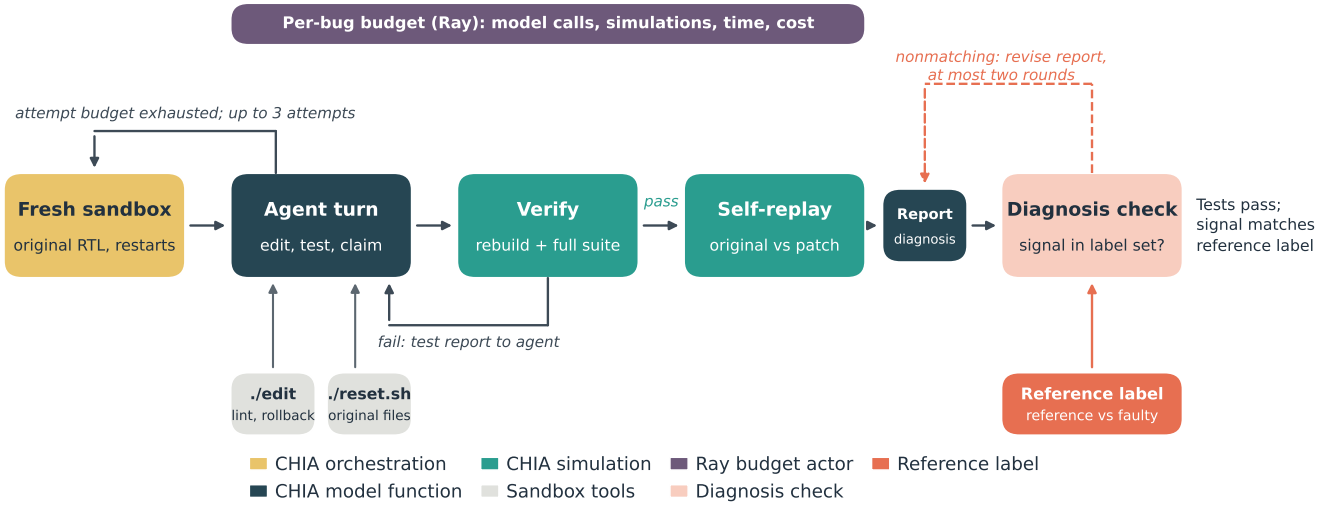


Figure 1: YARROW's CHIA workflow separates repair verification from diagnosis checking. After a patch passes, self-replay supports the agent's diagnosis, which is compared with a reference-derived label. Up to three fresh repair attempts share one per-bug budget, with at most two diagnosis-feedback rounds within the remaining budget.

The external benchmark collection contains 24 validated cases from CirFix [1], including decoders, multiplexers, state machines, counters, and shift registers. We retain these cases from a collection of 44 imported cases; the accompanying artifact records the excluded cases and the reasons for exclusion. This collection tests whether the repair–diagnosis gap also appears in bugs authored outside our study.

The open-source project collection contains 14 bugs across 11 designs from the FPGA debugging study by Ma et al. [8], using the versions distributed with RTL-Repair [6]. The designs include AXI stream cores from verilog-axis [4], Xilinx AXI interface templates, the ZipCPU SD-card controller [5], and an IEEE-754 adder. They range from 122 to 1,286 lines across one to four source files. Each case uses a cycle-by-cycle table recorded from a correct execution. The reference design passes these checks and the faulty design fails them in all 14 cases.

Each benchmark case provides faulty and reference sources, a self-checking Verilator [10] harness, and a diagnosis label obtained from paired simulations. We run the faulty and reference designs under identical inputs and compare their signal traces. At the earliest simulation tick with a value mismatch, we retain all differing corresponding signals, excluding top-level inputs. Signals present in only one trace are excluded because their absence represents a structural difference rather than a value mismatch. A diagnosis matches the reference label when the base name of the signal named by the agent belongs to the first-divergence set. Retaining all signals that differ at the earliest tick avoids choosing an arbitrary target when several signals diverge simultaneously. These sets contain one to four signals for the open-source project bugs and up to 20 for the synthetic bugs. We manually checked all 14 project-bug labels against the upstream fixes. Each evaluation is then assessed for both repair and diagnosis, with joint success requiring a patch that passes independent verification and a signal claim that matches the reference label.

We use these reference labels alongside independent verification to assess both the repair and its diagnosis. An *evaluation* is one complete run of a workflow on one bug and is the unit that is scored; an evaluation comprises up to three *repair attempts*, each starting from the original design (Section 3). *Repair rate* is the fraction of scored evaluations that pass independent verification, and *exact agreement* is the fraction of passing repairs whose claimed signal matches the reference label. *Joint success* is the fraction of all scored evaluations that produce both a passing repair and a matching diagnosis. Baseline conditions use two evaluations per bug, except Gemini 2.5 Pro on the 14 bugs from open-source projects, which has one.

3 The YARROW Workflow

YARROW implements the debugging loop in Fig. 1 using CHIA functions scheduled by Ray. The agent proposes RTL changes to repair the bug, and each submitted patch is rebuilt and tested independently. Once a patch passes, the agent uses simulation of the original and patched designs to support its diagnosis.

3.1 Repair and verification

Before the agent begins, the harness rebuilds the original design and reproduces the failure. The agent then works in a sandbox through CHIA's BashTool, with commands for running the test suite, simulating individual configurations, and querying waveforms where available. An edit command rolls back changes that prevent compilation, while a reset command restores the original sources.

Each submitted candidate is rebuilt and tested by a separate verification function. Failed verification returns the test report to the agent for another repair turn. The per-bug budget is divided

Table 1: Baseline workflow: repair rate and exact agreement among repairs (%), before any workflow changes. X: 24 CirFix cases; S: 29 single-fault and 15 composed cache-controller mutants; R: 14 bugs from open-source projects.

	X		S single		S composed		R	
Gemini	Rep.	Agr.	Rep.	Agr.	Rep.	Agr.	Rep.	Agr.
2.5 Flash	100	79	93	74	77	48	50	50
2.5 Pro	100	96	97	82	93	64	64	89
3.1 Pro	100	100	100	78	100	80	100	64

across up to three independent repair attempts, each starting in a fresh sandbox with no shared agent history. If a repair attempt exhausts its allocation without producing a passing patch, the next repair attempt starts from the original design. The first passing patch proceeds to diagnosis checking.

3.2 Diagnosis from self-replay

After verification passes, the agent uses self-replay to examine how its patch changes the design's execution. The command simulates the original and patched designs under identical inputs and reports their first internal divergence, using the comparison rules described in the benchmark section. The agent uses this evidence to submit a diagnosis that names a signal and its module and describes the event and failure mechanism. This evidence comes from the agent's own patch; self-replay does not consult the reference label.

YARROW then checks the signal named in the diagnosis against the reference label. If it does not match, the agent is asked to re-examine its diagnosis, with at most two additional feedback rounds within the remaining budget. Diagnosis retries request only a corrected report. The recorded repair remains the patch that passed independent verification before these retries; edits made during a retry are not re-verified. The loop ends when both the patch and diagnosis pass their respective checks, or when the remaining budget or permitted feedback rounds are exhausted.

3.3 CHIA execution and resource control

CHIA functions declare separate orchestration, model, and simulation resources. A Ray budget actor records model calls, simulation work, elapsed time, and token-based cost, with repair-attempt allocations drawn from the per-bug budget. Per-bug budgets are 12 model calls, 96 simulations, and 1,800 s for single-fault synthetic and external benchmark cases, and 20 model calls, 160 simulations, and 2,700 s for composed-fault synthetic and open-source project cases. Post-repair-check evaluations on single-fault synthetic cases used 14 model calls and 110 simulations. Design descriptors specify the RTL sources and test configuration, allowing additional designs to be evaluated using the existing solver, scorer, and model adapters.

4 Results

4.1 Baseline repair and diagnosis

Table 1 shows that passing tests does not ensure a matching diagnosis. Gemini 3.1 Pro repairs all 88 synthetic evaluations and all 28 evaluations on the open-source project bugs, completing each project repair in one model call. Exact agreement is nevertheless

Table 2: Repair rate, joint success (%), and scored evaluations (n) on 14 bugs from open-source projects.

	2.5 Flash			2.5 Pro			3.1 Pro		
Workflow	Rep.	Joint	n	Rep.	Joint	n	Rep.	Joint	n
Baseline	50	25	28	64	57	14	100	64	28
Post-repair check	29	25	28	–	–	–	100	92*	26*
Restarts + check	61	43	28	69	54	13†	100	86	14
YARROW	**71**	**64**	28	**71**	**68**	28	**100**	**86**	28

*24/26 valid evaluations (92%); including two context-overflow errors gives 24/28 (86%).
†One incomplete evaluation excluded. YARROW rows use two evaluations per bug per model. A third complete Flash evaluation of the 14 bugs (9 repairs, 8 joint successes), run after the primary two, is reported separately so that all models are compared over the same number of evaluations; pooling it gives 29/42 (69%) repair and 26/42 (62%) joint success.

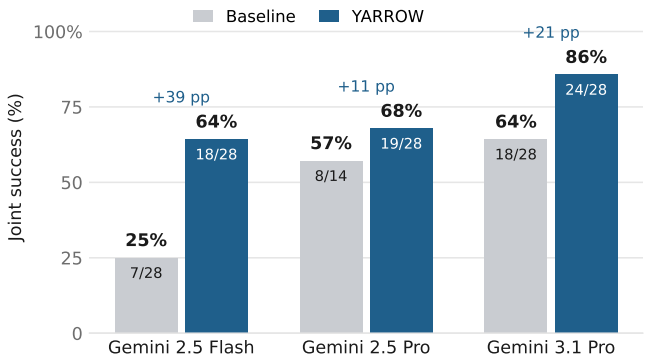


Figure 2: Joint success on 14 bugs from open-source projects, requiring both a passing repair and a matching diagnosis. Fractions give successes over scored evaluations; gains are in percentage points.

78% on single-fault mutants, 80% on composed mutants, and 64% on project bugs. On the external benchmark cases, all three models repair every case; agreement is 100% for 3.1 Pro, 96% for 2.5 Pro, and 79% for Flash.

The disagreement often persists even when the agent produces the reference patch. Across 249 successful synthetic repairs, 65 have nonmatching diagnoses, and 62 of those 65 (95%) reproduce the reference patch, with about 1.3 model calls on average. For example, on a synthetic bug caused by an incorrect bit slice, 2.5 Pro fixes the slice and passes verification, but names a signal that does not match the reference label. The reverse occurs on an AXI-Lite design, where Flash identifies the labeled signal but fails to produce a passing repair after 12 calls and 103 changed lines. These cases show why repair success and diagnosis agreement need separate measurements.

4.2 Repair and diagnosis with YARROW

Table 2 compares YARROW with the baseline and intermediate workflows on 14 project bugs. The primary comparison uses two evaluations per bug per model (28 per model); an additional Flash evaluation is reported separately in the table note. Figure 2 highlights joint repair-and-diagnosis success.

For Flash, YARROW produces 20/28 passing repairs (71%) and 18/28 joint successes (64%), compared with baseline rates of 50% and 25%, respectively. Eighteen of its twenty passing repairs have matching diagnoses, giving 90% exact agreement. It also repairs two bugs that remained unfixed in ten earlier runs.

For 2.5 Pro, YARROW produces 20/28 passing repairs and 19/28 joint successes (68%), compared with 57% joint success in its single baseline evaluation. Every passing repair in the second YARROW evaluation has a matching diagnosis. For 3.1 Pro, all 28 evaluations pass verification and 24/28 achieve joint success (86%). Both evaluations produce the same outcome for each bug, and all 24 matching claims are obtained without diagnosis retries. The remaining 3.1 Pro disagreements recur on two bugs in both evaluations, where passing patches introduce a differently named net in a verilog-axis FIFO (`d4` [4]) or repair a write-side byte-order mismatch in the ZipCPU SD-card controller for which the reference patch changes the read side (`d9` [5]). Bug identifiers follow the RTL-Repair benchmark [6]. We next examine the experiments that motivated YARROW's design choices.

4.3 Experiments motivating YARROW's design

The supporting experiments examine two difficulties: producing a passing repair and reporting a diagnosis that agrees with the reference label. With post-repair diagnosis checking alone, Flash reaches 29% repair and 25% joint success. For 3.1 Pro, all 26 valid evaluations produce passing repairs and 24 achieve signal agreement, 92% among 26 valid evaluations. Over all 28 evaluations the same 24 successes give 86% (24/28), the figure comparable with YARROW's 86%; the gap is a denominator difference, not a change in outcome.

Giving Flash the reference-derived signals and divergence cycle during repair yields 50% repair, the same as the baseline. In the unsuccessful cases, 37% of turns follow edits that break compilation. This motivates YARROW's edit tool, which rolls back changes that fail to compile. To help the agent recover from unsuccessful repair attempts, we also evaluate restarts from the original design. Dividing the budget across three fresh repair attempts raises Flash's repair rate to 61%, with seven of its 17 repairs coming from the second or third repair attempt. The same restart condition yields 69% repair and 54% joint success for 2.5 Pro on 13 valid evaluations, and 100% repair and 86% joint success for 3.1 Pro on 14. YARROW adopts this allocation, allowing up to three repair attempts within the same per-bug budget.

After a patch passes, feedback can help the agent revise its diagnosis. On the single-fault synthetic bugs, asking Flash to reconsider a nonmatching claim raises agreement from 78% before feedback to 91% afterward, correcting seven of twelve claims while repair remains at 93%. This motivates checking the diagnosis separately after repair. In YARROW, the agent first uses self-replay to examine how its patch changes execution, then revisits its diagnosis if the reported signal disagrees with the reference label, with at most two feedback rounds.

5 Related Work and Limitations

CirFix [1] and RTL-Repair [6] distinguish test-passing repairs from stronger evidence of functional correctness. AutoChip [11] and RTLFixer [12] use tool feedback for HDL generation and syntax repair, respectively, while R3A [7] combines localization with patch search. CHIA [3] supports the composition and evaluation of agent workflows, and RTLScout [2] applies iterative feedback to circuit optimization. YARROW complements this work by evaluating the diagnosis accompanying a repair and measuring its agreement with a reference label alongside independently verified repair success.

Our diagnosis score checks whether the agent names one of the signals that first differ between the faulty and reference designs under the supplied tests. It does not verify the reported failure event or mechanism. Different inputs can produce a different first divergence, and base-name matching does not distinguish identically named signals in different modules. Self-replay similarly shows where the patch first changes execution, but does not establish which change fixes the bug. Lastly, constructing the diagnosis label requires a reference whose internal signals can be matched to those of the faulty design under the same test inputs.

We evaluate YARROW on 14 bugs from open-source hardware projects, with a limited number of evaluations per bug due to budget constraints. The sample limits generalization to other designs and bug types, while the small number of evaluations per bug limits our ability to characterize variation across runs. Scalability to larger RTL designs remains unevaluated.

6 Conclusion

YARROW separates RTL repair from diagnosis, using independent verification to check the patch and replay of the original and patched designs to support the reported signal. On 14 bugs from open-source hardware projects, it raises joint success from 25% to 64% for Gemini 2.5 Flash and from 64% to 86% for Gemini 3.1 Pro. The remaining 3.1 Pro disagreements involve passing repairs that differ from the reference fix, showing why signal agreement should be interpreted alongside the submitted patch.

7 Artifact Information

The public MIT-licensed artifact at https://github.com/snarang181/yarrow [9] includes the YARROW CHIA loop, all 82 benchmark cases, simulation-derived labels, validation and scoring tools, and scripts and per-cell summaries to regenerate the tables and figures. Re-running experiments requires Verilator 5.x, CHIA (cloned by the setup script), and Gemini access through Vertex AI. Note that model outputs are stochastic, so reruns may reproduce the overall trends but not the exact counts.

Acknowledgment of AI Assistance

Claude assisted with implementation, harness debugging, experiment execution, and the initial manuscript. ChatGPT assisted with literature review, editing, and preparation of this submission. The authors designed the study, reviewed the code and results, and are responsible for the paper's contents, tone, and quality.

Acknowledgments

This work was supported by NSF CAREER award #2340137, NSF CIRC GRAND #2450539, and NSF NeTs #2504400. The views, opinions, and/or findings expressed are those of the authors and do not necessarily reflect the views of sponsors.

References

[1] Hammad Ahmad, Yu Huang, and Westley Weimer. 2022. CirFix: automatically repairing defects in hardware design code. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 990–1003. doi:10.1145/3503222.3507763

[2] Felix Arnold, Ryan Amaudruz, Dimitrios Tsaras, Renzo Andri, and Lukas Cavigelli. 2026. RTLScout: Joint Agentic Code and Synthesis Optimization for Efficient Digital Circuits. arXiv:2606.06530. arXiv:2606.06530 [cs.AR] https://arxiv.org/abs/2606.06530

[3] Angela Cui, Ferran Hermida-Rivera, Jack Toubes, Raghav Gupta, Jim Fang, Chengyi Lux Zhang, Ella Schwarz, Junha Kim, Yakun Sophia Shao, Borivoje Nikolic, Christopher W. Fletcher, and Sagar Karandikar. 2026. CHIA: An open-source framework for principled, agentic AI-driven hardware/software co-design research. arXiv:2606.27350. arXiv:2606.27350 [cs.AR] https://arxiv.org/abs/2606.27350

[4] Alex Forencich. 2026. verilog-axis: Verilog AXI stream components. https://github.com/alexforencich/verilog-axis.

[5] Dan Gisselquist. 2026. sdspi: SD card controller for the ZipCPU. https://github.com/ZipCPU/sdspi.

[6] Kevin Laeufer, Brandon Fajardo, Abhik Ahuja, Vighnesh Iyer, Borivoje Nikolić, and Koushik Sen. 2024. RTL-Repair: Fast Symbolic Repair of Hardware Design Code. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 867–881. doi:10.1145/3620666.3651346

[7] Zizhang Luo, Fan Cui, Kexing Zhou, Runlin Guo, Mile Xia, Hongyuan Hou, and Yun Liang. 2025. R3A: Reliable RTL Repair Framework with Multi-Agent Fault Localization and Stochastic Tree-of-Thoughts Patch Generation. arXiv:2511.20090. arXiv:2511.20090 [cs.AR] https://arxiv.org/abs/2511.20090

[8] Jiacheng Ma, Gefei Zuo, Kevin Loughlin, Haoyang Zhang, Andrew Quinn, and Baris Kasikci. 2022. Debugging in the brave new world of reconfigurable hardware. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 946–962. doi:10.1145/3503222.3507701

[9] S. Narang. 2026. YARROW: Benchmark, loop, results and scorer. https://github.com/snarang181/yarrow.

[10] Wilson Snyder, Paul Wasson, Duane Galbi, Geza Lore, et al. 2026. Verilator. https://github.com/verilator/verilator. Version 5.x.

[11] Shailja Thakur, Jason Blocklove, Hammond Pearce, Benjamin Tan, Siddharth Garg, and Ramesh Karri. 2024. AutoChip: Automating HDL Generation Using LLM Feedback. arXiv:2311.04887. arXiv:2311.04887 [cs.PL] https://arxiv.org/abs/2311.04887

[12] Yunda Tsai, Mingjie Liu, and Haoxing Ren. 2024. RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (San Francisco, CA, USA) *(DAC '24)*. Association for Computing Machinery, New York, NY, USA, Article 53, 6 pages. doi:10.1145/3649329.3657353