Fail2Bench: Turning RTL Agent Failures into a Self-Curating Benchmark

Aryan Chhabra Sumedh Narahari Shritan Settipalli Aadarsh Sivaraman

Abstract

LLM-based agents are increasingly used for RTL design and repair, yet benchmark suites remain static: once a suite is solved, it no longer reveals the agent's current failure modes. We present **Fail2Bench**, a closed-loop failure-to-benchmark framework for RTL repair agents. Fail2Bench executes agent repairs against real Verilog toolchains (Icarus Verilog and Verilator), mines reproducible failures into structured benchmark cards with provenance, shrinks failing artifacts via line-level delta debugging, assigns controlled-vocabulary failure tags, deduplicates near-duplicate cards via normalized AST hashes and sentence embeddings, and re-evaluates agents on the growing suite. On a 42-task hand-crafted Verilog repair suite, a real-toolchain validation pass uncovered nine silently-passing "buggy" seeds; after repair, all 42 seeds fail before any agent acts. We report reproducible baselines for a no-op agent ($pass@1 = 0/42$), a deterministic pattern-repair agent ($pass@1 = 3/42$), and a free local 1.5B-parameter code LLM ($pass@1 = 3/18$ on the easy subset). Fail2Bench is released as open-source infrastructure to study agent-driven hardware repair under an evaluation surface that evolves with the agents being evaluated.

Keywords—benchmark, RTL repair, hardware design, LLM agents, failure mining, delta debugging, self-curating evaluation, agentic AI

1. INTRODUCTION

LLM-based coding agents are routinely evaluated on fixed benchmark suites [4, 6, 2]. In hardware repair this is especially brittle: a passing score can reflect memorized bug templates, under-specified testbenches, or stale benchmark coverage once agents grow stronger. Benchmark contamination in pretraining data compounds the problem [7]. The standard response—adding more tasks—requires sustained manual effort and still yields a snapshot that ages immediately.

Fail2Bench asks a different question: *can every agent failure automatically become future evaluation material?* The system implements a closed-loop pipeline (Fig. 1): an agent proposes a patch for a Verilog design task; a real RTL toolchain executes and judges the patch; reproducible failures are minimized, diagnosed, tagged, and promoted to structured benchmark cards; cards are deduplicated to prevent suite inflation; and agents are re-evaluated on the growing registry, with a curriculum selector feeding hard cards back into the next prompt.

This paper makes four contributions. **(1)** A self-curating benchmark loop for RTL repair agents with no ongoing manual curation cost. **(2)** A versioned benchmark-card schema capturing provenance, minimized artifacts, structured failure traces, controlled-vocabulary tags, and full reproducibility metadata. **(3)** A real-toolchain validation pass over 42 Verilog repair seeds that exposed and fixed nine invalid seeds where "buggy" designs silently passed their testbenches. **(4)** Reproducible baselines (no-op, deterministic pattern, free local LLM) over the validated suite, plus an open-source artifact released under Apache 2.0.

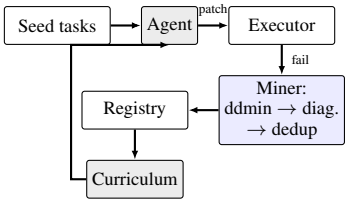

Figure 1: Fail2Bench closed-loop pipeline. Shaded block is the self-curating miner; the curriculum loops mined cards back as few-shot context for the next agent iteration.

2. RELATED WORK

Static code and hardware benchmarks. HumanEval [4] and MBPP [2] established pass@k for LLM-generated Python. SWE-bench [6] lifted this to real-world GitHub issues, but both are fixed: once solved, evaluations saturate. In hardware, VerilogEval [9] and VeriGen [14] benchmark LLM-generated Verilog on synthesis correctness; RTLCoder [10] and RTLLM [11] propose instruction-tuned models with curated corpora; BetterV [12] fine-tunes for bug repair. Fail2Bench complements these by providing *evolving* evaluation infrastructure rather than another fixed leaderboard.

Dynamic and evolving benchmarks. LiveCodeBench [5] continuously scrapes new competitive problems; EvoEval [16] mutates HumanEval prompts to probe robustness; DynaBench [7] uses adversarial human annotation. Fail2Bench applies the same principle to hardware repair: the suite grows from agents' own failures rather than from scrapers or annotators.

Fault localization and delta debugging. Spectrum-based fault localization [1] and delta debugging [17] shrink failing test cases to minimal reproducers. Fail2Bench adapts line-level ddmin to Verilog: the minimizer uses the RTL executor as its oracle, and the resulting compact snippet becomes the card's canonical artifact, improving readability and deduplication precision.

LLM agents for chip design. ChipNeMo [8] domain-adapts an LLM to chip-design workflows; ChatEDA [15] drives EDA tools through dialogue. These works target *generating* correct RTL or driving EDA flows; Fail2Bench is complementary infrastructure for capturing and recycling the failures those agents produce.

3. SYSTEM OVERVIEW

3.1. Task Execution and RTL Validation

Each task is described by a `TaskSpec`: a structured YAML card bundling the buggy Verilog source, an oracle testbench, a natural-language specification, oracle parameters (top module, timeout), and metadata (bug class, difficulty). An `Executor` compiles the design with Icarus Verilog, simulates it with `vvp`, and lints it with Verilator. Diagnostics are parsed into typed `Diagnostic` objects with fields `tool`, `severity`, `file`, `line`, `message`, and `category`. A real RTL executor and a `MockExecutor` share a common `Executor` base class so the same agent code runs under CI without toolchain dependencies. All inter-module communication uses Pydantic v2 models in `core/schemas.py`; changing a field bumps `SCHEMA_VERSION`.

3.2. Failure Mining

The `FailureMiner` is the system's central contribution and runs in three stages.

LineMinimizer. Line-level ddmin [17] with the RTL executor as the oracle. Starting from the full buggy source, the minimizer iteratively removes half-partitions of lines and re-runs the executor, retaining the deletion if the failure persists. `module`/`endmodule` declarations are protected from removal. The loop terminates when the design is 1-minimal with respect to the failure predicate or when `max_rounds` is exhausted.

Diagnoser. Parses structured diagnostics and executor logs to produce a natural-language `root_cause_hypothesis` and assign failure tags (e.g., `wrong-operator`, `width-mismatch`, `sign-extension`, `handshake`) drawn from a 20-class controlled vocabulary. The Diagnoser can call an LLM for complex hypotheses or fall back to a deterministic keyword-matching heuristic when no LLM is available.

Deduper. Filters near-duplicate cards before insertion. Stage 1 computes an AST-normalized hash of the Verilog source (strip comments, normalize whitespace, replace identifiers with positional placeholders); an exact match is a certain duplicate. Stage 2 encodes `(source + root_cause)` as a sentence-transformer embedding [13] and rejects cards whose cosine similarity to any existing card exceeds $\tau = 0.92$. A TF–IDF n-gram fallback runs when PyTorch is unavailable.

Cards that survive deduplication are persisted as JSON *BenchmarkCards* with a stable content-addressed ID, parent-run UUID, minimized artifact and testbench paths, and a `Provenance` record (agent ID, model ID, temperature, tool versions, random seed).

3.3. Promotion Policy

A failing run is promoted to a card only when: **(1)** the failure reproduces on $\geq$ `min_reproductions` $= 2$ runs; **(2)** the minimized artifact is no larger than the original; **(3)** it is not a near-duplicate; **(4)** the `root_cause_hypothesis` is non-empty; and **(5)** at least one controlled-vocabulary tag applies. Flaky failures go to a quarantine queue for inspection rather than being silently discarded.

Table 1: Seeds fixed during artifact validation.

Seed ID	Fix applied
`fifo_bug`	Re-injected write-pointer overflow
`gray_counter_bug`	Strengthened edge-case oracle
`latch_bug`	Restored missing sensitivity
`lzc_bug`	Corrected shift-count bug
`magnitude_cmp_bug`	Fixed signed-comparison oracle
`missing_sens_bug`	Added reset stimulus
`overflow_detect_bug`	Re-injected carry-out fault
`shift_reg_bug`	Widened testbench corner cases
`width_mismatch_bug`	Corrected truncation oracle

3.4. Registry, Scoreboard, and Curriculum

The `BenchmarkRegistry` is append-only, content-addressed, and stores cards indexed by ID, tag, and difficulty. The `Scoreboard` computes pass@k using the standard unbiased estimator and tracks per-card recurrence (how many independent runs reproduced the same failure), flagging persistent hard cases. After each loop iteration the `Curriculum` selects a diverse few-shot example set: greedy max-coverage over the tag vocabulary, biased toward cards the agent does not yet reliably solve, with difficulty ramped from easy toward hard as pass@1 improves—analogous to self-paced curriculum learning [3].

3.5. Seed Suite

We provide 42 hand-crafted Verilog repair seeds across three difficulty tiers (18 easy, 16 medium, 8 hard), spanning 28 distinct bug classes. The most common classes are `wrong-operator` (8), `wrong-priority` (3), `wrong-encoding` (2), `wrong-bit-order` (2), `width-mismatch` (2), `sign-extension` (2), `overflow` (2), and `off-by-one` (2). Each seed ships with a buggy module, an oracle testbench, a specification document, and a `task.yaml` card. All seeds are original works released under Apache 2.0.

4. BENCHMARK VALIDATION

A benchmark card is only useful if the seed's buggy design *actually fails* before any repair. Silently passing "buggy" seeds inflate pass@1 scores and mask real capabilities. We ran each of the 42 seeds through the real RTL executor before repair. Nine seeds passed, revealing bugs that had drifted back to correct behavior or testbenches that failed to stimulate the fault. We corrected both: buggy implementations were re-injected with the original fault, and testbenches were strengthened with corner-case stimuli (Table 1).

Validated invariant: all 42 seeds fail before repair with the real toolchain; zero seeds pass before repair.

This audit is itself a substantive result: it demonstrates that human-curated static suites can harbor correctness inversions, and that Fail2Bench's toolchain integration makes the invariant machine-checkable and re-runnable on any future suite revision.

5. EXPERIMENTAL SETUP

Toolchain. Python 3.13.1; Icarus Verilog 13.0; Verilator 5.048; Yosys 0.65; Ollama 0.24.0 with `qwen2.5-coder:1.5b`.

Baselines. Three agents share a common `Agent` base class

Table 2: Full validated 42-task suite (3 attempts per task).

Agent	Tasks	Invalid	Passed	pass@1
Mock	42	0	0	0.000
PatternRepair	42	0	3	0.071

Table 3: Easy subset (18 tasks) including a free local LLM.

Agent	Tasks	Invalid	Passed	pass@1
Mock	18	0	0	0.000
PatternRepair	18	0	2	0.111
OllamaQwen	18	0	3	0.167

with a single `propose` method. **Mock**: no-op repair; always fails. Establishes the zero-skill floor. **PatternRepair**: deterministic rule-based agent; applies regex transforms keyed to diagnostic keywords (operator flips, width truncations, missing-sensitivity insertions). **OllamaQwen**: `qwen2.5-coder:1.5b` served locally via Ollama; free, reproducible, no API cost. Used to provide an LLM floor without paid inference.

Commands. Full 42-task suite:

```
python -m fail2bench.cli.main experiment \
  --name rtl_seed_suite_20260520 \
  --agents pattern,mock --executor rtl \
  --max-attempts 3 --figures
```

Easy-subset local LLM:

```
FAIL2BENCH_OLLAMA_MODEL=qwen2.5-coder:1.5b \
python -m fail2bench.cli.main experiment \
  --name ollama_easy_suite_20260520 \
  --agents ollama,pattern,mock --executor rtl \
  --difficulty easy --max-attempts 1 --figures
```

6. RESULTS

Full 42-task suite (Table 2, Fig. 2). `PatternRepair` succeeds on `alu_bug`, `gray_counter_bug`, and `width_mismatch_bug`: all involve straightforward operator-substitution or width-truncation fixes that fall within its regex repertoire. The remaining 39 tasks expose failure modes the rule set cannot address—exactly the seeds the miner harvests into benchmark cards.

Easy-subset free local LLM (Table 3, Fig. 3). The 1.5B model solves `barrel_shift_bug`, `missing_sens_bug`, and `thermometer_enc_bug`: single-line repairs that a small code model can produce without multi-step reasoning. The result is intentionally a floor; stronger free or paid LLMs can be plugged in via the provider-agnostic `LLMClient` interface and are expected to cover more of the easy set and a meaningful fraction of medium tasks.

Generated paper figures. The `fail2bench figures` command produces five matplotlib figures from each experiment's JSON output: overall pass@1 by agent, pass@1 by difficulty tier, pass@1 heatmap by bug class and agent, seed difficulty distribution, and seed bug-class distribution. All figures in this paper were produced by the command-line pipeline without manual editing.

7. DISCUSSION

The audit as a result. The validation pass that uncovered nine silently-passing seeds is itself a contribution. Human-curated static suites can harbor correctness inversions in which the "buggy" design accidentally satisfies the oracle. Automated toolchain integration is necessary to maintain the invariant; Fail2Bench makes the check machine-runnable on any future suite revision.

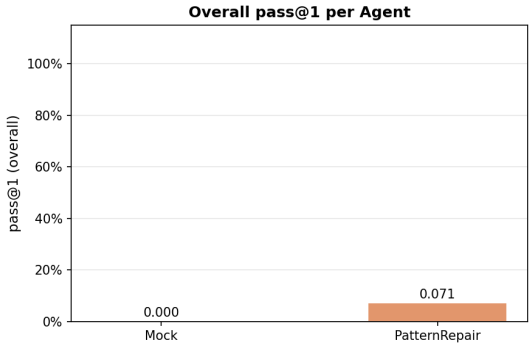

Figure 2: Overall pass@1 by agent on the full 42-task validated suite. `Mock` establishes the zero-skill floor; `PatternRepair` covers operator-substitution and width-truncation bugs, leaving 39 failure modes that the miner harvests into cards.

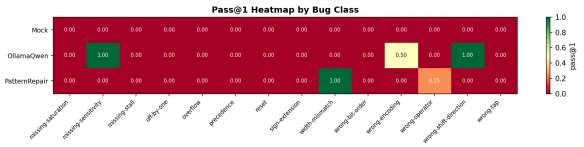

Figure 3: pass@1 heatmap by bug class on the easy subset for `OllamaQwen`, `PatternRepair`, and `Mock`. Cells are binary on a single attempt. Pattern repair specializes in `wrong-operator` and `width-mismatch`; the local LLM solves a disjoint set (`wrong-shift-direction`, `missing-sensitivity`, `wrong-encoding`), demonstrating that even a 1.5B-parameter LLM agent complements rule-based repair.

Scalability. The current suite is 42 RTL tasks. The `fail2bench generate` command applies four structural mutations per seed (width widening, width narrowing, reset-polarity flip, parameter wrapping), producing up to $42 \times 4 = 168$ synthetic variants without manual effort. Live evaluation of frontier LLMs will expand the registry further as stronger agents expose new failure modes that survive deduplication.

Extensibility. The `adapters/domain_registry.py` module maps domain names to (Agent, Executor, FailureMiner) triples. Adding a second domain (compiler-pass repair, microarchitectural DSE, kernel-module configuration) requires implementing three one-method abstract base classes and calling `DomainRegistry.register()`. The core loop is domain-agnostic.

Limitations. Our included LLM baseline is a 1.5B local model; results should not be interpreted as frontier-model performance. The pattern baseline is deliberately narrow. Coverage tests for paper-facing modules (`experiment`, `figures`) require additional work before a full 80% gate. The suite is RTL-only; domain adapters for other hardware workflows are future work.

8. CONCLUSION

Fail2Bench turns RTL agent failures into durable, structured benchmark material. The system validates that every seed fails before repair, mines reproducible failures via line-level ddmin, deduplicates near-duplicates by normalized hash and embedding similarity, and provides reproducible baseline experiments from no-op through a free local LLM. The central claim is that benchmark infrastructure for agentic hardware workflows should evolve with the agents being evaluated: the benchmark the agent faces today should be shaped by what the agent failed to fix yesterday.

The code, seed suite, benchmark-card schema, and all experiment scripts are released as open-source infrastructure under Apache 2.0.

REFERENCES

[1] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J.C̃. van Gemund. A practical evaluation of spectrum-based fault localisation. *Journal of Systems and Software*, 82(11):1780–1792, 2009.

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, 2009.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[5] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

[6] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[7] Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, et al. Dynabench: Rethinking benchmarking in NLP. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.

[8] Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. ChipNeMo: Domain-adapted LLMs for chip design. *arXiv preprint arXiv:2311.00176*, 2023.

[9] Mingjie Liu, Nathanael Fäßler, Haoxing Ren, Jiaqi Gu, Zhuofu Tao, Ruifan Xu, Yaran Fan, Stephen Boyd, Bhargav Grover, Haoke Chen, et al. VerilogEval: Evaluating large language models for Verilog code generation. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[10] Shang Liu, Wenji Fang, Yao Lu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. RTLCoder: Outperforming GPT-3.5 in design RTL generation with our open-source dataset and lightweight solution. *arXiv preprint arXiv:2312.08617*, 2024.

[11] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. RTLLM: An open-source benchmark for design RTL generation with large language model. *arXiv preprint arXiv:2308.05345*, 2023.

[12] Zhiyao Pei, Yuchen Hu, Shang Liu, Hongce Zhang, and Zhiyao Xie. BetterV: Controlled verilog generation with discriminative guidance for functional correctness. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[13] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[14] Shailja Thakur, Baleegh Ahmad, Zhenxing Fan, Hammond Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. VeriGen: A large language model for Verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3), 2024.

[15] Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. ChatEDA: A large language model powered autonomous agent for EDA. In *Proceedings of the ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD)*, 2023.

[16] Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. EvoEval: Evolving coding benchmarks via LLM-based mutation. *arXiv preprint arXiv:2403.19114*, 2024.

[17] Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2):183–200, 2002.