Beyond Synthetic Benchmarks: Reproducible Evaluation of Engineering-Intelligence Systems for Semiconductor Design

Srinivasan Subramanian

Founder & Chief Architect
XSYDA Technologies
XSYDA Research
Research Preprint · Version 1.0
contact@xsyda.com

***Abstract*—**Verification and debugging dominate modern semiconductor engineering effort, yet the knowledge produced—root causes, design rationale, recurring failure modes—is fragmented across tools and lost across projects and staff turnover. A wave of "engineering intelligence" tooling now claims to recover this value, but such claims are unusually hard to evaluate: systems are frequently tuned and scored on synthetic benchmarks against self-supplied ground truth, producing results that, in our own development, did not survive contact with real data. This paper makes the evaluation problem its primary subject. We present (i) a reproducible, anti-circular, provenance-aware evaluation framework built on a corpus of four real open-source RTL designs (picorv32, Ibex, CVA6, OpenTitan; 2,510 HDL files, 622,692 lines, parsed with 100 % success), with bug ground truth derived from upstream maintainer fix commits rather than self-injected labels; (ii) XRecall, a provenance-anchored cross-project engineering-memory architecture; and (iii) a candid account of what the framework reveals, including negative findings. Under strict pre-registration, naive cross-project recurrence detection on maintainer-labeled data yields a *trivial-only* verdict, driven by a dominant catch-all taxonomy class and label noise; on real failures, semantic clustering does not significantly outperform a trivial string-key baseline. What survives every test is a reproducible capability—cross-project aggregation surfaces structure that single-project views structurally cannot—and a determinism/provenance discipline verified by independent recomputation. *We deliberately make no superiority claim over existing tools.* The contribution is a rigorous evaluation methodology, an open real-RTL corpus with maintainer-fix labels, a provenance-aware memory architecture, and a synthetic-to-real generalization gap that we document as a cautionary case study from our own system.

Index Terms—semiconductor design, engineering intelligence, reproducible evaluation, provenance, knowledge reuse, RTL verification, empirical methodology.

I. INTRODUCTION

Semiconductor design is knowledge-intensive and increasingly knowledge-bound. As designs scale, verification consumes the majority of project effort, and the artifacts of that effort—failure signatures, root-cause determinations, the rationale behind design and fix decisions—constitute an organization's most valuable and most perishable asset. This knowledge is fragmented across vendor tools, scattered across regression logs and ticketing systems, and largely lost when projects close or engineers leave.

A growing class of tools markets itself as "engineering intelligence" for chip design: failure triage, design-quality analysis, optimization, and decision support. The promise is real, but the evidence is hard to trust. In our own development we observed a failure mode we suspect is common but demonstrate here only for our own system: a tool is tuned on synthetically generated designs and failures, then scored against ground truth that the same pipeline produced. Such evaluation is *circular*—it measures self-consistency, not capability—and in our own system the resulting numbers did not generalize to real designs.

This paper takes the unusual position that, for this class of system, *evaluation rigor is the bottleneck*, not algorithmic novelty. We build an anti-circular, provenance-aware evaluation framework on real open-source RTL and real maintainer-authored bug labels, and we report what it shows—favorable and unfavorable—without attempting to rescue weak claims.

This work asks three research questions (Section III), introduces the XRecall architecture (Section IV) and its evaluation corpus (Section V) and methodology (Section VI), reports experiments and results (Sections VII–VIII), and discusses—candidly—what worked, what did not, and why (Section IX). This work is part of an ongoing research effort at XSYDA Technologies on engineering-intelligence systems for semiconductor design; the present paper concerns how such systems should be evaluated.

Contributions. (1) A reproducible, pre-registered, anti-circular evaluation framework for semiconductor engineering-intelligence systems. (2) An open evaluation corpus of four real RTL designs with 372 validated maintainer-fix bug labels. (3) XRecall, a provenance-anchored cross-project engineering-memory architecture, and a demonstration of a capability single-project tools structurally lack. (4) A documented synthetic-to-real generalization gap and an honest negative-results account. *We explicitly do not claim superiority over commercial or academic tools.*

II. BACKGROUND AND MOTIVATION

A. Knowledge loss in semiconductor engineering

The cost of lost engineering knowledge is felt most acutely across project boundaries: a failure class debugged on one design re-appears on the next, and is re-debugged from scratch because no queryable memory connects the two. Existing analytics are largely tool-locked and single-project; none provides a vendor-neutral, cross-project memory whose primary unit is engineering knowledge rather than design artifacts. Because verification already consumes a large share of project effort, this repeated cross-project re-debugging is a direct, recurring cost, and staff turnover compounds it: the rationale behind past decisions departs with the engineers who made them.

B. The reproducibility concern

Machine learning for EDA has matured rapidly [1], [2], but the subfield has also experienced a prominent, publicly documented reproducibility dispute over the reinforcement-learning macro-placement results of [2], underscoring that headline results in this domain require independent verification. Engineering-intelligence systems inherit and amplify this risk: their outputs (clusters, rankings, recommendations) are easy to score favorably against self-authored ground truth.

C. Related work

Our work intersects six lines of research. *Machine learning for EDA* applies learning to placement, routing, congestion prediction, and design-space optimization [1], [2]; it has advanced rapidly but targets implementation quality within a single design and flow, not the persistence of engineering knowledge across projects. *Verification analytics* applies machine learning to failure triage, bucketing, and coverage optimization across regression runs; these systems are effective but operate within a single tool and project, and—directly relevant here—their reported gains are frequently measured on proprietary or synthetic data, a concern that open EDA datasets such as CircuitNet [3] have begun to address. *Provenance systems* [4], [5] and *reproducibility research* [6], [7], mature in scientific computing and machine learning, supply the conceptual basis for our signature-based determinism and end-to-end traceability, yet provenance is seldom a first-class concern in semiconductor-design tooling. *Knowledge management* and *engineering-memory* concepts from software and systems engineering motivate the cross-project persistence we pursue; however, semiconductor-specific engineering memory—linking failures, fixes, decisions, and rationale across projects with provenance—remains, to our knowledge, largely unaddressed. That gap is the space XRecall occupies.

III. RESEARCH QUESTIONS

> **RQ1.** Can a provenance-aware engineering-memory architecture support cross-project knowledge reuse that single-project tooling structurally cannot?

> **RQ2.** When cross-project failure recurrence is evaluated honestly on real, maintainer-labeled data, what does it reveal—and where does naive recurrence analysis break down?

> **RQ3.** How can engineering-intelligence systems be evaluated without circularity?

RQ2 is stated as an open question rather than a hypothesis to confirm: as Section VIII reports, the honest answer is partly negative, and that answer is itself a finding.

IV. XRECALL ARCHITECTURE

XRecall is the cross-project memory layer of a three-tier system. A reusable intelligence kernel provides lineage, propagation, causal-attribution, counterfactual, and benchmarking primitives. Two domain applications consume the kernel: a verification failure-intelligence app and an RTL design-intelligence app. XRecall sits above both: it ingests the structured, signed outputs they produce and accumulates them into a persistent, queryable cross-project store.

A. Engineering memory as a first-class system

Conventional EDA tools analyze *designs*: given RTL, a netlist, or waveforms, they reason about a single artifact within a single project. XRecall analyzes *accumulated engineering knowledge*: it takes the structured outputs that design and verification activity already produce and reasons across them, over time and across projects. This is a different object of analysis, and it is the paper's central architectural claim. A design-analysis tool cannot, by construction, observe that a failure class recurs across independent projects, that a fix re-applied on one design was already understood on another, or that prior art exists for a problem a team is currently re-solving—because each instance sees only its own project. An engineering-memory system makes that accumulated, cross-project knowledge a first-class, queryable, provenance-anchored object. We are careful about the nature of this claim: we do not assert that XRecall analyzes any single design more *accurately* than existing tools; we assert that it answers a different class of question that single-project tools structurally cannot.

B. Design principles

Three principles govern the system. *Determinism:* every analysis is reduced to a canonical input and hashed (SHA-256) to a stable signature, so identical inputs yield byte-identical results. *Idempotency:* ingestion is keyed on these signatures, so re-ingesting a run is a no-op. *No claim without provenance:* every stored fact carries the source signature (and, for external data, repository URL and commit SHA), so any query result is traceable to the run that produced it.

C. Pipeline

Ingestion adapters normalize app outputs into a typed store (projects, runs, modules, findings, root-cause families, decisions, metrics). A lightweight graph links shared entities across runs and projects; the kernel's lineage and propagation primitives are reused to compute cross-run relationships. A retrieval layer answers cross-project queries—module history, finding recurrence, "seen-before" lookups—each annotated with provenance.

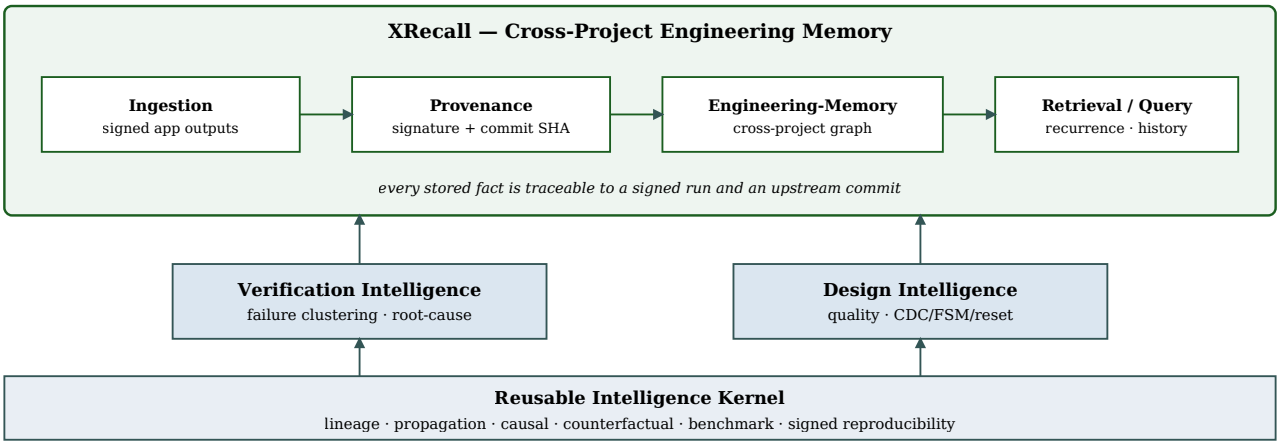

Fig. 1. System tiers and the XRecall pipeline. A reusable kernel feeds two parallel domain applications; their signed outputs flow into XRecall, where ingestion, a provenance layer, a cross-project engineering-memory graph, and a retrieval interface make accumulated knowledge queryable across projects. Each application, in isolation, observes only its own project; only the memory layer observes cross-project structure.

V. DATASET CONSTRUCTION

We evaluate on four widely used open-source RTL designs, pinned by commit, spanning a range of scale and maturity (Table I). The corpus is parsed by an elaborating SystemVerilog frontend (pyslang 11.0.0); all 2,510 HDL files parse successfully.

TABLE I. EVALUATION CORPUS (REAL OPEN-SOURCE RTL)

Design	Files	LOC	Modules	Signals	Parse
picorv32	41	9,133	61	862	100%
Ibex	383	61,099	157	1,552	100%
CVA6	433	110,682	384	3,756	100%
OpenTitan	1,653	441,778	683	25,975	100%
Total	**2,510**	**622,692**	**1,285**	**32,145**	**100%**

A. Bug ground truth from maintainer fix commits

Rather than inject faults and score against our own manifest, we mine bug ground truth from the designs' upstream git history: commits whose messages indicate a fix and which touch HDL. This yields 951 candidate fix commits (612 touching HDL). We then apply documented validation filters —single HDL file, not a refactor, fix keyword present, single identifiable module—retaining **372 validated maintainer-fix labels (61 %)**. Crucially, these labels are authored by the projects' own maintainers, independent of our system—an approach analogous to fault databases mined from real fix histories in software testing [8].

B. A candid look at the labels

The resulting taxonomy is dominated by a catch-all class: roughly 80 % of validated labels are `functional_other`, with specific classes (interrupt, control, memory, datapath, reset, CDC) forming the long tail (Fig. 2). This distribution is itself a finding (Section IX) and a primary limitation (Section X).

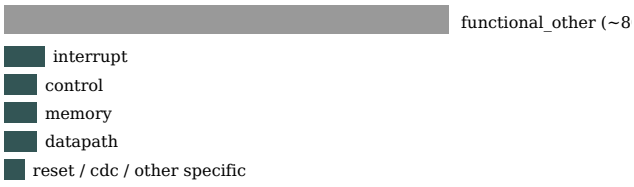

Fig. 2. Validated label distribution. A dominant catch-all class limits the granularity at which recurrence can be meaningfully assessed.

VI. EVALUATION METHODOLOGY

The methodology is the core methodological contribution; we state its rules explicitly so others can adopt them.

A. Anti-circularity

Ground truth must be produced by a process independent of the system under test; baselines must be external/established tools; results are never scored against any table the system's own generators authored. Where a metric is computed, it is also recomputed independently of the system and the two are required to agree.

B. Pre-registration and held-out testing

Before any held-out result is computed, the primary metric, the comparison, and a quantitative success criterion are committed to a pre-registration document. Tuning is restricted to a development split; the test split is never inspected during tuning. Any post-hoc change to the criterion invalidates the affected result.

C. Independent ground truth and external baselines

Across experiments we use: maintainer-fix commits (Section V); established linters (Verilator, Yosys) as an external reference for design findings; mutation faults [9] whose injection site is recorded *before* simulation; and an external solver (OR-Tools CP-SAT) for optimization comparisons. Where the appropriate reference tool was unavailable in our environment (notably Verible), we say so and treat the corresponding capability as *unvalidated* rather than validated.

D. Reproducibility

Determinism is enforced by SHA-256 signatures over canonicalized inputs and verified by tests asserting byte-identical signatures across independent cold runs; the scorer itself is unit-tested against partitions with known metric values.

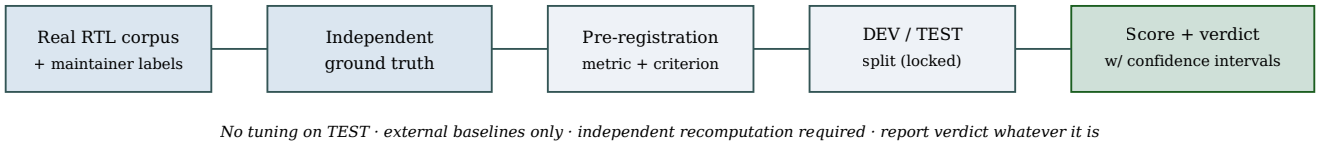

Fig. 3. Anti-circular evaluation workflow. Each result is pre-registered, scored on a held-out split against externally-derived ground truth, and reported regardless of outcome.

VII. EXPERIMENTS

E1 — Corpus parsing at scale. Parse the full corpus with the elaborating frontend; report coverage. **E2 — Recurrence: independent vs. system.** Compute cross-project recurrence directly from the validated labels, then via XRecall, and require agreement. **E3 — Cross-project capability contrast.** Compare what a single-project view can observe with what cross-project aggregation reveals. **E4 — Manual validation and secondary evaluations.** A stratified hand spot-check of surfaced recurrences, plus three external-baseline evaluations: failure clustering vs. a string-key baseline, design findings vs. linters, and optimization vs. OR-Tools.

VIII. RESULTS

A. Parsing and reproduction (E1–E2)

The elaborating frontend parses all 2,510 files (Table I). XRecall's cross-project recurrence query reproduces the independently-computed ground truth exactly for all 8 evaluated classes, satisfying the independence-of-recomputation requirement and confirming the query is not the source of any later finding.

B. The pre-registered recurrence verdict (E2, E4)

Three of four pre-registered criteria are met: six specific classes have at least three distinct bugs each; each spans at least two designs; and at least one (control, memory) spans three. The fourth criterion—an anti-artifact spot-check—fails: in a 23-record stratified manual check, 78 % of records were plausibly classed and only 70 % passed both plausibility and genuine-distinctness. The pre-registered verdict is therefore **TRIVIAL-ONLY** (Table II). The dominant cause is the catch-all class together with cherry-picked/re-applied fixes and verification-path files contaminating specific classes.

TABLE II. RECURRENCE: PRE-REGISTERED CRITERIA

Criterion	Result
≥3 specific classes, ≥3 distinct bugs each	Met (6 classes)
Each spans ≥2 designs	Met
≥1 class spans ≥3 designs	Met (control, memory)
Anti-artifact spot-check ≥80% pass	**Failed** (78% / 70%)
Verdict	**TRIVIAL-ONLY**

An *exploratory* strict-filtered view (de-duplicating cherry-picks and dropping verification-path files) leaves four specific classes with ≥10 RTL bugs each across ≥2 designs, which would likely pass a meaningfulness bar. Because the pre-registration committed to the raw recurring set, we report this only as exploratory and do not claim it.

C. Cross-project capability (E3)

Independent of the recurrence verdict, the capability claim holds: each design, viewed alone, observes only its own specific-class bugs; neither can know that a pattern recurs in another without aggregation. XRecall's cross-project view exposes this structure, with every linked record carrying its originating commit as provenance. This is a capability demonstration—single-project tooling cannot produce it—not a superiority comparison.

D. Secondary external-baseline evaluations (E4)

Failure clustering. On real mutation-induced failures (held-out split, anonymized to remove test-name leakage), semantic clustering and a trivial string-key baseline are statistically indistinguishable on the primary metric at the available sample size; we report no significant advantage (Table III). **Design findings vs. linters.** Adding cross-module elaboration to the design app sharply reduces false positives (e.g., on Ibex, unused-signal findings fall from 50 to 1; multi-driver from 33 to 1), but agreement F1 with the linter union remains low because the app's flagship classes (CDC, FSM, reset) are not emitted by the available reference tools; with Verible

unavailable, those classes remain *unvalidated*. **Optimization.** Against OR-Tools CP-SAT, the optimizer mostly ties at proven optima; a single regime (n=50, density 0.5) shows a metaheuristic advantage under a tight wall-clock budget. None of these is a superiority result, and we do not present them as such.

TABLE III. SECONDARY EVALUATIONS (HONEST OUTCOMES)

Evaluation	Reference	Outcome
Failure clustering	string-key baseline	no significant difference
Design findings	Verilator ∪ Yosys	FP↓; flagship classes unvalidated
Optimization	OR-Tools CP-SAT	mostly ties; one regime favorable
Reproducibility	byte-equality tests	holds (signatures identical)

IX. DISCUSSION

A. A synthetic-to-real generalization gap

The clearest lesson is that capabilities which appear strong on synthetic, self-labeled benchmarks weaken or vanish under honest, external evaluation. Our clustering component, tuned on synthetic data, did not generalize to real maintainer-labeled failures; our recurrence analysis, persuasive on synthetic samples, was trivial-only on real labels. We document this gap as a cautionary case study; whether it generalizes to other systems is a question a single system cannot answer.

B. Why recurrence was partially trivial

The catch-all `functional_other` class (~80 %) makes coarse "recurrence" almost tautological; cherry-picked and re-applied fixes inflate counts; and verification-path files leak into specific classes. Meaningful recurrence may well exist beneath this noise (the exploratory strict view suggests so), but a finer-grained, validated taxonomy is required before the claim can be made honestly.

C. What survived

Three things held under every test: the cross-project *capability*; the *provenance* chain (every result traceable to a signed run and a commit); and *reproducibility* (byte-identical signatures, independently verified). These—not benchmark wins—are the defensible value.

X. THREATS TO VALIDITY

Construct. Maintainer-fix commits identify which file a fix touched, not necessarily the precise root cause; commit messages mislabel. We treat them as noisy maintainer labels, not gold truth. **Internal.** The available linters cover only a subset of the design app's issue classes; CDC/FSM/reset are unvalidated for lack of an appropriate reference. **External.** The corpus is RISC-V-centric and limited to four designs; mutation-derived failures are small in number; results may not transfer to other ISAs, larger SoCs, or industrial regression data. **Conclusion.** Small sample sizes yield wide confidence intervals; we report intervals and avoid claims they do not support.

XI. IMPLICATIONS

Research. For this class of system, evaluation methodology deserves first-class treatment; provenance and pre-registration should accompany capability claims. **Industry.** A provenance-anchored cross-project memory addresses a concrete, costly pain—knowledge lost across projects and turnover—even before any predictive-accuracy claim is justified. The value proposition is integration, traceability, and reuse, not benchmark superiority.

XII. FUTURE WORK

A finer, validated bug taxonomy that dissolves the catch-all class; stronger semantic clustering evaluated on the same external corpus; a *retrieval-quality* benchmark (can the memory return relevant prior engineering knowledge more accurately than alternatives?) that better reflects the system's intended value than recurrence detection; an expanded, multi-ISA corpus; and, most valuably, evaluation on a real industrial regression dataset with root-cause labels. More broadly, these results point to engineering-memory systems—semiconductor knowledge graphs, cross-project intelligence, and provenance-aware decision support for design teams—as a research direction distinct from design-analysis tooling; we regard the rigorous, reproducible evaluation of such systems, rather than the assertion of their superiority, as the prerequisite next step.

XIII. CONCLUSION

We presented a reproducible, provenance-aware framework for evaluating semiconductor engineering-intelligence systems, an open real-RTL corpus with maintainer-fix labels, and XRecall, a cross-project engineering-memory architecture. Evaluated honestly, naive recurrence detection is trivial-only and semantic clustering does not beat a trivial baseline—findings we report rather than rescue. What endures is a reproducible cross-project capability and a provenance discipline that make the system's outputs traceable and its evaluation repeatable. Where synthetic-benchmark inflation is a risk, we argue that this kind of rigor—not another superiority claim—is the more durable contribution.

APPENDIX: REPRODUCIBILITY

All experiments are driven by versioned scripts with pinned tool versions (pyslang 11.0.0; Verilator 5.020; Yosys 0.33; Icarus 12.0; OR-Tools). Corpus designs are pinned by commit SHA and re-fetched from upstream by a manifest-driven fetcher with strict SHA validation. Run signatures, the pre-registration documents, the validated label set, and the per-experiment result tables are maintained as a versioned artifact bundle; a public artifact repository is planned for a future release.

REFERENCES

[1] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong, X. Ning, Y. Ma, H. Yang, B. Yu, H. Yang, and Y. Wang, "Machine learning for electronic design automation: A survey," *ACM Transactions on Design Automation of Electronic Systems*, vol. 26, no. 5, Art. 40, pp. 40:1–40:46, 2021, doi: 10.1145/3451179.

[2] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y. J. Lee, E. Johnson, O. Pathak, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021, doi: 10.1038/s41586-021-03544-w.

[3] Z. Chai, Y. Zhao, Y. Lin, W. Liu, R. Wang, and R. Huang, "CircuitNet: An open-source dataset for machine learning applications in electronic design automation (EDA)," arXiv preprint arXiv: 2208.01040, 2022.

[4] L. Moreau et al., "The open provenance model core specification," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.

[5] P. Groth and L. Moreau, "PROV-DM: The PROV data model," *W3C Recommendation*, World Wide Web Consortium (W3C), 2013.

[6] O. Gundersen and S. Kjensmo, "State of the art: Reproducibility in artificial intelligence," in *Proc. AAAI Workshop on Reproducible AI*, 2018.

[7] J. Pineau et al., "Improving reproducibility in machine learning research," *Journal of Machine Learning Research*, 2021.

[8] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," in *Proc. Int. Symp. Software Testing and Analysis (ISSTA)*, 2014.

[9] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.