

AI-Guided Stimulus and Debug Triage: A Methodology for Accelerating Pre-Silicon Emulation of Flagship SoC Programs

Abhishek Kumar Sinha

Independent Researcher

Received: 03 Aug 2026 | Received Revised Version: 24 Aug 2026 | Accepted: 09 Sep 2026 | Published: 03 Oct 2026

Volume 08 Issue 10 2026 | Crossref DOI: 10.37547/tajet/Volume08Issue10-01

Abstract

Raw emulation capacity for flagship system-on-chip (SoC) programs keeps growing, and engineer-hour supply keeps falling further behind it. That gap, not machine availability, is the actual bottleneck in pre-silicon verification today, driven by workload diversity across artificial intelligence (AI), extended reality (XR), and mobile platforms converging on a single die. This article argues that closing the gap requires a bounded, governed layer of AI-guided assistance rather than either extreme - full automation or continued reliance on engineer-only throughput. A three-layer adoption framework is proposed: scoping, which restricts AI involvement to stimulus generation for known-workload classes and anomaly-flagged debug triage while keeping root-cause sign-off and tape-out risk calls fully engineer-owned; validation, which requires every AI-generated output to be checked against a trusted reference - simulation, formal methods, or historical regression baselines - before an engineer may act on it; and governance, an operating-review cadence with rollback criteria defined before deployment, not improvised afterward. Pilot deployment of emulation-friendly RTL on a premium-tier mobile SoC program produced a shift of approximately 60 percent of RTL earlier in the program schedule, with no coverage-model exception granted to reach that outcome. The finding that emerges is not that AI accelerates verification - that much is already argued elsewhere - but that the acceleration only survives contact with a tape-out schedule when it is deliberately bounded. Emulation productivity under AI assistance is a governance problem before it is a technical one, and the three-layer framework is offered as a transferable operating model for programs where the cost of an unvalidated miss is unacceptable.

Keywords: Artificial Intelligence; SoC Emulation; RTL Verification; Stimulus Generation; Debug Triage; Coverage Closure; Verification Governance; Pre-Silicon Verification

© 2026 Abhishek Kumar Sinha. This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). The authors retain copyright and allow others to share, adapt, or redistribute the work with proper attribution.

Cite This Article: Abhishek Kumar Sinha. (2026). AI-Guided Stimulus and Debug Triage: A Methodology for Accelerating Pre-Silicon Emulation of Flagship SoC Programs. The American Journal of Engineering and Technology, 8(10), 1–7. <https://doi.org/10.37547/tajet/Volume08Issue10-01>

1. Introduction

A single premium-tier mobile SoC no longer verifies one thing. AI accelerators, XR-class graphics and sensor pipelines, and conventional application-processor logic now share a die, and each domain arrives with its own stimulus profile, its own coverage model, its own way of failing. Emulation capacity has kept pace on paper - capacity-

doubling initiatives and virtualization approaches that put multiple users on a single emulation system have both moved the needle - but capacity was never the actual constraint. Engineer-hours spent authoring stimulus, triaging regressions, and localizing debug have not scaled at anything close to the same rate. The bottleneck sits upstream of the hardware, and it has for some time.

That is the specific problem this article addresses. Register transfer level (RTL) scale and workload diversity are outpacing engineer-hour and machine-hour supply, even as raw emulation capacity keeps expanding through infrastructure investment (a detail not always visible in vendor-published capacity figures, which tend to report machine counts rather than effective throughput). The gap surfaces as compile turnaround delay, a growing backlog in stimulus authoring for new workload classes, and debug queues that outpace triage capacity. Operating reviews across recent flagship generations have converged on the same finding: stimulus generation and debug triage, not machine availability, sit on the critical path to tape-out readiness.

Unbounded automation is not the fix. Neither is treating AI assistance as categorically out of scope for tape-out-critical work - that position is becoming harder to defend as the productivity gap widens. A governed layer of AI assistance, scoped to specific task classes and validated before any output is trusted, can close a meaningful share of that gap without introducing unacceptable risk into signoff. This is a narrower claim than either extreme offers, and it is the claim a program-execution organization is actually forced to reason about under schedule and cost pressure - a research group answering "does the model work on this benchmark" is answering a different question entirely.

This article's contribution is a three-layer adoption framework built from direct experience leading AI-enabled emulation strategy across four flagship Snapdragon and XR chip generations. It specifies which emulation task classes are eligible for AI-guided assistance, what validation an AI output must clear before an engineer acts on it, and what governance cadence keeps the whole arrangement bounded as adoption scales. Pilot results from a premium-tier mobile SoC program, discussed later as the article's own primary data, show what a governed shift-left outcome looks like in practice - and where it nearly went wrong.

2. Literature Review

Three bodies of work inform this article's argument, and they answer three different questions rather than converging on one. The first concerns machine learning applied to RTL-level verification as a standalone algorithmic problem - stimulus generation, coverage prediction, bug localization. Early work in this thread establishes that learned models can meaningfully outperform undirected stimulus generation [1], and industrial experience with assertion-based verification affirms that formal methods deliver real value

while still demanding substantial engineer effort to construct and maintain [2]. Standardization around portable stimulus [3], SystemVerilog [4], and the Universal Verification Methodology [5] gave the field common ground, though none of these frameworks assumed AI-guided generation from the outset - each was adapted to it later, sometimes awkwardly.

Newer empirical work sits closer to production relevance but stays scoped at the module level rather than the program level. Broader surveys of machine learning applied to electronic design automation confirm the field's adoption has accelerated sharply since 2023 [6], and large language model (LLM) approaches to Verilog generation reveal considerable variance in specification-to-RTL correctness depending on model and prompting strategy [7], [8]. A comprehensive survey of LLM-assisted circuit verification catalogs a comparable pattern across the broader verification stack, not just RTL generation [9]. Reinforcement-learning-guided stimulus generation demonstrates strong efficiency gains over random approaches on bounded designs such as a JTAG test-access-port controller [10], and LLM-based assertion generation pipelines achieve high syntactic and functional correctness with strong design-logic coverage [11]. Automated testbench generation and evaluation pipelines built on LLMs push this further into the verification-environment layer itself [12], and an LLM-aided universal-verification-methodology (UVM) machine demonstrates that full verification-environment construction, not just stimulus or assertions, is within reach of current techniques [13]. LLM-based RTL debug-automation work extends the same production-relevance argument to failure triage specifically [14], and open, reproducible LLM-for-RTL benchmarks are beginning to make these comparisons less vendor-dependent [15]. Learned sampling for coverage closure separately outperforms both random and prior optimization-based baselines by a wide margin [16]. Useful results, all of them. But this thread asks a narrower question than the one this article poses - whether a technique beats a baseline on one task, not whether it can be trusted inside a schedule-constrained, tape-out-critical program without independent re-verification.

A second thread treats emulation and verification productivity more broadly, without a strong AI focus, and it complicates the first thread's implicit assumption that better algorithms are the main lever available. Surveys of directed test generation catalog decades of technique evolution without addressing how such techniques get adopted inside a large verification organization managing competing

schedule pressures [17]. Verification environment setup overhead reportedly consumes more than half of total project time in some co-design contexts, with scripted frameworks cutting configuration time from days down to roughly thirty minutes [18]. The overwhelming majority of relevant published work on ML-for-EDA is concentrated in just the past two years [19]. That concentration matters here: the research base is young relative to the maturity of the emulation infrastructure it is being layered onto, and that immaturity is precisely why adoption risk deserves central treatment rather than a footnote.

A third thread, thinner than the other two but the one this article leans on most directly, concerns trust in AI-generated outputs inside safety- or correctness-critical pipelines. Root-cause prediction research in semiconductor manufacturing demonstrates that AI-assisted failure

analysis can lift prediction accuracy over engineer-only baselines - but only within a scope that stays defined and monitored [20]. Bug-localization work built on information-retrieval techniques treats the AI output as a ranked candidate list awaiting human adjudication, never as an autonomous verdict [21]. Systematic reviews of formal-methods approaches to validating machine-learning behavior in safety-critical settings narrow a large initial literature down to a small handful of studies that actually address validating AI outputs themselves, as distinct from AI-assisted verification of something else [22]. None of these three threads, individually, answers the question a verification program actually needs answered: under what conditions can an organization trust an AI-generated stimulus or triage result enough to act on it without quietly re-deriving the answer by hand anyway.

Table 1. Comparison of the three literature threads by unit of analysis, typical evaluation metric, and applicability to program-execution adoption decisions.

Literature Thread	Unit of Analysis	Typical Evaluation Metric	Applicability to Adoption Decisions
ML for RTL-level verification (stimulus generation, coverage prediction, bug localization)	Single algorithm vs. baseline	Coverage gain, pass rate, speedup on a benchmark design	Answers whether a technique outperforms a baseline; silent on schedule-risk trust conditions
Emulation and verification productivity (broader, non-AI-specific)	Verification organization / project	Setup-time reduction, share of project time consumed by overhead	Identifies where effort is spent, but does not address AI-specific adoption governance
Trust in AI-generated outputs (safety/correctness-critical pipelines)	Validation methodology itself	Number of studies meeting rigor threshold; scope of validated claims	Most directly relevant to governance argument, but the literature base remains thin

3. Methodology / Adoption Framework

The framework proposed here has three layers, and each layer answers a distinct question about how AI-guided assistance is allowed to enter a verification program - not how well it performs in isolation. Scoping comes first. It determines eligibility, not performance. Two task categories qualify: stimulus generation for known-workload classes, where historical regression data and coverage models already exist as a baseline, and anomaly-flagged debug triage, where an AI-guided system surfaces candidate root-

cause hypothesis for engineer review rather than delivering a final verdict. Two categories are excluded outright, regardless of how well a model demonstrates it can perform: root-cause sign-off on any failure touching tape-out schedule or silicon risk, and the tape-out risk call itself. That exclusion is not a technical limitation of current models (it would likely hold even if model accuracy improved substantially). It is a governance decision, full stop, reflecting where a false negative is simply not affordable.

Validation is the second layer, and it is the one doing most of the trust-building work. Every AI-generated output must be checked against a trusted reference before an engineer is permitted to act on it - no exceptions carved out for time pressure. Three reference classes count as trusted: simulation results for the same design point, formal verification where applicable, and historical regression baselines drawn from prior silicon-qualified builds of comparable intellectual property (IP). An AI-generated stimulus set that has not cleared at least one of these checks holds the same status as an untested hand-written test: a candidate, not yet evidence. This stance is deliberately more conservative than what some published stimulus-generation results might suggest is achievable on their own terms [16], [10], because the cost of a bug discovered after tape-out and the cost of additional pre-tape-out validation effort are not remotely symmetric.

Governance, the third layer, sets the operating-review cadence at which AI-guided adoption gets measured against its own stated goals, plus the rollback criteria that pull back AI involvement when outputs underperform. Reviews run

on a cadence aligned to existing program milestones rather than bolted on as a separate process, tracked against baseline engineer-only performance on matching task classes. Rollback criteria are set before deployment, not improvised after something breaks: a task class reverts to engineer-only ownership if validated AI-output accuracy drops below an agreed threshold over a defined window, or if even one validated miss occurs on a task class feeding a tape-out risk decision. This last condition was tightened mid-pilot, after an early anomaly-triage run flagged a false negative that the validation layer caught before it reached a regression gate - a moment addressed in more detail in Section 4, because it shaped the framework more than any success did.

These three layers loop rather than run in sequence. Scoping decisions get revisited at every governance review based on what validation performance actually shows, which means a task class can move from eligible to excluded, or back again, as evidence accumulates over successive program generations.

Figure 1. Three-Layer AI-Guided Adoption Framework

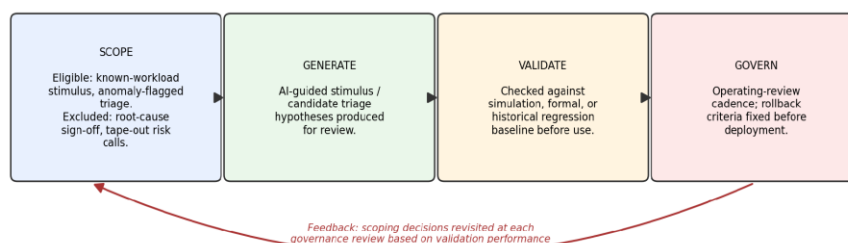


Figure 1 Three Layer AI Guided Adoption Framework

Notice what the framework does not specify: which particular ML technique performs the generation step at any layer. That silence is deliberate. The governance question - is this output trustworthy enough to act on - does not hinge on model architecture, and a framework that pretended otherwise would be solving the wrong problem.

4. Results / Case Findings

The primary evidence behind this framework comes from a pilot deployment of emulation-friendly RTL practices on a premium-tier mobile SoC program, presented here as the author's primary research data rather than an externally sourced result. The pilot restructured RTL delivery and verification sequencing so a defined share of design content

entered emulation earlier than prior-generation baselines allowed, with AI-flagged prioritization helping identify which RTL blocks stood to benefit most from earlier emulation exposure. The measured outcome was that approximately 60 percent of RTL was shifted earlier in the program schedule, reached without any coverage-model exception - none was requested, and none would have been granted.

Post-pilot operating reviews credited three factors with most of the shift-left result: earlier availability of stable RTL partitions suitable for emulation compile, AI-guided triage that trimmed engineer-hours spent on false-lead debug paths before a validated root cause surfaced, and a scoping decision - the same one specified in the framework above - that kept root-cause sign-off fully engineer-owned even as candidate generation accelerated. There was a fourth factor too, and it is less flattering, but it belongs in this record rather than out of it: one workload class initially scoped as AI-eligible for stimulus generation produced a stimulus set that cleared validation against the historical regression baseline, then required rework once a mid-program design-spec revision landed, because the AI-guided generation had been trained implicitly on the pre-revision workload profile. Validation caught this before it went further. That is the entire point of having a validation layer, and this article treats the incident as evidence the layer earns its cost rather than as a mark against the framework.

External evidence, drawn from different design scales and organizational settings, at least supports the plausibility of the mechanisms behind the pilot's result - it does not stand in for the pilot's own numbers. Coverage-closure acceleration using learned sampling reports gains large enough to be operationally meaningful, not marginal [16], and reinforcement-learning-guided stimulus generation on bounded designs reaches full coverage several times faster than undirected approaches [10]. Verification environment configuration-time reductions of the magnitude reported in scripted co-verification frameworks [18] are consistent with the kind of setup-time savings that made the pilot's compressed schedule is workable in the first place. None of these external figures is claimed as direct proof of the pilot's own outcome. They are cited as context for why the underlying mechanisms - faster stimulus availability, faster triage, lower setup overhead - are independently plausible at other scales too.

Table 2. Pilot-scale outcome alongside external verified productivity metrics from cited empirical studies.

Metric	Source	Design / Program Scale	Applicability Caveat
~60% of RTL shifted earlier in program schedule	Author's primary data (premium-tier mobile SoC pilot)	Flagship SoC program, full emulation estate	Program-specific outcome; not a controlled experiment
Coverage-closure gains from learned sampling exceed random and prior optimization-based techniques by a substantial margin	Gal, Haber, Ziv [8]	Academic/industrial benchmark designs	Cited as plausibility context, not direct evidence of pilot result
RL-guided stimulus generation reaches full FSM coverage several times faster than random	Gadde et al. [16]	JTAG TAP controller (bounded design)	Smaller-scale design; illustrates mechanism, not pilot-scale magnitude
Verification environment configuration time reduced from days to ~30 minutes	Zulberti et al. [21]	Hardware/software co-design environment	Supports plausibility of setup-time savings enabling earlier emulation entry

Capital efficiency, while not the pilot's stated objective, shaped how it was resourced in practice. Emulation hardware is a capital-intensive asset class, and a governance layer that reduces wasted machine-hours on invalidated or

mis-scoped stimulus runs carries a capacity-utilization benefit independent of the shift-left schedule result. That benefit was never formally measured in the pilot. It is reported here as a secondary observation, not a claim.

5. Discussion

The pilot findings and the literature converge on the same tension from different angles, and it helps to name the tension directly rather than let it stay implicit. RTL-level ML research keeps demonstrating that algorithmic performance on narrow verification tasks continues to improve, in some cases substantially [16], [11], [13], [10] - but improvement on a narrow task metric does not answer the question a program executive is actually stuck with, which is how much of that improvement can be trusted without independently re-verifying it anyway. The productivity literature suggests, separately, that a large share of verification effort is consumed by overhead unrelated to any single algorithm's performance - environment setup, regression maintenance, triage labor [18] - which indicates adoption gains may come as much from cutting that overhead as from a better generation model. The framework in Section 3 is built to capture value from both sources at once, rather than betting the entire productivity case on model improvement.

Trust-validation literature is where this article's governance argument finds its firmest external footing, and also its clearest limitation. Systematic reviews of formal validation approaches for AI behavior in safety-critical settings turn up only a small number of studies that directly address validating AI outputs themselves [22] - which suggests the field has not yet produced a mature, general answer to the exact question a tape-out-critical program needs answered: how much independent checking an AI-generated output requires before it earns trust. The three-layer framework's validation requirement is a practical, program-level answer to that open gap, not a theoretical resolution of it. It manages the trust problem operationally - by refusing an AI output any authority it has not earned against a trusted reference - rather than claiming to have solved it.

Two limits deserve to be stated plainly rather than buried in a caveat. The framework does not substitute for coverage-model rigor. Nothing in the pilot results should be read as license to relax coverage requirements in exchange for faster AI-guided cycles; the pilot's own governance layer treated coverage exceptions as categorically off the table, and this article holds the same position as a general rule, not a pilot-specific one. Separately, false-negative risk in tape-out-critical debug paths remains genuinely open - this framework manages that risk, it does not eliminate it. A validation layer built on historical regression baselines is only as strong as those baselines, and a truly novel failure mode with no precedent could in principle clear validation

and still be wrong (the mid-pilot incident in Section 4 is a mild version of exactly this scenario). The framework's answer to that residual risk is organizational rather than technical: keeping root-cause sign-off engineer-owned means a person, not a model, stays accountable for the call in precisely the cases where validation is weakest. That is a limitation. It should be treated as one by any program considering this framework, not smoothed over as a solved problem.

6. Conclusion

Capacity investment alone will not solve emulation productivity for flagship SoC programs. Unbounded AI automation will not solve it either. The argument this article has developed points to something narrower and, in the author's view, more defensible: the constraint on pre-silicon emulation throughput is increasingly a governance and integration problem before it is a technical one, and closing it requires deliberately scoping where AI assists, validating every AI-generated output against a trusted reference before anyone acts on it, and governing the whole arrangement through an operating-review cadence with rollback criteria fixed before deployment rather than invented after something goes wrong. The pilot outcome discussed here shows that this bounded approach can produce a measurable shift-left result without touching the coverage discipline tape-out signoff depends on. The three-layer framework - scoping, validation, governance - is offered as a transferable practice for any verification organization facing the same pressure: a schedule and cost environment that makes unbounded automation tempting, and a tape-out risk profile that makes it unacceptable.

References

1. S. Fine, A. Freund, I. Jaeger, Y. Naveh, A. Ziv, Y. Mansour, "Harnessing Machine Learning to Improve the Success Rate of Stimuli Generation," IEEE HLDVT, 2005.
<https://ieeexplore.ieee.org/document/1568823>
2. L. Pierre, F. Pancher, R. Suescun, J. Quévremont, "On the Effectiveness of Assertion-Based Verification in an Industrial Context," FMICS, LNCS vol. 8187, pp. 78-93, 2013.
https://link.springer.com/chapter/10.1007/978-3-642-41010-9_6
3. Accellera Systems Initiative, Portable Test and Stimulus Standard, Version 1.0a, 2019.
https://accellera.org/images/downloads/standards/pss/Portable_Test_Stimulus_Standard_v2.1.pdf

4. IEEE, IEEE Standard for SystemVerilog, IEEE Std 1800-2017, 2018. <https://ieeexplore.ieee.org/document/8299595>
5. Accellera Systems Initiative, Universal Verification Methodology (UVM) 1.2 User's Guide, 2015. <https://www.accellera.org/downloads/standards/uvm>
6. G. Huang et al., "Machine Learning for Electronic Design Automation: A Survey," ACM TODAES, vol. 26, no. 5, art. 40, 2021. <https://arxiv.org/abs/2102.03357>
7. M. Liu, N. Pinckney, B. Khailany, H. Ren, "VerilogEval: Evaluating Large Language Models for Verilog Code Generation," IEEE/ACM ICCAD, 2023. <https://arxiv.org/abs/2309.07544>
8. N. Pinckney, C. Batten, M. Liu, H. Ren, B. Khailany, "Revisiting VerilogEval," ACM TODAES, vol. 30, no. 6, art. 91, 2025. <https://arxiv.org/abs/2408.11053>
9. H. Liu, Y. Lu, M. Wang, X. Yao, B. Yu, "LLM-Assisted Circuit Verification: A Comprehensive Survey," ASP-DAC, 2026. <https://ieeexplore.ieee.org/document/11420692>
10. D. N. Gadde, T. Nalapat, A. Kumar, S. Simon, D. Lettnin, W. Kunz, "Efficient Stimuli Generation using Reinforcement Learning in Design Verification," SMACD, 2024. <https://arxiv.org/abs/2405.19815>
11. Z. Yan, W. Fang, M. Li, M. Li, S. Liu, Z. Xie, H. Zhang, "AssertLLM," ASP-DAC, 2025. https://zhiyaoxie.com/files/ASPDAC25_AssertLLM.pdf
12. R. Qiu, G. L. Zhang, R. Drechsler, U. Schlichtmann, B. Li, "AutoBench," ACM/IEEE MLCAD, 2024. <https://portal.fis.tum.de/en/publications/autobench-automatic-testbench-generation-and-evaluation-using-llm/>
13. J. Ye et al., "From Concept to Practice: an Automated LLM-aided UVM Machine for RTL Verification," IEEE/ACM ICCAD, 2025. <https://arxiv.org/abs/2504.19959>
14. K. Xu, J. Sun, Y. Hu, X. Fang, W. Shan, X. Wang, Z. Jiang, "MEIC: Re-thinking RTL Debug Automation using LLMs," IEEE/ACM ICCAD, 2024. <https://arxiv.org/abs/2405.06840>
15. S. Liu, Y. Lu, W. Fang, M. Li, Z. Xie, "OpenLLM-RTL," IEEE/ACM ICCAD, 2024. <https://arxiv.org/abs/2503.15112>
16. R. Gal, E. Haber, A. Ziv, "Using DNNs and Smart Sampling for Coverage Closure Acceleration," ACM/IEEE MLCAD, 2020. <https://ieeexplore.ieee.org/document/9394648>
17. A. Jayasena, P. Mishra, "Directed Test Generation for Hardware Validation: A Survey," ACM Computing Surveys, 2023. <https://dl.acm.org/doi/10.1145/3638046>
18. L. Zulberti, S. Di Matteo, P. Nannipieri, S. Saponara, L. Fanucci, "A Script-Based Cycle-True Verification Framework to Speed-Up Hardware and Software Co-Design," Electronics, vol. 11, no. 22, art. 3704, 2022. <https://www.mdpi.com/2079-9292/11/22/3704>
19. M. Abdollahi, S. F. Yeganli, M. A. Baharloo, A. Baniasadi, "Hardware Design and Verification with Large Language Models: A Scoping Review," Electronics, vol. 14, no. 1, art. 120, 2025. <https://www.mdpi.com/2079-9292/14/1/120>
20. A. Rammal, K. Ezukwoke, A. Hoayek, M. Batton-Hubert, "Root Cause Prediction for Failures in Semiconductor Industry," Scientific Reports, vol. 13, art. 4934, 2023. <https://www.nature.com/articles/s41598-023-30769-8>
21. S. Alsaedi, A. A. A. Gad-Elrab, A. Noaman, F. Eassa, "Two-Level Information-Retrieval-Based Model for Bug Localization," Electronics, vol. 13, no. 2, art. 321, 2024. <https://www.mdpi.com/2079-9292/13/2/321>
22. A. Newcomb, O. Ochoa, "Formal Methods for Safety-Critical Machine Learning: A Systematic Literature Review," Frontiers in AI, vol. 9, art. 1749956, 2026. <https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2026.1749956/full>