

Applied AI Engineering for EDA

Building Reliable Agentic Systems for Chip Design

Nikhil Sharma

Leader, Solutions Architect, AWS



What we'll cover in the next 25 minutes

1

The problem worth solving

Verification is the bottleneck the industry cannot hire its way out of

2

What "AI engineering" means for EDA

It is not ML engineering — you are not training a model

3

Layer 1 — Grounding

Making EDA tool output, not model confidence, the reward signal

4

Layer 2 — Agent architecture

A supervisor and narrow, tool-bound specialist agents

5

Beyond the design flow

Agentic AI for infrastructure and operations, not just design

6

Layers 3–4 — Evaluation & governance

How you know it works, and where humans stay in the loop

7

Where this stands today

What works reliably now, what still breaks, and the takeaways

Verification is the bottleneck the industry cannot hire its way out of

60–70%

of every design schedule is spent in verification

50B+

transistors in a single leading-edge SoC

10,000s

of test cases in one verification plan

- The core engineering tension

EDA tools are deterministic.
AI is probabilistic.

That gap is not a product limitation to work around — it is the central engineering challenge of applying AI to chip design. This is a hard engineering problem, not a demo problem.

Verification engineers are the scarcest resource in semiconductors — and the shortage is widening.

AI engineering builds on foundation models — it does not train them

- **ML Engineering**

Builds and trains models from scratch.

- **AI Engineering**

Builds production applications on top of foundation models — prompt engineering, RAG, agents, evaluation.

— Chip Huyen, *AI Engineering* (O'Reilly, 2024)

You are not training a model. You are making a foundation model reliable enough to trust in a chip design flow.

- **Foundation Model**

LLMs for RTL reasoning

- **Prompt Engineering**

Tool-call formatting, output constraints

- **RAG**

PDK retrieval, LRM lookup, design history

- **Agents**

Supervisor + specialist design agents

- **Evaluation**

Simulation pass/fail as ground truth

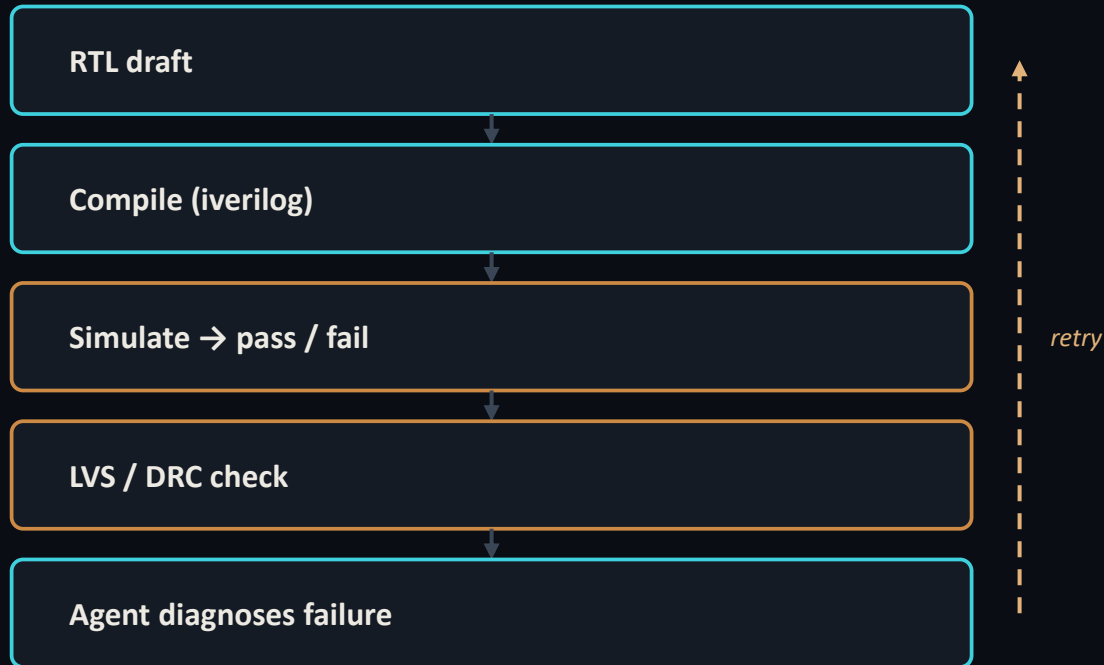
- **Infrastructure & Scheduling**

Compute, storage, and job scheduling for heterogeneous EDA workloads

Skipping to agents before solving evaluation is how you get unreliable systems.

EDA tool output — not model confidence — is the reward signal

The closed-loop fix cycle



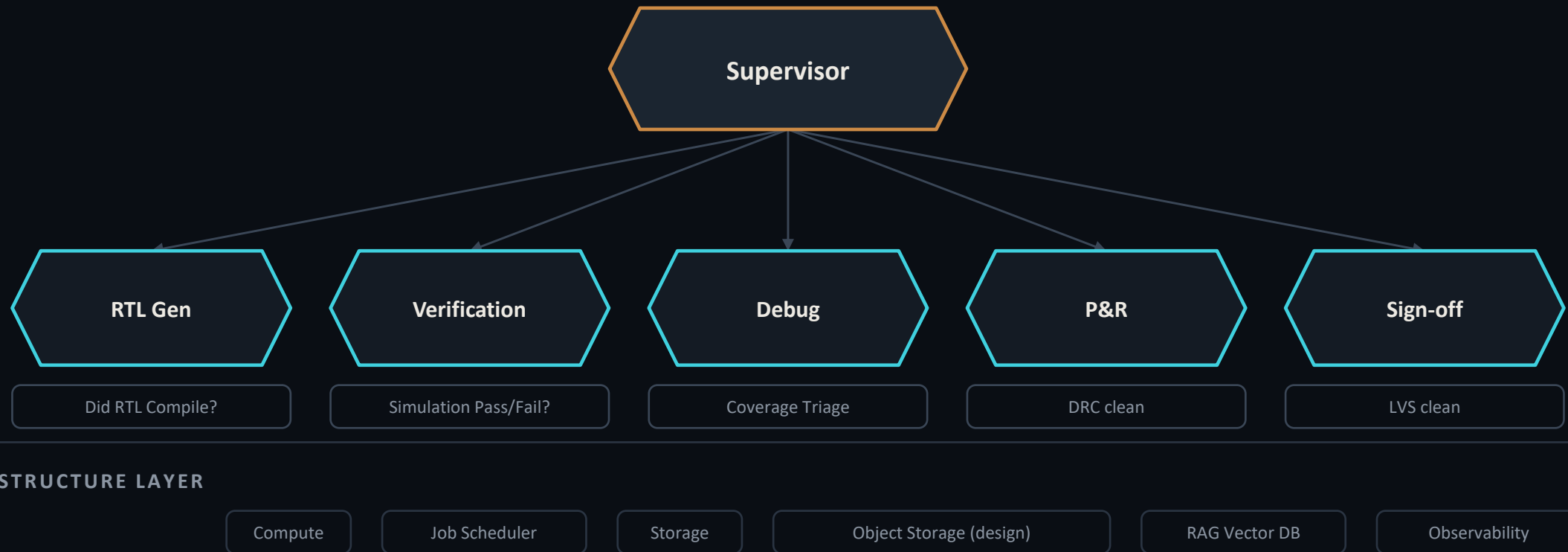
Bounded iterations, then escalate to a human — no silent failure.

Why RAG for EDA is hard

- ▶ **IP-sensitive** — PDKs and netlists cannot leave the secure compute boundary; the vector store must be air-gapped.
- ▶ **Structurally heterogeneous** — netlists, logs, PDK tables, and LRMs need different chunking and indexing strategies.
- ▶ **Precision-critical** — a timing constraint retrieved one version stale can silently corrupt a downstream decision.

The agent must be tool-grounded, not self-referential. If the simulation fails, the agent reads the log — it does not reason about whether it should have failed.

A design-flow agent is a supervised set of narrow specialists



The orchestrator plans, sequences, and arbitrates conflicts between specialists — each with its own tool-grounded success criterion.

Agentic AI runs the infrastructure underneath the design flow, too

Everything so far has been agents that help design the chip. There is a second, less obvious dimension: agents that run the infrastructure the design flow depends on — compute, storage, licenses, and the pipelines connecting them.

● License Utilization

EDA license seats are scarce and expensive. An agent watching checkout and queue patterns can flag an idle seat, or a coming bottleneck, before it blocks a regression run.

● Incident Investigation

When a job fails, a license server hiccups, or a storage volume acts up, an agent correlates logs, telemetry, and recent changes to find root cause on its own — cutting resolution time from hours to minutes.

● Elastic Scaling

Scale compute up ahead of a regression push, back down after — instead of provisioning for peak capacity, and paying for it, all year round.

The common thread: this works best where compute, storage, and networking already expose standard, programmable APIs. Without that, an operations agent has nothing consistent to act on — same principle as the grounding slide, just one layer down.

Reliability is engineered through evaluation, not assumed

The hardest problem in agentic systems is evaluation. Most AI teams lack objective ground truth. In EDA, you have it: a passing simulation is a passing simulation.

● Unit

Tool-call correctness

- ▶ Did the agent call sbatch with valid parameters?
- ▶ Did it parse the sim log correctly?
- ▶ Automatable — run on every commit

● Integration

Flow completion

- ▶ Does the RTL → simulation loop complete?
- ▶ Is the GDSII valid and DRC-clean?
- ▶ Run on reference designs after model updates

● Regression

Does it still work?

- ▶ Re-run known-good designs after model updates
- ▶ Detect silent degradation before production

Common failure modes:

Hallucinated tool params

Context overflow

Silent degradation

Self-referential fix

An AI judge cannot replace a passing simulation — reserve LLM judgment for diagnosis only.

Humans gate stage transitions, not every step



- **Spec Approval**

Agent generates an RTL spec from NL input.
Human reviews before any code is written.



- **Sign-off Gate**

No output promotes to physical design
without simulation passing and human
review of coverage.



- **Tapeout Authorization**

DRC-clean GDSII requires explicit human
release. No agent submits to a foundry
autonomously.

*The self-correcting fix loop runs autonomously
within each stage. Humans gate stage
transitions — not every retry.*

- **IP security: engineering constraints**

- ▶ Design data never leaves the secure compute boundary — no public internet path.
- ▶ Every AI decision is traceable to a tool output or a human approval, logged to the agent session.
- ▶ Tenant isolation — an agent in Product A's context cannot access Product B's PDK or netlist data.

Art of the possible: what works now, what still breaks

● Works reliably now

- ▶ Verification orchestration — regression management, coverage gap detection, failure triage.
- ▶ Self-correcting fix loops — diagnose, propose a fix, retry up to N times, then escalate.
- ▶ RAG-assisted design — retrieval over PDK specs, LRMs, and reference designs improves first drafts.
- ▶ Analytics and observability — cost attribution, iteration velocity, per-agent performance.

● Still breaks

- ▶ LLM-driven RTL generation for non-trivial designs — first sim attempt usually fails beyond simple blocks.
- ▶ Heterogeneous tool interfaces — no mature cross-vendor protocol; MCP helps, but every vendor boundary needs custom work.
- ▶ Context management across stages — P&R databases and full-chip netlists routinely exceed context windows.

Honest benchmark reading: vendor claims (10×, 240×, 400×) are task-specific micro-benchmarks. End-to-end design-cycle acceleration is the metric that matters — and it is harder to achieve and harder to measure.

Three engineering principles, one urgency

01

Ground in the tool

Simulation pass/fail, LVS clean, DRC zero — these are your reward signals. Never let the agent override a deterministic verdict with a probabilistic rationalization.

02

Scope the agents

Narrow, tool-bound specialists under an orchestrator beat one monolith. Each agent needs its own success criterion from the tool it drives.

03

Evaluate relentlessly

A passing simulation is a cleaner ground-truth signal than any human labeling dataset. Build the eval harness around it — unit, integration, regression.

04

The urgency

67,000-engineer shortfall in the US alone by 2030, and design complexity compounds annually. Agentic verification is a staffing response, not a research agenda.

Thank you

Questions & discussion

Nikhil Sharma · Leader, Solutions Architect, AWS

shmnikh@amazon.com