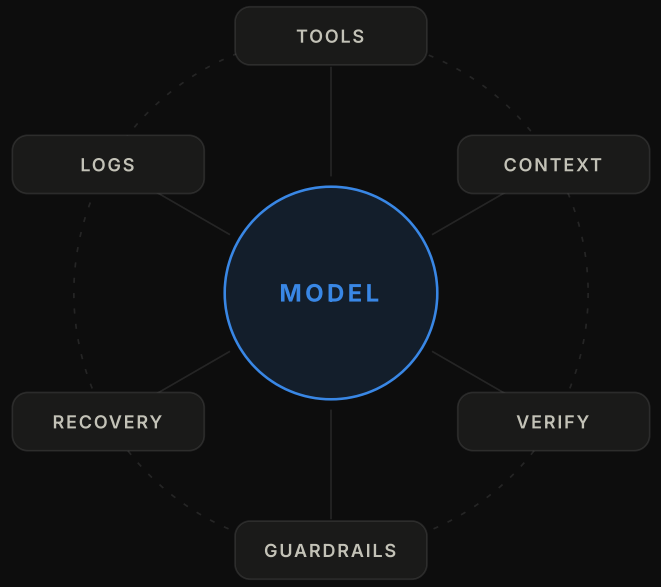
For builders & decision-makers

Harness Engineering

How reliable AI agents actually get built, and why the model was never the hard part.

Bharadwaj Pendyala / ACM Dallas / Aug 2026



WHO'S TALKING

Bharadwaj Pendyala

Lead Member of Technical Staff at Salesforce. I work on enterprise workflow systems and the tooling around agents.

Most of this deck comes from things that broke on me.

SLIDES	bharad.dev/talks
LINKEDIN	in/bharadwajpendyala
GITHUB	bharadwaj-pendyala
X	@bharadwaj_py

THE ONE IMAGE TO KEEP

The model is the engine. The harness is the car.

Nobody buys an engine to get to work. They buy transport they can trust.

THE UNCOMFORTABLE TRUTH

Your demo works. That's the problem.

A demo has to be right once. A product has to be right the ten-thousandth time. From what we've seen, that gap is structural rather than bad luck.

WHERE THE LEVERAGE ACTUALLY IS

Most pilots never ship, and everyone blames the model.

Majority

of GenAI & agent pilots (2024-25 surveys) never reached production

Top 3

blockers cited: *reliability, trust, governance*. Not raw capability

Fix

usually isn't a smarter model. It's the engineering around it.

SAME WEIGHTS. NOTHING RETRAINED.

~40% → 90%+ same model.

The gap wasn't intelligence. It was engineering. Most teams we've worked with aren't waiting for a smarter model. They're underinvesting in the harness.

WHAT A HARNESS IS

Onboarding for your AI hire.

A genius with no tools, no docs, no manager, and no review gives you confident, plausible, sometimes-wrong work.

LOOP

Plan · Act · Verify

TOOLS

Actions & permissions

CONTEXT

Memory & retrieval

GUARDRAILS

Sandbox & approvals

VERIFY

Evals & recovery

WATCH

Logs & human handoff

THE MATH NOBODY WANTS TO HEAR

The task is never as reliable as the step.

Reliability compounds multiplicatively. So does flakiness.

PER-STEP × STEPS = TASK

95% per step feels like an A.

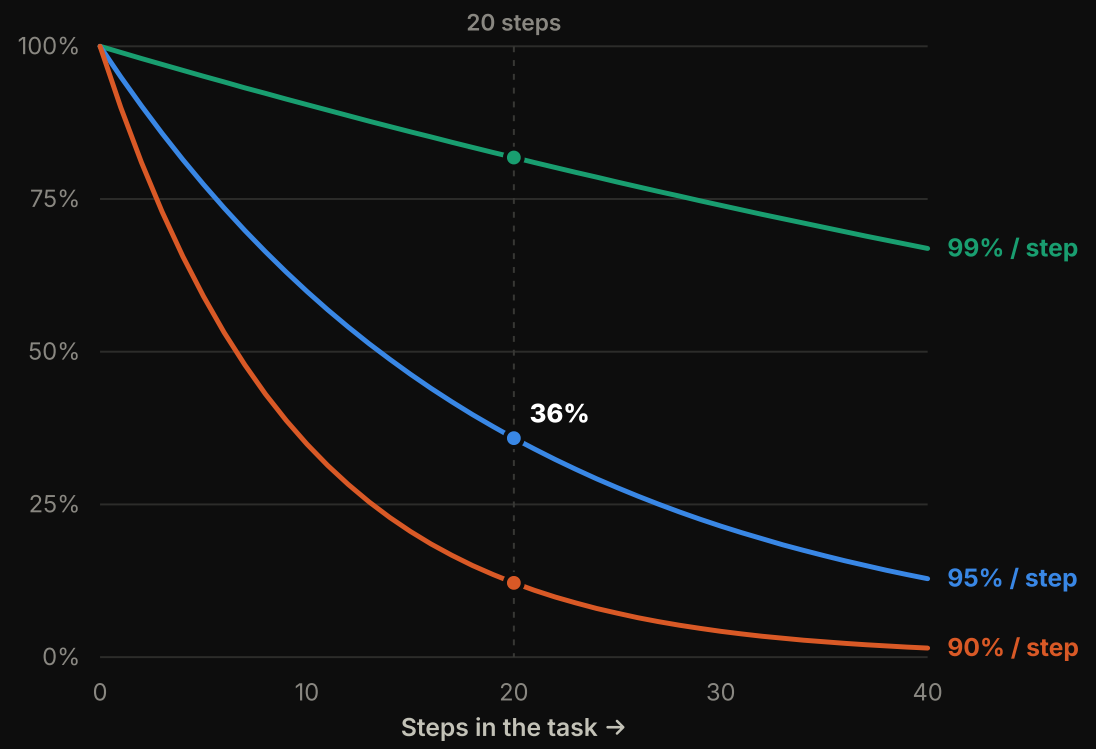
Over 20 steps it's a coin flip that loses two times out of three.

36%

0.95^{20} · 95% per step, 20 steps

82%

0.99^{20} · still fails 1 task in 5



TWO LEVERS, ONE DISCIPLINE

Chase nines, or verify and recover.

LEVER 1

Push per-step to 99.9%+

Expensive, asymptotic, and never enough for a truly long task. There's a ceiling.

LEVER 2

Verify each step & recover

A checked step with a retry has effective reliability far above its raw rate. No ceiling.

40 steps @ 97% → **30%** → verify + one retry (99.5%) → **82%** same model.

THE PATTERN THAT HAS PAID OFF MOST FOR US

Plan → Act → Verify → repeat.

“The model says it's done” is not verification. A failing test is.
The compiler doesn't hallucinate, so let it be the judge.

DESIGN THE SURFACES THE MODEL TOUCHES

Error messages are some of the highest-signal tokens you can hand a model.

A BUG YOU SHIPPED

`Error: invalid`

Nothing to recover from. The model guesses and flails.

A GUARDRAIL

`path must be absolute, got ./foo, use /repo/foo`

The model self-corrects on the next turn.

Then bound the retries. Most recoverable failures we've seen resolve in 2 or 3 attempts. Cap it, or the agent loops on burning your token budget.

THE RULE ABOUT RULES

If a rule must always hold, put it in code the model can't talk its way out of.

Not the system prompt. Constrained decoding,
allowlists, permission gates, pre/post-tool hooks.
Enforced by the harness, not requested of the model.

SHOW, DON'T TELL

Two guardrails. About 20 lines total.

“Put it in code” isn't abstract. Here it is in a tool you can open tonight: a rule the agent can't talk its way out of, then a door it physically can't open.

GUARDRAIL 1 · MAKE IT EARN THE EDIT

No green tests, no code edit.

The JSON *is* the hook. It registers a script to run *before* every edit. The script can say no, and the agent can't argue.

Verification becomes a wall, not a suggestion the model can skip.

.claude/settings.json — THIS IS THE HOOK

```
"PreToolUse": [{ "matcher": "Edit|Write",
  "hooks": [{ "command": "…/tdd-gate.sh" }] }]
```

↳ which runs this command:

.claude/hooks/tdd-gate.sh — THE SCRIPT IT RUNS

```
# the tool call arrives as JSON on stdin
FILE=$(jq -r '.tool_input.file_path')

if [[ "$FILE" == */src/* && ! -f .tests-passed ]]; then
  echo "Run the tests first" >&2
  exit 2   # 2 = block; stderr goes to the agent
fi
```

GUARDRAIL 2 · SHRINK THE BLAST RADIUS

Production is a door it can't open.

No script, just config. A `deny` list the model can't reach past.

deny is checked first and always wins. You can't allow-list around it.

.claude/settings.json — ENFORCED, NOT REQUESTED

```
{
  "permissions": {
    "deny": [
      "Bash(rm -rf:*)",        # no mass deletes
      "Edit(prod/**)",          # hands off prod
      "Read(.env.production)"   # can't read secrets
    ]
  }
}
# checked deny → ask → allow. deny always wins.
```

SAME PRINCIPLE, TWO LAYERS

One makes it work right. One caps the damage.

THE HOOK · DYNAMIC

Governs process

Runs logic, checks state, decides in the moment. “Earn the edit.” Plan, act, verify with teeth.

THE PERMISSION · STATIC

Governs blast radius

A flat rule that never runs code and can't be reasoned with. A confused agent still can't touch prod.

You changed the model by exactly **zero**. Twenty lines of the car around the engine, and you can try it this afternoon.

MANAGE IT, OR THE TRANSCRIPT POISONS THE RUN

Context is reliability.

Over a long run the agent's worst enemy is often its own transcript. One early false “fact” gets restated for 40 steps.

- **Compact** old turns so the goal survives.
- **Retrieve** only what's relevant, to beat lost-in-the-middle.
- **Pin** verified facts; re-ground against reality, not the log.
- **Sub-agents** get fresh context, a small job, a small answer back.

WHAT FAILURE ACTUALLY LOOKS LIKE

Agents don't crash. They succeed at doing the wrong thing.

COHERENT WRONG ANSWER

Zero rows moved

Infers the column is `user_email`. It's `email`. Every SQL step perfect. Migration reports clean.

THE REVERT LOOP

Fails the same way forever

Fix → flaky test fails → revert → rewrite the identical fix. No memory of failure.

SILENT NO-OP

"Deployed successfully"

Script early-returns, exits 0. Old version keeps serving. Nobody notices for hours.

"Exit 0" is not "it worked." Verify outcomes against intent, not return codes.

BLAST RADIUS IS A DESIGN PARAMETER

Give an agent root and a wrong assumption.

It will confidently execute the disaster.

- **Least privilege** and sandboxes, so a bad command hits a throwaway container.
- **Dry-run by default**; human approval on irreversible steps.
- **Checkpoints & git branches** per task; full trajectory logs.
- **Reversibility** is what lets you safely grant more autonomy.

ONE DIAL TO SET THEM ALL

Autonomy scales with your verifier and your undo button, not your optimism.

Short leash where the agent is blind. Long rope only where a test can catch the fall.

PUBLIC, NAMEABLE, EXPENSIVE

The harness, present and absent.

Top: no harness. Bottom: the harness, run as a business.

KNIGHT CAPITAL

~$440M in 45 minutes

Automated trading with no kill switch. The cautionary tale every risk officer already knows.

AIR CANADA

Invented a refund policy

The chatbot made one up; a court ordered them to honor it. A guardrail on approved policy would have prevented it.

KLARNA

Work of ~700 agents

Two-thirds of chats, after guardrails & escalation. Then in **2025 they re-hired humans** when quality slipped. The envelope, live.

WAYMO

Gated by the envelope

Redundancy, remote operators, telemetry. The rollout waited on the harness, not a smarter vision model.

RELIABILITY IS A MULTIPLIER, NOT A FEATURE

The math for the budget meeting.

Net value = (success × tasks × value) – (error + oversight + token cost)

Reliability is the multiplier on the left *and* the reducer on the right.

THE 90%-DONE TRAP

If a human re-checks everything to find the missing 10%, that's not automation. It's a scavenger hunt.

WORSE THAN NOTHING

An agent that takes the wrong action can be worse than no agent. Now someone has to notice and undo it.

THE PREREQUISITE FOR ALL OF IT

If you can't measure reliability, you can't sell it.

Evals are regression tests for agents. Invest in them first. You can't improve what you can't measure.

- **Programmatic graders**: did the test pass? did the right tool get called?
- **Run in CI** on every prompt, tool, or model change.
- **Turn "feels flaky"** into a number you drive up, sprint over sprint.
- **De-risk upgrades**: a scary model swap becomes a data-backed call.

THE DURABLE ASSET

The model is rented. The harness is owned.

COMMODITY

The model

You and your competitor call the same frontier weights. It's a rented input.

MOAT

The harness

Tools, evals, permissions, recovery, domain context. Nobody else has yours.

DURABILITY

Next year's model

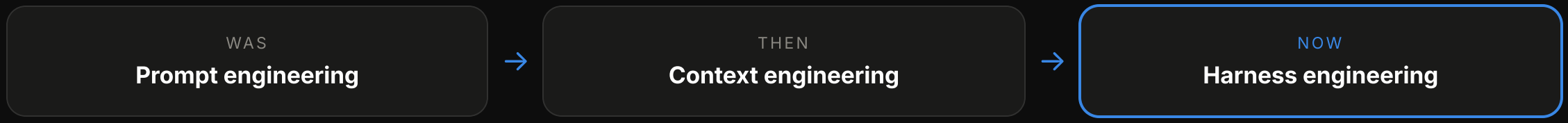
Build for the model you'll have, not the one you're fighting. Delete workarounds on schedule.

Prompts copy in an afternoon. A real eval suite takes months. We think that asymmetry is where the moat sits.

Every failure your harness catches is a test your competitor hasn't written yet.

WHERE WE THINK THIS IS HEADING

The harness is the new application layer.



Prompts are hope. Tools are leverage. Evals are proof. MCP did for tools what HTTP did for documents. Now harnesses compose, and agents are starting to build agents.

THE WHOLE TALK IN ONE SENTENCE

Reliability isn't a property of the model. It's a property of the system around it.

Smarter models raise the ceiling. Better harnesses raise the floor, and the floor is where reliability lives.

IF YOU KEEP THREE IDEAS

Stop writing prompts. Start writing systems.

01

The model isn't the agent

The harness turns raw capability into dependable behavior.

02

Reliability compounds

So verification and recovery beat waiting for a smarter model.

03

The harness is the moat

That's where trust, safety, and differentiation tend to live.

Thank you. Now, what's *your* "run the tests"?

bharad.dev/talks
Slides, speaker notes, and every link on this slide.

- Deck & notes: bharad.dev/talks
- in/bharadwajpendyala
- github.com/bharadwaj-pendyala
- @bharadwaj_py
- bharadwajpendyala@gmail.com