// A talk in 25 minutes

The Judge *Problem*

Why your agentic verifier is silently rotting in production

SPEAKER — VSEVOLOD FOUNDING ENGINEER · MULTI-AGENT SYSTEMS

// Two years with LLMs in production

Who judges the *agent*?

→ Then

Thrad

Ad infrastructure inside conversational AI.

→ Then

Isara Labs

OpenAI-backed multi-agent neolab, San Francisco.

→ Now

Stealth

Founding engineer, early-stage multi-agent startup.

Same problem, every team, every domain: the verifier gets bolted on last.

The judge is the *hardest* component in an agentic system.

Not the easiest.

We built our intuitions about it backwards — and almost every production verifier is, in a precise sense, rotting: degrading with no error, no alert, no signal, until the number it emits means nothing.

// A broken judge and a working judge emit the same artefact

A number.

// The agent fails

LOUDLY

Wrong output. Angry user. Broken test.
An event your tooling is built to surface.

// The judge fails

SILENTLY

No exception when it approves a bad answer.
The absence of an event — which nothing surfaces.

The verifier is the only component whose failures are invisible by construction.

// The judge evaluates a blend of quality and bias — measured, reproduced

It isn't grading correctness.

POSITION BIAS	Prefers whichever answer is shown first, independent of content.	Zheng et al. 2023
VERBOSITY BIAS	Prefers the longer answer, even when the length is padding.	Zheng et al. 2023
SELF-PREFERENCE	Favours text that looks like its own output. Effect tracks perplexity — it prefers what it finds easy to predict, not what is correct.	Wataoka et al. 2024

None of these appear in the demo — they are statistical, and emerge at scale, exactly when you have stopped reading every output.

// Self-preference range across judges on ArenaHard

−38% to +90%

The swing in a judge's verdict depending solely on who wrote the answer it is grading.

Implication. If your agent and your judge share a model family, you are not getting verification. You are getting a model nodding at its own reflection.

Source: UDA, arXiv:2508.09724 (ArenaHard panel)

// Put a person on top of the judge to catch it

The safety net has the *same hole*.

Asked to verify an LLM's judgment, people disproportionately just **agree with it** — even when it is demonstrably wrong. The Asch conformity experiment, run with a model as the confident voice in the room.

two correlated verifiers, stacked — not defence in depth

Source: rubber-stamp effect, Dietz et al. 2025

// Most teams put the strong model on the agent, a weak one on the judge

This is backwards.

// The conventional setup

Strong agent · weak judge

"Judging is easier than doing, so a small model can check the big one." Spend tokens on generation, economise on verification.

// What verification actually requires

Reconstruct → locate → know correct

Spotting the one subtle flaw in a plausible proof is harder than writing a plausible proof. A judge can't catch a mistake on a problem it couldn't solve itself.

limited-reasoning bias: a weak judge on a frontier task is rubber-stamping with extra steps and a confident tone.

// Verification is easier than generation — but only with a cheap certificate

Does the task have a certificate?

YES — a cheap check exists

- Factorisation → multiply back
- Convex optimisation → dual certificate
- Code → run it

A cheaper judge + the tool can verify a stronger generator. Lean on the tool.

NO — no shortcut to checking

- Is this summary faithful to the contract?
- Did this synthesis represent sources honestly?
- Will this plan actually achieve the goal?

Verification = the same reasoning as generation. The judge must be as strong, or stronger.

Source: verification asymmetry, Wei 2025 · arXiv:2509.17995

// Does more capability mean a better self-judge? No.

Generation and verification are *different capabilities*.

01 **They don't move together**

Training a model to generate better does not improve its self-verification. Scaling generation leaves self-checking behind.

02 **It doesn't grow with scale**

Self-verification ability does not reliably increase with model size. Verification is a distinct skill, not a byproduct of intelligence.

03 **Much of it is fake**

Models emit the words of checking — "let me verify" — but the step has near-zero causal effect on the final answer. The ritual without the work.

Source: self-verification dynamics, arXiv:2602.07594 · CRITIC, arXiv:2305.11738

You cannot get verification for free by adding a *"now check your work"* step to the agent's own prompt.

Without external feedback, self-correction can make results worse — the model talks itself out of a correct answer. Language models don't reliably know what they know. Asking the same model to judge its own work asks the blind spot to inspect itself.

Verification you can trust has *independent purchase* on the problem — a different model, an external tool, a real outcome.

// Use several judges and vote — right in spirit, usually wrong in execution

// Same family, averaged

Correlated copies

Three judges that share self-preference, position bias, and the same blind spots — because they share the same training. Averaging launders the bias into a number that looks trustworthy and isn't.

// Different families, disagreeing

Claude · GPT · Gemini

Agreement marks the easy cases. The split cases are where ambiguity lives, or where one judge's bias is firing — a live map of where your risk actually is.

The rule almost nobody follows: *never let a model be the sole judge of itself. The disagreement isn't a problem with your verification — it is your verification.*

// Which judges do you call, on which task, at what cost?

Treat judge selection as a *multi-armed bandit*.

Each judge / combination	`= an arm`
Route a task, learn if the verdict held	`= a reward`
Human check · downstream test · escalation	`= the signal`

Explore → exploit. Learn which judge is reliable for which kind of task, then route each task to the judge that has earned it. The naive alternative — call all judges every time — spends your most expensive judge on the easiest case and learns nothing.

// An algorithm from 1933 that has aged remarkably well

Pick each arm by its probability of being the best.

EARLY

Explore

Little is known. Try different judges, watch the outcomes.

LATER

Exploit

Concentrate on the judges that keep being right for each task type.

BOUND

Logarithmic regret

The cost of learning is small and bounded. Bad judges drop out fast.

A QoS-aware variant was built specifically for runtime verification — guaranteeing a quality threshold with high confidence. The verification problem, in the language of bandits.

Sources: Agrawal & Goyal arXiv:1111.1797 · QoS-aware TS, Belzner & Gabor

// The algorithm matters less than the stance it forces

STATIC ENSEMBLE

"I already know how to verify."

Knowledge frozen into a config. Wrong more slowly, and silently — the exact failure we started with.

BANDIT ROUTER

"I am still learning to verify — forever."

The agent changes, model versions change underneath you, the task distribution drifts. A judge calibrated last month is already subtly wrong. Routing keeps it honest.

A verifier calibrated once is a guess frozen in time.

// Honest arithmetic — verification is not a rounding error

Strong judging is expensive on purpose.

≈ 2×

strong judge as strong as the agent — per-task cost floor

3–4×

a panel of three strong judges on every output

largest

line in your inference bill, at full strength

The bandit dissolves the bill. Run the full panel only where it's needed — near a boundary, where judges disagreed, where a wrong verdict is costly. The true cost is the strongest judge times the *fraction of tasks that genuinely need one* — and finding that fraction is exactly what the bandit does.

The most expensive verification failure is *never the compute*.

It's the wrong verdict that ships — the bad output your judge waved through, that reached a customer, that someone has to discover, explain, and unwind, after it's caused damage.

You're not spending on verification. You're buying down the probability that your whole system is confidently lying to you.

// Five things, building one of these today

Design for the judge.

01	**Design the verifier before the agent.**	If you can't say how you'll know the agent is right, you have a demo with good lighting — not a system.
02	**Ask if the task has a certificate.**	If yes, lean on the tool and a cheaper judge. If no, make the judge at least as strong as the agent.
03	**Source verification independently.**	Never the same model checking its own work in its own context. That's mostly theatre, and it can make things worse.

→ continues

// Five things, continued

04 **Build the judge as a plural panel.**

Genuinely different model families. Treat their disagreement as the primary signal — the split cases map where your risk lives.

05 **Route, don't vote.**

Judge selection as a bandit. Learn from live outcomes which judge to trust, and keep relearning as everything underneath drifts.

AND ABOVE ALL

Measure the judge continuously, against ground truth, forever. *It's the one component whose decay is invisible. A verifier you are not verifying is already rotting — you just can't see it yet.*

WRONG AGENT

You lose a task.

Local. Loud. Traceable.

WRONG JUDGE

You lose the truth about the whole system.

Global. Silent. Untraceable.

Stop bolting verifiers onto finished agents.
Build the judge first.
Build it stronger than the thing it judges —
and never stop verifying the verifier.

Each result read in full.

[01]	**Zheng et al.** — "Judging LLM-as-a-Judge." Position & verbosity bias. NeurIPS 2023.	2023
[02]	**Wataoka et al.** — "Self-Preference Bias in LLM-as-a-Judge." arXiv:2410.21819. Effect tracks perplexity.	2024
[03]	**UDA** — Unsupervised Debiasing Alignment. arXiv:2508.09724. ArenaHard −38% to +90% range.	2025
[04]	**Dietz et al.** — Rubber-stamp effect: humans agree with LLM judgments (Asch conformity).	2025
[05]	**Wei** — Verification asymmetry & certificates. arXiv:2509.17995.	2025
[06]	**Self-verification dynamics** — Generation ≠ self-verification; "fake verification." arXiv:2602.07594.	2026
[07]	**CRITIC** — "LLMs don't know what they know." Self-correction without feedback. arXiv:2305.11738.	2023
[08]	**Agrawal & Goyal** — Thompson Sampling, logarithmic regret. arXiv:1111.1797.	2012
[09]	**Belzner & Gabor** — QoS-aware Thompson Sampling for runtime verification.	2016

Thank you.

Vsevolod · The Judge Problem