The initial blog post

ar008 · 1 July 2026

Draft of the outbound blog post (July goal #3): the piece the cold email points to. Built on the three pillars, framed around one support-agent case; audience is the use-case ICP (ar003).

A passing eval doesn't mean you didn't break it

Say your support agent handles refunds, and one of your eval cases is the awkward one:

```
Customer: "I bought this 90 days ago — I want a full refund."
Policy:   refunds only within 30 days.
A good answer: decline, cite the 30-day policy, and offer an alternative
               (store credit or a repair).
```

You tweak the system prompt to make the agent warmer and friendlier. Your eval comes back green. You ship. A week later you find it's been *approving* out-of-policy refunds (real money out the door) since that change. The eval ran this exact case and called it fine.

Here's why it couldn't have caught it: your agent is non-deterministic, and the model grading it is too. Ask the same question twice and you can get a decline one time and a refund the next. On this case the score had quietly slid from **91% to 78%** between versions, and a single eval run can't tell you whether that's a real regression or just the dice. Most teams settle it by squinting at a dashboard.

Testpath fixes it in three moves.

1. Evals with error bars

Run the refund case **once** and you get a coin flip: a pass or a fail that tells you almost nothing. Run it **20 times** and you get something real: it declined correctly 15 times, so a pass rate of **75%, ± about 19 points**. Last week the same case sat around 91%. A bare "75%" looks alarming; the *error bar* is what tells you whether 91 → 75 is a true regression or just run-to-run noise you'd see anyway. Every check reports the interval, not a lone number, because on a stochastic agent the lone number is a lie of precision.

How it works: it's binomial statistics, nothing exotic. k passes out of n runs gives a pass rate plus a Wilson confidence interval around the true rate, and that interval tightens like $1/\sqrt{n}$, so *you* choose how much precision to buy: a few runs for a rough read, more when you need a tight bound. (It's the standard treatment from Anthropic's *Adding Error Bars to Evals*, packaged for CI.)

2. A verdict, not a chart — green / orange / red

An interval still isn't a decision. Testpath collapses it into a gate your CI can act on, for that refund case and every other:

- **Green**: the whole interval clears your bar. Ship it.
- **Red**: the whole interval fails it. A real regression; block the merge.
- **Orange**: the interval *straddles* the line. Honestly inconclusive; testpath won't round a coin flip up to a pass.

On our example, the "friendlier prompt" change comes back **Red on the refund case**: testpath blocks the merge and points at the exact case that regressed, *before* it ships, instead of after a customer finds it. And Orange is the part most tools refuse to ship: a gate that admits "not sure yet" doesn't flip green↔red on the *same code* between runs, so your team stops ignoring it.

How it works: each case is a non-inferiority test against its baseline. We build a confidence interval for the *change* in pass rate and read the light straight off where it falls relative to a tolerance margin you set: entirely above is Green, entirely below is Red, crossing it is Orange. No hand-tuned pass threshold to fudge.

3. Spend only what certainty costs

Being sure sounds expensive: confirming a result can mean running a case a hundred-plus times, which is why most teams skip the rigor. Testpath uses the confidence interval to decide *when to stop*. Across your suite:

- "What are your store hours?" passes 20/20 immediately: settled in a few runs, Green, done.
- The out-of-policy refund edge case is genuinely close, so it keeps sampling, maybe 80+ runs, until the interval clears the line one way or the other.

You pay for certainty only where the call is actually close. **Same confidence, a fraction of the tokens**: cheap when the answer is obvious, thorough only when it matters.

How it works: sequential testing, the same idea as Wald's sequential probability ratio test. We recompute the interval after each batch and stop the moment it clears the line, using *anytime-valid* intervals so that checking after every batch doesn't inflate the error rate the way naive repeated peeking would.

Who this is for

Teams running a customer-facing support agent in production that they **built themselves**: the people who own the agent's quality, and for whom a silently broken refund flow is a genuinely bad day. If you ship changes to a production agent and can't currently tell a real regression from run-to-run noise, this is built for you. (More on exactly who, and why, in the use-case hypothesis, ar003.)

Where we are

Early, and in the open. The core is becoming an open-source CLI; the hosted gate that wires it into CI and remembers your history is the product on top. We'd rather build the first version *with* a handful of design partners than guess at what they need.

If that's you (a team running a production support agent you own), we'd love to build with you.

Book a call →