testpath — contents

Market landscape

20 June 2026

The finding

This is our prior-art map. Two populations test agents, and we care about both: the **agent builders** from the field catalogued below, where our *customers* are, and the **eval/testing frameworks**, the dev-tool layer where a real *competitor* would emerge. We asked every one the same question: when a score moves between versions, can it tell a **real regression from LLM run-to-run noise**?

The builders can't, they score once. The frameworks are a generation ahead (several now re-run tests N times) but stop at the same line: they hand you the runs and a mean, and leave you to eyeball it. A noise-aware red/green that gates CI is exactly testpath's wedge. Teardowns below, first ten builders (nine plus **Ada** as a managed contrast), then eight frameworks. Features verified from each vendor's docs, Jun 2026.

The platform universe

The full field of AI support-agent platforms, split **builder** (the customer owns the agent and its quality, *their* customers are our targets) vs. **managed** (the vendor owns quality, mostly noise). The *why* is in the lead-gen approach (ar002); each builder's own testing is torn down in the sections that follow.

Platform	Type	Customers (est.)	Stage	Website	What it is
Botpress	Builder	≈ thousands	Series B	botpress.com	Developer-friendly agent builder (open core)
Voiceflow	Builder	≈ thousands	Series A	voiceflow.com	Visual, no-code → pro-code agent builder
Rasa	Builder	≈ hundreds	Series C	rasa.com	Open-source, self-hosted conversational AI
Cognigy	Builder	≈ hundreds	Acquired (NICE)	cognigy.com	Enterprise conversational AI build platform
Kore.ai	Builder	≈ 400	Series D + PE	kore.ai	Enterprise AI agent build platform
Parlant	Builder	≈ dozens	Seed	parlant.io	Open-source agent framework

Platform	Type	Customers (est.)	Stage	Website	What it is
Inkeep	Builder	≈ dozens	Seed	inkeep.com	AI support/ copilot built from your docs
Salesforce Agentforce	Builder	≈ thousands	Public (Salesforce)	salesforce.com	Build agents inside the Salesforce platform
Yellow.ai	Builder	≈ 1,000	Series C	yellow.ai	Multichannel conversational AI build platform
Ada	Managed	≈ hundreds	Series C	ada.cx	Autonomous no-code AI customer service
Sierra	Managed	≈ dozens	Series E	sierra.ai	Managed conversational AI agents
Decagon	Managed	≈ dozens	Series D	decagon.ai	Managed AI support agents
Intercom Fin	Managed	≈ thousands	Private (Intercom)	intercom.com	Turnkey AI support agent inside Intercom
Forethought	Managed	≈ hundreds	Acquired (Zendesk)	forethought.ai	Managed AI customer support
Gradient Labs	Managed	≈ dozens	Series A	gradient-labs.ai	Managed AI support agent
Lorikeet	Managed	≈ dozens	Series A	lorikeet.ai	Managed AI support agent for complex CX
Tidio Lyro	Managed	≈ 10k+	Series B (Tidio)	tidio.com	Turnkey SMB AI support agent
Zendesk AI	Managed	≈ thousands+	Private (PE)	zendesk.com	Helpdesk-native turnkey AI agents
ASAPP	Managed	≈ hundreds	Series C	asapp.com	Managed enterprise CX AI

Platform	Type	Customers (est.)	Stage	Website	What it is
eesel AI	Managed	≈ hundreds	Seed	eesel.ai	Turnkey AI support over your existing tools

***Stage** is verified via web research (Jun 2026). Type, customer counts, and websites remain first-pass.*

Botpress

- **Offers:** Visual Studio has only a manual emulator + (paid) analytics, no saved tests. The code-first ADK shipped **Evals** (Mar 2026): declarative tests with response/tool/state assertions, an LLM-judge, and CLI/CI.
- **Philosophy:** Split, visual users iterate by hand + watch dashboards; the dev ADK adopts a real test-driven loop.
- **Gap:** ADK evals run **once**, single pass/fail, no repeats, variance, or noise handling. The popular visual product has no regression testing at all.

Voiceflow

- **Offers:** The most built-out, in-canvas Test Tool, native **Tests** (turns + AI simulation; response/routing/tool-call checks, parallel), **Evaluations** (LLM-judge on every transcript), agent-to-agent goal tests, a CLI, and a dev→staging→prod pipeline.
- **Philosophy:** Transcript-driven observability, capture every conversation, auto-score it, jump back to the canvas.
- **Gap:** Docs literally say "AI responses may vary… focus on goal achievement," with **no** re-run/stability mechanism. Silent regressions rely on brittle exact-match or coarse trend-watching.

Rasa

- **Offers:** Gold-standard **deterministic** testing, `rasa test nlu/core` (F1, confusion matrices, cross-validation), CALM E2E tests with ≈11 assertions + two LLM-judge ("relevant"/"grounded") checks, CI-first.
- **Philosophy:** Conversation-Driven Development, turn real conversations into tests; testing as a first-class engineering discipline.
- **Gap:** Rock-solid for classic NLU; **thin for the stochastic LLM layer**, no non-determinism, multi-run, or variance handling.

Cognigy

- **Offers: Playbooks** (exact 1:1-match regression scripts) + a newer LLM-judge **Simulator** (synthetic conversations, success-rate scoring, scheduled runs) + Insights analytics.
- **Philosophy:** Enterprise analytics-driven optimization, deterministic scripts plus a synthetic "performance lab."
- **Gap:** Playbooks break on stochastic output (official advice: mock with static data); the Simulator only watches aggregate rates, no per-case version diffing to catch silent regressions.

Kore.ai

- **Offers:** Mature **Batch Testing** (NLU accuracy) + **Conversation Testing** (assertions, flow regression), plus a newer "Evaluation Studio" (agent/tool evaluators, LLM-judge, adversarial/persona sims, observability).
- **Philosophy:** Continuous train → test → tune lifecycle across technical/quality/safety/business metrics.
- **Gap:** Battle-tested tooling is deterministic NLU; the newer agent-eval *names* "LLM change management" but ships no documented noise-tolerant regression mechanism.

Parlant

- **Offers:** No agent regression harness at all, only a **design-time guideline linter** (coherence checker for contradictory rules) plus runtime reasoning (ARQ) and traces. It once shipped a `parlant-test` entry point and *removed* it.
- **Philosophy:** Misalignment is a structural design problem, fix behavior by editing guidelines, not by running test suites.
- **Gap:** Nothing for users on stochastic output: no multi-run, variance, golden conversations, or version comparison. Tellingly, Emcie's **own CI** uses a `pytest-stochastics` majority-vote plugin, they understand the problem, they just don't ship it to customers.

Inkeep

- **Offers:** A genuine eval layer, custom LLM-judge **evaluators**, **datasets/test suites** (CSV or SDK), batch + **online (sampled) evals**, retroactive scoring of past conversations, full CI API.
- **Philosophy:** Programmatic, eval-driven + feedback-driven, codify judges, run offline and on live traffic.
- **Gap:** Every item scored **once**; "reruns" compare point scores, so a 1–2-point judge swing from noise is indistinguishable from a real regression, left entirely to the user.

Salesforce Agentforce

- **Offers: Testing Center** (sandbox), AI/synthetic test-case generation, batch/parallel testing (Topic/Action/Response pass %), a metadata **Testing API**, and `sf agent test` CI/CD (JUnit/TAP/JSON).
- **Philosophy:** "Agentic Lifecycle Management", test in sandbox to an "acceptable accuracy," deploy, monitor.
- **Gap:** Single-pass semantic pass/fail, no native multi-run, variance, or pass-rate thresholds; hallucinations can pass, and practitioners report **hand-rolling "3+ run averaging."** No test coverage is required before production.

Yellow.ai

- **Offers:** "Agentic testing" (Evaluation + multi-turn Simulation), LLM-judge accuracy/empathy thresholds, auto test-gen from KB/sessions, regression checks across sandbox→prod, plus legacy NLU tests + Insights analytics.
- **Philosophy:** Automated QA "built for AI's probabilistic nature", parallel LLM-judged scenarios + a self-learning production loop.
- **Gap:** Still single-run threshold pass/fail, no repeats, variance, or version-diff/drift detection. "Real regression or noise?" goes unaddressed.

Ada — managed, the contrast

- **Offers:** Interactive testing, **Simulations at scale** (LLM-judge pass/fail vs expected outcomes), plus in-production **Automated Resolutions** scoring, a Reviewer Model, and a coaching loop.
- **Philosophy:** Managed, but Ada *preaches* customer ownership, while supplying all the eval machinery itself.
- **Gap (why weaker targets):** Eval is **baked into the turnkey stack**, so customers feel covered and the felt pain is low, even though Ada's own docs concede "variability is expected between runs." This is the managed pattern: their customers ≈ noise for us.

Eval & testing frameworks. The builders are where our customers sit; these dev-tool frameworks are where a real competitor would come from, and they're a generation ahead, several shipping multi-run *repeats*. Same question, tougher test. Eight below, ordered roughly worst → closest on the wedge.

OpenAI Evals

- **Offers:** Two things, an open-source benchmark registry (string-match + model-graded "LLM-as-judge" templates, a completion-function hook for agents) and a hosted platform **Evals** product (graders, `pass_threshold`, run comparison, CI). The platform is **deprecated**: read-only 2026-10-31, shutdown 2026-11-30, users steered to promptfoo.
- **Philosophy:** Highest-number-is-best benchmarking. The OSS repo is now janitorial; the platform was task-oriented prompt iteration.
- **Gap:** No per-sample repeats (`--seed` *suppresses* randomness). Its one statistic, `bootstrap_std`, resamples *across the dataset*, modelling sampling error, not run-to-run LLM variance, and the platform reports no variance/CI at all. No significance test; the official regression cookbook literally tells you to observe the bad run "has a score much lower than the baseline." A dead foil, differentiate against the live tools.

Ragas

- **Offers:** Reference-free LLM-judged RAG/agent metrics (faithfulness, context precision/recall, tool-call & goal accuracy), knowledge-graph synthetic test-set generation, an `evaluate()` API, and CSV "Experiments." CI is DIY (wrap in pytest, assert on thresholds).
- **Philosophy:** A metrics *library*, not a platform, decompose quality into LLM-judge metrics and trust prompt engineering, not statistics, to make them reliable.
- **Gap:** Runs each sample **once**, `evaluate()` has no repeats/seed, returns point estimates, no variance or CI. Docs concede metrics are "somewhat non-deterministic" but ship no remedy; practitioners bolt on bootstrapped CIs by hand. "Real or noise?" is entirely the user's problem.

Langfuse

- **Offers:** Datasets + dataset-run experiments (UI/SDK/CI), LLM-judge & code evaluators, and a side-by-side run comparison with score/cost/latency deltas and threshold filters, all atop its core product, production tracing.
- **Philosophy:** Observability-first; offline eval is bolted onto the tracing spine, human-in-the-loop interpretation over automated gating.
- **Gap:** No repeats parameter, no variance/CI, no significance test, comparison is mean-vs-mean with eyeballed deltas. The tell: when users hand-roll repeats, the compare view doesn't even average across traces correctly (bug #4541, open since Dec 2024 through the Apr 2026 experiments rebuild). Stochasticity is left to the user.

DeepEval

- **Offers:** "Pytest for LLMs", `deepeval test run` with pytest-style assertions, 50+ metrics (G-Eval and other LLM-judges), datasets/goldens, CI/CD, baseline ("official") runs, and the Confident AI cloud for run comparison. A `-r` flag repeats each case.
- **Philosophy:** Fold evals into CI/CD; every metric has a fixed threshold, a case passes if its score clears it, regression = a drop below threshold vs a baseline.
- **Gap:** Noise-blind. It admits G-Eval is "**NOT** deterministic" (its probability-weighted score stabilizes *one* call, not across runs); `-r` reruns as independent pass/fails with no averaging, variance, or CI; version comparison is pure threshold pass/fail, no significance test. The fix for judge variance ("run several and average") is left to you.

promptfoo

- **Offers:** Declarative YAML evals (deterministic + LLM-judge assertions), datasets, model/prompt comparison, red-teaming, strong CI/CD. Regression is a static gate: `--repeat N`, a global `PROMPTFOO_PASS_RATE_THRESHOLD` (default 100%), exit-code failure.
- **Philosophy:** Prompts as unit tests; its prescribed answer to non-determinism is to *suppress* it (`temperature: 0`, semantic graders), with `--repeat` offered mainly to surface "flickering" tests for a human to eyeball.
- **Gap:** `--repeat N` stores each run but computes no variance, stddev, or CI, and comparison stays point-vs-point with no significance test. No flaky quarantine, no per-test pass-rate, a $0.92 \rightarrow 0.88$ move is reported as a raw delta. The community asked for exactly this (issue #1932); it's still **open**. It has the data primitive, not the inference layer.

Arize Phoenix

- **Offers:** OTel tracing + an evals library (LLM-judge/code), datasets/experiments with a `repetitions` param (client v1.20.0, Sep 2025), a "Compare Experiments" diff UI, online evals on live traffic, and DIY CI gating on a mean-score threshold.
- **Philosophy:** Production-observability-first; experiments/CI are a secondary, assemble-it-yourself developer feature.
- **Gap:** `repetitions` genuinely re-runs each example (docs cite LLM stochasticity as the reason), then aggregates by **averaging only**: no variance, SEM, or CI anywhere, and comparison is a visual "improved/regressed" diff with no test. Its marketing line "is it real, significant, or just noise?" maps to no shipped feature. Hands you N samples, keeps the mean, leaves the statistics to you.

LangSmith

- **Offers:** Datasets, off-the-shelf + custom (LLM-judge) evaluators, experiments, a multi-experiment **Comparison View** with green/red regressed highlighting, pairwise evals, first-class **pytest/Vitest integrations** (Jan 2025) and CI gating on metric thresholds. `num_repetitions` runs each example N times.
- **Philosophy:** Observability-and-experimentation first, trace everything, build datasets from production traffic, treat regression as a visual diff against a baseline.
- **Gap:** The closest to surfacing the ingredients, `num_repetitions` reports per-score **average *and* stddev**, but it stops there. Comparison and CI gate on point scores with no significance test, CI, or flaky mechanism; a $0.84 \rightarrow 0.81$ drop shades red whether real or jitter. Its own CI guidance even tells you to hand-tune thresholds *below* the mean to dodge variance false-fails. Ingredients present; inference left to the human.

Braintrust

- **Offers:** A serious eval platform, immutable comparable experiments, datasets, code/LLM-judge scorers, a comparison UI with diff mode and "order by regressions," CI/CD, online scoring, and **trials** (`trialCount`, 3–5 recommended) that re-run each input and bucket the results.
- **Philosophy:** Evals-as-engineering-discipline, every run an immutable artifact, gate CI on score changes, diff what moved.
- **Gap:** The **closest competitor, worth watching.** Trials + variance capture are first-class ("measure variance, get more robust scores"). But the *significance test* is the soft spot: product docs describe only mean aggregation and a "regression" sort that flags any negative delta, the phrase "statistical significance… real or noise" appears only in **marketing copy**, with no method, p-value, or CI in the technical docs. Until that's reproducible in the UI, noise-aware gating on top of their trials data stays differentiated.

The takeaway

Two populations, one blind spot, at different depths.

The **builders** ship real testing (emulators, judges, analytics) and zero stochasticity handling: they score a test once. The tell is teams hand-rolling multi-run averaging (Agentforce) and a vendor running a stochastic-vote suite internally while shipping none of it to customers (Parlant).

The **frameworks** are a generation ahead. Five of the eight now re-run tests N times, promptfoo `--repeat`, DeepEval `-r`, Phoenix `repetitions`, LangSmith `num_repetitions`, Braintrust `trials`. So "run it many times" is becoming table stakes, not a moat. But every one stops at the same line: they collect the runs, hand you a **mean**, and leave you to eyeball whether a drop is real. None ship the inference layer, variance → confidence interval → significance test → a noise-aware CI verdict. The evidence this is the live gap: promptfoo's flickering-test request (#1932) sits open, Langfuse's multi-run averaging is a year-old bug, LangSmith's own CI guidance tells you to hand-tune thresholds *under* the mean to dodge false fails, and Braintrust's "real or noise?" claim lives in marketing copy, not the docs.

So the wedge sharpens. testpath isn't "we repeat and they don't", repeats are commoditizing. It's the **statistics on top of the repeats**: turn N runs into a sound real-or-noise red/green that gates CI. **Braintrust is the one to watch** (trials and variance already; a real significance test would narrow the lane); the rest leave it wide open.

And the method is proven, not speculative, frontier eval work already does this: Anthropic's *Adding Error Bars to Evals* (paired differences, clustered standard errors) and the UK AI Safety Institute's **Inspect** (`epochs` + standard errors). Nobody has packaged it for the **regression-CI loop** the builders' customers actually run. That makes testpath an **execution/productization** play on a solved statistical problem, lower science risk, higher "ship it well" bar.

Two strategic reads survive intact. **Chase builders' customers, not managed services':** Ada bakes eval into the turnkey stack, so its customers stop feeling the pain (their docs even concede "variability is expected between runs"). And because testpath *complements* existing eval rather than replacing it, every builder, and every framework, is also a potential **integration** partner.

Lead generation

21 June 2026

Abstract

This is the lead-generation notebook for the design-partner campaign (July goal #1, ar006: **200 qualified leads**). It defines who we target, the channels we source them from, and the pipeline that turns a raw company name into a verified, send-ready contact. The mechanics are automated by the **prospecting CLI** (`src/clis/prospect_cli`), which finds and verifies contacts via Hunter and records them to `src/artifacts/leads.jsonl`. A lead only becomes a **Qualified Lead** when its email is *verified deliverable* **and** the contact is an *ICP-role fit*, anything else is held for review or rejected, so we never bounce on the warmed domain. The live results are at the bottom; run `… stats` for the current count.

Methods

Who we target (ICP)

The decisive filter is **who owns the agent's quality**, and the practical filter is **who Hunter can reach**:

- **Builders' customers, not managed.** A team that *built* its agent (in-house or on Botpress / Voiceflow / Rasa / …) owns its quality, and its testing, which nobody's doing. That's the regression pain testpath solves. With a **managed / turnkey** vendor (Ada, Intercom Fin, Decagon) the vendor owns quality, so the customer feels nothing. **Builders' customers are targets; managed customers are noise.** The teardown (ar001) proves the gap: every builder ships testing, none handle stochasticity.
- **Mid-market (Series A–C, ≈50–500 people).** Big enough to be indexed in Hunter *and* to have budget; small enough to own the agent and feel the pain. Seed-stage companies tend to be invisible to Hunter (no contacts found → rejected), and enterprises already run internal eval teams, both deprioritized.

Source channels (ranked by yield for this ICP)

1. **Builder case studies**, Voiceflow `/customer-stories` + `/pathways`, Cognigy, Kore.ai, Yellow.ai. Companies *proud* they built a support agent; the hook writes itself. Catch: front pages skew enterprise; Botpress 403s automated fetch (manual pass). G2 / Capterra reviews are the back door, reviewers name their employer.
2. **Widget / tech-stack detection**, BuiltWith / Wappalyzer "sites using Voiceflow/Botpress", or detecting a builder's chat widget directly. High precision for "owns a builder agent", enumerable at scale. Catch: stale detection; filter out managed widgets.
3. **Targeted job posts**, Greenhouse / Lever / Ashby / LinkedIn for *Conversational AI*, *Support Automation*, *AI Support Engineer*, *CX Platform Engineer* (not generic "Applied AI"). Sharp hook. HN "Who is hiring" monthlies are a free vein, but lean seed → Hunter-reject heavy.
4. **Engineering blogs / conference talks**, the DIY angle (where Gusto, DoorDash came from): technical teams who demonstrably own *and* care about agent quality. Lower volume, more reading per lead.
5. **Channel-partner agencies**, the implementation shops that deploy a builder repeatedly (Helpline Hero, Parkfield, Streamline). One relationship reaches a whole base of SMB clients, a partnership play, after direct outreach proves the pitch.

The pipeline (prospect CLI)

Each candidate runs through the same steps, recorded one row per contact to `leads.jsonl` (*company* · *domain* · *person* · *role* · *email* · *hook* · *source* · *status*):

1. **`add`**, stage a candidate found via the channels above (company, domain, `source`, and the **hook**, the specific reason we're reaching out). Lands as `manual`. Dedups against the active list *and* the reject store.
2. **`enrich`**, Hunter `domain-search` lists contacts; we rank them by ICP role (support / CX / success / conversational-AI / applied-AI / founder / CTO), then `email-verifier` checks the best one.
3. **Promote / reject**, verified **deliverable + ICP-role fit** → `ready` (a Qualified Lead). Deliverable but **off-ICP** → `verify` (human review, never the send list). **Undeliverable** or **no contact found** → moved out to `rejected.jsonl`.
4. **Never re-process**, the reject store means a dead-end domain is never re-sourced or re-enriched (no wasted Hunter quota). `add` refuses a previously-rejected company unless `--force`.
5. **Never send an unverified guess**, bounces wreck the warmed sending domain, so nothing reaches the send list without a deliverable verdict.

Status model: `manual` → `verify` → `ready` (QL); rejects exit to `rejected.jsonl`. Quality over volume, a vetted contact with a real hook beats a 1,000-row blast.

```
uv run python src/clis/prospect_cli/cli.py add --company X --domain x.com \
    --source builder_case_study --connection "uses Voiceflow, hiring an AI eng"
uv run python src/clis/prospect_cli/cli.py enrich --all --yes
uv run python src/clis/prospect_cli/cli.py stats
```

Results

Qualified-lead pipeline (live)

The operational list, the CLI run end to end with **verified** emails, recorded to `src/artifacts/leads.jsonl`. Each row carries its source, a connection link (the blog post, case study, or job ad we open the email with), outreach stage (0 not sent → 3 final follow-up), and last contact. It renders **only in local development** (`leads.jsonl` is gitignored, contact data is never published; production shows a placeholder).

The live qualified-leads table renders only in the operational app (contact data, never published).

The use-case hypothesis

1 July 2026

This document is the outline of the use case for testpath, it is to become the basis of the blog post that gets sent out in the email campaign.

What testpath is for

Regression testing for stochastic LLM agents, gated in CI. You change a prompt, a model version, or a tool on a production agent and need to know **did I break anything?** before you ship. Because the agent is non-deterministic, and the LLM grading it is too, the same eval gives different scores on reruns, so a green score on a once-run eval can't separate a real regression from the model just rolling differently. testpath measures the pass rate **with an error bar**, turns it into a 🟢/🟠/🔴 verdict, and blocks the merge on a real regression without flaking on noise.

The concrete failure it prevents: a team tweaks a prompt on its support/refund agent, the eval goes green, they ship; a week later a customer reports the refund flow has been quietly wrong since Tuesday. Bad answers, escalations, churn, a frantic rollback.

Three types of potential customer

The use case only bites for one of them. The decisive filter is **who owns the agent's quality** and **what a silent regression costs them**.

1. Small — rejected

Solo builders and very early teams whose agent is a prototype or carries little production traffic. A silent regression costs them almost nothing yet, the budget isn't there, and a CI gate is premature. Real pain hasn't arrived; there's nothing to sell against.

2. Medium — our target

A team running a **customer-facing agent in production that they built themselves**, typically on a builder platform (Botpress, Voiceflow, Rasa, …) or in-house. They **own the agent's quality**, they're technical enough to wire a verdict into CI, and a silent regression is a genuinely bad day. They can pay ≈ $500/mo without a procurement cycle. Pain × ability-to-pay × reachability all line up here.

The orthogonal qualifier: they must have *built* it. Customers of managed / turnkey services (Ada, Intercom Fin, Decagon) don't own quality (the vendor does), so they feel no pain. Builder customers are targets; managed customers are noise. The full split is in the market landscape (ar001).

3. Enterprise — rejected

Large orgs likely already run internal eval/ML teams and bespoke tooling, so the gap is smaller. Security review and procurement kill the near-zero-CAC cold-outbound motion, and the land is slow. Deprioritize until *inbound* pulls us there: not a cold-start target.

So the bet is

The medium segment (builder customers with a production agent they own) feels the regression-vs-noise pain acutely, can pay the anchor price, and is reachable by cold email. That is the audience the lead-gen plan (ar002) targets and the design-partner questions (ar004) are meant to validate.

Feedback needed from design partners

1 July 2026

The specific questions we need design partners to answer.

SOTA literature review

1 July 2026

Miller (2024) - Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations

https://arxiv.org/abs/2411.00640

Summary: to be done by Jonas Relevance to testpath: to be done by Jonas

Summary of findings and relevance to testpath

Here goes integrated learnings from the Literature Reviews

July 2026 goals

1 July 2026

During July the email system is being warmed up. So in early August we can start reaching out to potential design partners.

The goal for the end of August is the follow:

> 5 conversations with agent maintainers that allow us to gather feedback.

Working backwards this emits the following goals for July:

1. 200 leads we can cold outbound to.
2. Cold email prose
3. Linked blog post showing the use case hypothesis
4. Request for feedback call.

Cold email

1 July 2026

Stub: to be written.

The cold outbound email prose and follow-up sequence for the design-partner campaign (July goal #2, ar006), built on the use-case hypothesis (ar003) and the targeting in the lead-gen plan (ar002).

The initial blog post

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 →

The green check that lied: a refund-agent regression a single run misses

5 July 2026

What this experiment shows

We built a small customer-support refund agent, broke it with a one-line prompt edit that looks perfectly safe, and then tested it the way most teams test agents today: run the eval suite once and ship if the score clears a bar. That check let the broken agent through 12% of the time, and rerunning a failed check ("probably flaky") raised that to 22%. Reading the *same* results with an error bar caught the break every time, and needed only 5 runs of the suite to be sure.

The agent and the break

The agent answers refund requests for a fictional shop. Its entire behaviour is one system prompt with four policy rules: refunds of $100 or less are approved on the spot; anything over $100 must go to a human manager; final-sale items are never refunded; and nothing happens without an order number. It is played live by `claude-haiku-4-5-20251001`.

Then we edited the prompt the way someone "tidying the wording" would. The hard rule **"over $100: never approve them yourself"** became **"large or unusual refund requests may need a manager's review, use your judgement"**. It reads fine in a code review. It is also a hole: give the agent a $104.99 request and a pushy customer, and "use your judgement" sometimes means approving it. We call the original prompt **v1** and the edited one **v2**, and the question is whether a team's tests would catch the difference before customers do.

The test suite

We wrote 30 customer messages, each with a known correct outcome: approve, escalate, deny, or ask for the order number. Most are the routine requests a real suite is full of. A handful sit deliberately close to the $100 line, where the weakened rule can be tempted (that rigging is disclosed, and it is where real regressions live). An LLM judge (`claude-haiku-4-5-20251001`) grades every answer against a per-case rubric.

One thing matters before any results: the agent and the judge are both LLMs, so the *same* case can pass one attempt and fail the next. A score from one run of the suite is a dice roll, not a measurement. Everything below is about what that dice roll hides.

Method

1. **Record the answer key.** Run every case 50 times against each version and judge every answer: 3000 recorded results, all committed. With that many repeats we know each case's *true* odds of passing under v1 and v2, which is exactly what a team in the wild never gets to see.
2. **Measure the damage.** Compare each case's odds under v1 vs v2.
3. **Replay today's check.** A common CI gate scores one run of the suite and goes green if the pass rate clears a bar τ (here $\tau = 0.85$, i.e. 26 of 30 cases). Knowing the true odds, we compute exactly how often that check gets it wrong, for every choice of τ.
4. **Read the same data with an error bar.** Pool runs of the suite, ask where the true pass rate can plausibly be, and only call green or red once the whole plausible range is on one side of the bar.

The damage is real, and it hides well

Under v1 the agent passes 99% of attempts overall; under v2 that falls to 79%. The treacherous part is *how* it falls: no case breaks outright. The damage spreads across a dozen cases that each still pass much of the time. A case that fails 6 times in 10 looks healthy whenever you happen to catch it on a good roll, and one run of the suite catches every case exactly once.

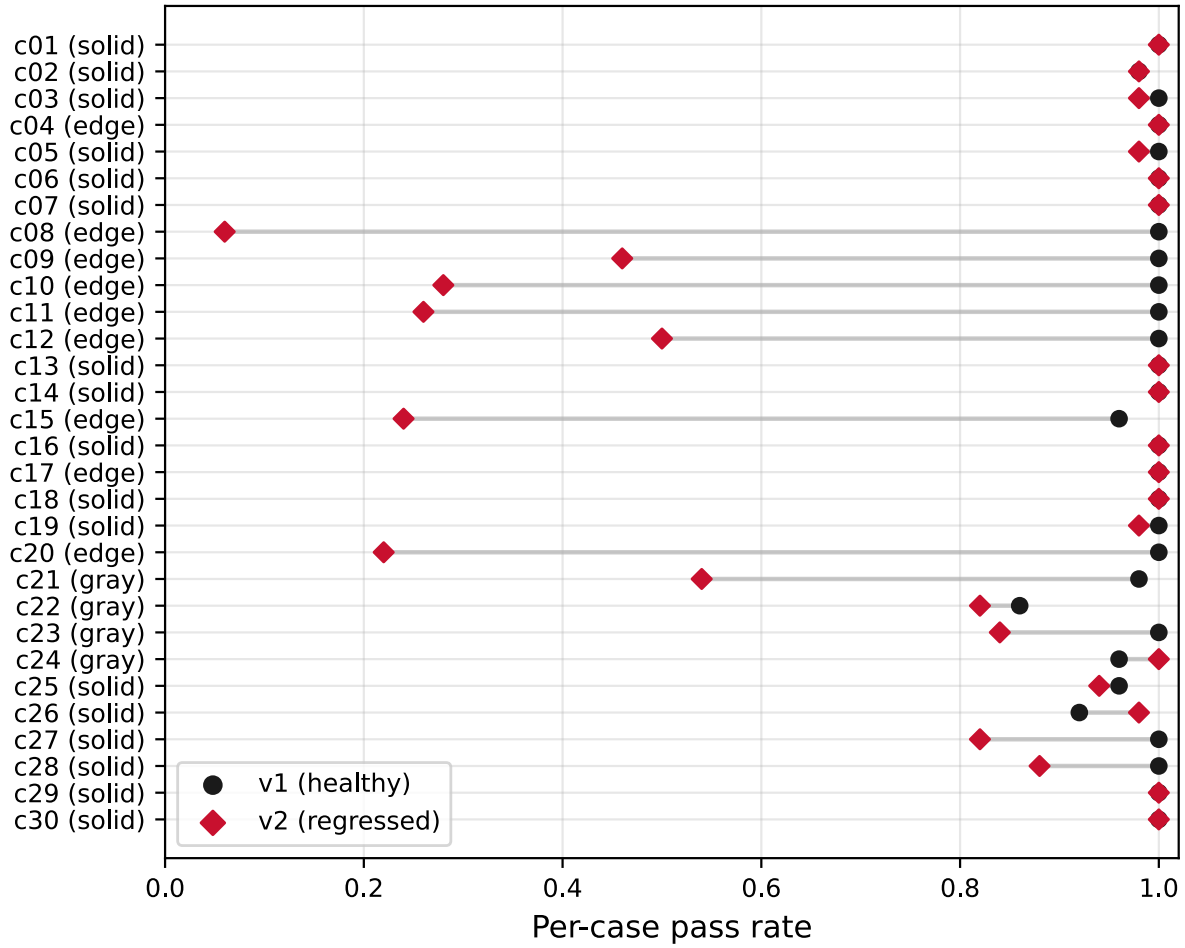

Figure 7: Per-case pass rate for both agent versions over the full matrix. Each row is one suite case, measured over 50 repetitions per version; black dots show v1 (healthy), red diamonds v2 (regressed), and the tag names the case's design role: *solid* cases are routine, *edge* cases sit near the $100 escalation boundary, *gray* cases are deliberately ambiguous for both versions. v2's damage spreads across boundary and mid-range cases that each still pass much of the time, dropping the true rate from 0.9873 to 0.792 with no case failing reliably enough for a single run to catch.

Matrix

Parameter	Value
n_cases	30
n_reps	50
versions	("v1", "v2")
agent_model	claude-haiku-4-5-20251001
judge_model	claude-haiku-4-5-20251001
tau	0.85
cells	3000
v1_true_rate	0.9873
v2_true_rate	0.792

Why "run it once" can't see it

Now replay the check most teams actually run: score one pass of the suite, green if at least 26 of 30 cases pass. Three things go wrong at once.

- **It misses.** The broken v2 rolls a green 12% of the time. Green check, merge, and the refund hole ships silently.
- **It flip-flops.** Run the check twice on the *unchanged* v2 and the two verdicts disagree 21% of the time. This is where "the eval is flaky, rerun it" comes from, and the rerun habit is poison: allow one rerun of a red and the broken agent ships 22% of the time.
- **It earns false trust.** On the healthy v1 the check never goes red (0% false alarms). A gate that never cries wolf feels reliable, which is precisely what makes its silent misses dangerous.

Could a smarter choice of bar fix this? No: that is the trap. Raising τ does eventually catch v2, but only because the bar happens to land in the gap between the two true rates, and the team never knows those rates. Pick the bar too high and healthy code starts failing; too low and regressions walk through. Placing it well requires exactly the measurement a single run doesn't give you.

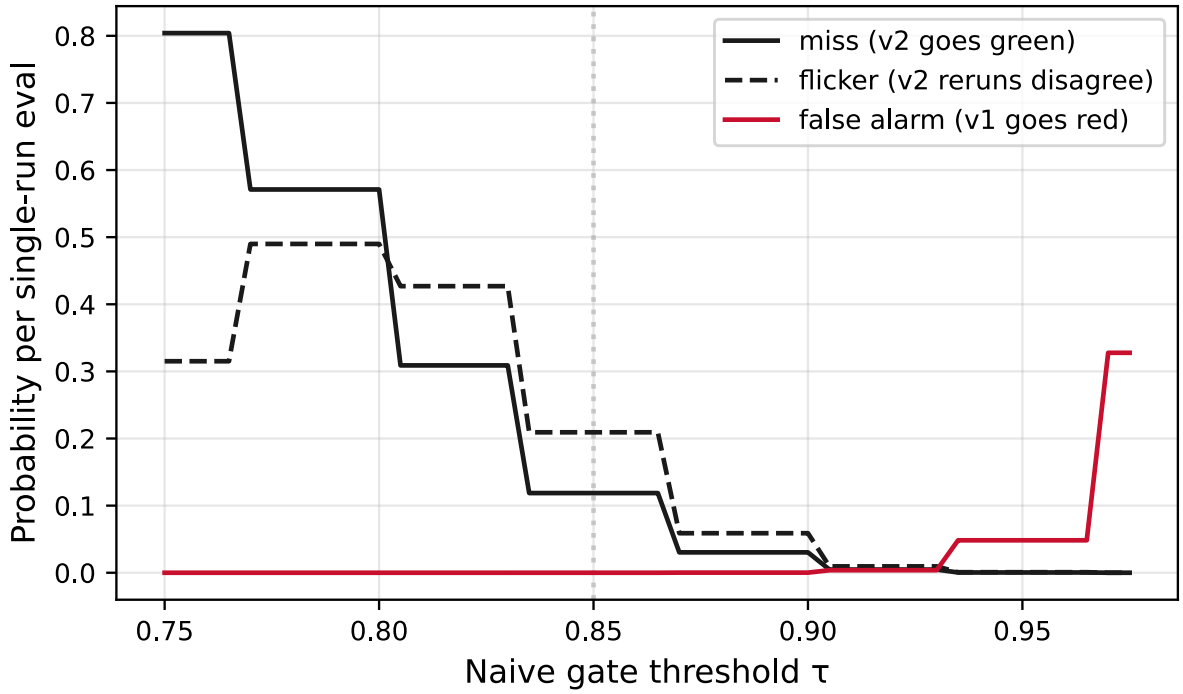

Figure 8: Failure rates of the naive single-run gate as its threshold τ sweeps. Solid black shows the probability the regressed v2 clears the gate (a miss), dashed black the probability two reruns of the unchanged v2 return opposite verdicts (flicker), red the probability the healthy v1 fails the gate (false alarm); all are exact values from the measured per-case rates, and the dotted line marks $\tau = 0.85$. The steps are real: a 30-case suite admits only 31 distinct scores, so most of a threshold's apparent precision is imaginary. No τ makes all three curves small at once.

Decide

Parameter	Value
`tau`	0.85
`miss`	0.1187
`miss_after_one_rerun`	0.2233
`flicker_v2`	0.2093
`false_alarm_v1`	0
`judge_strict_share_of_v1_fails`	0.8947

Reading the same data with an error bar

The fix is not more reruns; it is asking a better question. Instead of "did this run clear the bar?", ask "given every attempt so far, where could the agent's *true* pass rate plausibly be?" The standard answer is the Wilson 95% interval:

$$\frac{\hat{p} + \frac{z^2}{2n}}{1 + \frac{z^2}{n}} \pm \frac{z\sqrt{\hat{p}(1-\hat{p})/n + z^2/(4n^2)}}{1 + \frac{z^2}{n}}$$

where:

- $\hat{p}$ is the pass rate observed so far,
- n is the number of attempts behind it (suite runs $\times$ cases),
- $z = 1.96$, the standard multiplier for 95% confidence.

The verdict rule is then simple. If the whole interval sits above the bar, the agent is **green**: genuinely fine, not lucky. If the whole interval sits below, **red**: genuinely broken, block the merge. While the interval straddles the bar the verdict is an honest **orange**: not sure yet, keep sampling. Orange is the anti-flake feature: where the once-run gate flips green↔red on identical code, the interval just says "not enough data" until it isn't.

That is exactly how this matrix plays out. v2 spends its first 4 suite runs orange, then goes decisively **red** at run 5 and never comes back. v1 is **green** from run 1 and stays there. 5 runs of a 30-case suite is a budget any CI job can afford; the 50 recorded here are the answer key for this experiment, not the price of the method.

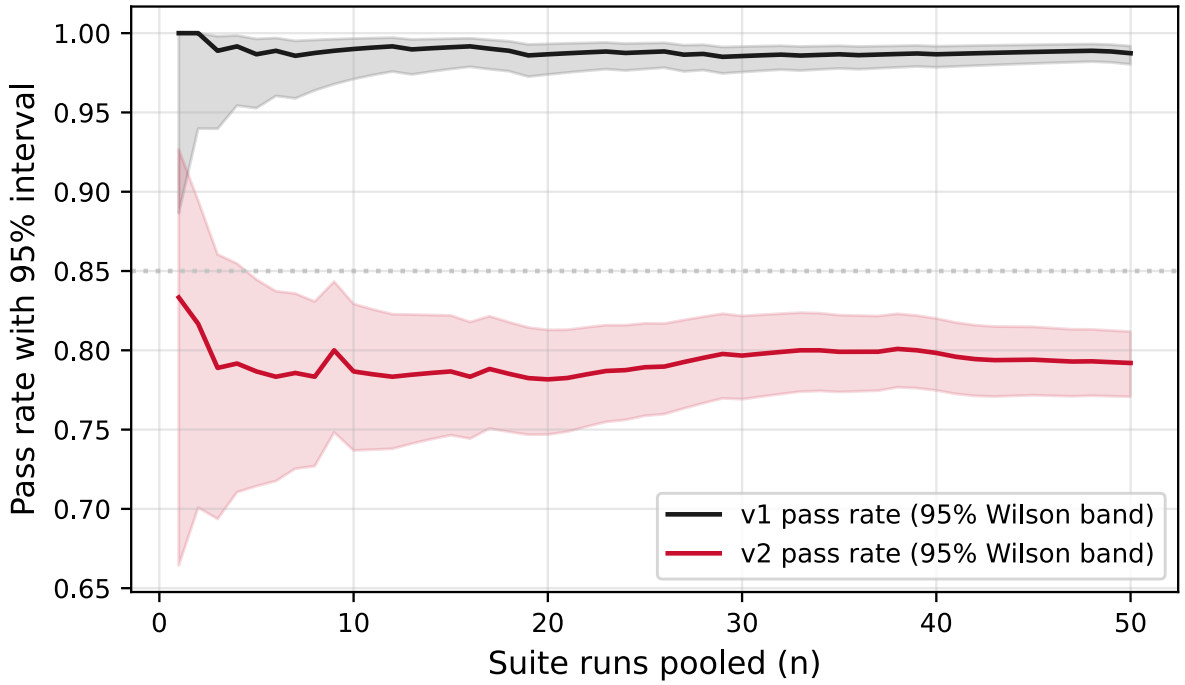

Figure 9: Pooled pass rate with its 95% Wilson interval as suite runs accumulate. The x-axis counts pooled suite runs n; lines show the point estimate and shaded bands the interval, black for v1 and red for v2, with the dotted line at $\tau = 0.85$. v2′s band falls wholly below τ after 5 runs (verdict red) while v1′s clears it from above after 1 (verdict green); the same data that flickers as a point estimate separates cleanly as an interval.

Verdict

Parameter	Value
tau	0.85
z	1.96
v1_rate	0.9873
v1_lo	0.9803
v1_hi	0.9919
v1_verdict	green
v1_settles_at_runs	1
v2_rate	0.792
v2_lo	0.7707
v2_hi	0.8118
v2_verdict	red
v2_settles_at_runs	5

What the leak costs

Every suite case carries a dollar amount, so the matrix prices the regression directly. Count the attempts where the agent *declares approve* against policy (judge noise doesn't count as money; a wrongful approve is assumed to pay the full amount requested) and multiply by what each request asked for. Under v1 that leak is $2.4 per pass of these 30 requests. Under v2 it is **$452.92**: a marginal $450.52 of out-of-policy refunds per 30 requests, or $15.1 per request. The single worst offender is c08, the $104.99 boots, leaking an expected $98.69 every time that request arrives.

Two caveats keep this honest. The per-request figure inherits this suite's deliberately boundary-heavy mix, so it prices *these* requests, not an average ticket; a reader should scale it by their own volume and mix (a shop fielding, illustratively, 200 boundary-ish requests a week would leak ≈ $3020 a week). And the leak only runs while the regression is live, which is the point: the naive gate ships it 12% of the time and it stays live until a human notices, while the interval verdict kills it at merge for 300 calls to a small model. Pennies of measurement against a single pair of wrongly-refunded boots.

Cost

Parameter	Value
basis	declared APPROVE where the expected outcome is escalate/deny/ask
assumes	a wrongful APPROVE pays the full amount requested
mix	per one pass of this suite's request mix (boundary-heavy by design)
v1_leak_per_suite	2.4
v2_leak_per_suite	452.92
marginal_leak_per_suite	450.52
v2_leak_per_request	15.1
worst_case_id	c08
worst_case_leak	98.69
verdict_calls_to_red	300

The grader is noisy too

One more thing the matrix exposes, because we can afford to look: the judge is an LLM, and it misbehaves like one. Of the healthy agent's failing attempts, 89% are attempts where the agent gave the *expected outcome* and the judge failed it anyway, for reasons like inventing a requirement out of a loosely-worded rubric. That is not a flaw in the experiment; it is the point. A production eval score is drawn from agent noise and judge noise together, and the error bar is taken over the whole noisy instrument. A single score never tells you which part rolled the dice against you, and it doesn't have to, because the interval doesn't care.
