INNGEST

AI in Production: The 2026 Benchmark Report

WITH PARTICIPATION FROM ENGINEERS AT

cohere · 11x · LiveKit · mintlify · BÆRSkin TACTICAL SUPPLY CO. · Stuut Technologies · MERCURY · wealthfront · GNOSIS FREIGHT · Remitly

TABLE OF CONTENTS

CONTENTS

EXECUTIVE SUMMARY

We surveyed 130 back-end, full-stack, and AI engineers across companies of all sizes about what they're building, and how they keep it reliable.
We specifically focused on how these teams run asynchronous work in production. Some findings were expected: 68% of respondents are building AI workflows in production.

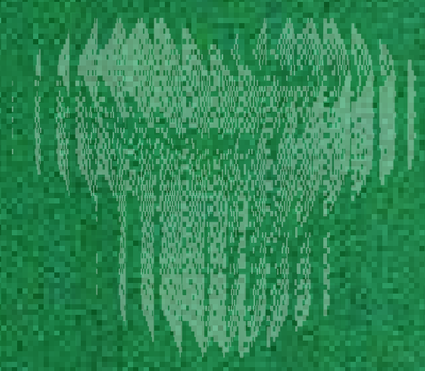
Other findings were less so: Only 19% of those teams are "very confident" their infrastructure can handle 2–3x current scale, and in teams with 500+ engineers, that number is 0%.

That confidence gap is the story of this report.

Why confidence in AI infra wanes as capacity increases, and how tooling choices might be dictating that directionality.

Three key findings rise above the rest:

1. **The reliability burden is growing for teams expanding into AI use cases**.
   20% of teams building AI in production spend up to half of their time on reliability work, with a clear split in burden expansion or reduction based on durability tool in use.
2. **Observability is the #1 problem engineers believe still hasn't been solved.**
   Even respondents using a mix of third-party and homegrown solutions were just as likely to cite a persistent gap in observability across their workflow.
3. **Confident teams use observable, composable stacks.**
   The strongest 3-factor predictor of confidence is using orchestration tools with built-in observability, running AI evals, and diagnosing failures fast. These teams are 3x more likely to report confidence in their stack.



SECTION 1

CONFIDENCE IN SCALING AI

Only

19%

of teams running AI workflows in production said they were very confident in their stack's ability to handle **2-3x** their current scale

0%

of teams with over 500 engineers reported any confidence.

We asked survey participants several questions about the tooling in their stack that supports how they run asynchronous workflows in production. While confidence in the ability to scale AI systems is relatively low, we see an important correlation to combinations of solutions in play.

What we asked

1. What kinds of AI workflows are you running in production?
2. How confident are you in your setup's ability to handle AI workloads reliably at 2-3x your current scale?

Smaller teams, with limited resource to handle scale, reported being more confident than larger teams in the ability to handle 2-3x the load.

This could be because these teams rate their use cases to be relatively simple—already designed for 2-3x scale by nature of the stage they're at.

In an attempt to add more dimension to this result, we looked at how stack choices might correlate with overall reported confidence. We found a few statistically significant trends worth noting.

Confidence in scaling AI workflows by team size

Q13: How confident are you in your setup's ability to handle AI workloads reliably at 2–3× current scale? AI teams only.

We started by asking about the tooling used to build workflows, and observe failures. We also asked about agent evals, and frameworks. These aren't the only layers of a production stack, but they are some of the more emergent, and in some cases, contentious (deeper dives on each in the subsequent sections).

The strongest statistically significant predictors are production evals and the ability to diagnose failures fast—both clearing p<0.05 independently, and together reaching p<0.01. Teams that can diagnose in minutes and have production evals are the most concentrated in the confident group; not one unconfident team has both.

The combination of using a durable execution solution as well as structured evals is the most actionable three-factor predictor (p=0.011), with 49% of confident teams having all three versus 13% of unconfident teams.

On the negative side, the absence of evals is the clearest anti-pattern in the dataset (p=0.038). Teams missing both evals and orchestration-native observability show the sharpest gap (p=0.008). Single-factor items like "any structured orchestration" and "logs only" don't reach significance on their own—they matter, but primarily in combination with other missing layers.

Most statistically significant signals for confidence in scaling AI

Net = % confident (n=73) minus % unconfident (n=15). *** p<0.01, ** p<0.05, * p<0.10, ns not significant. Greyed rows shown for context only.

SECTION 2

DURABLE EXECUTION: MAPPING USE CASES AND TOOLS

56%

of respondents use more than one tool for durable execution, often pairing a custom-built solution with a third-party platform. While Inngest is favored by growth-stage teams, users also report the widest variety of use cases served.

Durable execution is a fault-tolerant approach to running code that ensures functions complete—no matter what. These platforms persist state, handle failures, pause and retry when needed, and provide deep observability for quick troubleshooting and recovery. They are the full-context infrastructure layer that modern production systems are being built on top of.

While the survey showed that types of workflows and jobs run in production are fairly uniform across companies of all sizes, we saw quite a range in what folks were using to ensure durable execution at scale.

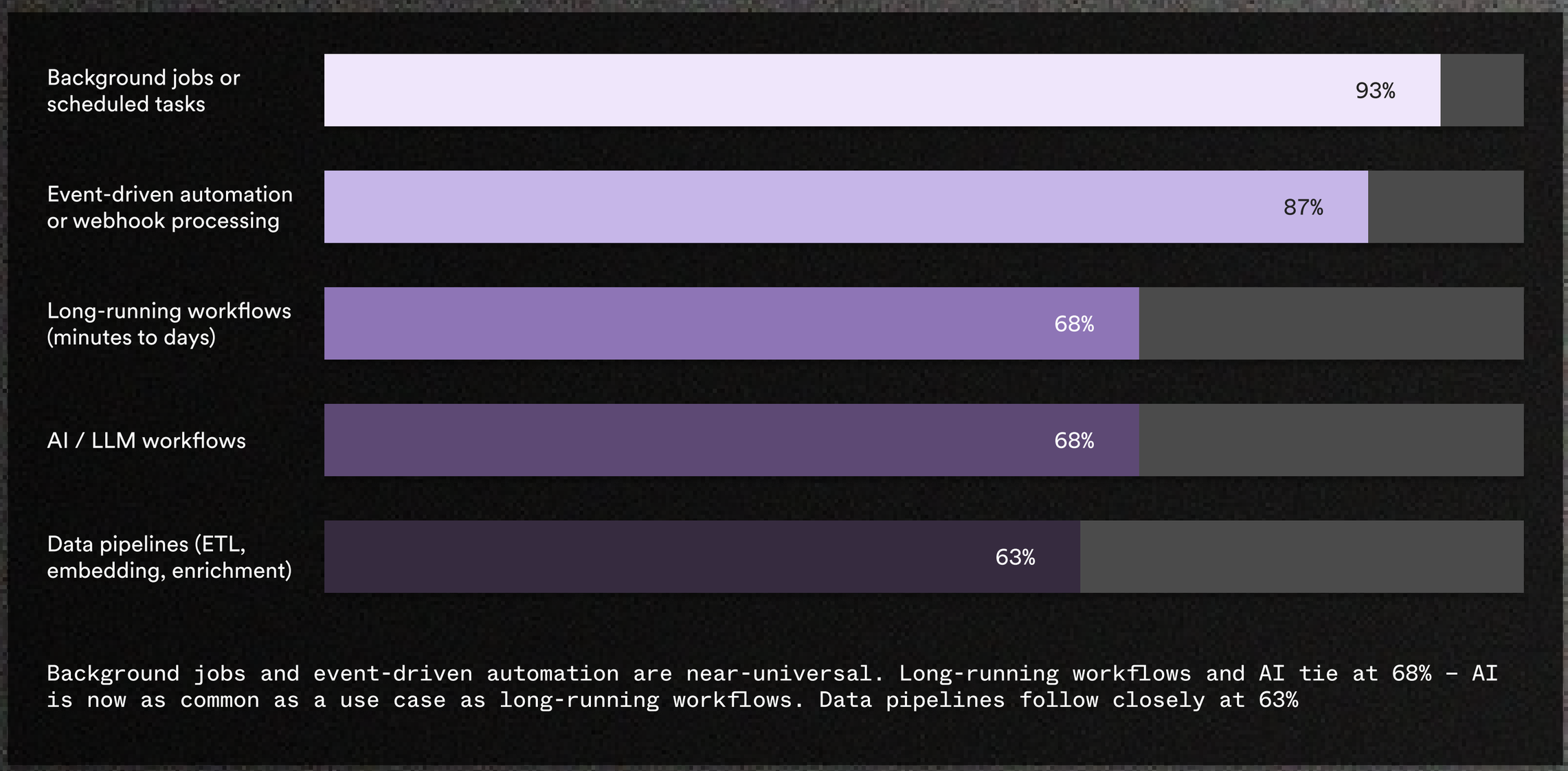
What we asked

1. What types of workflows or jobs are you running in production?
2. What are you using to ensure reliable execution of those workflows and jobs?

Top use cases for durable execution

Background jobs and event-driven workflows are a near-universal use cases for anyone building asynchronous workflows in production. These run at companies of every size, regardless of what teams are building. Long-running workflows and AI/LLM workflows are tied at 68%—AI is now as common a production use case as long-running workflows.

What types of workflows are teams running in production?



Q2: What types of workflows or jobs are you running in production? Q11: Are you currently running AI workflows in production? All respondents n=130. Multi-select – totals exceed 100%.

Orchestration platforms by team, and use case

What teams are building correlates strongly with which orchestration solution they use (or build themselves). 56% of respondents use more than one tool, often pairing a custom-built solution with a third-party platform.

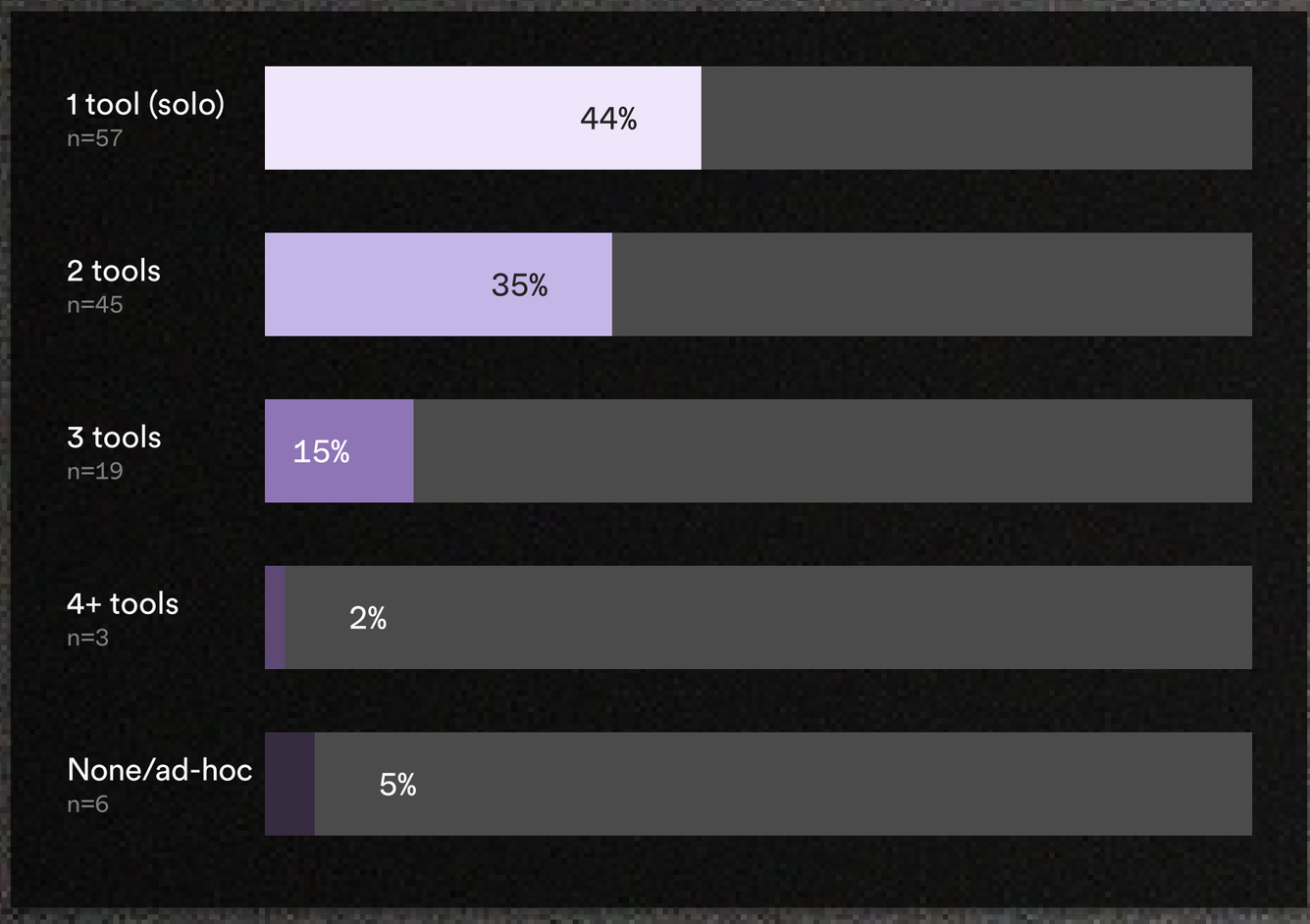
For example, 100% of Temporal users surveyed have another durable execution or orchestration solution running alongside to handle edge cases. Inngest is the platform most often run alone (56%), with 22% of those teams also running something custom-built.

This loyalty may be a function of scale: Inngest is popular among growing teams who want low-friction adoption that scales with them.

But it's also a reflection of capability breadth. Among statistically meaningful solo-tool cohorts, Inngest serves the widest variety of use cases.

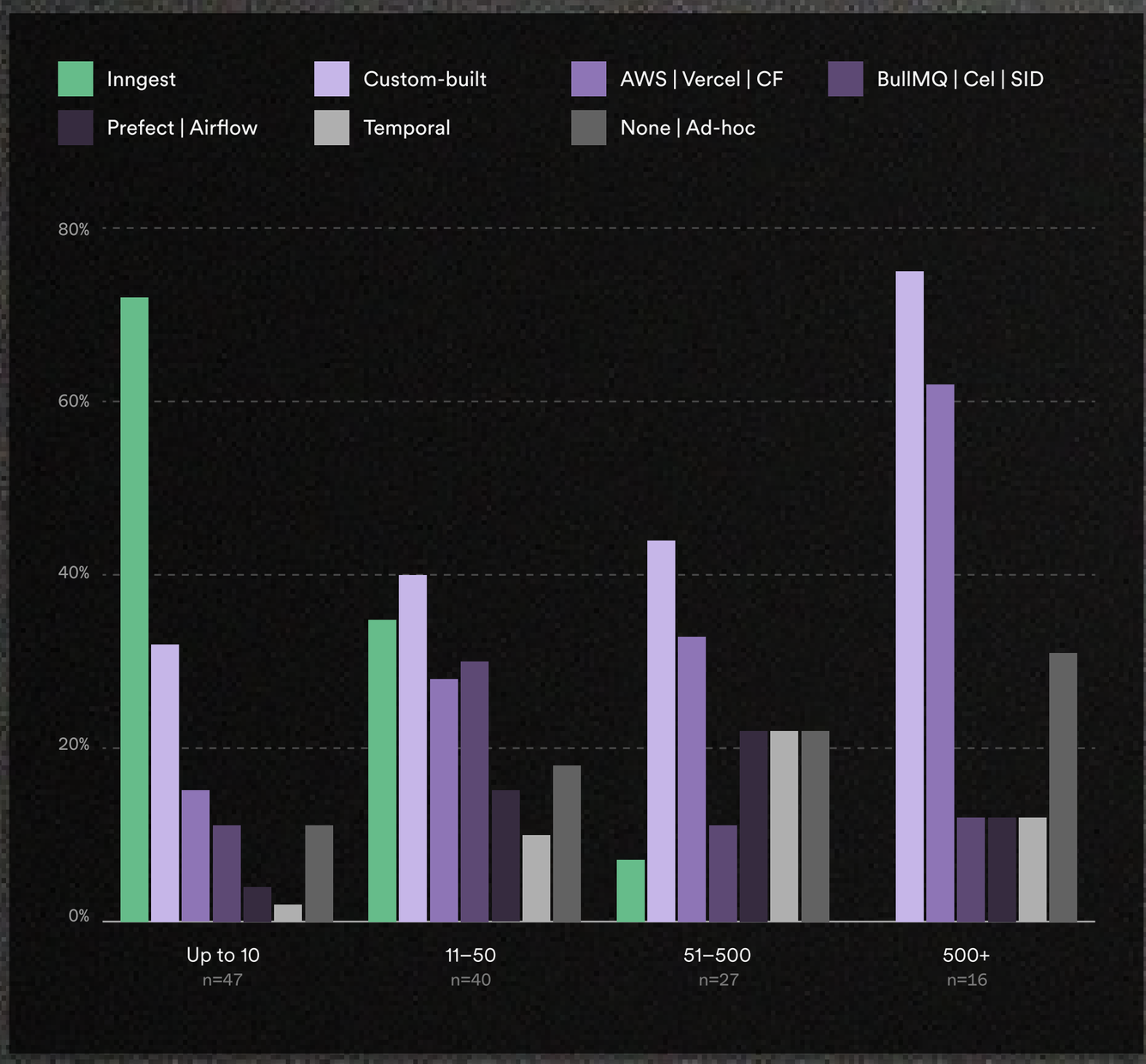
How many orchestration tools are teams using?

All respondents n=130. 56% use more than one tool.



Orchestration platform by team size

% of each size band using each tool. All respondents n=130. Multi-select — totals exceed 100%. % of each size band using each tool.



Q3: What are you using for workflow or job orchestration?

Use case breadth – team using only one orchestration tool

Q2 × Q3 × Q11. Solo users of each tool only. % running each workflow type. Sorted by average across all five use cases.

Temporal solo users (n=3) score the highest average (80%) but are too small to report. Prefect/Airflow solo users (n=2) are also too small – excluded. Among reportable cohorts, Inngest serves the widest variety at 75% average. AWS/Vercel/CF and BullMQ are the most specialized – AWS/Vercel/CF drops sharply on long-running (43%) and AI (29%); BullMQ on data pipelines (17%).

SECTION 3

THE RELIABILITY TAX: WHAT'S DRIVING A CHANGE IN TIME SPENT ON RELIABILITY?

20%

of teams building AI in production spend twice the time on reliability work than those running traditional workflows. Users of Inngest and homegrown solutions report a net reduction in burden to ensure reliability over the last 12 months.

Building AI in production carries different challenges from building traditional software — higher throughput, non-deterministic models, opaque evaluations. The question this section tries to answer is: is it the use case, the team size, or the tooling that makes the biggest difference in how much time engineers spend on reliability? And critically — can anything reduce that time?

What we asked

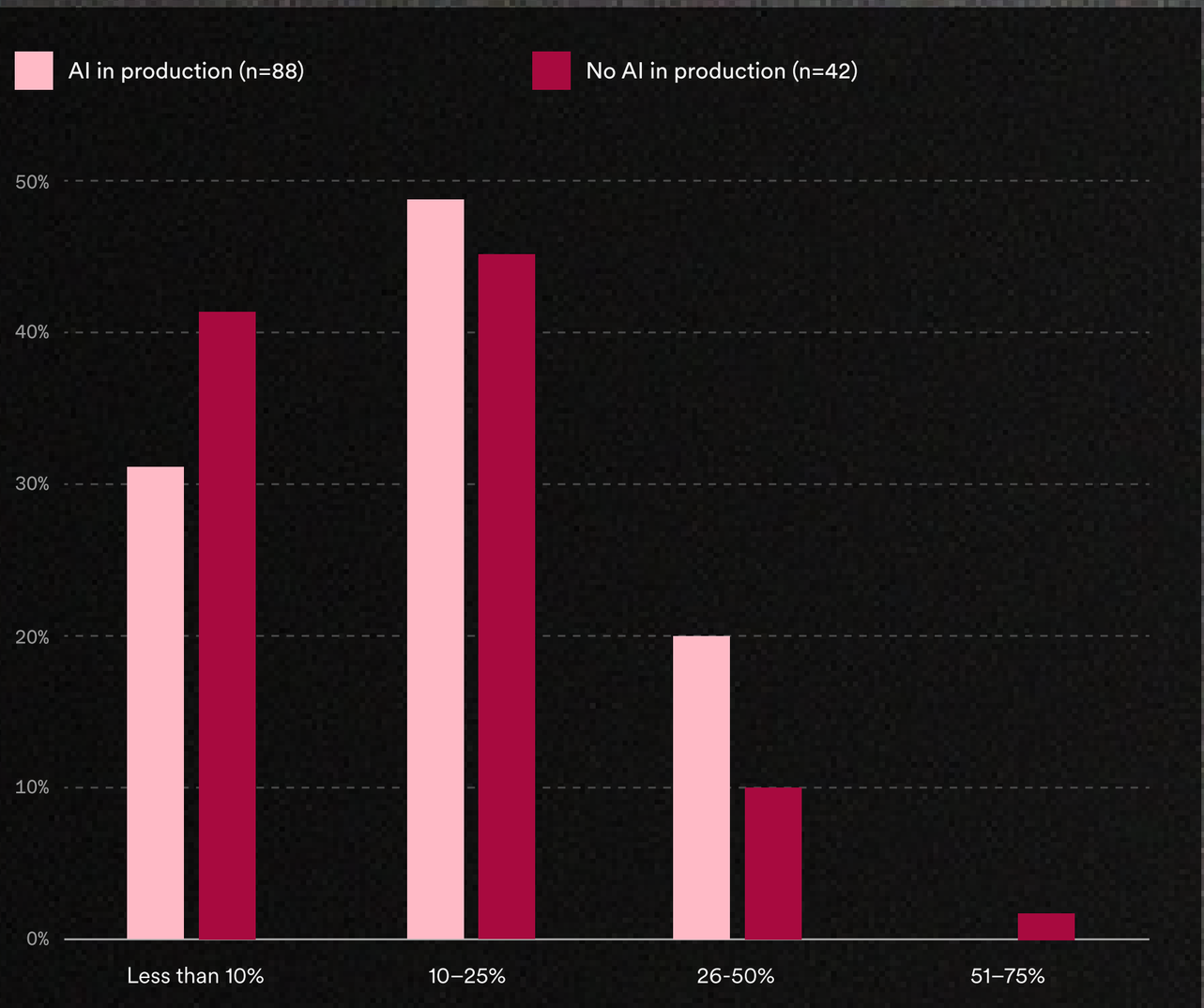
1. Roughly what share of your team's engineering time goes to reliability work?
2. Has that percentage gone up or down in the last 12 months? If up, what do you think is driving that increase?

Time spent on reliability: AI vs non-AI use cases

Teams building AI in production spend more time on reliability work. 20% of AI teams are in the 26–50% band — spending up to half their engineering capacity just keeping things running. That's twice the rate of non-AI teams (10%).

Nearly half of teams building AI (48%) are in the 10–25% band. That's a significant but not catastrophic spend—the kind of number that likely prevents the team from shipping as fast as it wants to.

Time spent on reliability work: Teams building AI vs traditional software



AI teams are twice as likely to be in the 26-50% band (20% vs. 10%). Non-AI teams are more likely to be lean, 43% spending less than 10% vs. 32% of AI teams.

Q8: Roughly what share of your team's engineering time goes to reliability work rather than shipping new features? % of each group in each band.

How has time spent on reliability changed over time?

While the survey showed that larger teams generally spend more of their time on reliability toil, the net change in time spent on reliability over the last year is relatively uniform across teams of all sizes. Change over time also appears to be relatively similar between AI and non-AI use cases. The more revealing split is by orchestration tool.

Teams building AI workflows with Inngest (–10pp) and custom-built orchestration tools (–9pp) correlate with declining reliability burden over the last year. Temporal (+22pp) and AWS/Vercel/CF (+9pp) users correlate with increasing burden. BullMQ/Cel/Sid sits in the middle (–5pp).

Has reliability burden gone up or down? AI teams by orchestration tool

Net = % up minus % down. AI teams only. Sorted best to worst.

Ingest (-10pp) and custom-built (-9pp) AI teams are trending toward lower burden. Temporal (+22pp) and AWS/Vercel/CF (+9pp) users are running the most complex workflows, complexity is compounding. BullMQ (-5pp) sits in the middle.

What causes reliability toil to increase?

For the 31% who reported their burden increasing, the top two drivers were equal in citation: higher traffic and scale (61%) and technical debt (61%). However, when splitting results by AI and non-AI use cases, we see higher traffic & scale take a slight lead for teams building AI at 62%, and technical debt take the lead for those not building AI in production at a significantly higher margin of 86%.

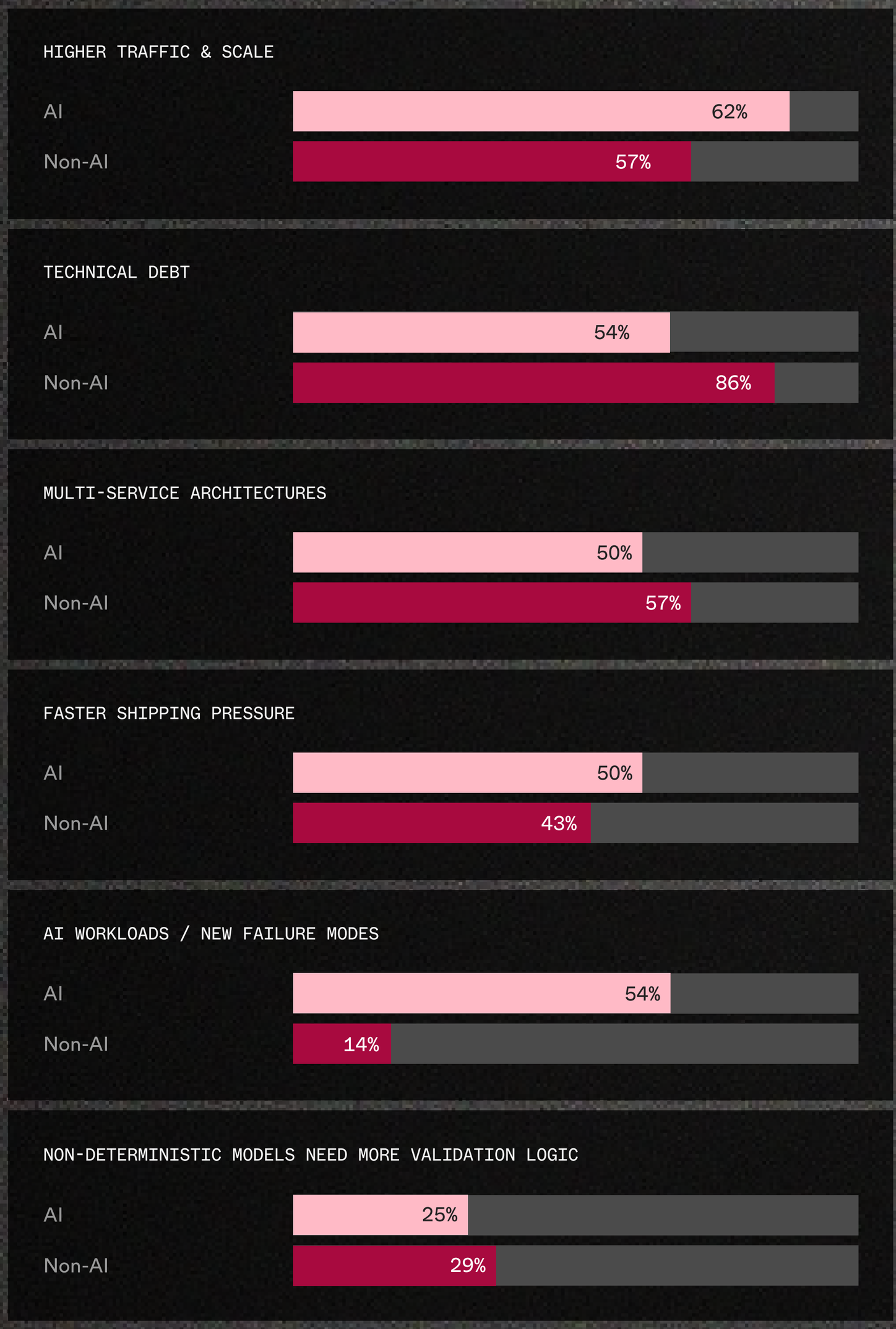
Multi-service architectures, faster shipping pressure, and AI workloads with new failure modes follow closely. Non-deterministic outputs requiring more validation logic is last.

KEY INSIGHT:
Orchestration tooling should help teams handle rapid or lumpy scale, as well as fit into their existing stack in order to reduce friction cited here.

> E-COMMERCE ENGINEERING LEADER
> 1,000+ ORG
>
> I think the biggest unsolved problem is making workflows truly reliable when things go wrong in production. Retries, duplicate events, partial failures, long-running jobs, and out-of-order execution can all break logic in subtle ways. It's still hard to build systems that are easy to debug, resume safely, and trust at scale.

What's driving reliability burden up?

Net = % confident (n=73) minus % unconfident (n=15). *** p<0.01, ** p<0.05, * p<0.10, ns not significant. Greyed rows shown for context only.



SECTION 4

THE OBSERVABILITY EDGE: HOW TOOLING CHOICES CORRELATE WITH RECOVERY TIME IN CUSTOMER-FACING INCIDENTS.

Engineers using observability dashboards built into their orchestration layer report the fastest path to understanding crashes when they occur:

40%
diagnose in minutes.

1%
note lengthy delays.

Teams using only APM tools,

29%
diagnose in minutes.

11%
note lengthy delays.

In this survey we asked one free-text question: "What is the biggest unsolved problem in building reliable workflows, agents, and endpoints?" One theme dominated the responses: observability.

While not exclusively an AI problem, observability was cited more frequently by teams building AI in production. Most observability tools were built on the premise that the same inputs produce the same outputs. AI systems break that assumption. When a workflow fails because an LLM returned a malformed response at step three of five, the error surfaces as a downstream consequence in a system that had no reason to expect it.

What we asked

1. How do you observe workflow runs?
2. When a workflow fails in production, how long does it typically take to understand what went wrong?
3. In the past 90 days, has a workflow or background job failure caused a customer-visible incident?
4. What are the most common causes of failures (pick up to 3).

Observability platforms and diagnostic speed

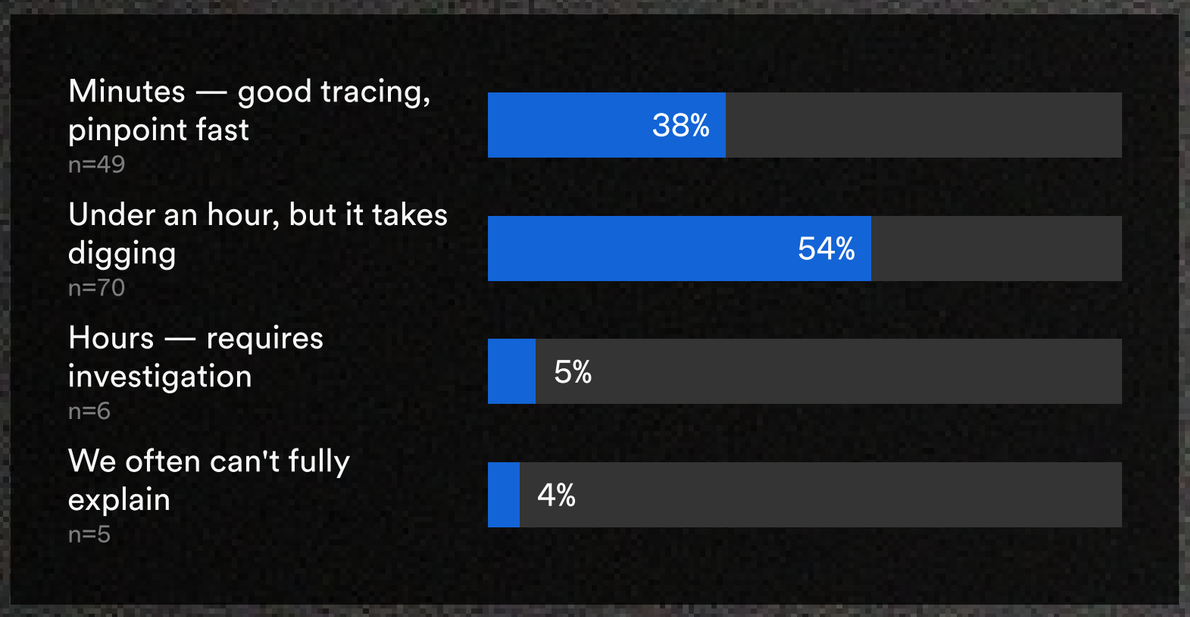
38% of teams can diagnose a production failure in minutes. Another 54% get there in under an hour but require active investigation. 9% take hours or can't fully explain what happened at all.

The gap between these cohorts is almost entirely explained by observability tooling. Platform-native dashboards — where observability is built into the orchestration layer — deliver the best outcomes: 40% diagnose in minutes, with only 1% in the slow-or-blind band.

Teams using only APM tools without an orchestration dashboard: 29% fast, 11% slow or blind. That's an 11x difference in worst-case outcomes.

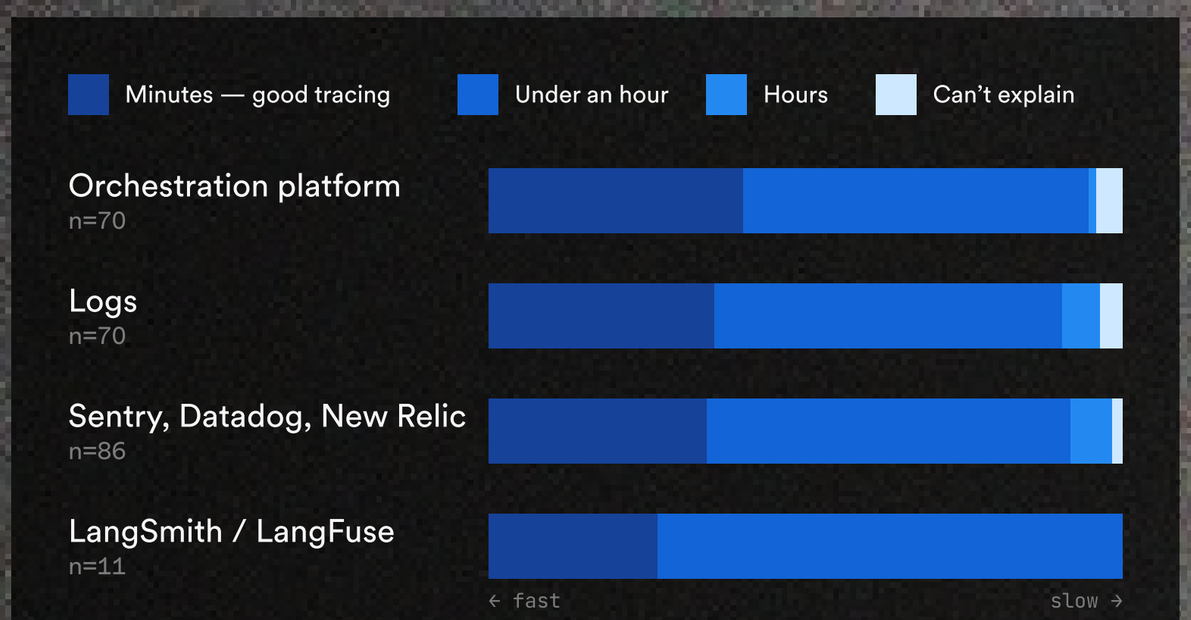
KEY INSIGHT:
Orchestration tools that provide granular observability are the best shot teams have to find what went wrong, quickly, when workflows fail.

Time to understand what went wrong



Q5: When a workflow fails in production, how long does it typically take to understand what went wrong? All respondents n=130.

Time to understand pipeline failures, by observability method



Q7: In the past 90 days, has a workflow or background job failure caused a customer-visible incident? AI vs. non-AI teams.

Orchestration platform dashboards lead on fast diagnosis (40%) and have the lowest combined slow/can't-explain rate (5%). Sentry, Datadog, and New Relic have the highest "hours" rate (7%) despite being the most widely used tools, broad adoption doesn't mean fast resolution. Teams with no structured observability (n=7, not shown) split sharply: 43% diagnose in minutes but 29% can't explain at all, either the failure is immediately obvious or you're completely in the dark.

Rate and cause of incidents in AI vs non-AI use cases

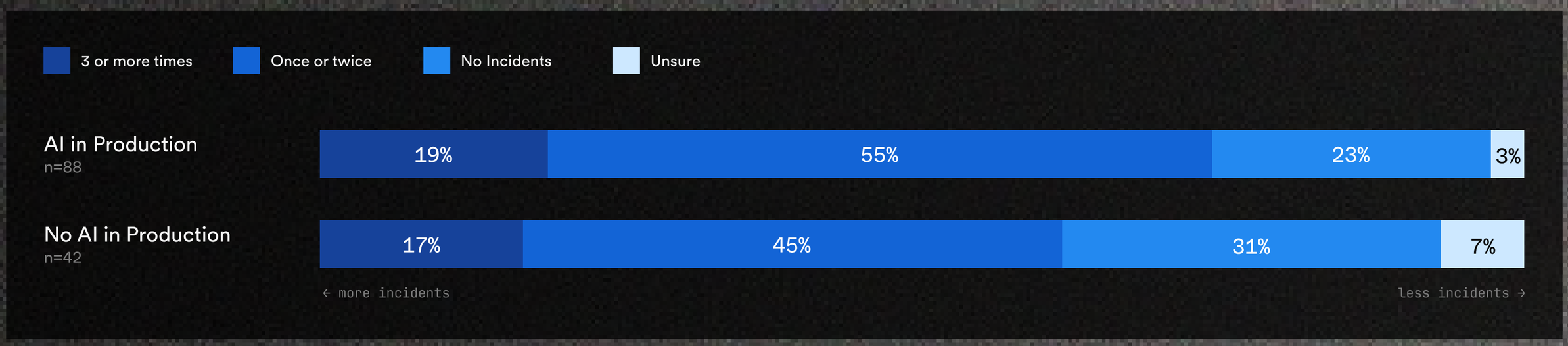
> **74% of AI teams building AI in production experienced at least one customer-visible incident in the past 90 days—vs 62% of non-AI teams.**

One in five AI teams had three or more, while teams larger than 500 engineers rarely accumulate 3+.

Infrastructure crashes are a top cause of workflow failure for both AI and non-AI use cases, though LLM / external API failures invert the top slot for those building with AI.

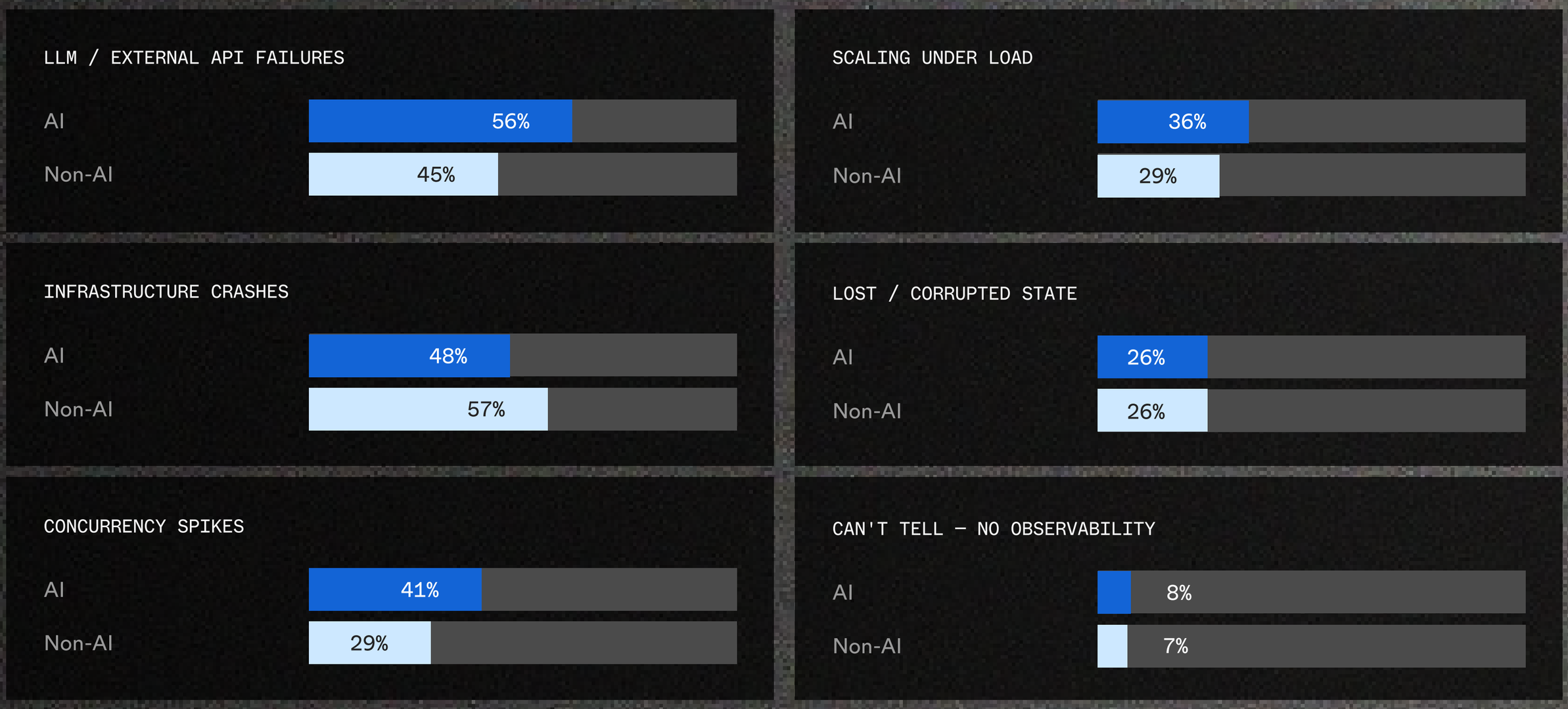
These trends mostly hold by team size, though the largest teams report a much higher occurrence of infra and scaling-related failures.

Customer-visible incidents in the past 90 days



Q7: In the past 90 days, has a workflow or background job failure caused a customer-visible incident? AI vs. non-AI teams.

What breaks — teams building AI vs traditional software



Q7: In the past 90 days, has a workflow or background job failure caused a customer-visible incident? AI vs. non-AI teams.

SECTION 5

AI FRAMEWORKS AND EVALS—HARNESSING UNPREDICTABLE MODELS

While evals are closely tied to confidence in scale, we're still early. 28% of respondents believe it's hard to write evals that matter,

35%

aren't using evals at all, and most who are, have built their own pipeline.

AI frameworks and evals exist in the layer of building AI where non-determinism becomes harder to manage through better infrastructure alone.

Evals aim to measure whether AI outputs are correct, safe, and within acceptable variance.

Agent frameworks are the libraries that sit between application code and the model — structuring calls, managing context, handling tool use.

Both categories barely existed two years ago and have grown fast. And both may share a common achilles heel: they were built as standalone layers, without a native connection to the orchestration layer that knows what actually ran.

What we asked

1. What are you using to evaluate AI outputs in production?
2. What do you believe are the biggest limitations, if any, of modern AI eval solutions?
3. What, if anything, are you using for an agent framework?
4. What do you believe are the biggest limitations, if any, of modern agent frameworks?

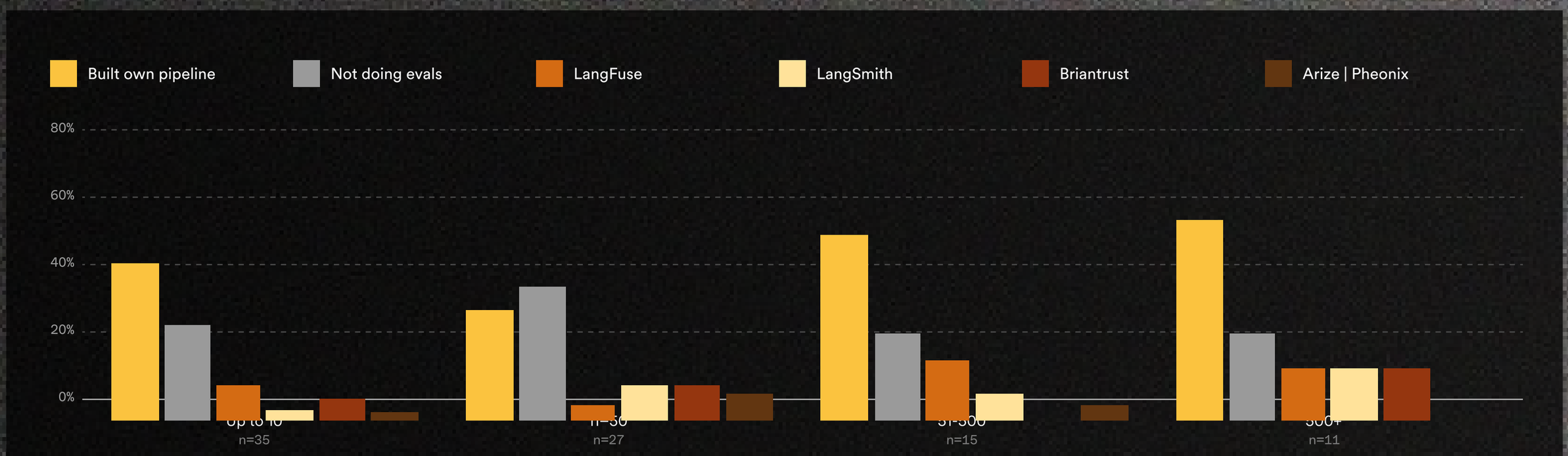
Using evals to improve AI quality in production

Maybe the most surprising finding in this survey is that 35% of respondents building AI in production are not doing AI evals at all. Of the 65% who are, most have built their own pipeline.

Tool selection spread seems to hold across all team sizes, though as with orchestration solutions, teams begin to default to building their own solution when they're small enough that complexity is low, or big enough that bespoke solutions seem like the only answer.

Growth phase teams (11–50 engineers) exhibit the highest rate of "not doing evals" at 44%. This may simply be because these teams are moving fast, shipping fast, and deferring infrastructure investments until something breaks.

Eval approach by team size



"Not doing evals" peaks at 11-50 (44%) — the growth phase where teams are moving fast and haven't invested in eval infrastructure. At 500+, 64% have built their own pipeline, often because commercial tools don't integrate with their bespoke orchestration stack.

Perceived opportunities for improvement in eval solutions

Regardless of whether teams invested in, built, or deferred eval solutions, they also answered what gaps they perceived to exist in the category.

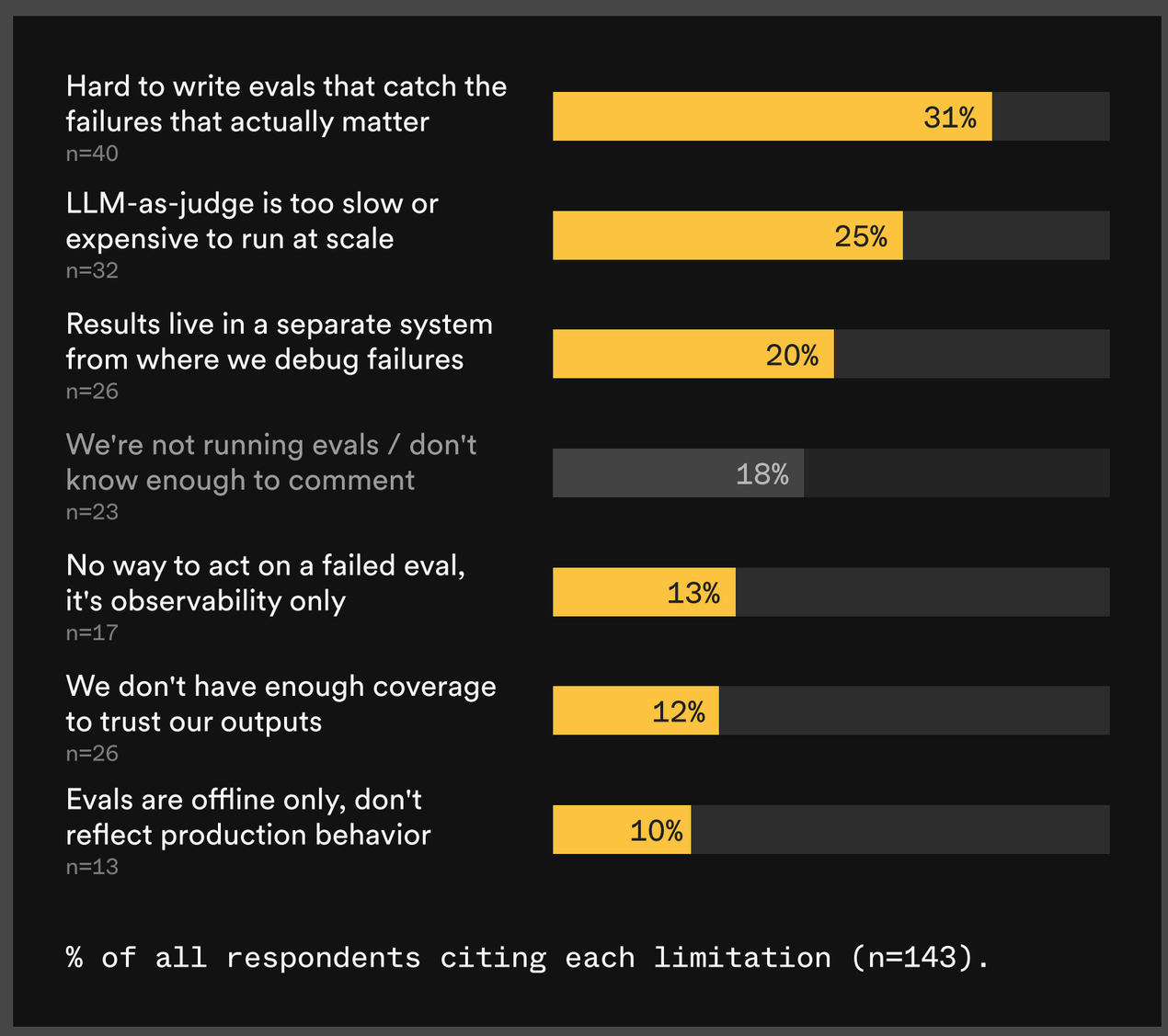
While the top two reasons cited pertain exclusively to the ergonomics of eval solutions, three are failures of context: results in a separate system (20%), no way to act on a failed eval (13%), we don’t have enough coverage to trust our output (12%), evals offline and disconnected from production behavior (10%). Nearly half of all AI teams (47%) cite at least one of these three. Each describes the same structural absence: eval state and orchestration state that can't see each other.

To add a little more color, we can compare gaps cited against tools in use, only to extrapolate potential opportunities for improvement in each. It’s important to note that respondents were not asked about gaps in their solutions—just in the category overall.

HEAD OF MACHINE LEARNING
200+ PERSON ORG

> Deeply integrated infrastructure, data sources, evaluations, and data migrations based on evaluation results — that's what's still missing.

Perceived gaps in eval solutions



% of all respondents citing each limitation (n=143).

Q15: What do you believe are the biggest limitations of modern AI eval solutions?

Perceived gaps correlated with eval solution in use

Q15: % of each tool's users citing each limitation. Tools with n<3 (W&B, Helicone) excluded.

Using agent frameworks to guide AI Agent production

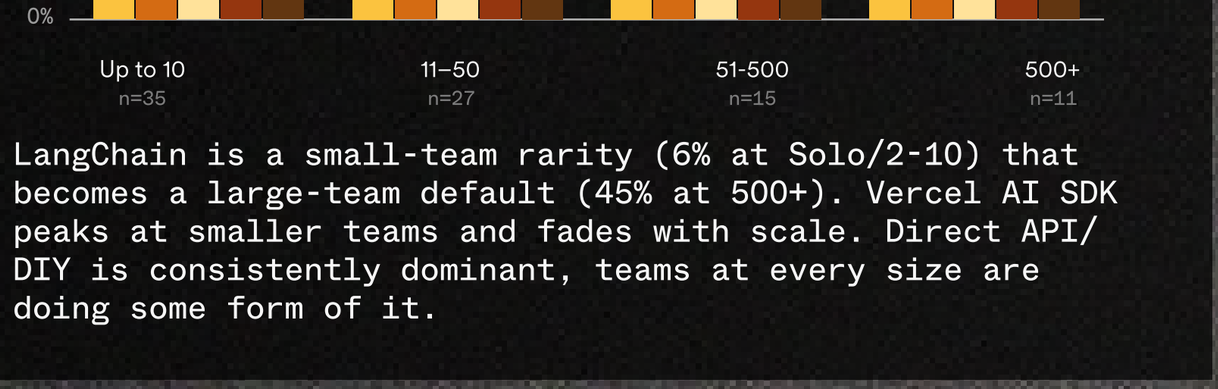
> **69% of respondents building AI in production not using third-party agent frameworks, instead choosing to call LLM APIs directly, or build their own abstractions. Observability-related fears top the list of perceived gaps, with 26% saying abstractions make failures harder to trace.**

Agent frameworks provide the foundational components to simplify the process of building, deploying, and managing autonomous AI agents, though the category is still quite new.

Given the popularity of frameworks like Langchain, it might be surprising that 69% of respondents building AI just call LLM APIs directly, or have built their own abstractions.

The remaining 31% are split between Vercel AI SDK (30%), LangChain/LangGraph (20%), and smaller tools. This trend hold for various size cohorts, though we do see adoption of Langchain/Langraph skewing towards larger teams.

Agent framework adoption by team size



LangChain is a small-team rarity (6% at Solo/2-10) that becomes a large-team default (45% at 500+). Vercel AI SDK peaks at smaller teams and fades with scale. Direct API/ DIY is consistently dominant, teams at every size are doing some form of it.

Q16: AI teams only. % of each size band. Multi-select — totals exceed 100%.

Perceived opportunities for improvement in framework solutions

Regardless of whether teams invested in, built, or deferred framework solutions, they answered what gaps they perceived to exist in the category, generally. The predominant gaps cited are that agent framework abstractions make failures harder to trace (26%), and there's poor support for long-running or stateful workflows (19%).

However, it should be noted that time to diagnose workflow failures is nearly identical between those using frameworks, and those calling LLMs directly without a framework.

Perceived gaps in agent frameworks

Q17: What do you believe are the biggest limitations of modern agent frameworks?

To add a little more color to these perceived gaps,we can compare against tools in use, only to extrapolate potential opportunities for improvement in each. It’s important to note that respondents were not asked about gaps in their solutions—just in the category overall.

Perceived gaps correlated with framework solution in place

50%+ 30–49% <30%

Framework	Harder to trace	Poor support for long-running workflows	App layer only	Too rigid	Lock-in	Doesn't compose well with job orchestration
Direct / DIY n=43	38%	26%	21%	25%	18%	20%
Vercel AI SDK n=11	58%	35%	38%	23%	35%	23%
LangChain n=18	50%	50%	28%	39%	28%	11%
LlamaIndex n=7	43%	14%	29%	71%	14%	29%
Mastra n=6	67%	17%	33%	17%	0%	33%

Q17: % of framework users citing each limitation. Tools with n<3 excluded.

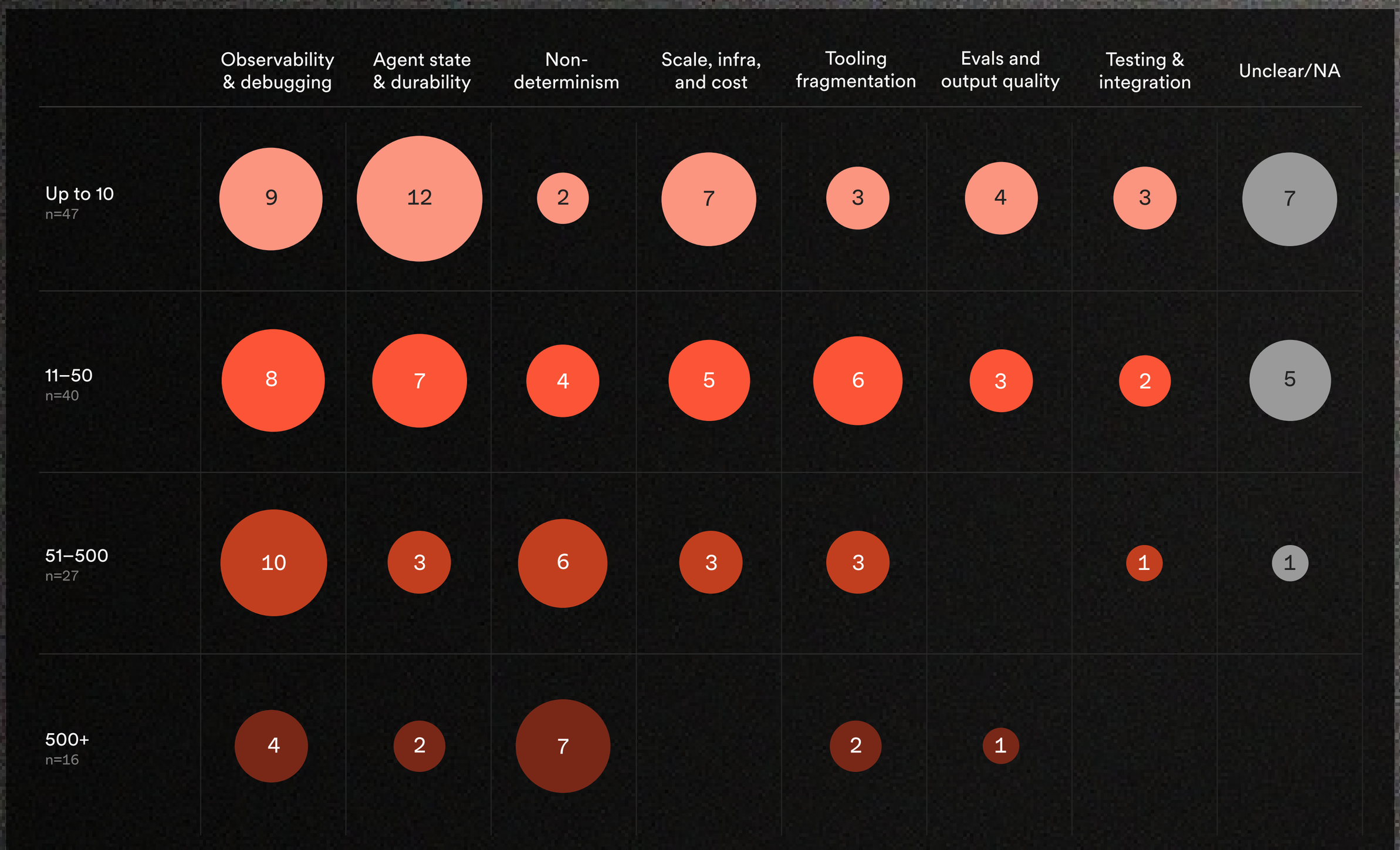
SECTION 6

WHAT ENGINEERS SAY IS STILL UNSOLVED

> Observability is the #1 mentioned unsolved problem (19% of responses)—the highest of any theme, and the only one that appears equally among AI (18%) and non-AI (21%) teams.

Our final question in the survey was open-ended: "What do you think is the biggest unsolved problem in building reliable workflows, agents, and endpoints?" The responses tell us a lot about universal problems, and problems of scale.

What's unsolved – by team size



19% of responses name observability as the core unsolved problem — the highest of any theme, and equal across AI (18%) and non-AI (21%) teams.

Q18: Themes in open-text responses by team size band. Bubble area = respondents mentioning each theme.

Observability: the named gap

19% of responses name observability as the core unsolved problem—the highest of any theme, and the only one that appears equally among AI (18%) and non-AI (21%) teams.

HEAD OF MACHINE LEARNING & DATA SCIENCE
11-50 ENG TEAM SIZE

> Observability is, I would say, still very unsolved. A lot of the workflows that we run, we are able to monitor, but it is not at the same level and requires a lot more instrumentation than we would like. Often the issues that are found are outside of the observability metrics that we have built, so they get missed.

HEAD OF MACHINE LEARNING & DATA SCIENCE
11-50 ENG TEAM SIZE

> The end to end observability in case of errors, digging takes take and recovering from failed steps requires deep architectural knowledge and failures embrace within an engineering and data department.

ENGINEER
11-50 ENG TEAM SIZE

> People are forgetting that the hardest part about software engineering is accounting for edge cases. This was always a problem, but how much more software is being built, thanks to AI, has made it orders of magnitude more difficult to solve. The surface area has exploded. People are very interested in building increasingly complex software, they are way less interested in building observability.

It’s also the primary concern of folks from companies large and small:

VP OF ENGINEERING
501-1000 ENG TEAM SIZE

> Our biggest challenge is observability. We seem to get more and more logic in our workflows getting pushed to LLMs, which not only makes them non-deterministic but just hard to evaluate for anything other than raw response time.

ENGINEER
2-10 ENG TEAM SIZE

> Observability gaps — When something breaks, it’s hard to trace why an agent made a decision or reproduce it. Debugging feels more like forensics than engineering.

Context: the unnamed gap

Survey responses point to observability as a felt gap, but the survey data reveals another related issue: the importance of shared context, in a layer that makes every other component in the stack actually effective. There are three signals that support this finding:

1

Platform-native observability outperforms bolted-on observability

Teams relying only on APM tools like Sentry or Datadog are more likely to end up slow or blind when a failure hits—11% take hours or can't explain what happened at all, vs. 1% for teams using insights native to their orchestration platform. The latter teams are also 38% more likely to pinpoint failures in minutes (40% vs. 29%).

2

Three of five major complaints of eval platforms are integration problems

The five most-cited eval limitations split into two types: two are intrinsic to evaluation (31% cite hard-to-write evals, 25% cite LLM-as-judge cost), while three are integration failures—results in a separate system (20%), no way to act on a failed eval (13%), evals offline and disconnected from production (10%). Each describes the same structural absence: eval state and orchestration state that can't see each other.

3

Framework composability is the missing link

69% of AI teams call LLM APIs directly or built their own abstractions rather than using a framework—and 20% of them still cite poor composability with job orchestration as a top limitation, comparable to the rate among LangChain and Vercel AI SDK users. Teams that rejected frameworks are already building orchestration primitives from scratch. The features aren't missing from the stack; the connection between AI logic and the orchestration layer is.

> DIRECTOR ENTERPRISE ARCHITECTURE AI CLOUD ARCHITECTURE AND APP INFRA MODERNIZATION
> 501-1000 ENG TEAM SIZE
>
> From my experience, the biggest unsolved problem is ensuring deterministic reliability in non-deterministic systems—especially when LLM-driven agents interact across multi-step workflows and external APIs. I often see failures in state management, error propagation, and tool orchestration, where edge cases break silently and are hard to trace or reproduce. Until observability, guardrails, and self-healing mechanisms mature, achieving production-grade reliability remains a core challenge.

Conclusion

The data in this report paints a pretty clear picture about missing context—about the lack of an AI stack where state is shared across layers—where a failed eval can see the execution that produced it, where an observability dashboard can trace inside the workflow, where an agent framework composes naturally with the infrastructure running beneath it.

The 19% of AI teams who are very confident in their ability to scale aren't using fundamentally different tools. They're using tools that are more tightly connected: purpose-built orchestration with native observability, eval approaches that integrate with where failures actually live, frameworks that compose with the job layer rather than sitting above it.

The ask that runs through open-text responses is simple: show me what ran, show me why it stopped, show me whether it's safe to resume, and what happens after it does. Most importantly, bring all of that information, into one place.