Zero-Shot Learning - Episode 4: Ankur Goyal from Braintrust

Stop guessing if your AI works

Hosts: Nancy Wang, Chief Technology Officer, 1Password Dev Tagare, Senior Director and Head of Engineering, Gemini Enterprise and Business, Google (personal capacity) Guest: Ankur Goyal, Founder and CEO, Braintrust

---

1. Introduction

Nancy Wang: Hi everyone. I'm Nancy Wang, Chief Technology Officer at 1Password. Welcome to Zero-Shot Learning, which is a podcast about the reality of developing with AI from the people actually doing the work. I host this show alongside Dev Tagare, who is a Senior Director and Head of Engineering for Gemini Enterprise and Business at Google. Today, we're going to be talking with Ankur Goyal, who is the founder and CEO of a company called Braintrust. Ankur has spent his career building data and ML systems - from distributed infrastructure at SingleStore to ML platform work at Figma. And now, of course, he's the founder of Braintrust.

Today's conversation gets at a problem almost every team runs into once they're actually building with AI. Once you have something running in production, how do you actually know whether it's good, whether it's getting better, and whether it's safe to ship? And that's the world Braintrust is being built for - not just prompts or dashboards in isolation, but a full loop. Ankur says the word "evals" a lot. What he's actually talking about is discipline - how teams create feedback loops around quality, catch regressions, and make faster decisions without flying blind. This is a great conversation that really shed light on an aspect of AI we don't normally discuss. So lock in. This is Zero-Shot Learning.

---

2. From Databases to Evals: The Origins of Braintrust

Dev Tagare: Today we're joined by Ankur Goyal, who is the founder and CEO of Braintrust. Digging a little bit into your origins as a builder - you've worked in distributed systems for a long time, obviously applied ML and AI products, and an ML platform. What's one lesson across all of those journeys or chapters that most directly shaped how you approach Braintrust? And really - why did you begin Braintrust?

Ankur Goyal: Yeah. Interestingly, the first thing I worked on was a relational database company called MemSQL. And one of the issues we had is we would try to test a bunch of SQL queries and find correctness issues and performance issues. Back then, performance meant speed, by the way. Nowadays when people talk about performance, they mean accuracy. We'd try to find issues, and then of course customers would use our product and they would find new issues that we didn't expect. And what we had to do is figure out how to reproduce an issue we were seeing - often without using the same data, because the data might be very sensitive - and then run it locally, fix it, and add it to a bank of queries that we could forever test on to make sure we don't regress. Does that sound familiar?

Dev Tagare: Absolutely - I was a big MemSQL user. That is actually like the first time I learned about evals. It's basically the same workflow as doing AI evals. You have a feedback loop from what people are experiencing in production. Of course, at that time - both because of how nascent the cloud was and also people's security requirements - we didn't automatically pull SQL queries. But still people ran into issues. We'd at least report them in the logs and then try to reproduce them and capture them in a bank of queries.

Ankur Goyal: I started a company called Impira, where we did document processing with AI - and this was in the Stone Ages, when you had to use computer vision. Pre-transformer. We learned a lot about that problem over time. One of the things that happened as we started to get customers is we'd have banking customers trying to extract bank statements with our product, and then accounting customers trying to extract data from invoices. And we might improve the performance - and when I say performance, I mean accuracy - on the invoices and then break the banking workflow. And obviously that doesn't fly.

And then Figma acquired us. This was right around when ChatGPT came out. And so we started working on using LLMs like GPT-3.5 and GPT-4 within the context of Figma - and guess what, we had exactly the same problem. Any time we tried to change the prompts or use a new kind of design file we hadn't seen before, everything would break. And so again, we had to build up a bank of examples and figure out how to automatically test them.

And I think there are always these two problems. One: how do you actually find interesting stuff from the ether and do it in a way that meets the security and compliance rules that apply to different use cases? Some of our customers are building consumer products with very different compliance rules; we also have banks as customers with a really high degree of sensitivity. Two: you have a bunch of data - how do you very efficiently and at scale make use of it to do as much testing and iteration as you can?

And I guess the fourth time I encountered this problem was when Elad - one of our big investors who started the company with me - we were catching up and he was like, "Why do you keep building this thing?" And we realized there actually isn't a really good solution out there for people. We talked to a bunch of companies, and our first customer was Zapier. Bryan from Zapier was really, really early into building AI products - they shipped their first AI product in January of 2023. One of his co-founders, Mike, started Arc Prize. As soon as the next team started building an AI feature at Zapier, Bryan was like, "Okay, I don't want to build an internal tool. Can we work with you guys and figure out how to capture this workflow so all the other teams at Zapier that want to build AI features can do this?" And that was like a light bulb moment for us. That's when Braintrust started.

Nancy Wang: Totally. And I mean, I've been a user - as you know, one of the early ones in a regulated space. So a lot of this resonates with me across the data plane: bringing our goldens, iterating really fast and continuously, and being able to share that experience across other users and systems. Like, you have a bunch of agents proliferating these days - what happens in one instantiation of an agent versus another? It's a hard problem to solve. People view agents differently, use them differently. You have a baseline you've got to tune to and tune over time. But you want to know what it's doing today - how good or bad it is. And quality has become an overloaded term these days, between latency and accuracy, and often it's neither. That's why we needed products like yours - as an excited new customer of Braintrust.

---

3. Designing for Developer Brains

Nancy Wang: There's this adage in product design where simplicity is a core product tenet. When you were first building Braintrust, what areas of the product did you figure out pretty quickly had to just be dead simple to use? I have to say - the simplicity for enterprise developers is actually what led me to Braintrust. I did try out other tools designed for AI observability, but just in how the input/output data pairs were presented in your console - that made a lot of sense to me from the enterprise world. Tell me about that.

Ankur Goyal: For sure. So when I was at Impira, I'm like a run-of-the-mill normal developer. I don't have a stats PhD. And when I was trying to first figure out ML stuff and do evals, I was using tools that I have a lot of respect for - I'm a big Hugging Face fan, and for a while I was one of the big Hugging Transformers contributors. But those tools are very difficult for my brain to understand. And I think it comes down to something quite simple: there are people in the world who understand NumPy, and there are people who don't. If you study stats or physics or do any kind of hardcore numerical work, your brain understands Matlab and NumPy. There are a lot of tools designed for people wired that way. But as a software engineer, you're thinking in terms of: what are the inputs and what are the outputs? I'm not even sure that term exists in traditional ML tooling. It's more tensor-based.

So a lot of the early Braintrust UI was me trying to boil down the painful experience of using tools built for NumPy brains into something more developer-friendly. Our early persona was squarely software engineers without an ML background. We showed the product to a bunch of software engineers, and they were like, "Yeah, this has the right amount of information for me." At one point we showed it to a machine learning person at a company we were trying to close as a customer, and they asked to add standard deviation to the product - so every cell

you could see the standard deviation of the score with respect to the other examples with the same input. We shipped that. And then I remember all of our developer users complained - "What? How do I interpret this?" So we actually un-shipped it, because even though it might theoretically be useful to someone who needs to dig that deep, it just wasn't valuable to our users.

And the other thing we noticed: when we showed the product to ML people, they'd say, "There's not enough buttons in the Braintrust UI - this is way less sophisticated than other stuff I've used." And if we showed it to product managers, they'd say, "There are too many buttons. This is too much." So we've realized there's this crossroads: we have a product that's pretty simple and approachable to developers, and we can either go in the direction of making it easier for product managers, designers, and subject matter experts - or in the direction of making it more approachable for data scientists and ML engineers. We've intentionally chosen the former. We see more and more product managers, subject matter experts, and people in areas like customer support getting to be part of the AI development process.

For the ML and data science audience, we just shipped a really nice SQL interface into Braintrust. Now you can run SQL queries to your heart's content against all the data in the product - standard deviation, percentile, you name it. You can do that analysis directly in the product, or hook it into a tool that has a lot more buttons.

Nancy Wang: That makes sense. And that goes back to what traditional software design calls "nerd knobs." For those who want it, there are always advanced options dropdowns. But the goal is to make the UI simple and easy to use. My firm belief is everyone's going to be an AI builder at some point. Right now you're seeing non-tech teams like finance and marketing build their own agents too. And look - agents need evals, because what's the good of an agent if you don't actually know and can't monitor what it's doing? Our marketing team is using Braintrust quite a bit now.

Ankur Goyal: Oh, wow. Nice. Well, your marketing team, having known some folks on their team, are also pretty data-driven - so that's good to hear.

---

4. Building a Production Feedback Loop

Dev Tagare: So maybe zooming out for a second - going to the core use case you're trying to solve - what do you think teams actually need in place to close the loop between "This is what happened in prod," to "Was that behavior acceptable?" to "What do we ship next?"

Ankur Goyal: That's a great question. I think at the core, every team should be thinking about building a feedback loop. Google Maps is one of my favorite examples - it predicts how long it'll take to get from point A to point B, but if it takes longer or shorter, Google Maps can implicitly use that as feedback to do a better job the next time. In some cases it gets a lot harder - if you're in a regulated industry and someone's interacting with your product and they have a bad experience, you may not even be able to log that in the first place, let alone take the input they typed into your chatbot and test with it. So it's not trivial. But at your core, you're trying to build a feedback loop.

And I think there are two core components to that. The first is the ability to capture interactions that happen in production - and that's all about instrumentation. How do you instrument your LLM calls? How do you instrument your agent? If your agent is calling across multiple services, how do you make a coherent trace? And then when your agent fails, you can go and understand why it failed and what happened.

The second thing is creating some kind of eval dataset. The eval dataset may literally approximate what the agent is experiencing in production. And really, your job as an engineering or product team is to build the ability to reconcile what happens in production with what you're testing in your evals. If you do that and take it seriously, you will systematically and inevitably make your AI product work. We've seen this with a bunch of customers. It can be a grind - figuring out how to prompt things well, which models to use, whatever it takes - but if you build this engine, you will get there.

Nancy Wang: Yeah, totally. And it's becoming more and more critical as we're living in an agent mesh, with agents proliferating everywhere. Companies like 1Password are building gateways and enforcing them with a lot of security and policy controls. In that world, having hop-by-hop visibility becomes super critical - you want to know what was invoked, how it was discovered, what the turnaround time was, what goal it achieved. In my mind, your product cuts across not just the instrumentation layers, but also goes into the actual data and the efficacy of it.

Ankur Goyal: Yeah. Actually, one of the things that is pretty early in the industry - and Nancy, I think we talked to your team briefly about this - is where we'll get as an industry in the next couple of years: taking the signals you learn from evals and using them at runtime to be more secure and safe. Being able to do that will help people use AI in more data-sensitive or regulated contexts. Right now, not being able to do that efficiently and with confidence around enforced guardrails is a huge bottleneck to adoption in the traditional enterprise.

---

5. Failure Patterns in Observability

Dev Tagare: One more related question, Ankur. Out of all the failure patterns you see with telemetry - or introducing telemetry into the agent tech mesh today - what would you say are the top few that you always see people go wrong with? And then when you introduce eval products or telemetry products, what do people start learning and unlearning?

Ankur Goyal: Great question. I'll give you two problems - one before you use a product like Braintrust, and one after you start using it.

Before Braintrust: you don't actually log the inputs and outputs. You log system information. Traditional tracing tools - we use them to figure out when our database performance is bad, for example. It's not super important to see the exact parameters to a SQL query or the exact inputs to a function call; what matters is understanding how long it takes and how it interleaves with other executions. But if you're debugging an AI problem, this is not the most important thing. It's relatively unlikely that you screwed up parallelizing two JavaScript functions, and that's the insight that fixes your agent. It's a lot more likely that a user came with a request, maybe attached some unexpected data, and you need to understand what they did that caused the system to fail - and reproduce data inputs that reflect that. That is the critical thing. So if you try to use traditional tracing, you're encouraged to trace the wrong thing.

And by the way, even if you trace the right thing, traditional tracing tools aren't designed to handle the data. One of the best - the most popular - tracing tools out there has a 100 megabyte limit on traces. In Braintrust, the average trace is over 200 MB now. If you're designed to track function name, start time, and end time, why would that ever be over 100 MB? Whereas if you're capturing prompts and each one is like 150 KB, of course you'll get there pretty quickly.

After Braintrust: the biggest thing we see people do is very literally trace what's happening in their application. There's nothing wrong with that to start. But what you should actually do is adopt the mindset that the tracing you do should make the downstream consumers of your traces more efficient. It's less about reflecting the state of your application and more about: I could trace exactly what the user input was and what the output was, plus a little bit of metadata that lets you recreate what the user was seeing - maybe inside of something like 1Password. Putting thought into actually organizing the data structure of your traces so that the person consuming it, or the eval you want to run, is already working with the right format - that is very, very important. Because it means that once you find something wrong with a trace, it's one click to save it in a dataset. Or you can trivially re-render what it looked like for a user to be in that state in your application.

We've seen this be really impactful. People will embed their UIs inside of our trace viewer - so you can actually see what the user would have seen in your app. Which ends up being really cool.

Nancy Wang: That is super powerful. I think the end feedback loop, especially with a user interface - I thought that was like one of the missing ingredients. I need to check that functionality.

---

6. Evals as the New PRD

Dev Tagare: Well, look - one of the things that always exists is this tension between launching features faster vs. making sure we're doing the right things - not introducing friction that could slow things down. So what are your best practices when teams are thinking about AI safety but don't know quite how to achieve that without slowing down their processes?

Ankur Goyal: I think the best way to solve the safety, quality, or taste problem - whatever your bar is - is to start thinking about evals as the new version of a PRD. In the previous world of building software, let's say you were an executive or product manager who really cared about a certain aspect of the product. You'd write that out in the PRD - that's how you translate business context to something approachable for the engineering team. In AI, it's very hard to do that. But instead of writing a PRD that you feed to the engineers, you can write an LLM-as-a-judge with really, really clear criteria about what matters to you in the business. As a product manager, that puts the burden on you to be very articulate about the actual criteria you're looking for. And if your agent behaves in a way that is unsafe with respect to your use case - or doesn't meet a quality or taste bar - it puts the onus on you to articulate that in a way that an LLM can identify. LLMs are obviously very smart now, so there tends not to be much randomness with LLM-as-a-judge if you can crystal clearly articulate your criteria. Once you do that, you have a quantifiable way of actually enforcing whatever guidelines make sense for you.

Evals you can apply offline - you can say "unless this score is met, we're not going to ship it yet." Or you can run them online and say, "Hey, we shipped, and this score went down by X percent, so maybe we should roll back or quickly fix it." But I think adopting the mindset of "I must enforce this quantitatively" is really important.

Dev Tagare: So this is about the on-ramp, right - for enterprises that want to do this but are afraid of slowing down. How fast should they actually move?

Ankur Goyal: If you use evals well, you ship way faster. And the reason is - let me tell you a story. At Impira, we were using BERT models, and then Microsoft released a new model called LayoutLM - it was like BERT but trained with 2D coordinates, very good for document processing. We spent six months debating whether we should move from BERT to LayoutLM. If you tried it in our product, in some cases it would be better, in some cases worse. Some of our unit tests were kind of flaky - very hard to make a decision. Then we built like the first version of what eventually became Braintrust, and we shipped LayoutLM two weeks later. Part of it was understanding where the trade-offs are. LayoutLM was clearly better at a bunch of document types we couldn't support before, and it very quickly illuminated where the problems were, and we were able to pound through them fast. Evals allow you to go way, way faster.

Nancy Wang: That makes sense - you're able to diagnose where the issues are, isolated down to the core component.

Ankur Goyal: Yes. You spend a lot less time debating philosophically about what might or might not work for a model. You literally look at the results and then try stuff and experiment.

---

7. Standardized Proxies, Gateways, and the Lingua Library

Dev Tagare: What do you think about the new way of doing things with agents - "I need a gateway, I need a proxy, I need to instrument something there"? Do you think there should be a standardized proxy across all agents - think along the lines of how MCP was really a protocol, but someone rolls out a standard on which a proxy is built, also called L7 proxies, but more tuned for the AI era? And how do you think eval products would fit into that from a policy enforcement lens?

Ankur Goyal: One of the things we're most in the business of at Braintrust is making it trivial for you to test performance across different models. We have a feature where if you run an eval or you're looking at a production trace, you can pick from a dropdown menu a different model to try it out on.

A year ago that was really easy because everything was using the chat completions format as a universal standard. There's an industry of internally built and commercial proxies that say "let's map everything to chat completions and then do all this stuff like enforcement and observability." We integrate with all of those products. But in the last year, that has completely turned on its head - every major vendor, OpenAI, Google, and Anthropic, have released significantly divergent model specs. And not only are the formats different, but even the features and the way you think about writing messages is quite different.

We're in a really weird state where when you're building an agent, you are rarely, if ever, thinking about the quirks of a specific LLM. But when you're writing the code to actually implement your agent with respect to Anthropic or OpenAI, you have to think about that. So we talked to a bunch of the model labs and other companies, and realized that everyone is sort of tangled up in their incentives. So we built a new library called Lingua - a translation-focused library that does literally nothing other than translation, and it is automatically verifiable. What it does is download the open API specs and protobufs from all the different LLM providers and automatically generate types. We wrote - with the help of LLMs - code that translates across the different model providers and has model-specific fallbacks. And because we're using the open API specs and auto-generating the types, we can also generate the translation tests. We can basically verify that the translation works without loss of any model-specific features.

Anytime you trace something now, no matter what format, we're able to detect it, parse it into a universal format, and use that to power cool Braintrust features like translating across models. It also powers the next version of our gateway. Our goal with the gateway is not to be the person who gives you free credits to OpenAI and Anthropic - there are actually some really great products that do that and that we partner with. Our goal is for enterprises to have an open source translation framework they can extend and customize - maybe even to their own models - and use that as a secure way of supporting the set of models they want inside their enterprise. Observability and guardrails are natively integrated with Braintrust, and we're very excited about what we'll be able to provide at runtime.

Nancy Wang: 100%, right. Because this is actually where I was having conversations with, for example, other network gateway providers - how do you actually know what kind of prompts are going into models? I think for many enterprise developers out there, they have a lot more inclination to move faster on changing prompts than they do on changing their code. So this plays really well into how they can observe - where are all the prompts going into the models, and what are the outputs.

So on that note - how do you think about the minimum release process for a prompt change?

Ankur Goyal: It's highly dependent on the use case. We see everything from "do nothing, observe in production what happens, maybe A/B test" as a way of gently rolling it out - all the way to "this group of seven selected subject matter experts must manually review every prompt change on these 50 golden examples before it goes out." Everything in between.

What I'd recommend is not subscribing to either extreme. What you want is to get as much information as possible, as time- and cost-efficiently as possible, while developing the prompt change - so that the probability of shipping something bad is lower. You need both subject matter expertise and tooling and enforcement in the dev process.

What works really well is: when you start early on, have humans - the people working on the product or subject matter experts - review every example and then translate that into scoring functions. Braintrust can do that automatically now: you can label stuff and click a button and it will generate a scoring function. Once you do that, you can scale the number of test cases. Let's say as a human you only want to look at ten things at once - maybe then you start running it on 100 or 1000, run the scoring functions, and pick the five or ten that were very high scoring and five or ten that were very low scoring. You look at those and audit them. You're going to find cases where the scoring function isn't very good, and you use that to improve it.

If you do that, you start to build a signal you can use anywhere. You can just type the idea, hit enter, and immediately get feedback on what the impact is. And as someone who does this all the time, I am very, very frequently - almost to a scary amount - surprised by what the ramifications of random prompt changes are. It's very useful to have a really quick signal. It also

means that as you build trust in the signal as an organization, you can still involve people in the review process, but you don't feel as nervous about always having people as the gate to release things. You can move into this world where you're using a relatively limited amount of precious human attention to look at stuff, but actually be testing effectively over tons of prompt iterations and tons of examples.

---

8. Model Families, Agent Layers, and Disposable Code

Dev Tagare: Related question - you alluded to writing your tests in one harness and then translating it to n number of models with a gateway in the middle. I think that's super powerful. Early on when we were chatting and bringing in Braintrust - fun fact, Ankur's head of sales called me in very early cycles for a recommendation on whether this product was going to work. It was maybe V0.5. I was like, "This is a powerful space, it's going to be fine." When you think about this flexible approach to building and having an eval framework that works across the spectrum - what are the trends you're seeing in terms of people sticking to a model family or a style of building agents? And how are you seeing the distribution of evals across the model layer, the agentic layer, and what I'd call copilot-style layers - managed agents that do one thing really well?

Ankur Goyal: I think early on I used to ask myself: are LLMs more like CPUs, or are they more like relational databases? Two CPUs that speak the same dialect of ARM are very directly comparable - literally the same instructions will produce the same output every time. Databases all speak ANSI SQL, but there are a lot of quirks: some have indexes, some don't; some queries will return different results; the type systems are different. Translating SQL from one database to another requires a lot of care. And I think it's turned out, at least so far, that LLMs are more like databases than CPUs.

So the choice of LLM you compile to - it can't be a last-minute thing you switch to just optimize for some performance tradeoff. It has to actually be core to the design of the system itself. You have to be intentional about which model you pick. And once you pick a model or model family, you have to put in intentional, sustained effort to optimize performance with respect to its quirks. Now, unlike relational databases, there's no state in an LLM - so if you realize a different LLM might perform better, you don't need to do a data migration. It's a lot easier to switch.

The predominant pattern at the LLM layer is that the best teams make a decision about which LLM makes sense for their use case, and then on a semi-regular basis - every four to eight weeks - they'll re-audit the best LLMs and make an intentional choice. Like, "Opus 4.6 came out yesterday at the time of recording, and it's so much better than what we had before - it makes sense for us to put in the engineering effort to switch now." Or, "It's actually only incrementally better than what we had, so even though it's performing better on the benchmark, let's just wait a bit and see."

Dev Tagare: I quite personally concur with that. And you know, my views from a year and a half ago to now - I feel like the model and agent coupling is becoming tighter, almost to the way agents are functioning like an application. A couple of years ago I was like, "It's going to be everyone's models, keep switching, keep evaling." Now I think: pick the model that suits you. And that also mirrors what happens during any internal AI rollout - we thought our developers would all conglomerate around one specific AI coding tool, but actually now we're rolling out two or three. And as we think about other parts of the SDLC, we're actually introducing different dev agents for different parts - so we can do almost like inter-agent evals or comparison.

Ankur Goyal: Oh yeah. I think the agent layer - or the harness layer - is very interesting right now. There's a lot of engineering leverage you can provide by making the model sing the right way, bringing the right context to it, taking advantage of each model's unique features - like web search, for example. So it's really exciting. And I think the main thing I've seen go wrong at that layer is: every time a major new model comes out, you have to be prepared to throw away all the code you wrote. If you look at all the major frameworks - even ADK is really popular right now - it's changing constantly. You have to be really careful about over-engineering these systems.

And I think the very best teams that we work with explicitly adopt the mindset that the code around the agent is disposable. The team's ego doesn't get too wrapped up in any of these decisions. And the best teams actually enjoy unwinding the stuff and changing it - because they've already embraced the fact that it's going to change.

Dev Tagare: Totally. My thesis there is your agents are going to eventually end up becoming a thinner and thinner layer - just like in a client-server architecture - because the models will get more and more powerful and self-learning over time. So the agent becomes like an orchestration and routing mechanism. What you control is really what you get out of the whole experience between the agent and the models - which is why evals become the cornerstone. I sometimes articulate it as a last-mile problem. Give the tools to the user to solve for it - and that's where a product like Braintrust comes in - that last 3 to 5 percent. Braintrust will solve for you because you will have control over your destiny. Let everything else move to the agents and the models.

Ankur Goyal: For sure. And I would submit a plea for any engineers watching this - something I talk to people a lot about. I meet a lot of founders right now starting coding agent products. I hate to break it to you, but I think - except for maybe a couple of exceptions - that ship has sailed. The user experience is such a ripe opportunity to reimagine these workflows yet again. We reimagined it a few years ago when LLMs went from not being able to code to being able to code. Now there's another opportunity to reimagine it. And I think that's where a lot of the value is going to be created.

Dev Tagare: 100% agreed. And a lot of agentic workflows now are basically just code gen inline. Add some guardrails, keep evaling it. And that gives you the flexibility between determinism and the YOLO-style agents - somewhere in the middle, you have enough control to guide them in a specific way.

Ankur Goyal: Yeah. I mean, spend like an hour implementing that. And then there are all these other really valuable problems to work on. I want to see a lot more cool new UIs come out over the next year.

Dev Tagare: Make sure the builders hearing this episode pay attention to that.

---

9. Crystal Ball: The Future of Evals and Observability

Nancy Wang: Well, that's actually a great segue to my favorite part of any episode - let's try to predict the future. The crystal ball moment. I guess I used to say 3 to 5 years, but given how things are moving, let's say four months. What pieces of the current agent eval or generic observability stack are going to be not as relevant? And what's going to take their place?

Ankur Goyal: So I think that even for Braintrust - and we have a really nice UI - we are completely rethinking our UI right now. Previously, if you used Braintrust, a human would look at each experiment, and if there's an issue with a trace, a human would look at each trace. Our task was to give you tools so when you run an experiment, you look at the least number of examples - or if you're running in production, you only look at the relevant traces. But now LLMs can do that layer of triage for you.

So we are rearchitecting our product - both at the infra layer and the UI layer - to assume that most of what people were doing in our product over the last year is going to be done by agents. And then there's this new problem of looking at all the intermediate work and making sense of it.

A good example: we are shipping a feature that lets an agent look at - not some of your traces, or a sample - but cost-efficiently look at 100% of your traces and try to derive insights automatically. That takes many steps of human workflow and pounds it into using different cheap and expensive LLMs to execute for you. And then once you do it, the human's job changes quite a bit. You go from looking at a bunch of data to try to generate trends in your brain, to taking the trends as input and validating them - understanding which is worth prioritizing, and figuring out how to make the corresponding changes in your actual product.

So to boil that down to a prediction: in three to five months from now, most traces are not going to be looked at by humans. The average human who interacts with a product like ours is going to be looking at some kind of analysis that's already been done for them.

Dev Tagare: Sort of like a human-in-the-loop, but at the end of the cycle - where iterations have happened and you get a "do you like this? Non-graded" and re-trigger the whole thing.

Ankur Goyal: Exactly. Yeah. And I think - I'm super curious what your predictions are, by the way. But I'll blab for a little bit longer. Andrej Karpathy was tweeting a lot over the holidays about the big inflection moment of coding models in December. And one of the things I resonated with from that is: the way people are doing AI coding today builds a lot of comprehension debt. Just like technical debt - where you can build, build, build, and then you have lingering problems you need to solve. If you let an LLM go, go, go, and no one takes the time to actually comprehend the work, you build up comprehension debt. And when the piper comes and your product breaks, someone needs to actually understand what's going on to have accountability for the product outcome. I think the way we're doing it with code gen is terrible right now. There are a lot of really crappy products, and a lot of good products that are becoming crappy because of comprehension debt. I'm maybe optimistic that some smart people - at existing companies or new companies - are going to come up with really creative solutions to this problem. I think it's probably the most important problem for our industry this year.

Dev Tagare: Well, I completely agree. Nancy, do you want to take a stab at your prediction for the next six months?

Nancy Wang: Man. Well, we've been thinking a lot about agent identities here at 1Password. Ultimately, traces and observability matter because you can trace them back to who the actor is and what they did. In fact, one of our VPs of AI this weekend put together a demo of SREs or DevOps using, I think, 500 agents - essentially a swarm of agents. Happy to send you both the demo. And what we need to know here at 1Password is: how do you effectively assign an agent an identity? Today we have constructs for workload identity - things like SPIFFE. But those are long-lived - you assign identity at issue time. But agents change so much. What they do at any given point in time, we can't predict - hence non-determinism. So I would say the agent identity war or battle will have been won. Curious - won by who?

Dev Tagare: My sense is that the agent market will shift away from being an automation-focused market - somewhere between "kind of works" and "I'm trying to get it to work" - to being very much focused on outcomes. I think ROI measurement and deflection rates are things people will start tracking and allocating dollars and capital to increasingly. And you'd measure any of these model companies on those outcomes specifically. So my thesis is twofold. One: we become a deflection-based AI economy. Two: ecosystem effects will start playing in. So you're not just doing a model - you're doing the gateway with it, you're doing the identity associated with the agents built on whatever framework you're using, and so on. That's my thesis or hope for 2026.

Ankur Goyal: That sounds like a good collab for the three of us.

Nancy Wang: I know, right? You represent different layers of the stack. Thanks so much, Ankur, for coming into our virtual studio. I feel like between the three of us, we could share anecdotes, use cases, and all the fun hot takes for another hour or two.

Ankur Goyal: Well, I'm glad we're also going to be able to share all the exciting releases with our listeners here. This is a great conversation. Thanks again.

Nancy Wang: Thanks for having us.

Zero-Shot Learning is presented by 1Password.