The Agent Development Lifecycle: Build, Test, Deploy, Monitor | Interrupt 26

45 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=JWY39WAVBJY
https://www.youtube.com/watch?v=jWy39wavbjY

SUMMARY

The presentation discusses advancements in agent development and production, emphasizing the importance of iteration and context management in building reliable AI agents. Key announcements include the launch of Deep Agents 0.6, Langsmith Context Hub, and SmithDB, all aimed at enhancing the agent development lifecycle and improving observability and efficiency.

- *Agents require a unique development lifecycle distinct from traditional software, focusing on iteration and context management.*
- *Deep Agents 0.6 introduces features like a virtual file system and CodeInterpreter for enhanced execution environments.*
- *Langsmith Context Hub allows for better management of agent context, including skills and agent.md files.*
- *SmithDB is a new database designed specifically for agent observability, improving query performance and handling large trace volumes.*
- *Langsmith Engine is a proactive agent that analyzes traces to suggest improvements and automate issue resolution.*
- *The integration of open-source models is emphasized as a response to rising costs and the need for better execution environments.*
- *The presentation highlights the importance of user interfaces and their integration with agent systems for improved user experience.*

[Music] Intelligence. They call it intelligence. But if it can't act, what does it do? It waits for us. So we push it. So it doesn't need us. We give it skills, tools, a harness with memory. Memory gives it context. Context becomes a plan. So it acts. But action isn't enough. It must learn.

So we observe it, measure it, personalize it and improve it. Improve it again. Tests evaluate, deploys, fix again, faster, more pressure. Tests evaluate, deploy, fix. Perfect. It's working. [Music] It's not just intelligence. It's a lie.

Please welcome to the stage Harrison, co-founder and CEO of LangChain. All right, let's go. I did not know that video was going to be made, so yeah, it was great though. It was great. We're really excited to be here. This is our second year doing it, and so we decided to make it two days instead of one. We've got a stacked lineup for everyone here. I think it's going to be a blast. We've got a lot to share.

And so we're thrilled and honored that you guys decided to join us for these two days. Last year, we kicked off

Interrupt by talking about how building agents was hard. It was easy to build a prototype, but it was hard to get to production. There was a lot of challenges that you encountered along the way. And over the past year, we've seen that a number of companies, including a lot that are speaking here today, have figured out how to overcome those hurdles and bring agents into production.

It's still not easy to build agents and bring them into production, but we've figured out some of the things that are needed to make that a reality. Building agents is different than building software. That's why it's a new challenge. That's why you need new tools and new skills. And I think it's important to think about why it's different than building software. There's two things that we think make it different. One is the inputs themselves. So agents take in natural language.

Natural language can be infinite in its dimensions. It can be any number of words, or now things can start to be images, or videos, or even audio files. And so the input space to agents is really, really large. The output space is also really large. LLMs themselves are non-deterministic, and even if they were deterministic, they're still really sensitive to small changes in the prompt. And so when you have this really big input space and this really non-robust system, you end up with an overall agentic system that's really hard to actually predict how it's going to do before you actually launch it and you actually put it in front of users and you actually have them experience it.

And the teams that have figured out how to build agents reliably, they ship early and then they iterate quickly. And this is a pattern that we've seen again and again from companies that have managed to bring agents into production. As they do this iteration, through some manner, they've landed on this new agent development lifecycle, parallel to the software development lifecycle that people use to build and ship software, but it's different because agents are different. They require more iteration and the steps at each of those phases are slightly different.

And so the mastery of this lifecycle, whether people call it that or not, is really what people are doing when they're building organizations and building tools to help ship agents. And everything that we do at Linkchain can be described by fitting into this lifecycle. We wanna figure out what the agents of the future look like, and we wanna build all the tools to help make them a reality. And so we think a lot about what this lifecycle looks like and how we can help that.

And so today, I want to talk about how we view this lifecycle. And I want to talk about a number of things that we are launching to make it easier for teams to iterate quickly through this lifecycle. So first, I want to talk about build, building agents. This is how we got started. So this is what LangChain was. This was a package launched a month before chat GPT. And it was really around building these LLM and agentic applications.

Fun fact, we had a class called Agent in Langchain before a chat GPT launch. I remember it. I was on, I think, I was in San Diego for Thanksgiving and was just trying to decide what to call it. And I ended up calling it agent-executor instead of agent. So, you know, we're not great at naming and that's maybe the first example of that. But we had this concept of an agent and it was in Langchain. Then we launched Langgraph a little over a

year later.

As we saw that the applications that people were building, they may have started off as chains, but they were becoming more complex graphs. And that was the idea behind LangGraph. We then launched LangChain 1.0, and then about nine months ago, we launched deep agents. Deep agents is an evolution of this as well. And so I wanna talk a little bit more about deep agents right now, because that's where we think the future is really headed. So the core idea of an agent, from the original agent executor days in LangChain to now, is basically, it's pretty simple.

It's an LLM running in a loop calling tools. There's a bunch of stuff that can happen around that loop. There's a bunch of specific tools you can add in. But when people talk about agents, this is essentially what they're talking about. A user request comes in, you call the LLM, the LLM decides to call a tool, you then execute that tool, and you keep on doing that until the LLM decides that it's finished. That's the core idea of an agent, and that's always been the case.

So what is deep agents? Deep agents is an agent harness, And it basically adds more batteries, included things. It supercharges this loop. What are some of the things that it adds? It adds an execution environment. So agent harnesses need a place to run. A really common way to run them is in a sandbox. You give it access to a file system, you give it access to code, it can read files, write files, execute code. That's its execution environment.

And so when people talk about agent harnesses, they often talk about coding agents part of that coding agent is its execution environment. Sam boxes are one extreme. You can write and execute code. And then on the other extreme, we have what we call a virtual file system in deep agents. This is basically a database that we expose to the agent as a file system. LLMs are great at reading and writing files. And so you can kind of trick it into having this kind of mock execution environment.

But the point is you give it an environment to interact with. Deep Agents and other agent harnesses come with a lot of built-in context management. So skills and memory are examples of this. Summarization is an example of short-term memory when the conversation gets too long and you summarize it. Context offloading, how do you deal with really long tool calls? This is logic that's built in to the agent harness. Prompt caching is another example of this where you want to make sure that you're cashing the initial part of your request so that it can be faster and cheaper in the future.

And so all of this context management is built into the harness and it does it automatically for you. Steering is really important. So these agents, these deep agents, they're long horizon agents, but that doesn't mean they're fully autonomous. You still need the human in the loop. And so deep agents comes with really good human in the loop controls. And then lastly, delegation. Agent harnesses can be used to kick off other agents, whether that's planning agents or other sub agents for different tasks.

And so all of that delegation communicating with those subagents and having them communicate back to you, all of that is specified in an agent harness. And so deep agents contains all of these different things. We spend a

ton of time doing both research and applied AI work to make sure that the way we do summarization is the best, the way that we do subagents is the best. And so today, one of the things that we're launching is DeepAgeant 0.6, a new version of DeepAgeants.

There's three big things in it that are informed by three trends that we see in the industry. So one trend we see in the industry is the rise of open models. DeepSeq V4 launched last week, and it's just as performant on certain tasks as frontier models. So open source models are getting better. And the other thing that's happening is cost is rising. The cost of frontier models keeps on going up and up, and usage keeps on going up and up.

And so one of the things that we think will happen more and more is more usage of open source models. And so with DeepAgent 0.6, we want to make it the best place to use open source models. So we're launching native support for GLM5, DeepSeek, and Nemotron models. We have best-in-class integrations with inference partners like Fireworks, Base10, and NVIDIA. And the easiest way to try out open source models in a harness is by coding. And so we're launching deep agents code as an open source example of how to build a coding agent on top of deep agents.

And we're making it really, really good for open source models. The second thing that we're seeing is a lot of attention around the execution environment. So we talked about this. We talked about the virtual file system, which a really simple way to let the agent interact with what it thinks is a file system, but it's just a pretty simple database under the hood. And then on the other extreme, we have a full-blown code sandbox where the agent can write and execute code, spin up a web server, any of that.

And so this is a spectrum. But what sits in the middle? And so in DeepAgent 0.6, we're launching CodeInterpreter. What CodeInterpreter is, we use QuickJS, which is a JavaScript runtime. And it basically lets the agent write and execute code in this kind of like, repel-like environment. So it's a subset of the JavaScript language, and you can't do everything that you can do in a sandbox. So you can't run Docker or things like that, but you can still write and call tools programmatically.

You can manipulate data files. You can read and write from the file system here as well. And so we think this is a really interesting middle ground where it's super lightweight to deploy. So this is the benefit of it. You don't have to spin up a separate sandbox for each agent. You can just deploy this, and it's really easy to run in a multi-tenant way, but you still get a lot of benefits that you get from coding.

The third trend that we've seen is that it's still really hard, but really important to build delightful UIs. UIs matter, interacting with the agents matter. And one thing that's happened is these agents have gotten more and more complicated, and the events that they're emitting are correspondingly more and more complicated. They emit text, they emit tool calls, images, reasoning, sub-agents, stuff, How do you do that? And so we want to make it as easy as possible for people to run agent harnesses and hook them up to delightful UIs that they build for their customers.

And so the third launch in Deep Agents 0.6 is better support for streaming. So we have a brand new streaming

protocol. We have four different front-end SDKs for different front-end languages. And we're partnering with UI frameworks, like Co-PilotKit, Assistant UI, and Vercel, to make sure it's really tightly integrated. UIs are a big part of building agents, and we want to make it the easiest way to do so. So if you haven't already tried out deep agents, please go try it out today.

This is where we think the future is heading. We think agent harnesses are getting more and more robust, more and more production ready. And deep agents is our version of that. I want to talk about the test phase next. So you've built your agent. How do you know if it's actually working? This is where Langsmith evaluations comes in, something we've been building over the past two years. Testing for agents looks different than testing for software. You want to build up data sets, so reference inputs and reference outputs, or maybe criteria to score how it's doing.

You want to define metrics. How do you know if the agent is passing? This could be correctness. This could be hallucinations. But you want to define some metrics that work for your use case. And then you run your agent over these data sets, and you score them on these metrics, and you create these experiments, and you can use these experiments to hill climb on certain things. Or you can use them to make sure that you're not regressing as you're making changes.

Evaluations are a key part of Lang Smith, and we're launching some stuff around this today. But I'm going to talk about that later. So I'm going to go on to deploy next. So you've built your agent. It's running locally. Now you want to go to production. There's a bunch of challenges that are going to emerge as you do that. First, you have to go from a single user, just you. Now you're serving many in production. Environment variables and memory, these now have to be specific to the users that you're interacting with.

Off, you need to control who can access it. You can't just open it up to everyone. And then when it runs locally and it dies, you're just testing it. You can resume from there. But you need to run it durably when you're running it at scale. And so a year ago, we launched Langsmith deployments to help with this. So there's a bunch of features built into this. We launched about 30 different API endpoints to handle streaming, human in the loop, auth, other things like that.

It's a single standard deployment pattern. Today it's served over 100 million agent runs and is trusted by companies like Workday, Cisco, Etsy, Podium, and ByteDance. But we also realized that Langsmith deployments isn't the only thing you'll need to bring an agent into production. We've talked about how agents need an execution environment, And one of the best execution environments is a sandbox. So whether the agent is a coding agent or not, reading and writing code can be really impactful for the agent.

It can manipulate data that way. It can use CLIs that way. And so we generally think that a lot of agents will need the ability to write and execute code in sandboxes. And so today, we're excited to announce that Lang Smith sandboxes is generally available. Yeah. (audience applauding) We think this is a big part of Agents in the future, and so we made it really easy to spin up sandboxes in less than a second. There's persistence for these sandboxes, so that will survive across interactions.

It supports snapshots and forks. One of the coolest things we've done is this off proxy. So if you wanna give the LLM the agent, the ability to use something that requires an API key, you don't actually want it to see that API key because then it could get prompt injected and leaked it. And so we have an off proxy that sits outside the sandbox and basically intercepts traffic and inserts that in. And this is all part of the Lang Smith SDK, and it's completely framework agnostic.

So you can use it with deep agents, you can use it without deep agents, you can use it for testing agents, you can use it for RL, you can use it for running agents from production. We're really excited to see how people use it. We've already had a number of customers using it. Monday, for example, uses it for their AI assistant sidekick, and we're excited by the feedback that we've gotten so far. Another big part of bringing agents to production is context.

So LLMs by themselves, they don't know anything, or they don't know everything. Humans don't know everything. When we need to look up things, we go to a library. We look at books to learn things, we read them, and that's exactly what agents do as well. They do that with context. And that context has evolved over the years as LLMs have gotten better. So we had a prompt hub in Lang Smith for the longest time where you could store inversion prompts.

And prompts were the things that kind of like guided agents, but over the past few months and years ago, we've seen that the context provided to agents has graduated from prompts into things like agent.md files, so really detailed instructions, and skills that these agents have used. A lot of these are shaped by coding agent standards. Both of these are open standards, by the way, so they're part of open foundations. And so as the context that LLMs have needed has evolved from prompts into these agent.md and skills, So has our tooling for that.

So today, we're excited to launch Lang Smith Context Hub. Woo! (audience applauding) And so, what's in this Context Hub? So, you can take your agents.md files, you can take your skills, you can take your like LLM wikis, so this thing that Carpathie did where he basically ran an LLM and had it generate wikis, which are condensed knowledge in markdown files. We think that's gonna become a more and more common pattern, So you can take all of this, you can store it in Context Hub, you get versioning, you get tags, you get comments, and then you can use these.

You can pull them down locally, you can run them in your coding CLI, you can run them in deep agents as a virtual file system so we have an integration there, or you can use them in whatever agent harness you have. We think Context Hub is a first start at memory. We think memory is really, really important to agents in the future. We think agents.md and skills, you can absolutely view those as an early form of memory.

And those are open standards. So that's a great thing because we think memory should be open. We think memory should not be locked in to an LLM, to a framework, or to a platform. And so even though we're building Context Hub, we want this idea of context to be really open. And so we're working with a number of companies, including Redis, Elastic, Mongo, and Pinecone, to turn this first stab at memory into something that's an open standard for memory, for agents.

So we think everything should be open. [APPLAUSE] And lastly, we've got monitor. So you've launched it. But how do you know what the agent's actually doing? This is where a lot of the core links with functionality around tracing. So you can make sure that you can see the step of every agent and what the inputs are and what the outputs are. Dashboards, so you can track cost and latency over time. online evals, so you can run LLM as a judge or code over these traces, get some feedback and then attach that or just capture user feedback. All of this is part of Ling Smith Observability. We've been building a lot here over the past two years and over the past few months. We've got a few very big launches here, but we're going to save that for a little bit later. So this is the agent development life cycle, and this is what it makes -- this is what it looks like to bring agents and maintain agents in production. And when you You do it once for an agent that's fine.

When you do it for 10 or when you do it for 100 agents, that's when you really need to start to think about the governance of all of this. How do you do this at scale? How do you do this in a cost efficient and secure way? And so specifically, two of the concerns that we've seen emerge around governance over the past few months have been cost and data exposure. LLMs are getting expensive. How do you know how much agents are spending, how much specific users are spending on agents, and how do you avoid surprise bills?

On the data exposure side, LLMs are great at working with data, but they shouldn't be able to access every source of data. And so how do you control what they can and can't see? Today, we're launching Langsmith LLM Gateway in beta to help with that. Woo! [APPLAUSE] You guys can't just pick and choose which announcements you clap for. You've got to applaud for all of them. They're all great. So what is Link Smith LLM Gateway? So you've got your agents, they're running.

Link Smith LLM Gateway basically sits between them and their LLM calls. You can set spend limits. You can have visibility, total visibility over spend. And then you can also set guardrails to help with PII and secret detection. It integrates with a bunch of coding agents out there. So you can use it. We know that coding agents is the most popular thing that people are using. And that's where a lot of these costs are happening. And so we've made sure to integrate it with a lot of coding agents out there.

It integrates with all the LLM providers that LangChain can help you access, and everything's traced automatically to LangSmith. So this is the full agent development lifecycle. Everything that we're building kind of fits into this. And we recognize that there's a lot in here. Taking an agent and going to production and going through this lifecycle involves a bunch of different pieces. And so we want to make it as easy as possible for people to take all these pieces, take deep agents, which is our agent harness under the hood, and really, really easily go to production.

And so to make that into a single API, today we're announcing in private preview, Manage Deep Agents. Woo! [APPLAUSE] So Manage Deep Agents is a single API for interacting with all these different components. So it runs the deep agent harness under the hood. It's deployed with Lang Smith deployments. And so that's where the agent will run. The models that you can access include any models out there. So we obviously integrate with OpenAI and Anthropic, but also with the open source model providers like Fireworks and Base10.

All of the agent instructions and memory are stored in Context Hub, so that as you or your users or the agent itself updates them, you can see them there and version them there and track them there. These agents, when they're deployed, you basically want to deploy the agent over here and then give it access to a sandbox to run and write tools over here. And you want those to be separate. And so that's the architecture that we took.

And we used Langsmith sandboxes to help power those sandboxes in the hood. MCPs are how you connect them to tools. So you can provide MCPs directly, or you can use Arcade or another partner that provides access to a lot of MCP servers to do that. And all of this streams out in the new streaming protocol that we announced. So you can integrate it super seamlessly with Co-Pilot Kit, Assistant UI, and other frameworks. So combining everything, LangChain Open Source and LangSmith power this whole agent development lifecycle.

And we think that traces sit at the center of this lifecycle. And so we spend a lot of time as a company thinking about traces and thinking about how to build the best experience around them. Everyone does. But there's no one at the company who thinks more about traces than my co-founder, Ankush. He's been thinking for the past almost a year around how we can make the experience around traces the best possible experience, because this is what powers that whole loop.

And so we're launching a lot of really cool things around this, but in order to talk about traces and his love of traces, I'd like to welcome onto the stage my co-founder, Ankesh Gola. (audience applauding) (upbeat music) [MUSIC PLAYING] Hey, everyone. I'm Ankush, co-founder and CTO here at LangChain.

As Harrison mentioned, I do spend a lot of time thinking about agent traces. And that's because we really think that agent traces are at the center of the agent development lifecycle. Each agent trace captures the unique behavioral record of what your agent actually did at a specific point in time. That being said, agent traces, or more generally, agent observability, poses a unique data infrastructure problem.

For one, agent traces are very deeply nested and can contain thousands, if not tens of thousands, of intermediate steps. The payloads associated with agent traces are large and unbounded. These two characteristics of agent traces are a direct result of agents running for longer time horizons and LLM context window sizes getting larger and larger.

We're seeing an increased number of agent traces being logged with modalities, such as images and voice. Voice is getting especially popular with applications like customer support. Finally, the access patterns or the query patterns needed to effectively mine your agent traces for useful insights are unique and complex. Agent traces are not only encoding more information and getting more complicated, they're also growing in volume as agents become more and more ubiquitous.

Here's a figure that highlights a real Langsmith customer's weekly trace volume over time. As you can see, in the early stages of testing and development, the weekly trace volumes are relatively small. But as the agent enters production And as new agents enter the picture, the weekly trace volume that we now see is over 150 million. So as mentioned earlier, the payloads associated with agent traces are large, and they're getting bigger over time.

Over the past couple of years, we've seen the P50 payload size associated with agent traces sent to Langsmith grow from 6 kilobytes to 37 kilobytes. And P99 has grown from still a pretty sizable 364 kilobytes to 12 megabytes today. And another data point. Earlier this year, we had a single customer send us 50 terabytes of trace data on a single day.

That's quite a lot of data. This video highlights what a modern agent trace in Langsmith looks like in practice. As you can see, lots of intermediate steps, very deeply nested. This trace is actually pulled from one of our internal go-to-market agents that's built with deep agents. And another thing to point out is that this trace actually encodes 8.1 million tokens.

Traditional data infrastructure was not built for the challenges associated with agent observability. If you're using suboptimal infrastructure to handle agent observability workloads, you will experience slow queries and ingestion bottlenecks. You'll also experience rising infrastructure costs and complexity as you try to scale up. When you're iterating on your agent, you cannot afford to have slow and cumbersome interactions with your agent traces.

And so today, we're super excited to be launching SmithDB. [APPLAUSE] SmithDB is the database we purpose built for agent observability workloads. We've been working on SmithDB for the past few months and are incredibly excited by the results that we're seeing in production. Please stay tuned for this video to learn a little bit more. Introducing SmithDB, the database purpose built for agent observability.

This is a trace, a record of what your agent does, how it thinks, decides, and performs. Now, imagine thousands of them, millions growing each day, because agents are everywhere. The problem is that traces aren't like traditional logs and metrics. They're larger, deeply nested, multimodal, and you don't query them the same way. You're searching across text and meaning, In traditional databases, they weren't built for this. So as traces grow, searches slow until now.

SmithDB, the database purpose built for agent observability. Search across Vultrace content, query entire agent conversations, access conflicts data instantly and at scale. Building reliable agents requires rapid iteration. With SmithDB, you can move from data to insight, to improvement faster. (audience applauding) - That video got me pretty hyped up, not gonna lie.

Whoever's creating our videos is doing an amazing job. So now let's take a closer look at what SmithDB actually is and how it's architected. The first thing to point out is that SmithDB is backed by object storage. This gives us a few nice properties. First, object storage is incredibly cheap, and it scales pretty much infinitely. The second is it gives us what's called compute storage separation. This allows us to elastically scale the services that Baxmyth DB without any complex shuffling or sharding of data as you scale up the services.

Third, this gives us an architecture that is relatively easy to set up in self-hosted environments where dated residency requirements are strict. Now at a high level, the Langsmith services connect to SmithDB after getting assignments from our cluster manager. Our cluster manager helps ensure that load is evenly distributed across

our servers and it also maintains some semblance of what's called sticky routing.

Sticky routing is important because it helps utilize cache, and it also helps batch our data effectively for ingestion. So during ingestion, raw trace data comes into our ingestion service. It gets batched, and it gets durably stored to object storage. These files are registered in a Postgres backed meta store. At query time, we figure out which files are necessary to be scanned for the queries. We download them from object storage, and we scan them.

We also heavily utilize SSD caching and memory to minimize round trips to object storage. Finally, we have a compaction service that helps shape our files for more optimal querying. SmithDB is specifically designed for agent observability access patterns. We'll walk through some of these access patterns in the next few slides. SmithDB makes clicking into individual traces and individual intermediate steps really snappy and fast.

Agent observability isn't just about asking what happened across millions of traces. It's also about asking what happened in this one particular agent execution. That's why random access is really important and something that we've optimized in SmithDB. Agent traces can contain megabytes of text data. And oftentimes, you need to search across your agent traces using keywords or phrases.

As you can see, SmithDB makes full text search really snappy and interactive. To accomplish this, we've actually built a custom inverted index layout specifically designed for object storage.

and you can You often need to pick a specific time horizon and apply filters like on metadata, tags, name, latency, and other attributes. Scanning and filtering speed is something that we've highly optimized within SmithDB. And as you can see, the results are nearly instant.

We're incredibly pleased by the performance that SmithDB brings to Langsmith across these key agent and observability workloads. Compared to before SmithDB, these Lang Smith workloads are now anywhere from 6x to 15x faster than before. This is absolutely massive. [APPLAUSE] But look, don t just take our word for it. We re super lucky to be working with customers like Clay and Vanta, who were early adopters for SmithDB.

to be on Langsmith. As you can see Jeff from clay and Andy from Banta both state how SmithDB has completely transform their experience with their traces within Langsmith. So SmithDB is purpose-built for agent observability and we built it using a modern tech stack. The entire system is written in Rust and we use two awesome open source projects. one is called a PEP. patchy data fusion, which is an extensible, rust-based query engine.

The second is Vortex, which is an extensible file format that allows us to pick custom encodings and custom chunking strategies for the different columns that we store. On top of this foundation, we have heavily-- we've added some heavy customizations around indexing, specifically for full-text search. We've added custom query planning and execution plans. And we also have invested in custom storage layouts for all the data that we're

storing within SmithDB.

Here are some of the technical challenges that we've had to solve when building SmithDB. These are just a few of them. They're quite a lot, but I try to choose. So in agent observability workloads, your spans are distributed. They come in different parts. And this is because agents run for long time horizons. And so you can have a start event that gets sent sometimes hours before your end event. And so finding and merging these distributed events together at query time and at compaction time is a technical challenge that we had to solve.

Doing it efficiently is something that we've invested a lot of time in. A lot of the queries within Lang Smith are top case style queries. So they contain an order by and limit. And a more naive query plan would essentially like find all the files that are in range and fan them out, scan them, do some type of merge and top K operation. This is actually like a little bit expensive on object storage, actually quite expensive on object storage.

So what we've done is we've taken a time window based approach and encoded that into a custom execution plan that feeds top case to alqueries within SmithDB. Finally, in observability workloads, you often need to serve the most recently ingested data as fast as possible. And in order to do that, what we do is we buffer data in the ingestion service even after it's been durably flushed to object storage.

and we buffer it in memory and on SSD. And we make that data available to the query service to avoid downloading a ton of small files for leading edge style queries. Langsmith performance is not only important for human UX, but also agent UX. Increasingly, agents are not just the thing that are being observed by Langsmith, they are also the users of Langsmith.

And it's a huge hit to have your agents slow down by inefficient tools. I really like this quote from Jeff Dean, who states that, you know, as we get agent-based systems that are operating multiple times faster than a human, your tools can become like an Amdell's Blah Blah bottleneck. We're super excited to announce that SmithDB is now serving core observability workloads across all of US Cloud on Langsmith.

So if you're using Langsmith on smith.langchain.com, all of your interactions and your tracing projects page are now powered by SmithDB. Soon-- [APPLAUSE] --we're working quickly to get the rest of the Langsmith UI back by SmithDB. And we're going to bring it soon to self-hosted and across all of global cloud traffic. So SmithDB sets a really, really strong technical foundation for everything that we want to build into Langsmith.

And to share some more exciting updates about Langsmith, I'd like to welcome Harrison back on stage. Thank you so much. [APPLAUSE] (audience applauding) - SmithDB is absolutely incredible. As you can see, I have the fun and easy job of just talking about what we build, but on Cush and the whole team there goes out and builds really complex engineering projects and databases that help power everything in the agent development life cycle.

And that's really how we think about what we do. We want to accelerate everything in this life cycle. We think

traces and SmithDB are a key part of that. And so with this solid foundation, a big part of what we're thinking about now is how can we accelerate this life cycle even more? Because even with traces and even with observability, the visibility that it provides is table stakes. Being able to see what your agent did at each step is absolutely needed, but that is table stakes.

It's still hard to spin this iteration flywheel. Finding issues among the tens of thousands, hundreds of thousands, millions of traces that you have is still hard. Once you find those traces, understanding issues in them is still hard. They can be really, really long. You have to comb through them and see exactly where the LLM is messing up, where it's calling the wrong tool. Once you understand that issue, Fixing them is hard.

Do you want to change the prompt? Do you want to change the code? Do you want to change some tools? There's a bunch of different things that you have to investigate and try to tie back to the core issue. And then once you fix it, avoiding repeating these issues is hard. Sometimes it can feel like whack-a-mole. You fix one thing, and then you change the prompt, and it reappears. And so how do you avoid this? And so we spend a lot of time thinking about how we can spin up this flywheel faster and faster because it is still hard.

All these things I mentioned are still annoying. And so what do you do when you have a lot of really annoying things that are hard and important that you don't want to do? You build an agent to help you with them, of course. And so that's why today we're really excited to launch an agent in Langsmith, an ambient, proactive, action-taking agent that we're calling Langsmith Engine. [APPLAUSE] Specifically, what it does is it will sit on top of your traces.

It'll go through them on a schedule in the background. It will detect issues and assign a priority to them. It will back its evidence with traces. And then it will suggest concrete resolution actions. We've been working with a number of startups over the past few weeks to roll this out to them. And as you can see, it is already proactively suggesting evals to add and code changes to make and has dramatically reduced time to detection and time to triage.

So specifically what Engine can do, it acts at all stages of this life cycle. So it sits on top of traces. It can suggest code changes if you hook it up to your GitHub or code base. It can suggest data points to add to data sets so that you can use these to avoid regressions in the future. If you connect it to Prompt or Context Hub, it can suggest changes there. And then it can also suggest online evals to add so that you can track these issues over time.

Let's see a little video of it in action. [MUSIC PLAYING] (dramatic music) (dramatic music) [MUSIC PLAYING] [APPLAUSE] LayingSmith Engine is in public beta as of today.

So I already see some people on laptops log on to LayingSmith and try it out. Overall, as a company, we're building the entire platform for the agent development lifecycle. Over the past few years, we've been building a lot of the different apps and infrastructure pieces from observability and evaluations and Prompt Hub that help power a lot of this lifecycle. Today, we launched SmithDB as a foundational database to power all of this.

And we built LangSmith Engine as an agent that sits on top and helps you spin this flywheel faster and faster. more than just a platform, we also want to be a partner to everyone in this room, in the community, in the entire space in general. And that's why we're really thrilled about Interrupt in General. This is an opportunity to interrupt your day and take a pause and learn from us, but also other presenters and other partners in the ecosystem and other companies and other people that you're sitting next to, and I would encourage everyone to take full advantage of not only today but also tomorrow as well.

[BLANK_AUDIO]