The agent development loop with LangSmith + Claude Code / Deepagents

8 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=ZPGFL4N4DIC
https://www.youtube.com/watch?v=zpgFl4N4DIc

SUMMARY

Lance discusses his recent blog post on debugging deep agents using Langsmith, emphasizing the integration of Langsmith as a system for recording traces from code agents like deep agents. He introduces a utility called Langmith fetch, which allows users to easily retrieve recent traces from their projects, facilitating an iterative feedback loop for improving agent performance.

- *Langsmith serves as a system of record for traces from code agents.*
- *The Langmith fetch utility provides a command line interface to easily fetch recent traces.*
- *A feedback loop is established where code agents can reflect on traces and update their code accordingly.*
- *Users can create skills for cloud code and deep agents to automate the fetching of traces.*
- *The utility supports debugging and iterative refinement of prompts and agent code.*
- *Traces can be logged to separate projects for better organization and analysis.*
- *The integration of Langsmith with code agents enhances the debugging workflow significantly.*

Hey, this is Lance. Recently put out this blog post called debugging deep agents with lang. And the big idea here was connecting lang as a system of record for your traces with code agents like deep agents, but it could be other code agents like clock code to create kind of an iterative feedback loop. So you're having a code agent produce some langraph code that's being run. Traces are going to lang. And there's a way for the code agents to pull traces back, reflect on them, and update your lane share langraph code. That's the feedback loop we want to implement. Now to support this, I recently created this little utility called Langmith fetch.

Super simple. All it is is just a command line interface for Langsmith. You can just very easily fetch recent traces or threads in any given project. Now let's see that in action. I'm in a project that has a single file agent.py. This is just our boilerplate agent demo from the docs. I will open up a terminal, create a virtual environment, activate it, and I'm going to pip install Langsmith fetch as well as the dependencies needed for my script just like this. Cool. That's all done. And I'm going to create a new Langmouth project. Test Lang Smmith fetch. This means that traces that I run here are going to be logged to that particular project in Langmith. Let's just run the script. I see a new project created in my tracing projects. I'm going to try to use a Lenmith fetch utility to grab the most recent trace. All you need to do is just lang fetch traces. By default, it'll grab the most recent one. It'll automatically fetch from the project set in the environment and we can see it grabs some recent trace. Great. Okay, that's all great. It's not super interesting, but there's a starting point. Now, let me show a code agent.

I'll start with cloud code executing a similar workflow. So, I spin up claw and I'll tell claw to run this script agent.py and then use the langu fetch utility to grab the trace and I'm going to tell it to just use the help

command in langu fetch to learn how to use the CLI. Cool. It wants to run engine.pay. Great. I'll let it do it. Now it's Now it wants to run length fetch help to learn about this CLI. Great. And indeed will call the correct command here. Lenmouth fetch traces. It supplies a directory which is nice. And it will specify only wants the most recent one. So that's all great.

Found the trace. Saved it locally. Good. It's reading it. And there we go. Now claw gives us a trace summary of everything that happened in our invocation of Asian.py. Now this seems kind of simple. Who cares? But it's actually pretty useful once code agent connect up to traces in langmmith then you can implement things like a feedback loop run some code trace will be logged to langmmith code agent can read the trace reflect on it and make updates so I use this all the time for iter refinement of things like prompts or correcting errors or improving code for anything that's being traced to lang so it's very generally useful now it's a bit tedious to tell cloud code about this CLI every time. So, we can just create a skill for it. Here are some instructions. Create a global claude skill in this directory that instructs claw code how to use lang fetch. Use the help command to learn about it. Follow the skill creation guidelines here. And easy enough. Let's kick that off. Good.

Grabs the docs. It's also running the command. Cool. It's going to make the skills directory. Good. And it will create the skill.md. Very nice. We can open up that skill.mmd that it created. And we can see here is our skill. Pretty nice. So this YAML front matter gets loaded into clot every time we kick off a new session and it's going to invoke this skill anytime we say something like grab or fetch get traces from Langmith and it tells Claude all about our CLI which is quite nice.

Now let's shut down cla kick off a new session. We'll ask it to list it skill. Good. There's our user skills. Fetch lang traces is now there. Ask cloud to grab the most recent lenith trace. And perfect. It will use our skill. Good. And it just automatically knows how to use our fetch utility. Fantastic. grabbed a few traces and saved them all locally. Fantastic. Okay, so look, now you know how to connect clock to Langsmith really effectively. And this is fantastic for doing iterative debugging like using cloud code to help you build a langraph agent. You want to use traces from agent invocation to help cloud code know what to improve, such as the prompt, such as the agent code itself, such as the tools. Now, let me show this all with the deep agent CLI and the principles are extremely similar. You can think about deep agent CLI as an alternative that's open source to cloud code. Now there's one subtle difference I want to explain.

I'm going to start deep agent CLI with two environment variables set. One is the deep agents lenith project. The second is length of project for general tracing. Now why am I doing this? Here's the issue. The deep agent itself will trace to a project and that can be configured right here. Specifically this debasent lenith project is the project name where all the debasent specific traces will go. But any code that the deep agent attempts to execute will be saved to your default lang project. This is really nice because it separates all the deep agent tracing itself from any code it's executing with for example shell tool. Now let's see an example of this. I'm in a repo. Here is agent.py.

It's just a different agent file. Very simple though. Kick off deep agents with those two environment variables set. Great. And deep agent kicks off. Now what's neat is you can actually see the two different Langsmith

projects that's going to be using. One is the deep agent itself will be tracing to this agent traces test and anything it executes will just be saved to the lang project by default that we set. Now let's tell the deep agent to run agent and we'll tell it to just go ahead and run the script as it is. Cool. So now it's going to call the shell tool to simply run the script. We approve that. Perfect. It ran the script but the script does not produce any results which is exactly what we expect. Now let's go ahead and check the traces. Okay, perfect. We can see tracing in two different projects here. Agent traces test lang fetch.

Let's first check this. Perfect. So this is the trace I just ran. Cool. And this is all what we just ran. This was the initial invocation. This is when we specified that I wanted to run option one which is just to run the script. And this was the response. I went ahead and ran the script. You can see the shell tool here. So this is actually a subtle but very cool point. You can actually log everything the deep agent's doing to Langsmith, which actually is quite nice for some other patterns I'll show in a future video.

But the point is that's all going to log to one Langmith project. And now the shell tool was used to actually run our script agent.py that was logged to a separate project. This test lang fetch. So we can see this was our invocation of the agent. What is 3+4? Perfect. And the full trace is here. Now let's tell the deep agent to use lang fetch cli which I've already installed my environment to get information about the most recent trace in the project. Cool. And it's going to ask me to run the shell command. That's exactly what you expect.

And very good. It read that and it run the traces command to get the most recent trace again. It can specify limit one. It can specify format and save it to this particular folder. Perfect. Run. And perfect. It grabbed the trace and now it can summarize the user message. What is 3+4? All right. This is perfect. So what's happening is it's using lines fetch to grab the most recent trace in our project which again was our agents right here agent.py PI that the deep agent ran and it fetched the trace. It reflected on it gives us a summary. So same principle we saw in cloud code but now running in deep agents. Very nice.

So let's just create the same skill that we already use with cloud code. I'll make a directory for it. Again skills live in your home directory deep agents. This is the default agent skills and fax lang traces. Create the skill.md. And I'll just paste over everything that we already created before. And then just run deep agents skills list. Search around and there it is. Our new skill is just picked up. Run deep agents. Supply langu project that you want to actually use for fetching traces. Ask it to fetch the most recent traces. Perfect. And ask me some follow-up questions because it read in the skill. It understands how to use lang fetch. And I'll say go ahead.

Cool. And you can have it like read and summarize the most recent tracer as an example. Perfect. So let's kind of pull back and what have we done here? One, it's clear that the ability to connect Langmith as a system of record for your traces with code agents is very very useful because lang is a great way to capture invocations of agents. As an example, long trajectories, code agents are great for reflecting those trajectories and making updates to your system prompt, to your tools, or your agent harness.

So, you want to connect these two worlds. A good way to do that is a very simple command line utility that just allows these agents to easily grab Langmith data. You just want something nice and simple and crisp. That's

what l fetch is. Super simple CLI. Grab traces. It's designed to work with code agents. The help command is very descriptive. So it can just code a just read that help output and use that to know what to do.

Bas it works very nicely as we saw with cloud code or with deep agents and in both cases you can just introduce a skill for with lang which encapsulates whatever logic you want about how to use this utility. So this is very useful. It's great for debugging and hopefully this is a helpful intro to using links with fetch with various code agents in a practical way for the debugging workflow. Thanks.