James Briggs LangChain Mastery in 2025 | Full 5 Hour Course

Summary

The video transcript presents an in-depth, comprehensive course on LangChain, a popular Python framework designed to simplify the integration and orchestration of large language models (LLMs) and AI agents. The course is structured to take learners from complete beginners in LangChain to proficient AI engineers capable of building sophisticated AI-driven applications. It covers foundational concepts, practical implementations, and advanced features such as agents, conversational memory, prompt engineering, LangChain's expression language (L-EL), streaming, asynchronous programming, and finally culminates in a capstone project that integrates all these concepts into a functional AI chat application.

The course begins with a theoretical overview explaining what LangChain is, when to use it, and the pros and cons of adopting this framework versus direct API calls. It stresses the value of LangChain as an on-ramp for engineers from diverse backgrounds, especially those without deep AI or ML expertise, by abstracting complex workflows while allowing gradual deep dives into the underlying logic.

Hands-on chapters progressively explore core LangChain components: from initializing and configuring language models, crafting effective prompts, and managing conversational memory (buffer, window, summary, and hybrid memory types), to building and customizing AI agents that can call external tools and APIs. The course also delves into LangChain's newer features like expression language and runnable abstractions that enable modular, composable AI workflows.

A significant portion is dedicated to agents, including the popular ReAct agent pattern, where LLMs iteratively reason, decide on actions, invoke tools, and process observations until they reach a final answer. The course covers designing custom agent executors, handling parallel tool calls, and managing agent states across multiple iterations.

Advanced topics include streaming and asynchronous operations, critical for responsive AI applications, particularly chatbots and web services. The course demonstrates how to implement token-by-token streaming, asynchronous callbacks, and integrate these with FastAPI to create a real-time streaming API backend.

The capstone project consolidates all earlier lessons by building a complete AI-powered chat application capable of multi-tool use, streaming real-time responses, structured output formatting, and conversational memory. It also introduces integration with third-party APIs like SerpAPI for web search and demonstrates best practices for environment setup, local and cloud execution, and running the frontend with Node.js.

Throughout, the course balances conceptual explanations, practical coding examples, and architectural insights, empowering learners to understand, customize, and extend LangChain-based AI systems in real-world scenarios.

Highlights

- 🤖 Comprehensive introduction to LangChain framework, from basics to advanced AI engineering.
- 🛠️ Practical demonstrations of building AI agents capable of tool use and iterative reasoning (ReAct pattern).
- 💬 Detailed exploration of prompt engineering and conversational memory types for chatbots.
- ⚡ Streaming and async programming techniques for responsive real-time AI applications.
- 🔄 Explanation of LangChain's expression language (L-EL) and runnable abstractions for modular workflows.
- 🌐 Integration with external APIs (e.g., SerpAPI) and multi-tool orchestration in AI agents.
- 🚀 Capstone project building a fully streaming, multi-tool AI chat application with backend and frontend.

Key Insights

- 🧩 **LangChain as an On-Ramp for AI Engineering:** The framework abstracts complex AI workflows, enabling developers from various backgrounds (not only ML specialists) to quickly prototype and deploy sophisticated language model applications. Its design supports a gradual learning curve where users can start simple and progressively understand and customize the underlying mechanisms. This duality of abstraction and transparency is a key strength for AI adoption.
- 🤔 **Prompt Engineering is Fundamental:** The course emphasizes that prompts are no longer just simple queries but structured instructions that define LLM behavior, context, constraints, and output format. Techniques like few-shot prompting, chain-of-thought prompting, and context augmentation (e.g., retrieval-augmented generation) are critical for controlling LLM outputs and reducing hallucinations, thereby improving reliability and usability in production.
- 🧠 **Conversational Memory Enhances Contextual AI:** Implementing different memory strategies (buffer, window, summary, hybrid) allows AI systems to

maintain relevant context over long interactions. The use of summaries balances token costs and context retention, which is vital for scalable chatbots and agents to provide coherent, personalized, and contextually aware responses.

- 🛠️ **Agents Enable Intelligent Tool Use and Iterative Reasoning:** Agents embody a core AI engineering paradigm where the LLM not only generates text but also reasons, decides on actions, invokes external tools, and integrates the results iteratively. The ReAct agent pattern exemplifies this cyclical reasoning-action-observation loop, significantly expanding AI capabilities beyond static completion to dynamic, multi-step problem solving.
- ⚙️ **Streaming and Asynchronous Execution are Crucial for UX and Scalability:** Token-by-token streaming provides immediate feedback in chat interfaces, reducing perceived latency and improving user engagement. Asynchronous programming is essential to handle API calls and parallel tool invocations efficiently, enabling the development of responsive, scalable AI services that can operate in real-world production environments.
- 🔄 **LangChain Expression Language Simplifies Workflow Composition:** The introduction of runnable abstractions and the pipe operator enables modular chaining of AI tasks, which simplifies complex workflows into composable units. This paradigm encourages reusable, maintainable, and flexible AI pipelines, facilitating rapid development and experimentation.
- 🌐 **Extensibility and Real-World Integration:** The course underlines the importance of integrating with external data sources and APIs (e.g., SerpAPI for search), showcasing how agents can leverage real-world information dynamically. The capstone project demonstrates how streaming, memory, agents, and multi-tool use come together in a practical application, preparing learners to build production-grade AI solutions.

Overall, this course offers a thorough, hands-on roadmap for mastering LangChain and building advanced AI applications, blending theory, best practices, and practical code examples to equip learners for AI engineering challenges.

Transcript

Chapter

00:00

welcome to the AI Engineers guide for the L chain this is a four course that will take you from the assumption that you know nothing about Lang chain to being able to

proficiently use the framework either you know within line chain within line graph or even elsewhere uh from the fundamentals that you will learn in this course now this course will be broken up into multiple chapters we're going to start by talking a little bit about what line chain is and when we should really be using it and when maybe we don't want to use it

00:37

we'll talk about the pros and cons and also about the the why the line chain ecosystem not just about the line chain framework itself from there we'll introduce Lang chain we'll just have a look at a few examples before diving into essentially the basics of the framework now I will just note that all this is for Lang chain 0.3 so that is latest current version although that being said we will cover a little bit of where line chain comes from as well so we'll be looking at pre 0.3 uh version methods for doing things

01:13

so that we can understand okay that's the old way doing things how do we do it now now that we're in version 0.3 and also how do we dive a little deeper into those methods as well and kind of customize those from there we'll be diving into what I believe is the S of future of AI I mean it's it's it's a now and the short term potentially even further into the future and that is Agents we'll be spending a lot of time on agents so we'll be starting with a simple introduction to agents so that is

01:48

how can we build an agent that's simple what are the main components of Agents what do they look like and then we'll be diving much deeper into them and we'll be building out our own Agent X computer which kind of like the framework around the AI components of an agent we're building our own and once we've done our Deep dive on agents we'll be diving into Lang chain expression language which we'll be using throughout this course so line chain expression language is the

02:19

recommended way of using line chain and the expression language or L cell takes kind like a break from standard python syntax so there's a bit of weirdness in there and yes we'll be using it throughout the course but we're leaving the ELO chapter until this you know kind of later on in the course because we really want to dive into the fundamentals of Elsa by that point but the idea is that by this point you already have a good grasp of at least how to use the basics of lell before we

02:50

really dig in that point then we'll be digging in streaming which is an essential ux feature of AI applications in general streaming it can just improve the user experience massively

and it's not just about streaming tokens you know that that interface where you have word by word the AI is generating text on the screen streaming is more than just that it is also the ability if you've seen the interface of perplexity where as the agent is thinking you're getting an update of what the agent is thinking

03:24

about what tools it is using and how it is using those tools that's also another essential feature that we need to have a good understanding of streaming to build so we'll also be taking a look at all of that then we'll finally we'll be topping it off with a Capstone project where we will be building our own AI agent application that is going to incorporate all of these features we're going to have an agent that can use tools web search we'll be using streaming and we'll see all of this in you know a nice

03:57

interface that we can that we can work with so that's an overview the course of course it's very high level what I've just gone through there's a ton of stuff in here and truly this course can take you from you know wherever you are with L chain at the moment whether you're a beginner or you've used it a bit or even inter mediate and you're probably going to learn a fair bit from it so without any further Ado let's dive into the first chapter okay so the first chapter

04:26

of the course we're going to focus on when should we actually use Lang chain and when should we use something else now through this chapter we're not really going to focus too much on the code we you know every other chapter is very code focused but this one is a little more just theoretical what is line chain where's fit in when should I use it when should I not so I want to just start by Framing this line chain is one of if not the most popular open source framework within the python

04:58

ecosystem at least for AI it works pretty well for a lot of things and also works terribly for a lot of things as well to be completely honest there are massive Pros massive cons to using Lang chain here we're just going to discuss a few of those and see how Lang chain maybe Compares a little bit against other Frameworks so the very first question we should be asking ourselves is do we even need a framework a is a framework actually needed when we can just hit an API you have the open AI API

05:31

other apis mral so on and we can get a response from an llm in five lines of code on average for those is incredibly incredibly simple however that can change very quickly when we start talking about agents or retrieval augmented generation research

assistance all this sort of stuff those use cases those methods can suddenly get quite complicated when we outside of Frameworks and that's not necessarily a bad thing right it can be incredibly useful to be able to uh just understand everything that is going on and build it

06:13

yourself but the problem is that to do that you need time like you need to learn all the intricacies of building these things the intricacies of these methods and themselves like what you know how do they even work and that kind of runs in the opposite direction of what we see with AI at the moment which is AI is being integrated into the world at an incredibly fast rate and because of this most Engineers coming into the space are not from a machine learning or AI background most people don't

06:46

necessarily have any experience with these systems a lot of Engineers coming in that could be devops Engineers generic backend python Engineers even you front end Engineers coming in and building all these things which is is great but they don't necessarily have the experience and that you know that might be you as well and that's not a bad thing because the idea is that obviously you're going to learn and you're going to pick up a lot of these things and in this scenario there's

07:12

quite a good argument for using the framework because a framework means that you can get started faster and a framework like Lang chain it abstracts away a lot of stuff and that's a that's a big complaint that a lot of people will have with L chain but that abstract in away of many things is also what made sing chain popular because it means that you can come in not really knowing okay what you know rag is for example and you can Implement a rag pipeline get the benefits of it without really needing to

07:43

understand it and yes there's an argument against that as well just implementing something without really understanding it but as we'll see throughout the course it is possible to work with line chain in a way as we will in this course where you kind of implement these things in an abstract way and then break them apart and start understanding the intricacies at least a little bit so that can actually be pretty good however again circling back to what we said at the start if the idea or your

08:18

application is just a very simple you know you need to generate some text based on some basic input maybe you should just use an API that's completely valid as well now we just said okay okay a lot of people coming to L chain might not be from an AI

background so another question for a lot of these Engineers might be okay if I want to learn about you know rag agents all these things should I skip line chain and just try and build it from scratch myself well line chain can help a lot with that Learning Journey so you

08:53

can start very abstract and as you gradually begin to understand the framework better you can strip away more and more of those abstractions and get more into the details and in my opinion this gradual shift towards more explicit code with less abstraction is a really nice feature and it's also what we focus on right throughout this course that's what we're going to be doing going sing abstract stripping away the abstractions and getting more explicit with what we're building so for example building

09:26

an agent in L chain there's in very simple and Incredibly abstract crate tools agent method that we can use and like it creates a tool agent for you it's it doesn't tell you anything so you can you can use that right and we will use that initially in the course but then you can actually go from that to defining your full agent execution logic which is basically a tools call to open AI you going to be getting that tool information back but then You' got to figure out okay how am I going to

10:04

execute that how am I going to Sol this information and then how am I going to iterate through this so we're going to be seeing that stripping way abstractions as we work through as we build agents as we do as we bu like our streaming use case among many other things even chat memory we'll see there as well so line chain can act as the onramp to your AI learning experience then what you might find and I do think this is quite true for most people is that if you if you're really serious

10:37

about AI engineering and that's what you want to do like that's your focus right which isn't for everyone for certain a lot of people just want to understand a bit of AI and they want to continue doing what they're doing and just integrate AI here and there and maybe those you know if that's your focus you might sick with ly chain you know there's not necessarily a reason to move on but in the other scenario where you're thinking okay I want to get really good at this I want to just learn

11:04

as much as I can and I'm going to dedicate basically my you know my short-term future of my career on becoming AI engineer then line chain might be the on-ramp it might be your initial learning curve but then after you've become competent with line chain you

might actually find that you want to move on to other Frameworks and that doesn't necessarily mean that you're going to have wasted your time with L chain because one L chain is a thing helping you learn and two one of the main

11:34

Frameworks that I recommend a lot of people to move on to is actually line graph which is still within the L chain ecosystem and it still uses a lot of L chain objects and methods and of course Concepts as well so even if you do move on from line chain you may move on to something like L graph which you can no line chain for anyway and let's say you do move on to another framework in set said in that scenario the concepts that you learn from Lang chain are still pretty important so to just finish up this

12:07

chapter I just want to summarize on that question of should you be using Lang chain what's important to remember is that Lang chain does abstract a lot now this abstraction of L chain is both a strength and a weakness with more experience those abstractions can feel like a limitation and that is why we sort of go with the idea that L chain is really good to get started with but as a project grows in complexity or the engineers get more experience they might move on something like Lang graph which in any case is

12:44

going to be using Lang chain to some degree so in either one of those scenarios L chain is going to be a core tool in an AI engineered toolkit so it's worth learning in our opinion but of course it comes with its you know it comes with its weaknesses and it's just good to be aware of that it's not a perfect framework but for the most part you will learn a lot from it and you will be able to build a lot with it so with all of that we'll move on to our first of Hands-On chapter with Lang

13:18

chain where we'll just introduce Lang chain some of the essential Concepts I'm not going to Dag too much into the syntax but we're just going to understand a little bit of what we can do with it okay so moving on to our next next chapter getting started with a line chain in this chapter we're going to be introducing a line Chain by building a simple LM powered assistant that will do various things for us it will multimodal generating some text generating images generate some stret shed outputs it will

13:47

do a few things now to get started we will go over to the course repo all of the code all the chapters are in here there are two ways of running this either locally or in Google collab we would recommend running in Google collab because it's just a lot simpler

with environments but you can also run it locally and actually for the cap Zone we will be running it locally there's no way of us doing that in collab so if you would like to run everything locally I'll show you how quickly now if you

14:20

would like to run in collab which I would recommend at least for the the first notebook chapters just skip ahead there will be chapter points in the timeline of the video so for running running it locally we just come down to here so this actually tells you everything that you need so you will need to install uvie all right so this is the package manager that we recommended by the python and package management Library you don't need to use uvie it's it's up to you uvie is is very simple it works really well so I would

14:56

recommend that so you would install it with this command here this is on Mac so it will be different otherwise if you are on Windows or otherwise you can uh look at the installation guide there and it'll tell you what to do and so before we actually do this what I will do is go ahead and just clone this REO so we'll come into here I'm going to create like a temp directory for me because I already have the line chain course in there and what I'm going to do is just get loan line chain course okay

15:29

so you will also need to install git if you don't have that okay so we have that then what we'll do is copy this okay so this will install python 3.2.7 for us with this command then this will create a new VM within that or using python 3.2.7 that we've installed and then UV sync we actually be looking at the Pi Project at TL file that's like the uh the package installation for the repo and using that to install everything that we need now we should actually make sure that we are within the line chain

16:09

course directory and then yes we can run those three and there we go so everything should install with that now if you are in cursor you can just do cursor dot or we can run code do if mvs code I'll just be running this this and then I've opened up the course now within that course you have your notebooks and then you just run through these making sure you select your kernel pth environment and making sure you're using the correct VN from here so that should pop up already as this VM bin Python and you'll

16:48

click that and then you can run through when you are running locally don't run these you don't need to you've already installed everything so you don't this specifically is for collab so that is running things locally now let's have a look at running things in collab so for running everything in collab we have our notebooks in here we click through and

then we have each of the chapters through here so starting with the first chapter the introduction which is where we are now so what you can do to open this in

17:21

collab is either just click this collab button here or if you really want to for example Maybe this it is not loading for you what you can do is you can copy the URL at the top here you can go over to collab you can go to open GitHub and then just paste that in there and press enter and there we go we have our notebook okay so we're in now uh what we will do first is just install the prerequisites so we have line chain just a load of line chain packages here line chain core line chain open a because

18:01

we're using open Ai and line chain Community which is needed for running what we're running okay so that has installed everything for us so we can move on to our first step which is initializing our LM so we're going to be using GT40 mini which is side of small but fast but also cheaper model uh that is also very good for open AI so what we need to do here is get an API key okay so for getting that API key we're going to go to open's website and you can see here that we're opening platform.

18:41

open.com and then we're going to go into settings organization API keys so you can copy that I'll just click it from here okay so I'm going to go ahead and create a new secret key actually just in case you're kind of looking for where this is It's settings organization API Keys again okay create a new API key I'm going to call it Line train course I'll just put it on the semantic router that's just my organization you you put it wherever you want it to be and then you would copy your API key you

19:17

can see mine here I'm obviously going to reval that before you see this but you can try and use it if you really like so I'm going to copy that and I'm going to place it into this little box here you could also just and place it put your uh full API key in here it's up to you but this little box just makes things easier now that what we've basically done there is just passed in our API key we're setting our opening model GT40 mini and what we're going to be doing now is

19:48

essentially just connecting and setting up our llm parameters with L chain so we run that we say okay we're using a GT4 mini and we're also setting ourselves up to use two different LMS here or two of the same LM with slightly different settings so the first of those is an LM with a temperature setting of zero the temperature setting basically

controls almost the randomness of the output of your llm and the way that it works is when an LM is predicting the sort of next token or next word in sequence know

20:28

provide a probability actually for all of the tokens within the lm's knowledge base or what the LM has been trained on so what we do when we set temperature of zero is we say you are going to give us the token with highest probability according to you okay whereas when we set a temperature of 0.9 what we're saying is okay there's actually an increased probability of you giving us a token that according to your generated output is not the token with the highest probability according to the lmm but

21:03

what that tends to do is give us more sort of creative outputs so that's what the temperature does so we are creating a normal llm and then a more creative llm with this so what are we going to be building we're going to be taking a article draft so like a draft article uh from the aelio learning page and we're going to be using line chain to generate various sces that we might um find helpful as we're you know we have this article draft and we're editing it and just kind of like finalizing it so what

21:39

are those going to be you can see them here we have the title for the article the description and SEO friendly description specifically third one we're going to be getting the LM to Providers advice on existing paragraph and essentially writing a new paragraph for us from the existing paragraph and what it's going to do this is the structured output part is going to write a new version of that paragraph for us and it's going to give us advice on where we can improve our writing then we're going

22:08

to generate a thumbnail hero image for our article so nice image that you would put at the top so here we're just going to input our article you can you can put something else in here if you like essentially this is just a big article that's written a little while back on agents and and now we can go ahead and start preparing out our prompts which are essentially the instructions for our llm so line chain comes with a lot of different uh like utilities for prompts and we're going to dive into them in a

22:43

lot more detail but I do want to just give you uh the Essentials now just so you can understand what we're looking at at least conceptually so prompts for chat agents are at a minimum broken up into three components those are the system pront this provides instructions to our LM on how it should behave what its objective is and how it

should go about achieving that objective generally system prompts are going to be a bit longer than what we have here depending on the use case then we have our user

23:13

prompts so these are user written messages usually sometimes we might want to pre-populate those if we want to encourage a particular type of um conversational patterns from our agent but for the most part yes these are going to be using generated then we have our AI promps so these are of course AI generated and again in some cases we might want to generate those ourselves beforehand or within a conversation if we have a particular reason for doing so but for the most part you can assume that these are actually user and AI

23:48

generated now the line chain provides us with templates for each one of these prompt types let's go ahead and have a look at what these look like within line chain so to begin we are looking at this one so we have our system message prom template and human message which the the user that we saw before so we have these two system prom keeping it quite simple here you are AI system that helps generate article titles right so so our first component where we want to generate is article title so we're

24:23

telling the AI that's what we want it to do and then here right so here we're actually providing kind of like a template for a user input so yes as I mentioned user input can be um it can be fully generated by user it might be kind of not generated by user it might be setting up a conversation beforehand which a user would later use or in this scenario we're actually creating a template and the what the user will providers will actually just be inserted here inside article and that's why we

25:03

have this import variables so what this is going to do is okay we have all of these instructions around here they're all going to be provided to openai as if it is the user saying this but it will actually just be this here that user will be providing okay and we might want to also format this a little nicer it kind of depends this will work as it is but we can also put you know something like this to make it a little bit clearer to the llm okay what is the article where the prompts so we have

25:39

that and you can see in this scenario there's not that much difference between what the system prompt and user prom is doing and this is It's a particular scenario it varies when you get into the more conversational stuff as we will do later uh you'll see that the user prompt is generally more fully user generated or mostly user generated and much

of these types of instructions we might actually be putting into the system prompt it varies and we'll see throughout of course many different ways

26:09

of using these different types of PRS in various different places then you'll see here so I just want to show you how this is working we can use this format method on our user prompt here to actually insert something within the uh article input here so we're going to go us prompt format and then we pass in something for article okay and we can also maybe format this a little nicer but I'll just show you this for now so we have our human message and then inside the content this is the the

26:42

text that we had right you can see that we have all this right and this is what we wrote before we wrote all this except from this part we didn't write this instead of this we had article right so let's format this a little nicer so we can see okay so this is exactly what we wrote up here exactly the same except from now we have test string instead of article so later when we insert our article it's going to go inside there allly doing it's like it's an it's an F string in Python okay and this is again

27:15

this is one of those things where people might complain about Lang chain you know this sort of thing can be you it seems excessive because you could just do this with an nring but there are as we'll see later particularly when you're streaming just really helpful features that come with using line chains kind of built-in uh prompt templates or at least uh message objects that we'll see so you we need to uh keep that in mind again as soon as you get more complicated line chain can be a bit more

27:47

useful so chat prom template uh this is basically just going to take what we have here our system promt user prompts you could also include some AI prompts in there and what it's going to do is merge both of those and then when we do format what it's going to do is put both of those together into a chat history okay so let's see what that looks like first uh in a more messy way okay so you can see we have just the content right so it doesn't include the whole you know before we had human message we're not

28:23

include we're not seeing anything like that here instead we're just seeing the string so now let's switch back to print and we can see that what we have is our system message here it's just prefixed with this system and then we have human and it's prefixed by human and then it continues right so that's that's all it's doing it's just kind of merging

those in some sort of chat lug we could also put in like AI messages and they would appear in there as well okay so we have that now that is our

28:52

prompt template let's put that together with an LM to create what would be in past line ch be called an llm chain uh now we wouldn't necessarily call it an llm chain because we're not using the llm chain abstraction it's not super important if that doesn't make sense we we'll go into it in more detail later particularly in the in the ELO chapter so what this chain will do you think L chain is just chains where're chaining together these multiple components it will perform the STS

29:25

prompt formatting so that's what I just showed you LM generation so sending our prom to open AI getting a response and getting that output so you can also add another set here if you want to format that in a particular way we're going to be outputting that in a particular format so that we can feed it into the next set more easily but there are also things called output passes which pass your output in a more dynamic or complicated way depending on what you're doing so this is our first look at Elsa

29:58

don't want us to focus too much on the syntax here because we will be doing that later but I do want you to just understand what is actually happening here and logically what are we writing so all we really need to know right now is we Define our inputs with the first dictionary segment here right so this is a you know our inputs which we have defined already okay so if we come up to our user prompt here we said the input variable is our article right and we might have also added input variables to

30:35

the system prompt here as well in that case you know let's say we had your AI assistant called name right that helps generate article titles in this scenario we might have an input variables name here right and then what we would have to do down here is we would also have to pass that in right so also we would have article but we would also have name so basically we just need to make sure that in here we're including the variables that we have Define as input variables for our our first prompts okay so we can

31:19

actually go ahead and let's add that uh so we can see it's in action so we'll run this again and just include that or or reinitialize is our first prompt so we see that and if we just have a look at what that means for this format function here it means we'll also need to pass in a name okay and call it Joe okay so Joe the AI right so you are an AI system called

Joe now okay so we have Joe our AI that is going to be fed in through these input variables then we have this pipe operator the pipe

31:53

operator is basically saying whatever is to the left of the pipe operator which in this case would be this is going to go into whatever is on the right of the pipe operator it's that's simple again we'll we'll dive into this and kind of break it apart in the Elso chapter but for now that's all we need to know so this is going to go into our first prompt that is going to format everything it's going to add the name and the article that we provided into our first prompt then it's going to

32:21

Output that right going to Output that we have our P operate here so the output of this is going to go into the input of our Next Step it's our creative LM then that is going to generate some tokens it's going to generate our output that output is going to be an AI message and as you saw before if I take this bit out within those message objects we have this content field okay so we are actually going to extract the content field out from our AI message to just get the content and that is what we do

32:59

here so we get the AI message out from ilm and then we're extracting the content from that AI message object and we're going to passing it into a dictionary that just contains article title like so okay we don't need to do that we can just get the AI message directly I just want to show you how we are using this sort of chain in Elso so once we have set up our chain we then call it or execute it using the invoke method into that we will need to pass in those variables so we have our article

33:30

already but we also gave our AI a name now so let's add that and we'll ruin this okay so Joe has generated us a article title unlocking the future the rise of neuros symbolic AI agents cool much better name than what I gave the article which was AI agents are neuros symbolic systems no I don't think I did too bad okay so we have that now let's continue and what we're going to be doing is building more of these types of LM chain pipelines where we're feeding in some prompts we're generating

34:10

something getting something and and doing something with it so as mentioned we have the title we're now moving on to the description so to generate description so we have our human message prompt template so this is actually going to go into a similar format as before we also want to redefine this because I think I'm using the same system

message there so let's let's go ahead and do modify that or what we could also do is let's just remove the name now because I've showing that so what we could do is

34:46

you're an AI system that helps build good articles right build good articles and we could just use this as our you know generic system prompt now so let's say that's our new system prompt now we have our user prompt your task creating description for the article the article is here fure examine article here is the article title okay so we need the article title now as well in our input variables and then we're going to Output an AO friendly article description and we're just saying you

35:17

just to be certain here do not output anything other than the description so you know sometimes an LM might say hey look this is what I've generated for you the reason I think this is good is because so on and so on so on right if you're programmatically taking some output from an LM you don't want all of that fluff around what the LM is generated you just want exactly what you've asked it for okay because otherwise you need to pass out with code and it can get messy and also just far

35:44

less reliable so we're just saying do iput anything else then we're putting all these together so system prompt and the second user prompt this one here putting those together into a new chat prompt template and then we're going to to feed all that in to another LOL chain as we have here to well to generate our our description so let's go ahead we invoke that as before we're just make sure we add in the article title that we got from before and let's see what we get okay so we have this explore the

36:16

transformative potential of neuros symbolic Ai ageny and a little bit long to be honest but yeah you can see what it's doing here right and of course we could then go in we see this is kind of too long right a yeah SEO friendly description not not really so we can modify this output the SEO friendly description um make sure we don't exceed let me put that on a new line make sure we don't exceed say 200 characters or maybe it's even less to se I don't I don't have a clue I'll just say 120

36:51

characters I do not outly anything other than the description right so we could just you know go back modify our prompting see what that generates again okay so much shorter probably too short now but that's fine cool so we have that we have a summary process that and that's now you know in this dictionary form that we have here cool now

the third step we want to consume that first article variable with our full article and we're going to generate a few different output Fields so for this

37:24

we're going to be using the structured output feature so let's scroll down we'll see what that is what that looks like so structured output is essentially we're forcing their lmic like it has to Output a dictionary with these you know particular Fields okay and we can modify this quite a bit but in this scenario what I want to do is I want there to be an original paragraph right so I just want it to regenerate the original paragraph cuz I'm lazy and I don't want to extract it out then I want to get the

37:58

new edited paragraph This is the LM generated improved paragraph and then we want to get some feedback because we we don't want to just automate ourselves we want to augment ourselves and get better with AI rather than just being like I you do you do this so that's what we do here and you can see that here we're using this pantic object and what pantic allows us to do is Define these particular fields and it also allows us to assign these descriptions to a field and and line chain is actually going to

38:29

go ahead read all of this right even reads so for example we could put integer here and we could actually get a numeric score for our paragraph right we can try that right so let's uh let's let's just try that quickly I'll show you so numeric numeric score in fact let's even just ignore let's not put anything here so I'm going to put constructive feedback on the original paragraph but I just put into here so let's see what happens okay so we have that and what I'm going to do is I'm

39:00

going to get our creative llm I'm going to use this with structured output method and that's actually going to modify that llm class create a new llm class that forces that llm to use this structure for the output right so passing in paragraph into here using this we're creating this new structure LM so let's run that and see what happens okay so we're going to modify our chain accordingly maybe what I can do let's also just remove this bit for now so we can just see what the strictured

39:32

llm outputs directly and let's see okay so now you can see that we actually have that paragraph object right the one we defined up here which is kind of cool and then in there we have the original paragraph right so this is where this is coming from I definitely remember writing something that looks a lot like that so I think that is correct we have

the edited par so this is okay what thinks it's better and then interestingly the feedback is three which is weird right because uh here we said the constructive feedback

40:08

on the original paragraph but what we're doing when we use this with structured output but what Lang chain is doing is is essentially performing a tool core to open Ai and what a tool core can do is force a particular structure in the output of an LM so when we say feedback has to be an integer no matter what we put here it's going to give us an integer because how do you provide constructive feedback with an integer it doesn't really make sense but because we've set that limitation that

40:39

restriction here that is what it does it just gives us the uh a numeric value so I'm going to shift that to string and then let's rerun this see what we get okay we should now see that we actually do get constructive feedback all right so yeah you can see it's quite quite long so the original paragraph effectively communicates the limitations of neuro AI systems in performing certain tests however it could benefit from slightly improved Clarity and conciseness for example the phrase was

41:08

becoming clear can be made more direct by changing it to became evident yeah true thank you very much so yeah now we actually get that that feedback which is pretty nice now let's add in this final setep to our chain okay and it's just going to pull out our paragraph object here and extracting into a dictionary we don't necessarily need to do this honestly I actually kind of prefer it within this paragraph object but just so we can see how we would pass things on the other side of the chain okay so now we can see

41:44

we've extracted that out cool so we have all of that interesting feedback again but let's leave it there for the text part of this now let's have a look at at the sort of multimodal features that we can work with so this is you know maybe one of those things that kind of seems a bit more abstracted a little bit complicated where it maybe could be improved but you know we're not going to really be focusing too much on the M time modal stuff sub be focusing on language but I did want to just show you

42:16

very quickly so we want this article to look better okay we want to generate a prompt based on the article it's self that we can then pass to DAR the the image generation model from open AI that will then generate an image like like a thumbnail image for us okay so the first step of that is we're actually going to get an LM to generate that right so

we have our prompt that we're going to use for that so I'm say generate a prompt with less than 500 characters to uh generate an image based

42:52

on the following article okay so that's our prompt yeah super simple uh using the generic prompt template here you can use that you can use user uh prompt template it's up to you this is just like the generic prom template then what we're going to be doing is based on what this outputs we're then going to feed that in to this generate and display image function via the image prompt parameter that is going to use the darly API rapper from line chain it's going to run that image prompt and we're going to

43:25

get a a eurl out from that essentially and then we're going to read that using SK image here right so we're just going to read that image URL going to get the image data and then we're just going to display it okay so pretty straightforward now again this is a lell thing here that we're doing we have this runable Lambda thing when we're running functions within lell we need to wrap them within this runable Lambda I you know I don't want to go too much into what this is doing here because we do

43:59

cover in the L cell chapter but it's just you know all you really need to know is we have a custom function wrap in runable Lambda and then what we get from that we can use within this here right the the L Sal syntax so what are we doing here let's figure this out we are taking our original that image prom that we defined just up here right input variable to that is article okay we have our article d being input here feeding that into our prompt from there we get our message that we then feed into our

44:34

llm from the LM it's going to generate us a like an image prompt like a prompt for generating our image for this article we can even Let's uh let's print that out so that we can see what it generates because I'm also kind of curious okay so we'll just run that and then let's see it will feed in that content into our room reable which is basically this function here and we'll see what it generates okay don't expect anything amazing from darly it's not it's not the best to be honest but we at

45:09

least we see how to use it okay so we can see the prom that was used here create an image that visually represents the concept of neuros symbolic agents depict a futuristic interface where large D interacts with traditional code symbolizing integration of oh my gosh uh something computation include elements like a brain to represent neur

networks gears or circuits or symbolic logic and web of connections illustrating vast use cases of AI agents oh my gosh look at all that big prompt then we get this so you

45:46

know dar's interesting I would say we could even take this let's just see what that comes up with in something like mid Journey you can see these way cooler images that we get from just another image generation model far better but pretty cool honestly so in terms of Generation image the phrasing the The Prompt itself is actually pretty good the image you know could be better but that's it right so with all of that we've seen a little introduction to what we might build in with lighing chain so

46:19

that's it for our introduction chapter as I mentioned we don't want to go too much into what each of these things is doing just really want to focus on okay this is kind of how we're building something with line chain this is the overall flow uh but we don't really want to be focusing too much on okay what exactly lell is doing or what exactly uh you know this prompt thing is that we're setting up we're going to be focusing much more on all of those things and much more in the upcoming chapters so

46:54

for now we've just seen a little bit of what we can build before diving in in more detail okay so now we're going to take a look at AI observability using lsmith now lsmith is another piece of the broader Lang chain ecosystem its focus is on allowing us to see what our llms agents Etc are actually doing and it's something that we would definitely recommend using if you are going to be using line chain Lang graph now let's take a look at how we would set L Smith up which is incredibly simple so I'm

47:30

going to open this in collab and I'm just going to install the prerequisites here you'll see these are all the same as before but we now have the Lin Smith Library here as well now we are going to be using Lin Smith throughout the course so in all the following chapters we're going to be importing limith and that will be tracking everything we're doing but you don't need Lin Smith to go through the course it's an an optional dependency but as mentioned I would recommend it so we'll come down to here

47:58

and first thing that we will need is the line chain API key now we do need an API key but that does come with a reasonable free tier so we can see here they have each of the plans and this is the one that we are by default on so it's free for one user up to 5,000 traces per month if you're building out an application I think it's fairly easy to go beyond

that but it really depends on what you're building so it's a good place to start with and then of course you can upgrade as

48:33

required so we would go to smith. L chain.com and you can see here that this will log me in automatically I have all of these tracing projects these are all from me running the various chapters of the course yours if you do use l Smith throughout course your L Smith dashboard will end up looking something like this now what we need is an API key so we go over to settings we have API keys and we're just going to create an API key because we're just going through some personal learning right now I would go with

49:06

personal access token we can give a name or description if you want okay and we'll just copy that and then we come over to our notebook and we enter our API key there and that is all we actually need to do that's absolutely everything supposed the one thing to be aware of is that you should set your L chain project to whatever project you're working within so of course Within within the course we have individual project names for each chapter but for your own projects of course you should

49:33

make sure this is something that you recognize and is useful to you so L Smith actually does a lot without needing to do anything so we can actually go through let's just initialize our LM and start invoking it and seeing what L Smith returns to us so we'll need our open API key enter it here and then let's just invoke hello okay so nothing has changed on this end right so us running the code there's nothing different here however now if we go to Lang Smith I'm going to go back to my dashboard okay and you can

50:08

see that the the order of these projects just changed a little bit and that's because the most recently used project I this one at the top Lang chain course Lang Smith openai which is the current chapter we're in that was just triggered so I can go into here and I can see oh look at this so we actually have something in the Lang Smith UI and we didn't all we did was enter our L train apid that's all we did and we set some environment variables and that's it so we can actually click through to this

50:36

and it will give us more information so you can see what was the input what was the output and some other metadata here you see you know there's not that much in here however when we do the same for agents we'll get a lot more information so I can even show you a quick example from the future chapters if we come through to agents intro

here for example and we just take a look at one of these okay so we have this input and output but then on the left here we get all this information and the reason we

51:12

get all this information because agents are they're performing multiple LM calls etc etc so there's a lot more going on so we can see okay what was the first LM call and then we get these tool use traces we get another LM another rmm call another tool use and another LM call so you can see all this information which is incredibly useful and Incredibly easy to do because all I did when setting this up in that agent chapter was simply set the API key and the environment variables as we have

51:42

done just now so you get a lot out of very little effort with Lang Smith which is great so let's return to our Lang Smith project here and let's invoke some more now I've already shown you you know we're going to see a lot of things just by default but we can also add other things that Lang Smith wouldn't typically Trace so to do that we will just import a traceable decorator from Lang Smith and then let's make these just random functions traceable within limith okay so we'll run those we have

52:19

three here so we're going to generate a random number we're going to modify how long a function takes and also generate a random number and then in this one we're going to either return this no error or we're going to raise an error so we're going to see how limith handles these different scenarios so let's just iterate through and run those a few times so we're just going to run each one of those 10 times okay so let's see what happens so they're running let's go over to our Lin

52:54

sth UI and see what is happening over here so we can see that everything is updating we adding that information through and we can see if we go into a couple of these we can see a little more information so have the input and the output took three seconds see random error here in this scenario random error passed without any issues let me just refresh the page quickly okay so now we have the rest of that information and we can see that occasionally if there is an error from our random error function it is is

53:27

signified with this and we can see the traceback as well that was returned there which is useful okay so we can see if an error has been raised we have to see what that error is we can see the various latencies of these functions so you can see that varying throughout here we see all the inputs to each one of our functions and then of course

the outputs so we can see a lot in there which is pretty good now another thing that we can do do is we can actually filter so if we come to here we can add

54:01

a filter let's filter for errors that would be value error and then we just get all of the cases where one of our functions has returned or raise an error or value error specifically okay so that's useful and then yeah there's there's various other filters that we can add there so we could add a name for example if we want to look for the generate string delay function only we could also do that okay and then we can see the varying latencies of that function as well cool so we have

54:38

that now one final thing that we might want to do is maybe we want to make those function names a bit more descriptive or easy to search for for example and we can do that by saying the name of the traceable decorator like so so let's run that we'll run this a few times and then let's jump over to limith again going to limith project okay and you can see those coming through as well so then we could also search for those based on that new name so what was it chitchat maker like so and then we can see all

55:13

that information being streamed through to limith so that is our introduction to limith there is really not all that much to go through here it's very easy to sell up and as we scen it gives us a lot of observability into what we are building and we will be using this throughout the course we don't rely on it too much it's a completely optional dependency so you don't want to use l space you don't need to but it's there and I would recommend doing so so that's it for this chapter

55:43

we'll move on to the next one now we're going to move on to the chapter on prompts in Lang chain now prompts they seem like a simple concept and they are a simple concept but there's actually quite a lot to them when you start diving into them and they truly have been a very fundamental part of what has propelled us forwards from pre llm times to the current llm times you have to think until llms became widespread the way to fine-tune a AI model or ml model back then was to get loads of data for

56:24

your particular use case spend a load of training your specific Transformer or part of the Transformer to essentially adapt it for that particular task that could take a long time depending on the the task it could take you you know months or in some times if it was a simpler task it might take probably days potentially weeks now the interesting thing with

L LMS is that rather than needing to go through this whole fine-tuning process to to modify a model for one task over another task rather than doing that we

57:03

just prompt it differently we literally tell the model hey I want you to do this in this particular way and that is a you know that's a paradigm shift in what you're doing it's so much faster it's going to take you you know a couple of minutes rather than days weeks or months and LMS are incredibly powerful when it comes to just generalizing to you know across these many different tasks so prompts which control those instructions are a fundamental part of that now line chain naturally has many functionalities

57:38

around prompts and we can build very Dynamic prompting pipelines that modify the structure and content of what we're actually feeding into our llm depending on different variables different inputs and we'll see that in this chapter so we're going to work through prompting within the scope of of a rag example so let's start by just dissecting the various parts of a prompt that we might expect to see for a use case like rag so our typical prompt for rag or retrieval augmented generation will include rules

58:15

for the LM and this is this you will see in most prompts if not all this part of the promt sets up the behavior of the llm that is how it should be responding to user queries what sort of Personality it should be taking on what it should be focusing on when it is responding any particular rules or boundaries that we want to set and really what we're trying to do here is just to Simply provide as much information as possible to the llm about well what we're doing we just want to give the llm context as to the the

58:57

place that it finds itself in because an LM has no idea where it is it's just it's a it takes in some information and spits out information if the only information it receives is from the user you know user query it has you know doesn't know the context what is the application that it is within what is its objective what is its aim what are the boundaries all of this we need to just assume the llm has absolutely no idea about because it it truly does not so as much context as we can provide but

59:32

it's important that we don't overdo it it's uh we see this all the time people will over prompt an llm you want to be concise you don't want fluff and in general every single part of your prompt the more concise and less fluffy you can make it the better now those rules or instructions are typically in the system prompt of your llm now the

second one is context which is rag specific the context refers to some sort of external information that you are feeding into your llm we may have received this

01:00:06

information from like web search database query or quite often in this case of rag it's a vector database this external information that we provide is essentially the r retrieval augmentation of rag we are augmenting the knowledge of our llm which the the knowledge of our LM is contained within the llm model weights we're augmenting that knowledge with some external knowledge that's what we're doing here now for chat LMS this context is typically placed within a conversational context within the

01:00:47

user or assistant messages uh and with more recent models it can also be placed within uh tool messages as well then we have the question this pretty straightforward this is the query from the user this is or is this usually a user message of course there might be some additional formatting around this you might add a little bit of extra context or you might add some additional instructions if you find that you L them sometimes VAR off the rules that you've set within the system prompt you might you know append

01:01:25

or prefix something something here but for the most part it's probably just going to be the user's input and finally uh so these are all the inputs for our prompt here is going to be the output that we get so the answer from the assistant again I mean that's not even specific to rag it's just what you would expect in a in a chat llm or any LM and of course that would be an assistant message so putting all of that together in an actual prompt you can see everything we have here so we have the

01:01:56

uh rules for our prompt here the instructions we're just saying okay answer the question based on the context below if you cannot answer the question using the information answer with I don't know then we have some context here okay in this scenario that context that we're feeding in here because it's the first message we might putting that into the system prompt but that may also be turned around okay if you if you for example have an agent you might have your question up here before the context

01:02:27

and then that would be coming from a user message and then this context would follow the question and be recognized as a tool message it would be fed in that way as well kind of depends on on what sort of structure you're going for there but you can do either you can feed it into the system message if it's less conversational whereas if it's more

conversational you might feed it in as a tool message okay and then we have a user query which is here and then we'd have the AI answer okay and obviously

01:02:57

that would be generated here okay so let's switch across to the code we're in the L chain course repo notebooks 03 prompts and I'm just going to open this in collab okay let scroll down and we'll start just by installing the prerequisites okay so we just have the various libraries again as I mentioned before Lang Smith is optional you don't need to install it but if you would like to see your tracers and everything in Lang Smith then I would recommend doing that and if you are using L Smith you

01:03:27

will need to enter your API key here again if you're not using Lang Smith you don't need to enter anything here you just skip that cell okay cool and let's jump into the basic prompting then so we're going to start with this prompt answer used query based on the question below so we're just structuring what we just saw uh in code and we're going to be using the chat problem template because generally speaking we're using chat llms in most most cases nowadays so we have our chat

01:04:01

promp template and that is going to contain a list of messages system message to begin with which is just going to contain this and we're feeding in the the context within that there and we have our user query here okay so we'll run this and if we take a look uh here we haven't specified what our input variables are okay but we can see that we have query and we have context up here right so we can see that okay these are the input variables we just haven't explicitly defined them here so

01:04:40

let's just confirm with this that line chain did pick those up and we can see that it did so it has context and query as our input variables for the prompt template that we just defined okay so we can also see the structure of our temp plates let's have a look okay so we can see that within messages here we have a system message prompt template the way that we Define this you can see here that we have from messages and this will consume various uh different structures so you can see here that it has a from messages it is a

01:05:19

sequence of message like representation so we could pass in a system prompt template object and then a user prompt template object or we can just use a tupol like this and this actually defines okay this system this is a user and you could also do assistant or tool messages and stuff here as well using the same structure and then we

can look in here and of course that is being translated into the system message prom template and human message prompt template okay we have our input variables in there and

01:05:56

there and we have the template too okay now let's uh continue we'll see here what I what I just said so we're importing our system message prompt template and human message prompt template and you can see we're using the same from messages method here right and you can see it's so sequence of message like representation it's just you know what that actually means it can vary right so here we have system message prom template from template here from template query you know there's various

01:06:28

ways that you might want to do this it just depends on how explicit you want to be generally speaking I think for myself I would prefer that we stick with the objects themselves and be explicit but it is definitely a little harder to pass when you're when you're reading this so I understand why you might also prefer this it's definitely cleaner and it is it does look simpler so it just depends I suppose on preference okay so we can see again that this is exactly the same okay with chat

01:07:07

prom template and it contains this and this okay you probably want to see the exact output so it was messages okay exactly the same as what I output before cool so we have all that let's see how we would invoke our l L with these we're going to be using 40 mini again we do need our open API key so enter that and we'll just initialize our LM we are going with a low temperature here so less Randomness or less creativity and you in in many cases this is actually what I would be doing the

01:07:49

reason in this scenario that we're going with a low temperature is we're doing Rag and if you you remember before if we scroll up a little bit here our template says answer the user's query based on the context below if you cannot answer the question using the provided answer information answer with I don't know right so just from reading that we know that we want our llm to be as truthful and accurate as possible so a more creative llm is going to struggle with that and is more likely to

01:08:23

hallucinate whereas a low creativity or low temperature llm will probably stick with the rules a little better so again it depends on your use case you know if you're creative writing you might want to go with a higher temperature there but for things like rag where

the information being output should be accurate and truthful it's important I think that we keep temperature low okay I talk about that a little bit here so um of course lower temperature of zero makes the LM output more deterministic

01:08:57

which in theory should lead to less hallucination okay so we're going to go with L cell again here this is for those of you that use LINE chain pass this is equivalent to an llm chain object so our prompt template is being fed into our LM okay and from now we have this pipeline now let's see how we would use that pipeline so going to get some uh create some context here so so this just some Contex around orelio AI mention that we built sematic routers SM junkers there AI platform and development

01:09:40

services we mentioned I think we specifically outline this later on in the example so the Align chain experts little piece of information now most LMS would have not been trained on the recent internet so the fact that this came in September M 2024 is relatively recent so a lot of LMS out of the box you wouldn't expect them to know that so that is a good little bit of information to ask about so we invoke we have our query so what do we do and we have that context okay so we're feeding that into

01:10:14

that pipeline that we defined here all right so when we invoke that that is automatically going to take query and context and actually feed it into our prompt template okay if we want to we can also be a little more explicit so you you will probably see me doing this uh throughout the course because I do like to be explicit with everything to be honest and you'll probably see me doing this okay and this is doing the same thing or you'll see it will in a moment this is doing the exact same thing

01:10:57

again this is just a outo thing so all I'm doing in this scenario is I'm saying okay take from the dictionary query and then also take from that input dictionary the context key okay so this is doing the exact same thing uh the reason that we might want to write this is mainly for clarity to be honest just to explicit say okay these are the inputs because otherwise we don't really have them in the code other than within our original prompts up here which is not super clear so I think it's usually a good

01:11:40

idea to just be more expc with these things and of course if you decide you're going to modify things a little bit let's say you modify this to input down the line you can still feed in the same input here you're just you know mapping it between different Keys essentially or if you would like to just modify that I don't know you need to locase it on

the way in or something you can do so you have that I'll just redefine actually and we'll invoke again okay we see that it does the exact

01:12:14

same thing okay so R AI so this is the AI message just generated by the llm okay expertise in building AI agents several open source framework router AI platform okay right so they have everything there other than the line train experts thing it didn't mention that but we will yeah we'll test it later on that okay so on to Future prompting this is a specific prompting technique now many sort of State ofthe art or also to LMS are very good at instruction following so you'll find that fuch shop prompting is less common

01:12:55

now than it used to be at least for the sort of bigger more safy art models but when you start using smaller models not really what we can use here but let's say you're using a open source model like llama 3 or llama 2 which is much smaller you will probably need to consider things like f shot prompting although that being said with the open AI models you're at least the current open AI models this is not so important nonetheless it can be useful so the idea behind fuchia prompting is that you are

01:13:32

providing a few examples to your llm of how it should behave before you are actually going into the main part of the conversation so let's see how that would look so we create an example prom so we have our human in AI so human input AI response so we're basically saying up okay this with this type of input you should provide this type of output that's what we're doing here and we're just going to provide some examples okay so we have our input here's query one here is the

01:14:07

answer one right this is just I just want to show you how it works this is not what we'd actually feed into our LM then with both these examples and our example prompt we'd feed both of these into uh line chains few shot chat message prompt template okay and well you'll see what we get out of it okay so we basically get it formats everything and structures everything for us okay and using this of course it depends on let's say you see that your user is talking about a particular topic and you

01:14:45

would like to guide your llm to talk about that particular topic and a particular way right so you could identify that the user is talking about that topic either like a keyword match or a semantic similarity match and based on that you might want to modify these examples that you feed into your few sh chat message promp template and then

obviously for that could be what you do for topic a for topic B you might have another set of examples that you feed into this all all this time your example prompt is remaining the same but you're

01:15:16

you're just modifying the examples that are going in so that they're more relevant to whatever it is your user is actually talking about so that can be useful now let's see an example of that so when we are using a tiny LM It's ability would be limited although I think we are we're probably fine here we're going to say answer the US query based on the context below always answering mark down format you know being very specific the self system prompt okay that's nice but what we've

01:15:46

kind of said here is okay always answering mod down for I did do that but when doing so please provide headers short summary and follow bullet points then conclude okay so you see this here okay so we get this overview of already you have this and this it's actually quite good but if we come down here what I specifically want is to always follow this structure right so we have the double header for the topic summary header a couple of bullet points and then I always want to follow this pattern where it's like to conclude

01:16:24

always it's always bold you know I want to be very specific on what I want and to be you know fully honest with GT40 mini you can actually just prompt most of this in but for the sake of the example we're going to provide a few shot um examples in our few shot prompt examples instead to get this so we're going to provide one example here second example here and you'll see we're just following that same pattern we're just setting up the pattern that the llm should use so we're

01:16:58

going to set that up here we have our main header a little summary some subheaders bullet points subheader bullet points subheader bullet points to conclude so on and so on same with this one here okay and let's see what we got okay so this is the structure of our new F shop prompt template you can see what all this looks like let's come down and we're going to do we're basically going to insert that directly into our chat prompt template so we have for messages system prompt user prompt and then we have in

01:17:39

there these so let me actually show you very quickly right so we just have um this few shot chat to message prompt template which will be fed into the middle here run that and then feed all this back into our pipeline okay and this will you know modify the

structure so that we have that bold to conclude at the end here okay we can see nicely here so we get a bit more of that exact structure that we were getting again with GT40 models and many other opening air models you don't really need to do this but you

01:18:15

will see it in other examples we do have an example of this where we're using a llama and we're using I think llama 2 if I'm not wrong and you can see that adding this fuse shot promp template is actually a very good way of getting those smaller less capable models to follow your instructions so this is RAR when you're working those smaller lenss this can be super useful but even for so models like gp40 if you do find that you're struggling with the prompting it's just not quite following exactly what you

01:18:48

want it to do this is a very good technique for actually getting it to follow a very straight structure or behavior okay so moving on we have Chain of Thought prompting so this is a more common prompting technique that encourages the LM to think through its reasoning or its thoughts step by step so it's Chain of Thought the idea behind this is that okay in math class when you're a kid the teachers would always push you to put down your your working out right and there was a more reasons

01:19:25

for that one of them is to get you to think because they they know in a lot of cases actually you know you're a kid and you're in Aran you don't really care about this test and the you know they're just trying to get you to slow down a little bit and actually put down your reasoning and that kind of forc you to think oh actually I'm skipping a little bit in my head because I'm trying to just do everything up here if I write it down all of a sudden it's like oh actually I yeah I need to actually do

01:19:51

that slightly differently you you realize okay you're probably rushing now I'm not saying an LM is rushing but it's a similar effect by an LM writing everything down they tend to actually get things right more frequently and at the same time also similar to when you're a child and a teacher is reviewing your exam work by having the LM write down its reasoning you as a as a human or engineer you can see where the llm went wrong if it did go wrong which can be very useful when you're

01:20:22

trying to diagnose problems so with train of thought we should see uh less hallucinations and generally better performance now to implement train of thought in line chain there's no specific like line chain objects that do that instead it's it's just prompting okay so let's go down and just see how we might do that okay so be helpful

assistant and answer users question you must answer the question directly without any other text or explanation okay so that's our no Chain of Thought

01:20:51

system problems I will just note here especially with open AI again this is one of those things where you'll see it more with the smaller models most LMS are actually trained to use train thought prompting by default so we're actually specifically telling it here you must answer the question directly without any other text or explanation okay so we're actually kind of reverse prompting it to not use train of thought otherwise by default it actually will try and do that because it's been

01:21:18

trained to that's how that's how relevant Chain of Thought is okay so I'm going to say how many key strokes I need to type in type the numbers from 1 to 500 okay we set up our like llm chain Pipeline and we're going to just invoke our query and we'll see what we get total number of key strokes needed to type the numbers from one to 500 is 1,511 uh the actual an as I've written here is 1,392 without chain thought is hallucinating okay now let's go ahead and see okay with Chain of Thought

01:21:55

apprting what does it do so be helpful assistant and answer user question to answer the question you must list systematically and in precise detail all sub problems that are needed to be solved to answer the question solve each sub problem individually you have to shout at the LM sometimes to get them to listen and in sequence finally use everything you've worked through to provide the final answer okay so we're getting it we're forcing it to kind of go through the full problem there can

01:22:25

remove that not sure why that's there so run that again I don't know why we have context there I remove that and let's see you can see straight away that's taking a lot longer to generate output that's because it's generating so many more tokens so that's just one one drawback of this but let's see what we have so to determine how many keystrokes to tie those numbers we is breaking down several sub problems so count number of digits from 1 to 9 10 to 99 so so on and

01:22:58

count the digits in number 500 okay interesting so that's how it's breaking it up some more digits count in the previous steps so we go through total digits and we see that's okay nine digits for those for here 180 for here 1,200 and then of course three here so it gets all those sums those digits and actually comes to the right answer okay so that that

is you that's the difference with with Chain of Thought versus without so without it we just get the wrong answer basically guessing with

01:23:38

chain of thought we get the right answer just by the llm writing down its reasoning and breaking the problem down into multiple Parts which is I found that super interesting that it it does that so that's pretty cool now I will just see so as I as we mentioned before most llms nowadays are actually training to use train of thought prompting by default so let's just see if we don't mention anything right be a helpful assistant and answer the users question so we're not telling it not to think

01:24:08

through it's reasoning and we're not telling it to think through its reasoning let's just see what it does okay so you can see again it's actually doing the exact same reasoning okay it doesn't it doesn't give us like the sub problems that the start but it is going through and it's breaking everything apart okay which is quite interesting and we get the same correct answer so the formatting here is slightly different it's probably a little cleaner actually although I think

01:24:39

uh I don't know I here we get a lot more information so both are fine and in this scenario we actually do get the the right answer as well so you can see that that Chain of Thought prompting has actually been quite literally trained into the model and you'll see that with most well I think all save the-art lenss Okay cool so that is our our chapter on prompting again we're focusing very much on a lot of the fundamentals of prompting there and of course tying that back to the actual objects and methods

01:25:19

within langing but for now that's it for prompting and we'll move on to the next ch chapter in this chapter we're going to be taking a look at conversational memory in line chain we're going to be taking a look at the core like chat memory components that have really been in line chain since the start but are essentially no longer in the library and we'll be seeing how we actually Implement those historic conversational memory Utilities in the new versions of Lang chain so 0.3 now as a pre-warning

01:25:57

this chapter is fairly long but that is because conversational memory is just such a critical part of chatbots and agents conversational memory is what allows them to remember previous interactions and without it our chat boox and agents would just be responding to the most recent message without any understanding of previous interactions within a conversations so they would just not be coners ational and

depending on the type of conversation we might want to go with various approaches to how we remember those interactions

01:26:35

within a conversation now throughout this chapter we're going to be focusing on these for memory types we'll be referring to these and I'll be showing you actually how each one of these works but what we're really focusing on is rewriting these for the latest version of Lang chain using the what it's called the runable with message history so we're going to be essentially taking a look at the original implementations for each of these four original memory types and then we'll be rewriting them with the

01:27:12

the runnable memory history class so just taking a look at each of these four very quickly conversational buffer memory is I think the simplest and most intuitive of these meor types it is literally just you have your messages they come into this object they are sold in this object as essentially a list and when you need them again it will return them to you there's nothing and nothing else to it's super simple the conversation Buffet window memory okay so new word in the middle of the window

01:27:49

this works in pretty much the same way but those messages that it has stored is not going to return all of them for you instead it's just going to return the most recent let's say the most recent three for example okay and that is defined by a parameter K conation of summary memory rather than keeping track of the entire uh interaction memory directly what it's doing is as those interactions come in it's actually going to take them and it's going to compress them into a smaller little summary of

01:28:21

what has been within that conversation and as every new interaction is coming in it's going to do that going to keep iterating on that summary and then that is going to be return to us when we need it and finally we have the conversational summary buffer memory so this is it's taking so the buffer part of this is actually referring to very similar thing to the buffer window memory but rather than it being a you know most K messages it's looking at the number of tokens within your memory and

01:28:53

it's returning the most recent K tokens that's what the buffer part is there and then it's also merging that with the summary memory here so essentially what you're getting is almost like a list of the most recent messages based on the token length rather than the number of interactions plus a summary which would you know come at the the top here

so you get kind of both the idea is that obviously this summary here would maintain all of your interactions in a very compressed form so you're you're losing

01:29:30

less information and you're still maintaining you know maybe the very first interaction the user might have introduced themselves giving you their name hopefully that would be maintained within the summary and it would not be lost and then you have almost like higher resolution on the most recent um K or k tokens from your memory okay so let's jump over to the code we're going into the 04 chat memory notebook open that in collab okay now here we are let's go ahead and install the

01:30:01

prerequisites run all we again can or cannot use align Smith it is up to you enter that and let's come down and start so first just initialize our LM using 40 mini in this example again low temperature and we're going to start with conversation buffer memory right so this is the original version of this uh memory type so let me where are we we're here so memory conversation both of memory and we're returning messages that needs to be set to true so the reason that we set return messages true it it

01:30:44

mentions up here is if you do not do this it's going to returning your chat history as a string to an llm whereas well chat lm's nowadays would expect message objects so yeah you just want to be returning these as messages rather than as strings okay otherwise yeah you're going to get some kind of strange Behavior out from your llms if you return them strings so you do want to make sure that it's true I think by default it might not be true but this is coming this is deprecated right it does

01:31:18

tell you here as de creation warning this is coming from older BL chain but it's a good place to start just to understand this and then we're going to rewrite this with the runnables which is the recommended way of doing so nowadays okay so adding messages to our memory we're going to write this okay so it's just a just a conversation user AI user AI so on and so on random chat main things to not here is I do provide my name we have the the model's name right towards the start of those interactions

01:31:50

okay so I'm just going to add all of those with do it like this okay then we can just see we can load our history like so so let's just see what we have there okay so we have human message AI message human message right this is exactly what we I showed you just here it's just in that message format from line chain okay so we can do that alternatively

we can actually do this so we can get our memory we initialize the conation buffer memory as we did before and we can actually add it directly the

01:32:26

message into our memory like that so we can use this add us message add AI message so on and so on load again and it's going to give us the exact same thing again there's multiple ways to do the same thing cool so we have that to pass all of this into our LM again this is all deprecated so we're going to learn how to properly in a moment but this is how L chain was doing in the past so to pass all of this into our LM we' be using this conversation chain right again this is deprecated nowadays

01:32:59

we would be using lell for this so I I just want to show you okay how this would all go together and then we would invoke okay what is my name again let's run that and we'll see what we get it's remembering everything remember so this conversation buffer memory it doesn't drop messages it just remembers everything right and honestly with the sort of high context Windows of many LMS might be what you do it depends on how long you expect the conversation to go on for but you could you probably in

01:33:29

most cases would get away with this okay so what let's see what we get um I say what is my name again okay let's see what it gives me says your name is chains great thank you that works now as I mentioned all of this that I just showed you is actually deprecated that's the old way doing things let's see how we actually do this in modern or up toate L chain so we're going to be using this runable with message history to implement that we will need to use LL and for that we will need to just Define

01:34:03

prompt templates our LM as we usually would okay so we're going to set up our system prompt which is just a helpful assist called Zeta okay we're going to put in this messages placeholder okay so that's important essentially that is where our messages are coming from our conversation Buffet for memory is going to be inserted right so it's going to be that chat history is going to be inserted after our system prompt but before our most recent query which is going to be inserted last here

01:34:36

okay so messages placeholder item that's important and we use that throughout the course as well so we use it both for chat history and we'll see later on we also use it for the intermediate thoughts that a agent would go through as well so important to remember that little thing well link our prompt template to our LM again if we would like

we could also add in the I think we only have the query here oh we would probably also want our history as well but I'm not going to do that right now

01:35:10

okay so we have our Pipeline and we can go ahead and actually Define our runnable with message history now this class or object when we are initializing it does require a few items we can see them here okay so we' see that we have our Pipeline with history so it's basically going to be uh you can you can see here right we have that history messages key right this here has to align with what we provided as a meses placeholder in our pipeline right so we have our pipeline prompt template here

01:35:44

and here right so that's where it's coming from it's coming from messes placeholder variable name is history right that's important that links to this then for the input messages key here we have query that again links to this okay so both important to have that the other thing that is important is obviously we're passing in that pipeline from before but then we also have this get session history basically what this is doing is it's saying okay I need to get uh the list of messages that

01:36:16

make up my chat history that are going to be inserted into this variable so that is a function that we Define okay and with within this function what we're trying to do here is actually replicate what we have with the previous conversation buffer memory okay so that's what we're doing here so it's very simple right so we have this in memory chat message history okay so that's just the object that we're going to be returning what this will do is it will set up a session ID the session ID

01:36:49

is essentially like a unique identifier so that eachers ation or interaction within a single conversation is being mapped to a specific conversation so you don't have overlapping let say have multiple users using the same system you want to have a unique session ID for each one of those okay and what it's doing is saying okay if session ID is not in the chat map which is this empty dictionary we defined here we are going to initialize that session with an inmemory chat message history okay

01:37:21

that's it and we return okay and all that's going to do is it's going to basically append our messages they will be appended within this chat map session ID and they're going to get returned there's nothing R there's nothing else to it to be honest so we invoke our rable let's see what we get I need to ruin this okay note that we do have this config so we have

a session ID that's to again as I mentioned keep different conversations separate Okay so we've run that now let's run a few more so what is

01:37:58

my name again let's see if it remembers your name is James how can I help you today James okay so it's what we've just done there is literally conversation buter memory but for upto-date L chain with L cell with Runner BS so you the recommended way of doing it nowadays so that's a very simple example okay really and not that much to it it gets a little more complicated as we start thinking about the different types of memory although that being said it's not massively complicated we're only rarely

01:38:35

going to be changing the way that we're getting our interactions so let's uh let's dive into that and see how we will do something similar with the conation buff for window memory but first let's actually just understand okay what is Con station buffer window memory so as I mentioned near the start it's going to keep track of the last K messages so there's a few things to keep in mind here more messages does mean more tokens send with each request and if we have more tokens in each request it means

01:39:05

that we're increasing the latency of our responses and also the cost so with the previous memory type we're just sending everything and because we're sending everything that is going to be increasing our cost it's going to be increasing our latency for every message especially as a conversation gets longer and longer and we don't we might not necessarily want to do that so with this conversation buffer window memory we're going to just say okay just return me the most recent messages okay so let's

01:39:35

well let's see how that would work here we're going to return the most recent four messages okay we are again make sure we've turned messages is set to True again this is deprecated this is just the old way of doing it in a moment we'll see the updated way of doing this we'll add all of our messages okay so we have this and just see here right so we've added in all these messages there's more than four messages here and we can actually see that here so we have human message AI

01:40:08

human AI human AI human AI right so we've got four pairs of human AI interactions there but here we don't have there's more than four pairs so four pairs will take us back all the way to here I'm researching different types of conversational uh memory okay and if we take a look here the most the first message we have is I'm researching different

types of conversational memory so it's cut off these two here which will be a bit problematic when we ask you what our name is okay so let's just

01:40:40

see going to be using conversation chain object again again just remember that is deprecated and I want to say what is my name again let's see let's see what it says uh I'm sorry I but I don't have access to your name or any personal information if you like you can tell me your name right so it doesn't actually remember uh so that's kind of like a negative of the conversation Buffet window memory of course the uh to fix that in this scenario we might just want to increase K maybe we say remember the

01:41:12

previous eight interaction Pairs and it will actually remember so what is my name again your name is James so now it remembers we've just modified how much it is remembering but of course you know pros and cons to this it really depends on what you're trying to build so let's take a look at how we would actually implement this with the runable with message history okay so you getting a little more complicated here although it it's it's not it's not complicated but well we'll see okay so we have buffer window

01:41:47

message history we're creating a class here this class is going to inherit from the base chat message history object from line chain okay and in all of our other message history objects can do the same thing before with the inmemory message object that was basically replicating the buffer memory so we didn't actually need to do anything we didn't need to Define our own class here so in this case we do so we follow the same pattern that line chain follows with this base chat message history and

01:42:22

you can see a few of the functions here that are important so add messages and clear are the ones that we're going to be focusing on we also need to have messages which this object attribute here okay so we're just implementing the synchronous methods here if we want this to be async if we want to support async we would have to add a add messages um a get messages and a clay as well so let's go ahead and do that we have messages we have K again we're looking at remembering the top K messages or most

01:42:54

recent K messages only so it's important that we have that variable we are adding messages through this class this is going to be used by line chain within our runnable so we need to make sure that we do have this method and all we're going to be doing is extending the self messages uh list here and then we're actually just going to be

trimming that down so that we're not remembering anything beyond those you know most recent K messages that we have set from here and then we also have the clear method

01:43:26

as well so we need to include that that's just going to clear the history okay so it's not this isn't complicated right it just gives us this nice default standard interface for message history and we just need to make sure we're following that pattern okay I've included the uh this print here just so we can see what's happening okay so we have that and now for that get chat history function that we defined earlier rather than using the buin method we're going to be using our own object which

01:43:57

is a buffer window message history which will be defined just here okay so if session ID is not in the chat map as we did before we're going to be initializing our buffer window message history we're setting K up here with a default value of four and then we just return it okay and and that is it so let's run this we have our runable with message history we have all of these variables which are exactly the same as before four but then we also have these variables here with it's history Factory

01:44:28

config and this is where if we have um new variables that we've added to our message history in this case k that we have down here we need to provide that to line train and sell it this is a new configurable field okay and we've also added it for the session ID here as well so we're just being explicit and have everything in that so we have that and we run okay now let's go ahead and invoke and see what we get okay so important here this history Factory config that is kind of being fed through

01:45:06

into our invoke so that we can actually modify those variables from here okay so we have config configurable session ID okay we just put whatever we want in here and then we also have the number K okay so remember the previous four interaction I think in this one we're doing something slightly different I think we're remembering the four interactions rather than the previous four interaction pairs okay so my name is James uh we're going to go through I'm just going to actually clear this

01:45:36

and now I'm going to start again and we're going to use the exact same ad user message ad AI message that we used before we're just manually inserting all that into our history so that we can then just see okay what is the result and you can see that k equal 4 is actually unlike before where we were having the uh saving the top four interaction pairs

we now saving the most recent four interactions not pairs just interactions and honestly I just think that's clearer I think it's weird that

01:46:09

the number four for K would actually save the most recent eight messages right I I think that's odd so I'm just not replicating that weirdness we could if we wanted to I just don't like it so I'm not doing that and anyway we can see from messages that we're returning just the most four recent messages okay which should be these four Okay cool so we've just using the runable we've replicated the old way of having a window memory and okay I'm going to say what is my

01:46:44

name again as before it's not going to remember so we can come to here I'm sorry about I don't have access to personal information so on and so on if you like to tell me your name doesn't know now let's try a new one where we initialize a new session okay so we're going with ID K4 so that's going to create a new conversation there and we're going to say we're going to set K to 14 okay great I'm going to manually insert the other uh messages as we did before okay and we can see all of those

01:47:16

and see at the top here we are still maintaining that hi my name is James message now let's see if it remembers my name your name is James okay there we go cool so that is working we can also see so we just added this what is my name again let's just see if did that get added to our list of messages right what is my name again nice and then we also have the response your name is James so just by invoking this because we're using the the runable with message history it's just automatically adding

01:47:49

all of that into our message history which is nice cool all right so that is the buffer window memory now we are going to take a look at how we might do something a little more complicated which is the the summaries okay so when you think about the summary you know what are we doing we're actually taking the messages we're using that LM call to summarize them to compress them and then we're storing them within messages so let's see how we would actually uh do that so to start with

01:48:24

let's just see how it was done in Old Line chain so we have conversation summary memory go through that and let's just see what we get so again same interactions right I'm just invoking invoking invoking I'm not adding these directly to the messages because it actually needs to go through a um like that summarization process and if we have a look we can see it happening okay current conversation so sorry current

conversation hello there my name is James AI is generating current conversation the human introduces

01:49:02

himself as James AI greets James warmly and expresses its Readiness to chat and assist inquiring about how his day is going right so it's summarizing the the previous interactions and then we have you know after that summary we have the most recent human message and then the AI is going to generate its response okay and that continues your own Contin is going and you see that the the final summary here is going to be a lot longer okay it's different that first summary of course asking about his Day Men

01:49:31

researching different types of conversational memory the AI responds enthusiastically explaining that conversational memory includes short-term memory longterm memory contextual memory personalized memory and then inquires if James is focused on a specific type of memory Okay cool so we get essentially the summary is just getting uh longer and longer as we go but at some point the idea is that it's not going to keep growing and it should actually be shorter than if you were saving every single interaction whilst

01:49:59

maintaining as much all the information as possible but of course you're not going to maintain all of the information that you would with for example the the buffer memory right with the summary you are going to lose information but hopefully less information than if you're just cutting interactions so you're trying to reduce your token count whilst maintaining as much information as possible now let's go and ask what is my name again it should be able to answer because we can see in the summary here

01:50:34

that I introduced myself as James okay respondents your name is James how is your research going okay so has that cool let's see how we'd Implement that so again as before we're going to go with that conversation summary message history we're going to be importing this system message uh we're going to be using that not for the LM that we're chatting with but for the LM that will be generating our summary so actually that is not quite correct there is create a summary not that it

01:51:06

matters it's just the doct string so we have our messages and we also have the LM so different different attribute here to what we had before when we initialize a conversation summary message history we need to passing in our LM we have the same methods as

before we have ADD messages and clear and what we're doing is as messages coming we extend with our current messages but then we're modifying those okay so we construct our like instructions to make a summary okay so that is here we have the system front uh

01:51:40

giving the existing conversation summary and the new messages generate a new summary of the conversation ensuring to maintain as much relevant information as possible okay then we have a human message here through that we're passing the existing summary okay and then we're passing in the new messages Okay cool so we format those invoke the llm here and then what we're doing is in the messages we're actually replacing the existing history that we had before with a new history which is just a

01:52:16

single system summary message okay let's see what we get as before we have that get chat history exactly the same as before the only real difference is that we're passing in the llm parameter here and of course as we're passing in the LM parameter in here it does also mean that we're going to have to include that in the configurable field spec and that we're going to need to include that when we're invoking our pipeline okay so we run that pass in the LM now of course one side effect of

01:52:51

generating summaries for everything is that way actually you know we're generating more so you are actually using quite a lot of tokens whether or not you are saving tokens or not actually depends on the length of a conversation as the conversation gets longer if you're storing everything after a little while that the token usage is actually going to increase so if in your use case you expect to have shorter conversations you would be saving money and tokens by just using this standard buffer memory

01:53:23

whereas if you're expecting very long conversations you would be saving tokens and money by using the summary history okay so let's see what we got from there we have a summary of the conversation James introduced himself by saying hi name James a I responded War asking hi James Interac include details about token usage okay so we actually included everything here which we probably should not have done why did we do that as so in here we're including all of the out in here so we using or including

01:54:02

all of the content from the messages so I think maybe we just do X content for X in messages that should resolve that okay there we go so we quickly fli that so yeah before

we pass them in the entire mage object which obviously includes all of this information whereas actually we just want to be passing into the content so we modified that and now we're getting what we would expect okay cool and then we can keep going right so as we as we keep going the summary should get more like abstract like as we just saw here is

01:54:44

literally just giving us the messages directly almost okay so we're getting a bit of summary there and we can keep going we're going to add just more messages to that we'll see the you as we'll get send those we'll get a response send it again get a response and we just adding all of that invoking all of that and that will be of course adding everything into our message history Okay cool so we've run that let's see what the latest summary is okay and then we have this so this is

01:55:16

a summary that we have inside of our our chat history okay cool now finally let's see what is my name again we can just double check you know it has my name in there so it should be able to tell us okay cool so your name is James pretty interesting so let's have a quick look over at limith so the reason I want to do this is just to point out okay the different essentially token usage that we're getting with each one of these okay so we can see that we have these Runner mess history which probably uh

01:55:53

improved in naming there but we can see okay how long is each one of these taken how many tokens are they also using come back to here we have this runable message history this is we'll go through a few of these maybe to here I think we can see here this is that first interaction where we're using the buffer memory and we can see how many tokens we used here so 112 tokens when we're asking what is my name again okay then we modified this to include I think it was like 14 interactions or something on

01:56:30

those lines obviously increases the number of tokens that we're using right so we can could see that actually happening all in Lang which is quite nice and we can compare okay how many tokens is each one of these using now this is looking at the buffer window and then if we come down to here and look at this one so this is using our summary okay so our summary with what is my name again actually use more tokens in this scenario right which is interesting because we're trying to compress

01:56:57

information the reason there more is because there's not there hasn't been that many interactions as the conversation length increases with the summary this total number of

tokens especially if we prompt it correctly to keep that low that should remain relatively small whereas with the buffer memory that will just keep increasing and increasing as the as the conversation gets longer so useful little way of using Lang Smith there to just kind of figure out okay in terms of tokens and costs of what we're

01:57:31

looking at for each of these memory types okay so our final memory type acts as a mix of the summary memory and the buffer memory so what it's going to do is keep the buffer up until an N number of tokens and then once a message exceeds the N number of tokens limit for the buffer it is actually going to be added into our summary so this memory has the benefit of remembering in detail the most recent interactions whilst also not having the limitation of using too many tokens as a conversation gets

01:58:13

longer and even potentially exceeding context Windows if you try super hard so this is a very interesting approach now as before let's try the original way of implementing this then we will go ahead and use our update method for implementing this so we come down to here and we're going to do L chain memory import conversation summary buffer memory okay a few things here LM for summary we have the N number of tokens that we can keep before they get added to the summary and then return messages of course okay you can see

01:58:51

again this is dicated we use the conversation chain and then we just passing our memory there and then we can chat okay so super straightforward first message we'll add a few more here and we have to invoke because how memory type here is using NM to create those summaries as it goes and let's see what they look like okay so we can see for the first message here we have human message and then an AI message then we come a little bit lower down again it's same thing human message is the first thing in our history here then

01:59:29

it's a system message so this is at the point where we've exceeded that 300 token limit and the memory type here is generating those summaries so that summary comes in as a system message and we can see okay the human named James introduces himself and mentions he's researching different types of conversational memory and so on and so on right okay cool so we have that then let's come down a little bit further we can see okay so the summary there okay so that's what we that's what we have

02:00:01

that is the implementation for the old version of this memory again we can see it's deprecated so how do we implement this for our more recent versions of Lang chain and specifically 0.3 well again we're using that runable message history and it looks a little more complicated than we were getting before but it's actually just you know it's nothing too complex we're just creating a summary as we did with the previous memory type but the decision for adding to that summary is based on in this case

02:00:39

actually the number of messages so I didn't go with the the Lang chain version where it's a number of tokens I don't like that I prefer to go with messages so what I'm doing is saying okay let K messages okay once we exceed K messages the messages beyond that are going to be added to the memory Okay cool so let's see we first initialize our conversation summary buffer message history class with llm and K okay so these two here so LM of course to create summaries and K is just the the limit of the number of

02:01:18

messages that we want to keep before adding them to the summary or dropping them from now messages and adding them to the summary okay so we will begin with okay do we have an existing summary so the reason we set this in none is we can't extract the summary the existing summary unless it already exists and the only way we can do that is by checking okay do we have any messages if yes we want to check if within those messages we have a system message because we're we're doing the same structure is what

02:01:53

we have up here where the system message that first system message is actually our summary so that's what we're doing here we're checking if there is a summary message already stored within our messages okay so we're checking for that if we find it we'll just do we have this little print statement so we can see that we found something and then we just make our existing summary I should actually move this to the first instance here yeah okay so that existing summary will be set to the first

02:02:30

message okay and this would be a system message rather than a string cool so we have that then we want to add any new messages to our history okay so we're extending the history there and then we're saying okay if the length of our history is exceeds the K value that we set we're going say okay we found that many messages we're going to be dropping the latest it's going to be the latest two messages this I will say here one thing or one problem with this is that we're not going to be saving that many tokens

02:03:05

if we're summarizing every two messages so what I would probably do is in in an actual like production setting I would probably say let's go up to 20 messages and once we hit 20 messages let's take the previous 10 we're going to summarize them and put them into our summary alongside any you know previous summary that already existed but in in you know this is also fine as well okay so we say we found those mes we're going to drop the latest two messages okay so we pull the the oldest messages out I should say

02:03:46

not the latest it's the oldest not the latest I want to keep the latest drop the oldest so we pull out the oldest messages and keep only the most recent messages okay then I'm saying okay if we if we don't have any old messages to summarize we don't do anything we just return okay so this in the case that this has not been triggered we would hit this but in the case this has been triggered and we do have old messages we're going to come to here okay okay so this is we can see

02:04:25

have a system message prompt template saying giving the existing conversation summary and the new messages generate a new summary of the conversation ensuring to maintain as much relevant information as possible so if you want to be more conservative with tokens we could modify this prompt here to say keep the summary to within the length of a single paragraph for example and then we have our human M prom template which is going to say okay here's the existing conversation summary in here on new

02:04:52

messages now new messages here is actually the old messages but the way that we're framing it to the llm here is that we want to summarize the whole conversation right it doesn't need to have the most recent messages that we're storing within our buffer it doesn't need to know about those that's irrelevant to the summary so we just tell it that we have these Zoom mes and as far as this LM is concerned this is like the full set of interactions okay so then we would format those and invoke

02:05:22

our LM and then we'll print out our new summary so we can see what's going on there and we would prend that new summary to our conversation history okay and and this will work so we can just prend it like this because we've already popped where was it up here if we have an existing summary we already pop that from the list it's already been pulled out of that list so it's okay for us to just we don't need to say like we don't need to do this because we've already dropped

02:05:58

that initial system message if it existed okay and then we have the clear method as before so that's all of the logic for our conversational summary buffer memory we redefine our get chat history function with the LM and K parameters there and then we'll also want to set the configurable Fields again so that is just going to be of course session ID LM and K okay so now we can invoke the K value to begin with is going to be four okay so we can see no old messages to update summary with it's good let's

02:06:42

invoke this a few times and let's see what we get okay so now M to summary with found six messages dropping the aest 2 and then we have new summary in the conversation James Inu himself and first is interestes in researching different types of conversational memory right so you can see there's quite a lot in here at the moment so we would definitely want to prompt the LM the summary LM to keep that short otherwise we're just getting a ton of stuff right but we can see that that is you know it's it's working it's

02:07:19

functional so let's go back and see if we can prompt it to be a little more concise so we come to here ensuring to maintain as much relevant information as possible however we need to keep our summary concise the limit is a single short paragraph okay something like this let's try and let's see what we get with that okay so message one again nothing to update see this so new summary you can see it's a bit shorter it doesn't have all those bullet points okay so that seems better let's

02:08:06

see so you can see the first summary is a bit shorter but then as soon as we get to the second and third summaries the second summary is actually slightly longer than the third one okay so we're going to be we're going to be losing a bit of information in this case more than we were before but we're saving a ton of tokens so that's of course a good thing and of course we could keep going and adding many interactions here and we should see that this conversation summary will be it should maintain that

02:08:37

sort of length of around one short paragraph So that is it for this chapter on conation memory we've seen a few different memory types we've implemented their old deprecated version so we can see what they were like and then we've reimplemented them for the latest versions of Lang chain and to be honest using logic where we are getting much more into the weees and that is in some ways okay it complicates things that is true but in other ways it gives us a ton of control so we can modify those memory

02:09:13

types as we did with that final summary buffer memory type we can modify those to our liking which is incredibly useful when you're actually building applications for the real world so that is it for this chapter we'll move on to the next one in this chapter we are going to introduce agents now agents I think are one of the most important components in the world of AI and I don't see that going away anytime soon I think the majority of AI applications the intelligent part of those will be was always an

02:09:52

implementation of an AI agent or mle AI agents so in this chapter we are just going to introduce agents within the context of line chain we're going to keep it relatively simple we're going to go into much more depth in agents in the next chapter where we'll do a bit of a deep dive but we'll focus on just introducing the Core Concepts and of course agents within line chain here so jumping thing straight into our notebook let's run our prerequisites you'll see that we do have

02:10:28

an additional prerequisite here which is Google search results that's because we're going to be using the sub API to allow our llm as an agent to search a web which is one of the great things about agents is that they can do all of these additional things and LM by itself obviously cannot so we come down to here we have our lsmith parameters again of course so you enter your Lang chain API if you have one and now we're going to take a look at tools which is a very essential part of Agents so tools are a

02:11:04

way for us to augment our llms with essentially anything that we can write in code so we mentioned that that we're going to have a Google Search tool that Google Search tool it's some code that gets executed by our llm in order to search Google and get some results so a tool can be thought of as any code logic or any function in the C in the case of python any function that has been formatted in a way so that our LM can understand how to use it and then actually use it although the the LM

02:11:41

itself is not using the tool it's more our agent execution logic which uses the tool for the llm so we're going to go ahead and actually create a few simple tools we're going to be using what is called the tool decorator from Lang chain and there are a few things to keep in mind when we're building tools so for Optimal Performance our tool needs to be just very readable and what I mean by readable is we need three main things one is a DOT string that is written in natural language and it is going to be

02:12:14

used to explain to the Alm when and why and how it should use this tool we should also have clear parameter names those parameter names should tell the llm okay what each one of these parameters are they should be self-explanatory if they are not self-explanatory we should be including an explanation for those parameters within the doc string then finally we should have type annotations for both our parameters and also what we're returning from the tool so let's jump in and see how we would Implement all of

02:12:51

that so we come down to here and we have line chain core tools import tool okay so these are just four incredibly simple tools we have the addition or add tool multiply the exponentiate and the subtract tools okay so a few calculator S tools now when we add this tool decorator it is turning each of these tools into what we call a structured tool object so we can see that here we can see we have this structured tool we have a name description okay and then we have this Al schema we'll see this in a moment and a function right so

02:13:32

this function is literally just the original function it's it's a mapping to the original function so in this case it it's the add function now the description we can see is coming from our doc string and of course the name as well is just coming from the function name okay and then we can also see let's just print the name and description but then we can also see the ARs schema right we can so this thing here that we can't read at the moment to read it we're just going to look at the

02:14:04

model Json schema method and then we can see what that contains which is all of this information so this actually contains everything includes properties so we have the X it C or title for that and it also specifies the type okay so the type that we Define is float float for open AI gets mapped to number rather than just being float and then we also see that we have this required field so this is telling how LM which parameters are required which ones are optional so we yeah in some cases you would we can

02:14:40

even do that here let's do Z that is going to be float or none okay and we just going to say it is 0.3 all right well I'm going to remove this in a minute because it's kind of weird but let's just see what that looks like so you see that we now have X Y and Z but then in Z we have some additional information okay so it can be any of it can be a number or it can just be nothing the default value for that is 0.3 okay and then if we look here we can see that the required field does not

02:15:18

include Z so it's just X and Y so it's describing the full function schema for us but let's remove that okay and we can see that again with our exponentiate tool similar thing okay so how how are we going to invoke our tool so the llm the underlying LM is actually going to generate a string okay so we'll look something like this this is going to be our llm output so it is it's a string that is some Json and of course to load a string into a dictionary format we just use Json loes

02:16:01

okay so let's see that so this could be the output Fromm we load it into a dictionary and then we get an actual dictionary and then what we would do is we can take our exponentiate uh tool we access the underlying function and then we pass it the keyword arguments from our diction here okay and that will execute our tool that is the tool execution log you that line chain implements and then later on in the next chapter we'll be implementing ourselves cool so let's move on to creating an agent now we're going to be

02:16:39

constructing a simple tool calling agent we're going to be using Lang chain expression language to do this now we will be covering Lang chain expression language or also more in a upcoming chapter but for now all we need to know is that our agent will be constructed using syntax and components that like this so we would start with our input parameters that is going to include our user query and of course the chat history because we need our agent to be conversational and remember previous interactions within the conversation

02:17:13

these input parameters will also include a placeholder for what we call the agent scratch Pad now the agent scratch Pad is essentially where we are storing the internal thoughts or the internal dialogue of the agent as it is using tools getting observations from those tools and working through those multiple internal steps so in the case that we will see it will be using for example the addition tool getting the result using the multiply tool getting the result and then providing a final answer

02:17:43

to us as a user so let's jump in and see what it looks like okay so we'll just start with defining our prompt so our prompt is going to include the system message there nothing we're not putting anything special in there we're going to include the chat history which is a messages placeholder then we include our human message and then we include a placeholder for the agent scratch Pad now the way that we implement this later is going to be slightly different for the scratch Pad we actually use this

02:18:14

message's placeholder but this is how we use it with the built-in create tool agent from BL chain next we sign our LM we do need our Opening Our API key for that so we'll enter that here like so okay so come down Okay so we're going to be creating this agent we need conversation memory and we are going to use the older conversation buffer memory class rather than the newer renable with message history class that's just because we're also using this older create tool calling agent and this is

02:18:46

this is the older way of doing things in the next chapter we are going to be using the more recent basically what we already learned on chat history we're going to be using all of that to implement our chat history but for now we're going to be using the older method uh which is deprecated just as a pre-warning but again as I mentioned at the very solid of course we're starting abstract and then we're getting into the details so we're going to initialize our agent for that we need these four things

02:19:18

LM as we defined tools as we have defined prompt as we have defined and then the memory which is our old conversation buffer memory so with all of that we are going to go ahead and we create a tool calling agent and then we just provide it with everything okay there we go now H you'll see here I didn't pass in the the memory I'm passing it in down here instead so we're going to start with this question which is what is 10.7 MTI 7.68 eight okay so given the Precision of these numbers our l a normal LM would

02:19:58

not be able to answer that or almost definitely will not be able to answer that correctly we need a external tool to answer that accurately and we'll see that that is exactly what it's going to do so we can see that the tool agent action message here we see that it decided okay I'm going to use the multiply tool and here at parameters that I want to use for that tool okay we can see X is 10.7 and Y is 7.68 you can see here that this is already a dictionary and that is because Lang chain has taken the string from our

02:20:36

llm C and already converted it into a dictionary for us okay so that's just it's happening behind the scenes there and you can actually see if we go into the details a little bit we can see that we have these arguments and this is the original string that was coming fromn okay which has already been of of course processed by line chain so we have that now the one thing missing here is that okay we've got that the LM wants us to use multiply and we've got what the LM wants us to put into modly

02:21:09

but where's the answer right there is no answer because the tool itself has not been executed because it can't be executed by the llm but then okay didn't we already Define our agent here yes redefined the part of our agent that is how llm has our tools and it is going to generate which tool to use but it actually doesn't include the agent execution part which is okay the agent executor is a broader thing it's it's broader logic like just code logic which acts as a scaffolding within which

02:21:48

we have the iteration through multiple steps of our llm calls followed by the llm outputting what tools use followed by us actually executing that for the llm and then providing the output back into the llm for another decision or another step so the agent itself here is not the full agentic flow that we might expect instead for that we need to implement this agent executor class this agent executor includes our agent from before then it also includes the tools and one thing here is okay we we already

02:22:27

passed the tools to our agent why do we need to pass them again well the tools being passed to our agent up here that is being used so that is essentially extracting out those function schemers and passing it to our LM so that our LM knows how to use the tools then we're down here we're passing the tools again to our agent executor and this is rather than looking at how to use those tools this is just looking at okay I want the functions for those tools so that I can actually execute them for the llm or for the

02:22:57

agent okay so that's why it's happening there now we can also pass in our memory directly so you see if we scroll up a little bit here I actually had to pass in the memory like this with our agent that's just because we weren't using the agent executor now we have the agent executor it's going to handle that for us and another thing that's going to handle for us is it intermediate steps so you'll see in a moment that when we invoke the agent executor we don't include the intermediate steps and

02:23:27

that's because it that is already handled by the agent executor now so we'll come down we'll set the both equal to true so we can see what is happening and then we can see here there's no intermediate steps anymore and we we do still pass in the chat history like this but then the addition of those new interactions to our memory is going to be handled by the executor so let me actually show that very quickly before we jump in okay so that's cently empty we're going to execute

02:24:03

this okay we entered that new Asian execute chain let's just have a quick look at our messages again and now you can see that the agent executor automatically handled the addition of our human message and then the responding AI message for us okay which is useful now what happened so we can see that the multiply tool was invoked with these parameters and then this pink text here that we got that is the observation from the tool assist what the tool output back to us okay then this final message here it's not

02:24:37

formatted very nicely well this final message here is coming from our llm so the green is our llm output the pink is our tool output okay so the LM after seeing this output says 10.7 MTI by 7.68 is approximately 82.8 okay cool use and then we can also see the the chat history which we we already just saw great so that has been used correctly we can just also confirm that that is correct okay 82 1759 recurring which is exactly what we get here okay and we the reason for all that is obviously how multiply tool is

02:25:22

just doing this exact operation cool so let's try this with a bit of memory so I'm going to ask or I'm going to sayate to the agent hello my name is James we'll leave that as the it's not actually the first interaction because we already have these but it's an early interaction with my name in there then we're going to try and perform more tool calls within a single execution Loop and what you'll see with when it is calling these tools is that it can actually use

02:25:56

multiple Tools in parallel so for sure I think two or three of these were used in parallel and then the final subtract had to wait for those previous results so it would have been executed afterwards and we should actually be able to see this in Langs Smith so if we go here yeah we can see that we have this initial cord and then we have add and multiply and exponentially we all use in parallel then we have another call which use subtract and then we get the response okay which is pretty cool and

02:26:27

then the final result there is11 now when you look at whether the answer is accurate I think the order here of calculations is not quite correct so if we put the actual computation here it gets it right but otherwise if I use natural language it's like I'm doing maybe I'm phrasing it in a in a poor way okay so I suppose that is pretty important so okay if we put the computation in here we get the13 so it's something to be careful with and probably requires a little bit of prompting to promting and maybe

02:27:07

examples in order to get that smooth so that it does do things in the way that we might expect or maybe we as humans are just bad and misus the systems one or the other okay so now we've gone through that a few times let's go and see if our agent can still recall our name okay and it remembers my name is James good so it still has that memory in there as well that's good let's move on to another quick example where we're just going to use Google search so we're going to be using the Ser

02:27:39

API you can okay you can get the API key that you need from here so Ser ai.com usersign in and just enter that in here so you will get it up to 100 stes per month for free so just be aware of that if you overuse it I don't think they charge you cuz I don't think you enter your card details straight away but yeah just be aware of that limit now there are certain tools that line chain have already built for us so they're pre-built tools and we can just load them using the load tools function

02:28:17

so we do that like so we have our load tools and we just pass in the Ser API tool only we could pass in more there if we want to and then we also pass in our LM now I'm going to one use that tool but I'm also going to Define my own tool which is to get the current location based on the IP address now this is we're in collab at the moment so it's actually going to get the IP address for the collab instance that I'm currently on and we'll find out where that is so that is going to get the IP address and

02:28:47

then it's going to provide the data back to our LM this format here so we're going to latitude longitude City and Country okay we're also going to get the current day and time so now we're going to redefine our prompt I'm not going to include chat history here I just want this to be like a one shot thing I'm going to redefine our agent and agent executor using our new tools which is our set API plus to get current date time and get location from IP then I'm going to invoke our agent executor

02:29:20

with I have a few questions what is the date and time right now how is the weather where I am and please give me degrees in celce so when it gives me that weather okay and let's see what we get okay so apparently we're in Council Bluffs in the US it is 13° fah which I think is absolutely freezing oh my gosh it is yes minus 10 so it's super cold over there and you can see that okay it did give us Fahrenheits that is because the tool that we were using provided us with Fahrenheit which is fine but it did

02:30:00

translate that over into a estimate of Celsius fours which is pretty cool so let's actually output that so we get this which I is correct we do us approximately this and we also get an description of the conditions as well as partly cloudy with z % precipitation lucky for them and humidity of 66% okay well pretty cool so that is it for this introduction to Lang chain agents as I mentioned next chapter we're going to dive much deeper into agents and also Implement that for Lang chain version 0.3 so we'll leave this chapter

02:30:39

here and jump into the next one in this chapter we're going to be taking a deep dive into agents with the Lang chain and we're going to be covering what an agent is we're going to talk a little bit conceptually about agents the react agent and the type of agent that we're going to be building and based on that knowledge we are actually going to build out our own agent execution logic which we refer to as the agent executor so in comparison to the previous video on agents in line chain which is more of

02:31:17

an introduction this is far far more detailed we'll be getting into the weeds a lot more with both what agents are and also agents within Lang chain now when we talk about agents a significant part of the agent is actually relatively simple code logic that iteratively runs llm calls and processes their outputs potentially running or executing tools the exact logic for each approach to building an agent will actually vary pretty significantly but we'll focus on one of those which is the react agent now react

02:32:02

is it's a very common pattern and although being relatively old now most of the tool agents that we see used by openai and essentially every LM company they all use a very similar pattern now the reactor agent follows a patter and like this okay so we would have our user input up here okay so our input here is a question right aside from the Apple remote what other device you can control the program Apple remote was originally designed to interact with now probably most LMS would actually be able to

02:32:37

answer this directly now this is from the paper which was a few years back now in this scenario assuming our LM didn't already know the answer there are most steps that an llm or an agent might take in order to find out the answer okay so the first of those is we say our question here is what other device can control the program Apple remote was originally designed to interact with so the first thing is okay what was the program that the Apple remote was originally designed to interact with that's the first

02:33:10

question we have here so what we do is I need to search Apple remote and find a program it was use for this is a reasoning step so the llm is reasoning about what it needs to do I need to search for that and find a program useful so we are taking an action this is a tool call here okay so we're going to use the search tool and our query will be apple remote and the observation is the response we get from executing that tool okay so the response here would be the Apple remote it's designed

02:33:39

to control the front row mediate Center so now we know the programmer for was originally designed to interact with now we're going to go through another it okay so this is one iteration of our reasoning action and observation so when we're talking about react here although again this sort of pattern is very common across many agents when we're talking about react the name actually is reasoning or the first two characters of re reasoning followed by action okay so that's where the react comes from so this is one of

02:34:19

our react agent Loops or iterations we're going to go and do another one so next step we have this information the LM is now provided with this information now we want to do a search for front row okay so we do that this is the reasoning step we per the action search front row okay tool search query front row observation this is the response front row is controlled by an apple remote or keyboard function keys all right cool so we know keyboard function keys are the other device that we were asking about

02:34:56

up here so now we have all the information we need we can provide an answer to our user so we go through another iteration here reasoning and action our reasoning is I can now provide the answer of keyboard function keys to the user okay great so then we use the answer tool like Final Answer In more common tool agent use and the answer would be keyboard function keys which we then output to our user okay so that is the react Loop okay so looking at this how where are we actually calling an llm and what and in what way are we

02:35:43

actually calling llm so we have our reasoning step our LM is generating the text here right so LM is generating okay what should I do then our LM is going to generate input parameters to our action step here that will th those input parameters and and the tool being used will be taken by our code logic our agent executor logic and they'll be used to execute some code in which we will get an output that output might be taken directly to our observation or our llm might take that output and then generate an observation

02:36:22

based on that it depends on how you've implemented everything so our LM could potentially being be being used at every single step there and of course that will repeat through every iteration so we have further iterations down here so you're potentially using LM multiple times throughout this whole process which of course in terms of latency and token cost it does mean that you're going to be paying more for an agent than you are with just a sun LM but that that is of course expected because you

02:36:58

have all of these different things going on but the idea is that what you can get out of an agent is of course much better than what you can get out of an LM alone so when we're looking at all of this all of this iterative Chain of Thought and Tool use all this needs to be controlled by what we call the agent executor okay which is our code logic which is hitting our llm processing its outputs and repeating that process until we get to our answer so breaking that part down what does it actually look like it looks

02:37:32

kind of like this so we have our user input goes into our llm okay and then we move on to the reasoning and action steps is the action the answer if it is the answer so as we saw here where is the answer if the action is the answer so true we would just go straight to our outputs otherwise we're going to use our select tool agent executor is going to handle all this it's going to execute our tool and then from that we get our you know three reasoning action observation inputs and outputs and then we're

02:38:09

feeding all that information back into our llm okay in which case we go back through that Loop so we could be looping for a little while until we get to that final but okay so let's go across to the code when be going into the agent executor notebook we'll open that up in coab and we'll go ahead and just install our prerequisites nothing different here is just L chain L Smith optionally as before again optionally line chain API key if you do want to use l Smith okay and then we'll come down to our first

02:38:47

section where it's going to define a few quick tools I'm not necessarily going to go through these because we've already covered them in the agent introduction but very quickly Lang chain core tool is we're just importing this tool decorator which transforms each of our functions here into what we would call a structured tool object this thing here okay which we can see just having a quick look here and then if we want to we can extract all of the sort of key information from that structure tool

02:39:22

using these parameters here or attributes so name description AR schemer model Json streer which give us essentially how the llm should use our function okay so I'm going to keep pushing through that now very quickly again we did cover this in the intro video so I don't want to necessar go over again into much detail but our agent EX future logic is going to need this part so we're going to be getting a string from our llm we're going to be loading that into to a dictionary object

02:39:58

and we're going to be using that to actually execute our tool as we do here using keyword arguments okay like that okay so with the tools out of the way let's take a look at how we create our agent so when I say agent here I'm specifically talking about the part that is generating our reasoning St then generating which tool and what the input parameters to that tool will be then the rest of that is not actually covered by the agent okay the rest of that would be covered by the agent execution logic which would

02:40:36

be taking the tool to be used the parameters executing the tool getting the response aka the observation and then iterating through that until the llm is satisfied and we have enough information to answer a question so looking at that our agent we look something like this it's pretty simple so we have our input parameters including the chat history user query we have our input parameters including the chat history us query and actually would also have any intermediate STS that have happened in here as well we have our

02:41:09

prompt template and then we have our llm binded with tools so let's see how all this would look starting with we'll Define our promp template searching look like this we have our system message your helpful assistant when answering these question you should use on to provide after using a tool tool outp will provide in the scratch Pad below okay which we naming here if you have an answer in scratch Pad you should not use any more tools and set answer directly to the user okay so we have that as our

02:41:42

system message we could obviously modify that based on what we're actually doing then following our system message we're going to have our chat history so any previous interactions between the user and the AI then we have our current message from the user okay we should be fed into the input field there and then following this we have our agent stretch pad or the intermediate thoughts so this is where things like the llm deciding okay this is what I need to do this is how I'm going to do it AKA The Tool call

02:42:14

and this is the observation that's where all of that information will be going right so each of those to pass in as a message okay and the way that we look is that any tool call generation from the llm so when the llm is saying use this tool please that will be a assistant message and then the responses from our tool so the observations they will be returned as tool messages great so we'll run that to Define our prompt template we're going to Define our LM we're going to be using

02:42:50

J2 40 mini with a temperature of zero because we want less creativity here particularly when we're doing tour calling there's just no need for us to use a high temperature here so we need to enter our open ey API key which we would get from platform open ey.com we enter this then we're going to continue and we're just going to add tools to our LM here okay these and we're going to bind them here then we have tool Choice any so tool Choice any we we'll see in a moment I'll go through this a little bit more

02:43:26

in a second but that's going to essentially force a tool call you can also put required which is actually a bit more uh it's bit clearer but I'm using any here so I'll stick with it so these are our tools we're going through we have our inputs into the agent runable we have our prom template and then that will get fed into our llm so let's run that now we would invoke the agent part of everything here with this okay so let's see what it outputs this is important so I'm asking what is 10+ 10

02:44:00

obviously that should use the addition tool and we can actually see that happening so the agent message content is actually empty here this is where you'd usually get an answer but if we go and have a look we have additional keyword dos in there we have tool calls and then we have function arguments Okay so we're calling a function Arguments for that function are this okay so we can see this is string again the way that we would pass that as we do Json loads and that becomes a dictionary and

02:44:31

then we can see which function is being called and it is the add function and that is all we need in order to actually execute our function or our our tool okay we can see it's a little more detail here now what do we do from here we're going to map the to name to the tool function and then we're just going to execute the tool function with the generated ARS I those I'll also just point out quickly that here we are getting the dictionary directly which I think is coming from somewhere else in this which is prob

02:45:05

which is here okay so even that step here where we're passing this out we don't necessarily need to do that because I think on the L chain side they're doing it for us so we're already getting that so Json loads we don't necessarily need here okay so we're just creating this tool name to function mapping dictionary here so we're taking the well the tool names and we're just mapping those back to our tool functions and this is coming from our tools list so that tools list that we defined here

02:45:38

okay or can even just see quickly that that will include everything or each of the tools you define there okay that's all it is now we're going to execute using our name to Tool mapping okay so this here will get us the function so it will get us this function and then to that function we're going to pass the arguments that we generated okay let's see what it looks like all right so the response so the observation is 20 now we are going to feed that back into our llm using the tool message and

02:46:19

we're actually going to put a little bit of text around this to make it a little bit nice so we don't necessarily need to do this to be completely honest we could just return the answer directly uh I don't understand I don't even think there would really be any difference so we we could do either in some cases that could be very useful in other cases like here it doesn't really make too much difference particularly because we have this tool call ID and what this tool call ID is doing is it's being used by

02:46:49

AI is being read by the LM so that the LM knows that the response we got here is actually mapped back to the the tool execution that it's identified here because you see that we have this ID right we have an ID here the LM is going to see the ID it's going see the ID that we pass back in here and it's going to see those two are connected so see okay this is the tool I called and this is the response I got from because of that you don't necessarily need to say which tool you used here you

02:47:24

can it it depends on what you're doing okay so what do we get here we have okay just running everything again we've added our tool call so that's the original AI message that includes okay user add tool and then we have the tool execution tool message which is the observation we map those to the agent scratch pad and then what do we get we have an AI message but the content is empty again which is interesting because we we said to our llm up here if you have an answer to the in the scratchpad

02:48:00

you should not use any more tools and said answer directly to the user so why why is our llm not answering well the reason for that is down here we specify tool Choice equals any which again it's the same tool Choice required which is telling the L land that it cannot actually answer directly it has to use a tool and I usually do this right I would usually put tool Choice equals any or required and for the LM to use a tool every single time so then the question is if it has to use a tool every time how does

02:48:41

it answer our user well we'll see in a moment first I just want to show you the two options essentially that we have the second is what I would usually use but let's let's start with the first so the first option is that we set tool Choice equal to Auto and this tells the Ln that it can either use a tool or it can answer the user directly using the the final answer or using that content field so if we run that like we're specifying to choices Auto we run that let's invoke okay initially you see ah wait

02:49:20

there's still no content that's because we didn't add anything into the agent scratch Pad here there's no information right it's all empty um actually it's empty because sorry so here you have the chat history that's empty we didn't specify the agent scratch Cad and the reason that we can do that is because we're using if you look here we're using get so essentially it's saying try and get agent scratch pad from this dictionary but if it hasn't been provided we're just going to

02:49:48

give an empty list so that's what that's why we don't need to specify it here but that means that oh okay the the agent doesn't actually know anything here it hasn't used a tool yet so we're going to just go through our iteration again right so we're going to get our tool output we're going to use that to create the tool message and then we're going to add our tool call from the AI and the observation we're going to pass those to the agent scratch pad and this time we

02:50:19

see we run that okay now we get the content okay so now it's not calling you see here there's no to call or anything going on we just get content so that is this is the standard way of doing or building a tool calling agent the other option which I mentioned this is what I would usually go with so number two here I would usually create a final answer tool so why would we even do that why would we create a final answer tool rather than just you know this method is actually perfectly you know it works so why would

02:50:59

we not just use this there are a few reasons the main ones are that with option two where we're forcing tool calling this removes possibility of an agent using that content field directly and the reason at least the reason I found this good when building agents in the past is that occasionally when you do want to use a tool it's actually going to go with the content field and it can get quite annoying and and use the content field quite frequently when you actually do want it to be using one

02:51:31

of the tools and this is particularly noticeable with smaller models with bigger models it's not as common although does so happen now the second thing that I quite like about using a tool as your final answer is that you can enforce a structured output in your answer so this is something we're stting I think the first yes the first line chain example where we were using the structured output tool of Lang chain and what that actually is the structured outputs feature of Lang chain it's actually just

02:52:11

a tool call right so it's forcing a tool call from your LM it's just abstracted away so you don't realize that that's what it's doing but that is what it's doing so I find that structured outputs are very useful particularly when you have a lot of code around your agent so when that output needs to go Downstream into some logic that can be very useful because you can you have a reliable output format that you know is going to be output and it's also incredibly useful if you have multiple outputs or

02:52:47

multiple fields that you need to generate for so those can be very useful now to implement this so to implement option two we need to create a final answer tool we as with our other tools we're actually going to description and you can or you cannot do this so you can you can also just return non and actually just use the generated action as the essentially what you're going to send out of your agent execution logic or you can actually just execute the tool and just pass that information directly through perhaps in

02:53:27

some cases you might have some additional postprocessing for your final answer maybe you do some checks to make sure it hasn't said anything weird you could add that in this tool here but yeah in in this case we're just trying to pass those through directly so let's run this we've added where are we Final Answer we've added the final answer tool to our named tool mapping so our agent can now use it we redefine our agent setting tool choice to any because we're forcing the tool Choice here and

02:54:01

let's go with what is 10 + 10 see what happens okay we get this right we can also one thing nice thing here is that we don't need to check is out up in the content field or is it in the tool course field we know it's going to be in the tool course field because we're forcing that tool use quite nice so okay we know we're using the ad tool and these are the arguments great we go or go through our process again we're going to create our tool message and then we're going to add those messages into

02:54:32

our scratch pad or intermediate sets and then we can see again ah okay content field is empty that is expected we we're forcing tool users no way that this can be this can be or have anything inside it but then if we come down here to our to calls nice final answer arbs answer 10 + 10 = 20 all right we also have this tools used where's tools used coming from okay while I mentioned before that you can add additional things or or outputs when you're using this tool use for your final answer so if you just

02:55:11

come up here to here you can see that I asked the llm to use that Tool's use field which I defined here it's a list of strings use this to tell me what tools you used in your answer right so I'm getting the normal answer but I'm also getting this information as well which is kind of nice so that's where that is coming from see that okay so we have our actual answer here and then we just have some additional information okay and we've also defined a type here it's just a list of strings which is

02:55:42

really nice it's giving us a lot of control over what we're outputting which is perfect that's you know when you're building with agents the biggest problem in most cases is control of your llm so here we're getting a honestly pretty unbelievable amount of control over what our LM is going to be doing which is perfect for when you're building in the real world so this is everything we need this is our answer and we would of course be passing that Downstream into whatever

02:56:18

log our AI application would be using okay so maybe that goes directly to a front end and we're displaying this as our answer and we're maybe providing some information about okay where did this answer come from or maybe there's some additional steps Downstream where we're actually doing some more processing or Transformations but yeah we have that that's great now everything we've just done here we've been executing everything one by one and that's to help us understand what process we go through

02:56:53

when we're building an agent executor but we're not going to want to do that all the time are we most of the time we probably want to abstract all this away and that's what we're going to do now so we're going to build essentially everything we've just taken we're going to abstra take that and Abstract it away into a custom agent executor class so let's have a quick look at what we're doing here although it's it's literally just what we we just did okay so custom maor

02:57:27

executor we initialize it we set this m Max iterations I'll talk about this in a moment we initialize it that is going to set out chat history to just being empty okay it's a new agent there should be no chat history in this case then we actually Define our agent right so that poted logic that is going to be taking out inputs and generating what to do next AKA what tool call to do okay and we set everything as attributes of our class and then we're going to Define an invoke method this invoke method is

02:58:01

going to take an input which just a string so it's going to be our message from the user and what it's going to do is it's going to iterate through essentially everything we just did okay until we hit the The Final Answer tool Okay so well what does that mean we have our tool call right which is we're just invoking our agent right so it's going to generate what tool to use and what parameters should go into that okay and that's a that's an AI message so we would append that to our

02:58:36

agent stretch pad and then we're going to use the information from our tool call so the name of the tool and the ARs and also the ID we're going to use all of that information to execute our tool and then provide the observation back to our llm okay so we execute our tool here we then format the tool output into a tool message see here that I'm just using the the output directly I'm not adding that additional information there we need do need to always pass in the tool call ID so that our LM knows which

02:59:12

output is mapped to which tool I didn't mention this before in in this video at least but that is that's important when we have multiple toour calls happening in parallel because that can happen when we have multiple toour calls happening in parallel let's say we have 10 tool calls all those responses might come back at different times so then the order of those can get messed up so we wouldn't necessarily always see that it's a AI message beginning a tool call followed by the answer to that tool call instead

02:59:44

it might be AI message followed by like 10 different tool call responses so you need to have those IDs in there okay so then we pass our tool output back to our agent scratch pad or intermediate steps I'm sing a print in here so that we can see what's happening whilst everything is running then we increment this count number we'll talk about that in a moment so com past that we say okay if the tool name here is final answer that means we should stop okay so so once we get the final answer that means we can actually

03:00:21

extract our final answer from the the final tool call okay and in this case I'm going to say that we're going to extract the answer from the tool call or the the observation we're going to extract the answer that was generated we're going to pass that into our chat history so we're going to have our user message is the one the user came up with followed by our answer which is just the the natur answer field and that's going to be an AI message but then we're actually going to be including all of

03:00:54

the information so this is the the answer natural language answer and also the tools used output we're going to be feeding all of that out to some Downstream process as preferred so we have that now one thing that can happen if we're not careful is that our agent executor might may run many many times and particularly if we've done something wrong in our logic as we're building these things it can happen that maybe we've not connected the observation back up into our agent executor logic and in

03:01:33

that case what we might see is our agent executor runs again and again and again and I mean that's fine we're going to stop it but if we don't realize straight away and we're doing a lot of llm cords that can get quite expensive quite quickly so what we can do is we can set a limit right so that's what we've done up here with this Max iterations we said okay if we go past three max iterations by default I'm going to say stop all right so that's that's why we have the count here while

03:02:02

count is less than the max iterations we're going to keep going once we hit the number of Max iterations we stop okay so the while loop will will just stop looping okay so it just protects Us in case of that and it also potentially maybe it's Point your agent might be doing too much to answer a question so this will force it to stop and just provide an answer although if that does happen I just realize there's a bit of a fault in the logic here if that does happen we wouldn't necessarily have the

03:02:33

answer here right so we would probably want to handle that nicely but in this scenario a very simple use case we're not going to see that happening so we initialize our custom agent executor and then we invoke it okay and let's see what happens all right there we go so that just wrapped everything into a single single invoke so everything is handled for us uh we could say okay what is 10 you know we can modify that and say 7.4 for example and that we'll go through we'll use the multiply tool instead and then we'll

03:03:16

come back to the final answer again okay so we can see that with this custom agent executor we've built an agent and we have a lot more control over everything that is going on in here one thing that we would probably need to add in this scenario is right now I'm assuming that only one tool call will happen at once it's also why I'm asking here I'm not asking a complicated question because I don't want it to go and try and execute multiple tool Calles at once uh which which can happen so

03:03:49

let's just try this okay so this is actually completely fine so this did just execute it one after the other so you can see that when asking this more complicated question it first did the exponentiate tool followed by the ad tool and then they actually gave us our final answer which is cool also told us we use both of those tools which it did but one thing that we should just be aware of is that from open AI open AI can actually execute multiple tool calls in parallel so by specifying that we're just using this

03:04:27

zero here we're actually assuming that we're only ever going to be calling one tool at any one time which is not always going to be the case so you would probably need to add a little bit of exual logic there in case of scenarios if you're building an an agent that is likely to be running parallel to calls but yeah you can see here actually it's completely fine so it's running one after the other okay so with that we built our agent executor I know there's a lot to that and of course you can just

03:04:57

use the very abstract agent executor in L chain but I think it's very good to understand what is actually going on to build our own agent executor in this case and it sets you up nicely for building more complicated or use case specific agent logic as well so that is it for this chapter in this chapter we're going to be taking a look at line chains expression language we'll be looking at the runnables the serializable and parallel of those the runable pass through and essentially how we use l

03:05:33

cell in its full capacity now to do that well what I want to do is actually start by looking at the traditional approach to building chains in L chain so to do do that we're going to go over to the ELO chapter and open that Cur up okay so let's come down we'll do the prerequisites as before nothing measure in here the one thing that is new is Doc array because later on as you see we're going to be using this as an example of the parallel capabilities in L cell if you want to use Lim Smith you just need

03:06:14

to add in your lime train API key okay and then let's okay so now let's dive into the traditional approach to chains in line chain so the LM chain I think is probably one of the first things introduced in line chain if I'm not wrong this take it to prompt and feeds into an l and that that's it it you can also you can add like output passing to that as well but that's optional and I don't think we're going to cover here so what that might look like is we have for example this promp template here give me

03:06:52

a small report on topic okay so that would be our prompt template we set up as we usually do with the prom templates as we've seen before we then Define our LM need our open a key for this which as usual we would get from platform. open.com then we go ahead I'm just just showing you that you can Ino the LM there then we go ahead actually Define a output POS so we do do this I wasn't sure we did but we would then Define our LM chain like this okay so LM chain we adding our prompt adding

03:07:32

our LM adding our alasa okay this is the traditional approach so I would then say Okay retrieve Org the generation and what's going to do it's going to give me a little report back on on rag okay t a moment but you can see that that's what we get here we can format out nicely as we usually do and we get okay look we get a nice little report however the LM chain is one it's quite restrictive right we have to have like particular parameters that have been predefined as being usable which is you know restrictive and

03:08:13

it's also been deprecated so you know this isn't the standard way of doing this anymore but we can still use it however the preferred method to building this and building anything else really or chains in general in L chain is using El cell right and it's super simple right so we just actually take the prompt lemon Apple P that we had before and then we just chain them together with these pipe operators so the pipe operator here is saying take what is output from here and input it into here

03:08:44

take wi's output from here and input it into here it's all it does super simple so put those together and we invoke it in the same way and we'll get the same output okay and that's what we get there is actually a slight difference on what we're getting out from there you can see here we got actually a dictionary but that is pretty much the same okay so we get that and as before we can display that in markdown with this okay so we saw just now that we have this pipe operator here it's not really

03:09:23

standard P python syntax to use this or at least it's definitely not common it's it's it's an aberration of the intended use of python I think but anyway it does it looks cool and when you understand it I kind of get why they do because it make it does make things quite simple in comparison to what it could be otherwise so I kind get it it's a little bit weird but it's what they're doing and I'm teaching that so that's what we're going to learn so what is that pipe operator

03:09:59

actually doing well it's as I mentioned it's taking the output from this putting it as input into into what is ever under right but how does that actually work well let's actually implement it ourselves without line chain so we're going to create this class called runnable this class when we initialize it it's going to take a function okay so this is literally a python function it's going to take that and it's going to essentially turn it into what we would call a runnable in line chain and what

03:10:33

does that actually mean well it doesn't really mean anything it just means that when you use run the invoke method on it it's going to call that function in the way that you would have done otherwise all right so using just function you know brackets open parameters brackets closed it's going to do that but it's also going to add this method this all method now this all method in typical python syntax now this all method is essentially going to take your runnable function the one that you initialize

03:11:08

with and it's also going to take an other function okay this other function is actually going to be a runnable I believe yes it's going to be runnable just like this and what it's going to do is it's going to run this runnable based on the output of your current runable okay that's what this or is going to do seems a bit weird maybe but I'll explain in a moment we'll see why that works so I'm going to chain a few functions together using this or method so first we're just going to turn

03:11:45

them all into runnables Okay so these are normal functions as you can see normal python functions we then turn them into this runnable using our runnable class then look what we can do right so we we're going to create a chain that is going to be our runnable chained with another runnable chained with another runnable okay let's see what happens so we're going to invoke that chain of runnables with three so what is this going to do okay we start with five we're going to add five to three so we'll get eight

03:12:22

then we're going to subtract five from8 to give us three again and then we're going to multiply three by five to give us 15 and we can inval that and we get 15 okay pretty cool so that is interesting how does that relate to the pipe operator well that pipe operator in Python is actually a shortcut for the or method so what we just implemented is the pipe operator so we can actually run that now with the pipe operator here and we'll get the same get 15 right so that's that's what

03:13:04

line chain is doing like under the hood that is what that pipe operator is it's just chaining together these multiple runnables as we'd call them using their own internal or operator okay which is cool I I I will give them that it's kind of a cool way of doing this creative I wouldn't have thought about it myself so yeah that is a pipe operator then we have these runnable things okay so this is a this is different to the runable I just defined here this is we Define this ourselves it's not a lang

03:13:38

chain thing we didn't get this from Lang chain Instead This runnable Lambda object here that is actually exactly the same as what we just defined all right so what we did here this runnable this runnable Lambda is the same thing but in Lang chain okay so if we use that okay we use that to now Define three runnables from the functions that we defined earlier we can actually pair those together now using the the pipe operator you could also pair them together if you want with the or operator right

03:14:18

so we could do what we did earlier we can invoke that okay or as we were doing originally we use pipe operator exactly the same so this runnable Lambda from line chain is just what we what we just built with the runable cool so we have that now let's try and do something a little more interesting we're going to generate a report and we're going to try and edit that report using this this functionality okay so give me a small report about topic okay we'll Z through here we're going to get our report on

03:14:54

AI okay so we have this you can see that AI is mentioned many times in here then we're going to take a very simple function right so I'm extract fact this is basically going to take uh what is it see taking the first okay so we're actually trying to remove the introduction here I'm not sure if this actually will work as expected but it's it's fine try it anyway but then more importantly we're going to replace this word okay so we're going to replace an old word with a new

03:15:32

word our old word going to be Ai and the word is going to be Skynet okay so we can wrap both of these functions as runable lambas okay we can add those as additional steps inside our entire chain all right so we're going to extract try and remove the introduction although I think it needs a bit more processing than just splitting here and then we're going to replace the word we need that actually to be AI run that run this okay so now we get artificial intelligence Skynet refers to the simulation of human intelligent process

03:16:09

by machines uh we have narrow Skynet weak Skynet and strong Skynet applications of Skynet Skynet Technologies is being applied in numerous Fields including all these things scary despite potential sky that poses several challenges systems can perpetrate exist and biases it ra significant privacy concerns it can be exploited for malicious purposes okay so we have all these you know it's just a silly little example we can see also the introduction didn't work here the reason for that is because our introduction includes

03:16:46

multiple new lines here so I would actually if I want to remove the introduction we should remove it from here I think and this is a I I would never actually recommend you do that uh because it's not it's not very flexible it's not very robust but just so I show you that that is actually working so this extract fact runnable right so now we're essentially just removing the introduction right why what do we want to do that I don't know but it's there just so you can see that we can have

03:17:21

multiple of these runnable operations running and they can be whatever you want them to be okay it is worth knowing that the inputs to our functions here were all single arguments okay if you have function that is accepting multiple arguments you can do that the way that I would probably do it or you can do it in multiple ways one of the ways that you can do that is actually write your function to except for arguments but actually do them through a single argument so just like a single like X which would be like a dictionary or

03:17:55

something and then just unpack them within the function and and use them as needed that's just yeah that's one way you can do it now we also have these different uh runnable objects that we can use so here we have runnable parallel and runnable pass through kind of self-explanatory to some degree so let me let just go through those so runable parallel allow you to run multiple runnable instances in parallel runnable pass through May was less self-explanatory allows us to pass a variable through to the next runnable

03:18:29

without modifying it okay so let's see how they would work so we're going to come down here and we're going to set these two dock arrays obiously these two sources of information and we're going to need our LM to pull information from both of these sources of information in parallel which is going to look like this so we have these two sources of information Vector store a vector store B this is our dock array a and dock array B these are both going to be fed in as context into our prompt then our LM is

03:19:04

going to use all of that to answer the question okay so to actually Implement that we have our we need an embedding model so he open our embeddings we have our vetur a a vector B they're not you know real vectors they're not full-on vectors SS here we're just passing in a very small amount of information to both so we're saying okay we're going to create an inmemory vect S using these two bits of information so when say half the information is here this would be an irrelevant piece of information then we

03:19:37

had the relevant information which is deep seek re3 was released in December 2024 okay then we're going to have some other information in our other Vector sore again irrelevant piece here and relevant piece here okay the Deep seek V3 LM is a mixure of experts model with 671 billion parameters at its largest okay so based on that we're also going to build this prompt string so we're going to pass in both of those contexts into our prompt then I'm going to ask a question we don't actually need we don't

03:20:12

need that bit and actually we don't even need that bit what am I doing so we just need this so we have the both the contexts there and we would run them through our prompt template okay so we have our system promp template which is this and then we're just going to have okay our question is going to go into here as a user message cool so we have that and then let me make this easier to read we're going to convert both those STS to retrievers which just means we can retrieve stuff from them and we're

03:20:46

going to use this runnable parallel to run both of these in parallel right so these are being both being run in parallel but then we're also running our question in parallel because this needs to be essentially passed through this component without us modifying anything so when we look at this here it's almost like okay the this section here would be our runable parallel and these are being running parallel but also our query is being passed through so it's almost like there's another line there which is our

03:21:21

runable pass through okay so that's what we're doing here these running in parallel one of them is a pass through I need to run here I just realized here we're using the uh deprecated embeddings just switch it to this so L chain open AI we run that run this run that and now this is set up okay so we then put our initial so this using our runable parallel and runnable pass through that is our initial step we then have our prompt LM now pass which would being chained together with usual

03:22:07

you know the usual type operator okay and now we're going to invoke question what architecture does the mod deep seek release in December use okay okay so for the elm to answer this question it's here to need to tell us what it needs the information about the Deep seek model that was released in December which we have specified in one half uh here and then it also needs to know what architecture that model uses which is defined in the other half over here okay so let's run this okay there we go deep SE V3 model

03:22:44

released in December 2024 is a mix experts model with 671 billion parameters okay so mixture of experts and this many parameters pretty cool so we've put together our pipeline using elol using the pipe operator the runnables specifically we've looked at the runable parallel runable pass through and also the runable lampas so that's it for this chapter on lell and we'll move on to the next one in this chapter we're going to cover streaming and async in Lang chain now both using

03:23:20

async code and using streaming are incredibly important components of I think almost any conversational chat interface or at least any good conversational chat interface for async if your application is not async and you're spending a load of time in your API or whatever else waiting for llm calls because a lot of those are behind apis you are waiting and your application is doing nothing because you've written synchronous code and that well there are many problems with that mainly it doesn't scale so asyn code

03:24:00

generally performs much better and especially for AI where a lot of the time we're kind of waiting for API calls so asyn is incredibly important for that for streaming now streaming is slightly different thing so let's say I want to to tell me a story okay I'm using gbt 4 here it's a bit slower so we can achieve string we can see that token by token this text is being produced and sent to us now this is not just a visual thing this is the LM when it is generating tokens or words

03:24:38

it is generating them one by one and and that's because these llms literally generate tokens one by one so they're looking at all of the previous tokens in order to generate the next one and then generate next one generate next one that's how they work so when we are implementing streaming we're getting that feed of tokens directly from the LM through to our you know our back end or our front end that is what we see when when we see that token by token interface right so that's one thing what

03:25:10

one other thing that I can do that let me switch across to 40 is I can say okay we just got this story I'm going to ask are there any standard storytelling techniques to follow used above please use search okay so look we we get this very briefly there we saw that it was searching the web and the way it's not because we told it okay we told the llm to use the search tool but then the llm output some tokens to say use the search tool that is going to use a Search tool and it also would have output the token

03:26:01

saying what that search query would have been although we didn't see it there but what the chat GPT interface is doing there so it received those tokens saying hey I'm going to use a Search tool it didn't just send us those tokens like it does with the tokens here instead it used those tokens to show us that searching the web little text box so streaming is not just the streaming of these direct tokens it's also the streaming of these intermediate steps that the llm may be thinking through

03:26:37

which is particularly important when it comes to agents and agentic interfaces so it's also a feature thing right streaming does doesn't just look nice is also a feature then finally of course when we're looking at this okay let's say we go back to GT4 and I say okay use all of this information to generate a long story for me right and okay we are getting the first token now we know something is happening and we need start reading now imagine if we were not streaming anything here and we're just waiting

03:27:23

right we're still waiting now we're still waiting and we wouldn't see anything we're just like oh it's just blank or maybe there's a little loading spinner so we'd still be waiting and even now we're still waiting right this is an extreme example but can you imagine just waiting for so long and not seeing anything as a user right now just now we would have got our answer if we were not streaming I mean that that would be painful as a user you you not want to wait especially

03:27:59

in a chat interface you don't want to wait that long it's okay with okay for example deep research takes a long time to process but you know it's going to take a long time to process and it's a different user case right you're getting a report this is a chat interface and yes most messages are not going to take that long to generate we're also probably not going to be using GPT 4 depending on I don't know maybe some people still do but in some scenarios it's painful to need to wait that long

03:28:32

okay and it's also the same for agents it's nice when you're using agents again update on okay we're using this tool it's using this tool this is how it's using them perplexity for example have a very nice example of this so okay what what's this open I founder joins morati sub let's see right so we see this is really nice it's we're using Pro search it's searching for news sharing with the results like we're getting all this information as we're waiting which is

03:29:00

really cool and it helps to understand what is actually happening right it's not needed in all use cases but it's super nice to have those intermediate steps right so then we're not waiting and then I think this bit probably also streamed but it was just super fast so I I didn't see it but that's pretty cool so streaming is pretty important let's dive into our example okay we'll open that in cab enough we out so starting with the prerequisites same as always Lang chain optionally L Smith we'll also

03:29:34

enter our L chain API key if you'd like to use l Smith we'll also enter our openi API key so that is platform. open.com and then as usual we can just invoke our l m right so we have that it's working now let's see how we would stream with a stream okay so whenever a method so stream is actually a method as well we could use that but it's not acing right so whenever we see a method in line chain that has a prefixed onto what would be another method that's like the async version of this

03:30:12

so we can actually stream using async super easily using just LM a stream okay now this is just a an example in to be completely honest you probably will not be able to use this in an actual application but it's just an example and we're going to see how we would use this or how we would stream asynchronously in an application further down in this notebook so starting with this you can see here that we're getting these tokens right we're just appending it to token here we don't actually need to do that I

03:30:49

don't think we're using this but maybe we yeah we do it here it's fine so we're just pending the tokens as they come back from our LM pending it to this we'll see what that is in a moment and then I'm just printing the token content right so the content of the token so in this case that would be l in this case it would be LP it would be Sans for so on and so on so you can see for the most part it's it's tends to be Word level but it can also be subword level as you see scent iment is one word

03:31:23

of course so you know they get broken up in in various ways then adding this pipe character onto the end here so we can see okay where are our individual tokens then we also have flush so flush uh you can actually turn this off and it's still going to stream you're still going to see everything which going to be a bit more you can see it's kind of a it's like bit by bit when we use flush it forces the console to update what is being shown to us immediately all right so we get a much

03:31:57

smoother um when we're looking at this it's much smoother versus when flush is not set true so yeah when you're printing that is good to do just so you can see you don't necessarily need to okay now we added all those tokens to the tokens list so we can have a look at each individual object that was returned turn to us right and this is interesting so we see that we have the AI message chunk right that's an object and then you have the content the first one's actually empty second one has that n for NLP and

03:32:30

yeah I mean that's all we rarely need to know they're very simple objects but they're actually quite useful because uh just look at this right so we can add each one of our AI message chunks right let's see what that does it doesn't create a list it creates this right so we still just have one AI message chunk uh but it's combined the content within those AI message chunks which is kind of cool right so for example like we could remove these right and then we just see NLP so

03:33:04

that's kind of nice little feature there I do I actually quite like that but uh you do need to just be a little bit careful because obviously you can do that the wrong way and you're going to get like a I don't know all that is some weird token salad so yeah you need to just make sure you are going to be merging those into correct order unless you I don't know unless you're doing something weird Okay cool so streaming that that was streaming from a LM let's have look at streaming with agents so we

03:33:39

it gets a bit more complicated to be completely honest but we also need to things are going to get a bit more complicated so that we can implement this in for example an API right so is it's kind of like a necessary thing in any case so to just very quickly we're going to construct our agent executor like we did in the agent execution chapter and for that for the agent executor we're going to need tools chat prompt template llm agent and the agent H itself okay very quickly I'm not going

03:34:13

to go through these uh in detail we just def find our tools have ADD multiply exponentiate subtract and Final Answer tool merge those into a single list of tools then we have our prompt template again same as before we just have system message we have chat history we have you query and then we have the agent scratch pad for those intermediate sets then we Define our agent using L cell L cell works quite well with both streaming and async by the way it supports both out of the box which is nice so we Define our

03:34:51

agent then coming down here we're going to create the agan ice fter this is the same as before right so there's nothing new in here I don't think so just initialize our agent things there then it's you know We're looping through looping through yeah nothing nothing new there so we're just executing invoking our agent seeing if there's a tool call uh this is slightly we could shift this to before or after it doesn't actually matter that much so we're checking if it's final

03:35:26

answer if not we continue X to our tools and so on Okay cool so then we can invoke that okay we go what is 10 + 10 there we go right so we have our agent executor it is working now now when we are running our agent executor with every new query if we're putting this into an API we're probably going to need to provide it with a a fresh callback Handler okay so this is the corat Handler that's going to handle taking the tokens that are being generated by a LMO agent and giving them to some other

03:36:10

piece of code like for example the the streaming response for a API and our Corbat Handler is going to put those tokens in a queue in our case and then our for example the streaming object is going to pick them up from the queue and put them wherever they need to be so to allow us to do that with every new query or is needing to initialize everything when we actually initialize our agent we can add a configural field to our llm okay so we set the configural Fields here oh also one thing is that

03:36:48

way we set streaming equal to true that's very manting but just so you see that there we do do that so we add some configurable fields to our LM which means we can basically pass an object in for these on every new invocation so we set our configurable field it's going to be called callbacks and we we just add a description right there's nothing more to it so this will now allow us to provide that field when we're invoking our agent okay now we need to Define our callback Handler and

03:37:25

as I mentioned what is basically going to be happening is this callback Handler is going to be passing tokens into our a sync IO Q object and then we're going to be picking them up from the que elsewhere okay so we can call it a q callback Handler okay and that is inhering from the async Callback Handler cuz we want all this to be done asynchronously because we're we're thinking here about okay how do we Implement all this stuff within apis and actual real world code and we we do want

03:37:55

to be doing all this in aing so let me execute that and I'll just explain a little bit of what we're looking at so we have the initialization right it's nothing nothing specific here we just what we really want to be doing is we want to be setting our Q object assigning that to the class attributes and then there's also this Final Answer scene which we're setting to fults so what we're going to be using that for is we our llm will be streaming tokens towards whilst it's using its tool

03:38:28

calling and we might not want to display those immediately or we remember to display them in a different way so by setting this Final Answer scene to false whilst our LM is outputting those tool tokens we can handle them in a different way and then as soon as we see that it's done with the tool calls and it's on to the final answer which is actually another tool call but once we see that it's on to the final answer tool call we can set this true and then we can start processing our tokens in a

03:38:58

you know different way essentially okay so we have that then we have this AER method this is required for any async generator object so what that is going to be doing is going to iterating through right it's a generator it's going to be going iterating through and it's going saying okay if our queue is empty right this is the que that we set up here if it's empty wait a moment right we use the Sleep Method here and this is an async Sleep Method this is super important we're using we are waiting for an

03:39:33

asynchronous sleep all right so whilst we're whilst we're waiting for that 0.1 seconds our our code can be doing other things right that that is important if we if we use I think the standard is time dos sleep that is not asynchronous and so it will actually block the thread for that 0. one seconds so we don't want that to happen generally our Q should probably not be empty that frequently given how quickly uh tokens are going to be added to the queue so the only way that this would potentially be empty is maybe our

03:40:09

LM stops maybe there's like a connection Interruption for it you know a brief second or something and no tokens are added so in that case we don't actually do anything we don't keep the checking the queue we just wait a moment okay and then we check again now if it was empty we wait and then we continue onto the next iteration otherwise it probably won't be empty we get whatever is from our inside our queue we get that out pull it out then we say Okay if that token is a done token we're going to

03:40:43

return so we're going to stop this generator right we're finished otherwise if it's something else we're going to yield that token which means we're we're returning that token but then we're continuing through that loop again right so that is our generator logic then we have some other methods here these are L these are line chain specific okay we have on LM new token and we have on LM end starting with on LM new token this is basically when an LM returns a token to us line chain is

03:41:18

going to run or execute this method okay this is the method that will be called what this is going to do is it's going to go into the keyword arguments and it's going to get the chunk object so this is coming Fromm if there is something in that chunk it's going to check for a final answer tool call First okay so we get our tool calls and we say if the name within our ch chunk right probably this will be emptying most of the tokens we return right so you remember before when we're looking at

03:41:52

the chunks here this is what we're looking at right the content for us is actually always going to be empty and instead we're actually going to get the additional keyword objects here and inside there we're going to have our tool calling our tool calls as we s in the the previous videos right so that's what we're extracting we're extracting that information that's why we're going additional keyword ARS right and get those tool the tool call information right or it will be nonone right so if

03:42:22

if it is nonone I don't think it ever would be none to be honest it would be strange if it's none I think that means something would be wrong okay so here we're using the wars operator so the wars operator what it's doing here is whil we're checking the if logic here whilst we do that it's also assigning whatever is inside this it's assigning over to Tool Calles and then with the if we're checking whether tool cause is something or nonone right because we're using get here so if if this get

03:42:55

operation fails and there is no tool calls this object here will be equal to none which gets assigned to Tool calls here and then this this if none will return false and this logic will not run okay and it will just continue if this is true so if there is something returned here we're going to check if that's something returned is using the function name or tool name final answer if it is we're going to set that final answer see equal to True otherwise we're just going to add our chunk into Q okay

03:43:28

we use put no weight here because we're we're using async otherwise if you were not using async I think you might just put weight or maybe even put put no okay you you use put if it's a synchronous code but I I don't think I've ever implemented a synchronous so it would actually just be put no weight for Asing okay and then return so we have that then we have on llm end okay so this is when line chain sees that the llm has returned or indicated that it is finished with the response line chain

03:44:06

will call this so you you have to be aware that this will happen multiple times during an agent execution because if you think within our agent executor we're hitting the LM multiple times we have that first step where it's deciding oh I'm going to use the add tool or the multiply tool and then that response gets back towards we execute that tool and then we pass the output from that tool and all the original user query in the chat history pass that back to our LM again all right so that's another call to our LM that's

03:44:42

going to come back it's going to finish it's going to give us something else right so there's multiple llm cods happening throughout our agent execution logic so this on LM call will actually get called at the end of every single one of those llm calls now if we get to the end of our llm call and it was just a it was a tool invocation so we had the you know it called the ad tool we don't want to put the done token into our Cube because when the done token is added to our Cube we're going to stop

03:45:17

iterating okay instead if it was just a tool call we're going to say step end right and we'll actually get this token back so this is useful on for example the front end you could have okay I've I've used the ad tool the these are the parameters and it's the end of the step so you could have that your tool callers being used on some front end and then as soon as it sees step end it knows okay we're done with here was a response right and and it can just show you that and we're

03:45:49

going to use that we'll see that soon but let's say we get to the final an tool we're on the final answer tool and then we get this signal that the llm has finished then we need to stop iterating otherwise our our streaming generator is just going to keep going forever right nothing's going to stop it or maybe it will time out I don't think it will though so at that point we need to send okay stop right we need to say we're done and then that will that will come back to here to our a iterator and to our asnc

03:46:24

iterator and it will return and stop the generator okay so that's the core logic that we have inside there I know there's a lot going on there it's but we need all of this so it's important to be aware of it okay so now let's see how we might actually call our agent with all of this streaming uh in this way so we're going to initialize our queue we're going to use that to initialize a streamer okay using the the custom streamer that we just sell custom callback Handler whatever you want to

03:46:59

call it okay then I'm going to define a function so this is an asynchronous function it has to be if if we're using async and what it's going to do is it's going to call our agent with a config here and we're going to pass it that call the the Callback which is the streamer right note here I'm not calling the agent executor I'm just calling the agent right so the uh if we come back up here we're calling this all right so that's not going to include all tool execution logic and

03:47:31

importantly we're calling the agent with the config that uses callbacks right so this this configurable field here from our LM is actually being fed through it propagates through to our agent object as well to the runable so realizable all right so that's what we're executing here we see agent with config and we're passing in those callbacks which is just one actually okay so that sets up our agent and then we invoke it with a stram okay like we did before and we're just going to

03:48:02

return everything so let's uh run that okay and we see all the token or the chunk objects that are being returned and this is useful to understand what we're actually doing up here right so when we're doing this chunk message additional C keyword arguments right we can see that in here so this would be the chunk message object we get the additional keyword objects go into tool calls and we get the information here so we have the ID for that tool call as we saw in the previous chapters

03:48:34

then we have our function right so the function includes the name right so we know what tool we're calling from this first chunk but we don't know the arguments right those arguments are going to be streamed to us so we can see them begin to come through in the next chunk so the next chunk is just it's just a first token for for the ad function right and we can see these all come together over multiple steps and we actually get all of our arguments okay that's pretty cool so actually one thing I would like to

03:49:10

show you here as well so if we just do token equals token sorry and we do tokens. pen token okay we have all of our tokens in here now right you see that they're all AI message chunks so we can actually add those together right so let's we'll go with these here and based on these we're going to get all the arguments okay so this is kind of interesting so it's one until I think like the second to last maybe right so we have these and actually we just want to add those together so I'm going to go with tokens

03:50:01

one I'm just going to go four uh four token in we're going to go from the second onwards I'm going to TK plus token right and let's see what TK looks like at the end here TK okay so now you see I kind of merged with all those um arguments here sorry plus equal okay so run that and you can see here that it's merged those arguments it didn't get all of them so I kind of missed some at the end there but it's merging them right so we can see that that logic where it's you know before it

03:50:44

was adding the content from various trunks it also does the same for the other parameters within your trunk object which is is I I think it's pretty cool you can see here the name wasn't included that's because we started on token one or on token zero where the name was so if we actually started from token zero and let's just let's just pull them in there right so from one onwards we're going to get a complete AI message chunk which includes the name here and all of those arguments and you

03:51:19

you'll see also here right populate everything which is pretty cool okay so we have that now based on this we're going to want to modify our custom agent executor because we're streaming everything right so we want to add streaming inside our agent executor which we're doing here right so this is async death stream and we're sharing async for token in the a string okay so this is like the very first instance if output is none we're just going to be adding our token so the the chunk sorry

03:51:56

to our output like the first token becomes our output otherwise we're just appending our tokens to the output okay if the token content is empty which it should be right because we're using tool cores all the time we're just going to print content okay I just added these as so we see like print everything I just want to want to be able to see that I wouldn't expect this to run because we're saying it has to use tool calling okay so within our agent if we come up to here we said tool Choice any so it's

03:52:30

been forced to use tool calling so it should never really be returning anything inside the content field but just in case it's there right so we'll we'll see if that is actually true then we're just getting out our tool CES information okay from our trunk and we're going to say okay if there's something in there we're going to print what is in there okay and then we're going to extract our tool name if there is some if there is a tool name I'm going to show you the tool

03:52:55

name then we're going to go to ORS and if the ORS are not empty we're going to see what we get in there okay and then from all of this we're actually going to we merge all of it into our AI message right because we're merging everything as we're going through we're merging everything into outputs as I showed you before okay cool and then we just awaiting our stream that will like kick it off okay and then we do the the standard agent execut stuff again here right so we're just pulling out tool

03:53:23

name Tool logs tool call ID and then we're using all that to execute our tool here and then we're creating a new tool message and passing that back in and then also here I move the break for The Final Answer into the final step so that is our custom Asian executor with streaming and let's see what let's see what it does okay St for b equal true so we see all those print statements okay so you can kind of see it's a little bit messy but you can see we have tool calls that had some stuff

03:53:58

inside it had add here and what we're printing out here is we're printing out the full AI message chunk with tool calls and then I'm just printing out okay what are we actually pulling out from from that so these are actually coming from the same thing okay and then the same here right so we're looking at full message and then we're looking okay we're getting this argument out from it okay so we can see everything that is being pulled out you know chunk by chunk or token by to token and that's it okay

03:54:28

so we could just get everything like that however right so I'm I'm printing everything so we can see that it's streaming what if I don't print okay so we're setting the bo or by default the both is equal to false here so what happens if we invoke now plus C okay cool we got nothing so the reason we got nothing is because we're not printing but we don't if you are if you're building an an API for example you're you're pulling your tokens through you can't print them to your

03:55:14

like like a front end or or print them as to the output of your API printing goes to your terminal right your console window it doesn't go anywhere else instead what we want to do is we actually want to get those tokens out right but if but how do we do that all right so we we printed them but another place that those tokens are is in our que all right because we set them up to go to the que so we can actually pull them out of our queue whilst our agent executor is running and then we can do whatever we

03:55:53

want with them because our code is async so it can be doing multiple things at the same time so whilst our code is running the agent executor whilst that is happening our code can also be pulling out from our queue tokens that are in there and sending them to like an API for example right or whatever Downstream you you have so let's see what that looks like we start by just initializing our que initializing our streamer with that que then we create a task so this is basically saying okay I I want to run

03:56:27

this but don't run it right now I'm not ready yet the reason that I say I'm not ready yet is because I also want to Define here my async Loop which is going to be printing those tokens right but this is async right so we we set this up this is like get ready to run this because it is async this is running right this is just running like it there it's already running so we get this we continue we continue this none of this is actually executed yet right only here when we await the

03:57:01

task that we set up here only then does our agent executor run and our async object here begin getting tokens right and here again printing but I don't need to print I could I could have like a let's say where this is within an API or something let's say I'm I'm saying okay send token to XYZ token right that's sending up token somewhere or if we're maybe we're yielding this to our some sort streamer object within our API right we can do whatever we want with those tokens okay

03:57:43

I'm just printing them because I want to see them okay but just important here is that we're not printing them within our agent executor we're printing them outside the agent executor we've got them out and we can put them wherever we want which is perfect when you're building an actual sort real world use KS we using an API or something else okay so let's run that let's see what we get look at that we get all of the information we could need and a little bit more right because now we're using

03:58:13

the agent executor and now we can also see oh we have this step end right so I know all I I know just from looking at this right this is my first tool use so what tool is it let's have a look it's the add tool and then we have these arguments I can then pass them right Downstream then we have the next tool use which is here down here so we can then pass them in the way that we like so that's pretty cool let's I mean let's see right so we're getting those fers out can we can we do

03:58:50

something with them before I before I print them and show them yes let's see okay so we're now modifying our our Loop here same stuff right we're still initializing our queue initializing our streamer initializing our task okay and we're still doing this aing for token streamer okay but then we're doing stuff with our tokens so I'm saying okay if if we're on stream end I'm not actually going to print stream end I'm going to print new line okay otherwise if we're getting a tool

03:59:23

call here we're going to say if that tool call is the tool name I am going to Sprint calling tool name okay if it's the arguments I'm going to print the tool argument and I'm going to end up with nothing so that we don't go to a new line so we're actually going to be streaming everything okay so let's just see what this looks like oh my bad I just added that okay you see that so it go it goes very fast so it's kind of hard to see it I'm going to slow it down so you can see so

04:00:02

you can see that we as soon as we get the toour name we stream that we're calling the add tool then we stream token by token the actual Arguments for that tool then for the next one again we do the same we're calling this tool name then we're streaming token by token again we're processing everything Downstream from outside of the agent executor and this is an essential thing to be able to do when we're actually implementing streaming and acing and everything else in an actual application

04:00:36

so I know that's a lot but it's important so that is it for our chapter on streaming and Asing I hope this all been useful thanks now we're on to the final Capstone chapter we're going to be taking everything that we've learned so far and using it to build a actual chat application now the chat application is what you can see right now and we can go into this and ask some pretty interesting questions and because it's an agent because as iess is tools it will be able to answer

04:01:10

them for us so we'll see inside our application that we can ask questions s that require tool use such as this and because of the streaming that we've implemented we can see all this information real time so we can see that sub API tool is being used these are the queries we saw all that was in parallel as well so each one of those tools were being used in parallel we modified the code a little bit to enable that and we see that we have the answer we can also see the structured output being used

04:01:40

here so we can see our answer followed by the tools used here and then we could ask followup questions as well because this is conversational so we say how is the weather in each of those cities okay that's pretty cool so this is what we're going to be building we are of course going to be focusing on the API the back end I'm not front end engineer so I can't take you through that but the code is there so for those of you that do want to go through the front end code you can of course go and

04:02:18

do that but we'll be focusing on how we build the API that powers all of this using of course everything that we've learned so far so let's jump into it the first thing we going to want to do is clone this repo so we'll copy this URL this is repo orelio Labs line chain course and you'll just clone your repo like so I've already done this so I'm not going to do it again instead I'll just navigate to the line chain course repo now there's a few setup things that you do need to do all

04:02:53

of those can be found in the read me so we just open a new tab here and I'll open the read me okay so this explains everything we need we have if you were running this locally already you will have seen this or you will have already done all of this but for those of you that haven't we go through quickly now so you will need to install the UV Library so this is how we manage our pyth environment our packages we use UV on Mac you would install it like so if you're on Windows or Linux just

04:03:32

double check how you'd install over here once you have installed this you then go to install python so UV python install then we want to to create our VM our virtual environment using that version of python so the VM here then as you can see here we need to activate that virtual environment which I did miss from here so let me quickly add that so you just run that for me I'm using fish so I just add fish onto the end there but if you're using bash or zsh I think you can you can just run

04:04:10

that directly and then finally we need to sync I install all of our packages using UV sync and you see that will install everything for you great so we have that and we can go ahead and actually open cursor or vs code and then we should find ourselves within cursor or vs code so in here you'll find a few things that we will need so first is environment variables so we can come over to here and we have open AI API key larning chain API key and ser API API key create a copy of this and you would make this

04:04:54

your EMV file or if you want to run it with Source you can well I like to use mac. EnV when I'm on Mac and I just add export onto the start there and then enter my API Keys now I actually already have these in this local. EMV file which over in my terminal I'll just activate with Source again like that now we'll need that when we are running our API and application later but for now let's just focus on understanding what the API actually looks like so navigating into the 09 Capstone chapter we'll find a few things

04:05:36

what we're going to focus on is the API here and we have a couple of notebooks that help us just understand okay what are we actually doing here so let me give you a quick overview of the API first so the API we're using fast API for this we have a few functions in here the one we'll start with is this okay so this is our post Endo for invoke and this essentially sends something to our llm and begins a streaming response so we can go ahead and actually start the API and we can just see what this looks

04:06:11

like so we'll go into chapter 09 caps there and API after setting our environment variables here and we just want to do UV run uicorn main colon app reload we don't need to reload but if we're modifying the code that can be useful okay and we can see that our API is now running on Local Host Port 8000 and if we go to our browser we can actually open the dots for our API so we go to 8,000 slash dos okay we just see that we have that single invoke method it stripes the content and it gives us a small amount

04:06:53

of information there now we could try out here so if we say say hello we can run that and we'll see that we get a response we get this okay now the thing that we're missing here is that this is actually being streamed back to us okay so this is not a just a direct response this is a stream to see that we're going to navigate over to here to this streaming test notebook and we'll run this so we are using request here we are not just doing a you know the standard post request because we want to stream

04:07:34

the output and then print the output as we are receiving them okay so that's why this looks a little more complicated than just a typical request. post or request. getet so what we're doing here is we're starting our session which is our our post request and then we're just iterating through the content as we receive it from that request when we receive a token because sometimes this might be non we print that okay and we have that flush equals TRS we have used in the past so let's define that and

04:08:10

then let's just ask a simple question what is 5 + 5 okay and we we saw that that was it was pretty quick so it generated this response first and then it went ahead and actually continued streaming with all of this okay and we can see that there these special tokens are being provided this is to help the front end basically decide okay what should go where so here where we're showing these multiple steps of tool use and the parameters the way the front end is deciding how to display those is it's just it's being provided

04:08:52

the single stream but it has the SE tokens has a SE has se name then it has the parameters followed by the sort of ending of the step token and it's looking at each one of these and then the one step name that it treats differently is where it will see The Final Answer step name when it sees the final answer step name rather than displaying this tool interface it instead begins streaming the tokens directly at like typical chat interface and if we look at what we actually get in our final answer it's not just the

04:09:25

answer itself right so we have the answer here this is streamed into that typical chat output but then we also have tools used and then this is added into the little boxes that we have below the chat here so there's quite a lot going on just within this little stream now we can try with some other questions here so we can say okay tell me about the latest news in the world you can see that there's a little bit of a wait here whilst it's waiting to get the response and then yeah it's streaming a lot of

04:09:57

stuff quite quickly okay so there's a lot coming through here okay and then we can ask other questions like okay this one here how called is in Osa right now is five mtip by five right so these two are going to be executed in parallel and then it will after it has the answers for those the agent will use the another multiply tool to multiply those two values together and all of that will get streamed okay and then as we saw earlier we have the what is the current Daye and time in these places same thing so three

04:10:31

questions three questions here what is the current date and time in Dubai what is the current date and time in Tokyo and what is the current date and time in Berlin those three questions get executed in parallel against St I search at all and then all answers get returned within that final answer okay so that is how our API is working now let's dive a little bit into the code and understand how it is working so there are a lot of important things here there's some complexity but at the same time we've

04:11:05

tried to make this as simple as possible as well so let's just fast API syntax here with the app post invoke so our invoke endpoint we consume some some content which just a string and then if you remember from the agent execut a deep dive which is what we've implemented here or a modified version of that we have to initialize our asyn q and our streamer which is the Q coreback Handler which I believe is exactly the same as what we defined in that earlier chapter there's no differences there so

04:11:40

we Define that and then we return and this streaming response object right again this is a fast API thing this is so that you are streaming a response that streaming response has a few attributes here which again are fast API things or just generic API things so some headers giving instructions to the API and then the media type here which is text event stream you can also use I think it's text plane possibly as well but I believe this standard here would be to use event screen and then the more

04:12:15

important part for us is this token generator okay so what is this token generator well it is this function that we defined up here now if you again if you remember that earlier chapter at the end of the chapter we set up a a for Loop where we were printing out different tokens in various formats so we kind of pro postprocessing them before deciding how to display them that's exactly what doing here so in this block here We're looping through every token that we're receiving from our streamer We're looping through and

04:12:55

we're just saying okay if this is the end of a step we're going to yield this end of Step token which we we saw here okay so it's this end of end of St token there otherwise if this is a tool call so again we've got that W operator here so what we're doing is saying okay get the tool calls out from our current message if there is something there so if this is not nonone we're going to execute what inside here and what is being executed inside here is we're checking for the tool name if we have

04:13:29

the tool name we return this okay so we have the start step token the start of Step name token the tool name or set name whichever those you want to call it and then the end of the set name token okay and then this of course comes through to the front end like that okay that's what we have there otherwise we should only be seeing the tool name returned as part of first token for every step after that it should just be tool arguments so in this case we say okay if we have those tool or function arguments we're going to

04:14:07

just return them directly so then that is the part that would stream all of this here okay like these would be individual tokens right for example right so we might have the open curly brackets followed by query could be a token latest could be a token world could be a token news could be a token Etc okay so that is what is happening there this should not get executed but we have a we just handle that just in case so we have any issues with tokens being returned there we're just going to print as error and we're going to

04:14:41

continue with the streaming but that should not really be happening cool so that is our like token streaming Loop now the way that we are picking up tokens from our stream object here is of course through our agent execution logic which is happening in parallel okay so all of this is asynchronous we have this async definition here so all of this is happening asynchronously so what has happened here is here we have created a task which is the agent ex you to invoke and we passing our content we passing

04:15:18

that streamer which we're going to be pulling tokens from and we also set Theos to true we can actually remove that but that would just allow us to see additional output in our terminal window if we want it I don't think there's anything particularly interesting to look at in there but particularly if you are debugging that can be useful so we create our task here but this does not begin the task right this is a asyn iio create task but this does not begin until we await it down here so what is

04:15:54

happening here is essentially this code here is still being run and like a we're in an asynchronous Loop here but then we await this task as soon as we await this task tokens will start being placed within our que which then get picked up by the streamer object here so then this begins receiving tokens I know asyn code is always a little bit more confusing given the strange order of things but that is essentially what is happening you can imagine all this is essentially being executed all at the same time so

04:16:31

we have that is there anything else to go through here I don't think so it's all sort of boiler plates stuff for fast API rather than the actual AI code itself so we have that that's our streaming function now let's have a look at the agent code itself okay so agent code where would that be so we're using this agent executor invoke and we're importing this from the agent file so we can have a look in here for this now you can see straight away we're pulling in our API

04:17:04

Keys here just yeah make sure that you do have those now all of our C okay this is what we've seen before in that agent execut to Deep dive chapter this is all practically the same so we have our LM we've set those configurable fields as we did in the earlier chapters that configurable field is for our callbacks we have our prompt this has been modified a little bit so essentially just telling it okay make sure you use the tools provided we say You must use the final answer tool to provide a final answer to

04:17:42

the user and one thing that I added that I notice every now and again so I have explicitly said Ed to answer to users current question not pre-used questions so I found with this setup it will occasionally if I just have a little bit of small talk with the agent and beforehand I was asking questions about okay like what was the weather in this place or that place the agent will kind of hang on to those previous questions and try and use a tool again to answer and that is just something that you can

04:18:14

more or less prompt out of it okay so we have that this is all exactly the same as before okay so we have our chat history to make this conversational we have our human message and then our agent scratchpad so that agent can think through multiple tool use messages great so we also have the article class so this is to process results from Sur API we have our Ser API function here I will talk about that a little more in a moment because this is also a little bit different to what we covered before what

04:18:48

we covered before with C API if you remember was synchronous because we're using the Ser API client directly or the Ser API tool directly from line chain and because we want everything to be asynchronous we have had to recreate that tool in a asynchronous fashion which we'll talk about a little bit later but for now let's move on from that we see our final answer being used here so this is I think we defined the exact same thing before probably in that deep dive chapter again where we have

04:19:25

just the answer and the tools that have been used great so we have that one thing that is a little different here is when we are defining our name to Tool function so this takes a tool name and it Maps it to a tool to function when we have synchronous tools we actually use tool Funk here okay so rather than tool cartin it would be tool Funk however we are using a synchronous tools and so this is actually tool co-routine and this is why this is why if you if you come up here I've made every single tool

04:20:09

asynchronous now that is not really NE for a tool like final answer because there is no there's no API calls happening an API call is a very typical scenario where you do want to use async because if you make an API call with a synchronous function your code is just going to be waiting for the response from the API while the API is processing and doing whatever it's doing so that is an ideal scenario where you would want to use async because rather than your code just waiting for the response from the API it can instead go

04:20:45

and do something else whilst it's waiting right so that's an ideal scenario where you'd use async which is why we would use it for example with a Ser API tool here but for final answer and for all of these calculator tools that we built there's actually no need to have these as async because our code is just running through its executing this code there's no waiting involved so it doesn't necessarily make sense have these a synchronous however by making them asynchronous it means that I can do

04:21:17

tool care routine for all of them rather than saying oh if this tool is synchronous use tool. Funk whereas if this one is async use tool. cartin so just simplifies the code for us a lot more but yeah not directly necessary but it does help us write cleaner code here this is also true later on because we actually have to await our tool code which we can see over here right so we have to await those tool calls that would get Messier if we were using the like some sync tools some async tools so we have that we have our Q callback

04:21:59

Handler this is again that's the same as before so I'm not going to go through I'm not going to go through that we covered that in the earlier Deep dive chapter we have our execute tool function here again that is a synchronous this just helps us you know clean up code a little bit this would I think in the Deep dive chapter we had this directly placed within our agent executor function and you can do that it's fine it's just a bit cleaner to kind of pull this out and we can also

04:22:28

add more type annotations here which I like so execute tool expects us to provide an AI message which includes a tool call within it and it will return as a tool message okay agent exor this is all the same as before and we're actually not even using verose here so we could fully remove it but I I will leave it of course if you would like to use that you can just add a ifos and then log or print some stuff where you need it okay so what do we have in here we have our streaming function so this

04:23:03

is what actually calls our agent right so we have a query this will call our agent just here and we could even make this a little clearer so for example this could be configured agent because this is this is not the response this is a configured agent so I think this is may be a lot clearer so we are configuring our agent with our callbacks okay which is just our streamer then we're iterating through the tokens are returned by our agent using a stream here okay and as we are iterating through this because we

04:23:42

pass our streamer to the Callback here what that is going to do is every single token that our agent returns is going to get processed through our Q callback Handler here okay so this on LM new token on LMN these are going to get executed and then all of those tokens you can see here I'll pass to our Q okay then we come up here and we have this a it so that this aor method here is used by our generator over in our API is used by this token generator to pick up from the queue the tokens that have been put in the queue

04:24:29

by these other methods here okay so it's putting tokens into the queue and pulling them out with this okay so that is just happening in parallel as well as this code is running here now the reason that we extract the tokens out here is that we want to pull out our tokens and we append them all to our outputs now those outputs that becomes a list of AI messages which are essentially the AI telling as what tool to use and what parameters to pass to each one of those tools this is very similar to what we covered in that deep

04:25:07

dive chapter but the one thing that I have modified here is I've enabled us to use parallel tool calls so that is what we see here with this these four lines of code we're saying okay if our tool call includes an ID that means we have a new tool call or a new AI message so what we do is we append that AI message which is the AI message chunk to our outputs and then following that if we don't get an ID that means we're getting the tool arguments so following that we're just adding our AI message chunk to the most

04:25:46

recent AI message Chunk from our outputs okay so what that will do is it it will create that list of AI messages would be like AI message one and then this will just append everything to that AI message one then we'll get our next AI message chunk this will then just append everything to that until we get a complete AI message and so on and so on okay so what we do here is here we've collected all our AI message chunk objects then finally what we do is just transform all those AI message chunk

04:26:26

objects into actual AI message objects and then return them from our function which we then receive over here so into the tool cuse variable okay now this is very similar to The Deep dive chapter again we're going through that that count that Loop where we have a Max iterations at which point we will just stop but until then we continue iterating through and making more tool calls executing those tool calls and so on so what what is going on here let's see so we got our tool calls there's

04:26:59

going to be a list of AI message objects then what we do with those AI message objects is we pass them to this ex cuute tool function if you remember what is that that is this function here so we pass each AI message individually to this function and that will execute the tool force and then return us that observation from the tool okay so that is what you see happening here but this is an async method so typically what you'd have to do is you'd have to do await X you tool and we could do that so we could do a

04:27:40

okay let me let me make this a little bigger for us okay and so what we could do for example which might be a bit clearer is you could do tool OBS equals an empty list and what you can do is you can say for Tool call oops in tool calls the tool observation is we're going to append execute tool call which would have to be in a weit so we' actually put your weight in there and what this would do is actually the exact same thing as what we're doing here the difference being that we're doing this tool by Tool

04:28:17

okay so we are we're executing async here but we're doing them sequentially whereas what we can do which is better is we can use asyn I gather so what this does is gathers all those Co routines and then we await them all at the same time to run them all asynchronously they all begin at the same time or almost exactly at the same time and we get those responses kind of in parallel but of course it's saying so it's not fully in parallel but practically in parallel cool so we have that and then that okay we get all of

04:28:55

our tool observations from that so that's all of our tool messages and then one interesting thing here is if we let's say we have all of our AI messages of all of our tool cores and we just append all of those to our agent scratch Pad right so let's say here we're just like oh okay scratch Pad extend and then we would just have okay we' have our tool calls and then we do agent stretch PCT send tool OBS all right so what what is happening here is this would essentially give us something that looks like this

04:29:34

so we have our AI message say I'm just going to put okay we'll just put tool call IDs in here to simplify a little bit this would be tool call ID a then we would have ai message tool call ID B then we'd have tool message let's just remove this content field I don't want that and Tool message tool call ID B right so it would look something like this so the the order is the tool message is not following the AI message which you would think okay we have this tool qual ID that's probably

04:30:11

fine actually when we're running this if you add these to your agent scratch pad in this order what you'll see is your response just hangs like nothing nothing happens when you come through to your second uh iteration of your agent call so actually what you need to do is these need to be sorted so that they are actually in order and it doesn't actually doesn't necessarily matter which order in terms of like a or b or c or whatever you use so you could have this order we have ai message tool message AI message tool

04:30:45

message just as long as you have your tool call IDs are both together or you could know invert this for example right so you could have this right and that that will work as well it's essentially just as long as you have your AI message followed by your tool message and both of those are sharing that tool call ID you need to make sure you have that order okay so that of course would not happen if we do this and instead what we need to do is something like this okay so I made this a lot easier to read okay so we're

04:31:21

taking the tool call ID we are pointing it to the tool observation and we're doing that for every tool call and to Observation within like a zip of those okay then what we're saying is for each tool call within our tool calls we are extending our agent scratch pad with that tool call followed by by the tool observation message which is the tool message so this would be our this is the AI message and that is the tool messages down there okay so that is what it's happening and that is

04:31:55

how we get this correct order which will run otherwise things will not run so that's important to be aware of okay now we're we're almost done I know there's we just been through quite a lot so we continue we incre increment our count as we were doing before then we need to check for the final answer tool okay and because we're running these tools in parallel okay because we're allowing multiple tool calls in one step we can't just look at the most recent tool and look if it is

04:32:25

it has the name Final Answer instead we need to iterate through all of our tool calls and check if any of them have the name final answer if they do we say okay we extract that final answer call we extract the final answer as well so this is the direct text content and we say okay we have found found the final answer so this will we set to True okay which should happen every time but let's say if our agent gets stuck in a loop of calling multiple tools this might not happen before we break based on the max

04:32:58

iterations here so we might end up breaking based on Max iterations rather than we found a final answer okay so that can happen so anyway if we find that final answer we break out of this for Loop here and then of course we do need to break out of our wow Loop which is here so we say if we found the final answer break okay cool so we have that finally after all of that so this is our you know we've executed our tool our agent steps and iterations has process we've been through those finally we come

04:33:35

down to here where we say okay we're going to add that final output to our chat history so this is just going to be the text content right so this here get direct answer but then what we do is we return the full final answer call the full final answer call is basically this here right so this answer and tools used but of course populated so we're saying here that if we have a final answer okay if we have that we're going to return the final answer call which was generated by our llm otherwise we're

04:34:10

going to return this one so this is in the scenario that maybe the agent got caught in a loop and just kept iterating if that happens we'll say it will come back with okay no answer found and it will just return okay we didn't use any tools which is not technically true but it's this is like a exception handling event so it ideally it shouldn't happen but it's not really a big deal if we're saying okay there were no tools use in my opinion anyway cool so we have all of that

04:34:43

and yeah we just we initialize our agent executor and then I mean that that is our agent execution code the one last thing we want to go through is the Ser API tool which we will do in a moment okay so Ser API let's see what let's see how we build our Ser API tool okay so we'll start with the synchronous Ser API now the reason we're starting with this is that it's actually it's just a bit simpler so I'll show you this quickly before we move on to the async implementation which is what we're using

04:35:20

within our app so we want to get our set API API key so I'll run that and we just enter it at the top there and this will R so we're going to use the sub API SDK first we're importing Google search and these are the input prameters so we have our API key we're using we say want use Google we our question is so query so Q for query we're searching for the latest news in the world it will return quite a lot of stuff you can see there's a ton of stuff in there right now what we want is contained within

04:36:00

this organic results key so we can run that and we'll see K is talking about you various things pretty recent stuff at the moment so we can tell okay that is that is in fact working now this is quite messy so what I would like to do first is just clean that up a little bit so we Define this article base model which is pantic and we're saying okay from a set of results okay so we're going to iterate through each of these we're going to extract the title source link and the snippet so you can see

04:36:36

title source link and snippet here okay so that's all usedful we'll run that and what we do is we go through each of the results in organic results and we just load them into our article using this class method here and then we can see okay let's have a look at what those look like it's much nicer okay we get this nicely formatted object here cool that's great now all of this what we just did here so this is using sub apis SDK which is great super easy to use the problem is that they don't offer

04:37:18

a async SDK which is a shame but it's not that hard for us to set up ourselves so typically with a synchronous requests what we can use is the aiio HTTP Library it's well it's you can see what we're doing here so this is equivalent to requests Dot get okay that's essentially what we're doing here and the equivalent is literally this okay so this is the equivalent using requests that we are running here but we're using asyn Code so we're using AI Hep client session and

04:38:00

then session. getet okay with this async width here and then we just await our response so this is all yeah this is what we do rather than this to make our code async so it's really simple and then the output that we get is exactly the same right so we still get this exact same output so that means of course that we can use that articles method like this in the exact same way and we get we get the same result there's no need to make this article from sub API result asnc because again like this this bit of code

04:38:37

here is fully local it's just our python running everything so this does not need to be async okay and we can see that we get literally the exact same result there so with that we have everything that we would need to build a fully asynchronous Sur API tool which is exactly what we do here for Lang chain so we import those tools and I mean there's nothing is there anything different here no this is exactly what we we just said but I will run this because I would like to show you very quickly this okay so this is how we were

04:39:13

initially calling our Tools in previous chapters because we we were okay mostly with using the the synchronous tools however you can see that the funk here is just empty right so if I do type just a non-type that is because well this is an async function okay it's an async tool sorry so it was defined with async here what happens when you do that is you get this Co routine object so rather than Funk which is it isn't here you get that cartine if we then modified this which would be kind of okay let's just remove

04:39:58

all the ayns here and the await if we modify that like so and then we look at the set API structure tool we go across we see that we now get that funk okay so that is that is just the difference between an async structured tool versus a sync structured tool we of course on async okay now we have K again so important to be aware of that and of course we we run using the sub API care routine so that is that's how we build the sub API tool uh there's nothing I mean that is exactly what we did here so I don't need

04:40:45

to I don't think we need to go through that any further so yeah I think that is basically all of our code behind this API with all of that we can then go ahead so we have our API running already let's go ahead and actually run also our front end so we're going to go to documents orelo line chain course and then we want to go to Chapters 09 Capstone app and you will need to have npm installed so to do that what do we do we can take a look at this answer for example this is probably what I would

04:41:22

recommend okay so I would run Brew install node followed by Brew install mpm if you're on Mac of course it's different if you're on Linux or Windows once you have those you can do npm install and this will just install all of the oop sorry mpm install and this would just install all of the node packages that we need and then we can just run npm run Dev okay and now we have our app running on locost 3000 so we can come over to here open that up and we have our application can ignore this so in here we can begin just

04:42:00

asking questions okay so we can start with quick question what is 5 + 5 and you see so we have our streaming happening here it said the agent wants to use ad tool and these are the input parameters to the ad tool and then we get the streamed response so this is the final answer tool where we're outputting that answer key and value and then here we're outputting that tools used key and value which is just an array of the tools being used which just functions add so we have that then let's ask

04:42:35

another question this time we'll trigger Ser API with tell me about the latest news in the world okay so we can see that's using C API and a query is latest world news and then it comes down here and we actually get some citations here which is kind of cool so you can also come through to here okay and it teses through to here so that's pretty cool unfortunately I just lost my chat so fine let me I can ask that question again okay we can see that to us set API there now let's continue with the next

04:43:21

question from our notebook which is how cold is in I like right now what is five M by five what do you get when multiplying those two numbers together I'm just going to modify that to say in Celsius so that I can understand thank you okay so for this one we can see what did we get so we got current temperature in ow we got multiply 5 by five which our second question and then we also got subtract interesting that I I don't know why it did that it's kind of weird so it it decided to use oh ah okay so this is

04:44:00

okay so then here it was okay that kind of makes sense does that make sense roughly okay so I think the the conversion for Fahrenheits Celsius is say like subtract 32 okay yes so to go from Fahrenheit to Celsius you are doing basically Fahrenheit minus 32 and then you're multiplying by this number here which the iume the AI did not oh it roughly did okay so subtracting 36 like 32 would have given us four and it gave us approximately two so if you think okay multiply by this it's practically multiplying by 0.5 five

04:44:42

so halfing the value and that would give us roughly 2° so that's what this was doing here kind of interesting Okay cool so we've gone through we have seen how to build a fully fledged chat application using what we've learned throughout a course and we've built quite a lot if you think about this application you're getting the real time updates on what tools are being used the parameters being input to those tools and then that is all being returned in a streamed output and even

04:45:17

in a structured output for your final answer including the answer and the tools that we use so of course you know what we built here is fairly limited but it's super easy to extend this like you could maybe something that you might want to go and do is take what we've built here like Fork this application and just go and add different tools to it and see what happens because this is very extensible you can do a lot with it but yeah that is the end of the course of course this is just the beginning of

04:45:50

whatever it is you're wanting to learn or build with AI treat this as the beginning and just go out and find all the other cool interesting stuff that you can go and build so I hope this course has been useful informative and gives you an advantage in whatever it is is you're going out of this build so thank you very much for watching and taking the course and sticking through right to the end I know it's pretty long so I appreciate it a lot and I hope you get a lot out of it thanks bye