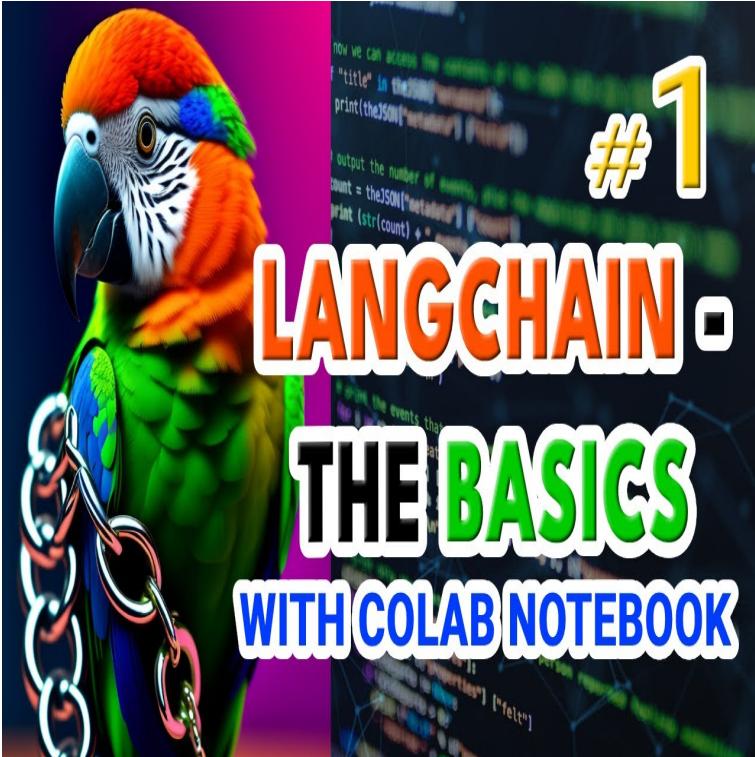


# **LangChain How to and guides**



## LangChain Basics Tutorial #1 - LLMs & PromptTemplates with Colab

hi everyone so in this video I'm going to be talking a little bit about an intro to Lang chain and actually over the series of videos I'm going to go quite in depth about what you can do with Lang chain all about how different parts of it work and how you can build applications with this in this particular video I'm going to just start off by talking about why we want Lang chain and then jumping into a little bit about what it is and how we can get started with the basics of using it to access large language models so first up large language models as apps so large language models are great at generating something new from scratch or via sort of a conditional generation so for example basically all they're doing is giving us probability distribution of the next token based on a set of tokens that we're putting into this and it turns out this is great for making stories for doing a whole bunch of different tasks only recently in the past year or so people have worked out ways to fine-tune these with rlhf and a variety of other sort of add-ons in the sort of prompt space to get the best out of these kind of models and it turns out once we can do that we can get some really good results from these large language models the problem is that they still can't access the traditional software stack in a normal way so if we

want to build like an app with one of these things we need some kind of way to interface between the large language model and the traditional software stack if we want to build things like chat Bots or something that uses search or external data then often a large language model alone is not going to be enough a perfect example of this is the way that large language models are not storing state so each call to the large language model is individual if we want to have a conversation State going with the large language model then each time that we call that model we're going to have to pass back in at least if not all of the conversation we've had at least the key parts of that and that raises a whole bunch of challenges or do we just pass it in verbatim which might be fine if the conversation only goes for a few rounds but if you've got a long conversation you're going to probably have to summarize it or find other ways to do this because the large language models have a very finite span of tokens that we can pass into these things most of these things are going up to sort of 10 24 2048 tokens that we can pass in some of the new ones can go much bigger and maybe this is something we'll see or get better over time but it's still you know that's still a finite amount and if you wanted to do something with large documents with other things you need ways to think about how do we interact with the language model so what is Lang chain so Lang chain really is a tool or a framework that allows us to build fully featured apps that interact with the normal software stack it allows us to manage our use of large language models and prompts and then how we deal with both the language models and prompts it also allows us to integrate to the traditional software stack like apis tools things like calculators Etc and then also things like databases and various data sources that we want to look things up on so often people ask okay why not just make our own for this kind of thing and while it can be easy to build something simple with prompting large language model once you start doing things that get a little bit more complicated with manipulating The Prompt with adding in memory Etc things get complicated very quickly so the open source Community is really picked up on Lang chain as being an exciting project it's been developed really quickly by a large group of people and it's becoming a de facto standard for ways to build apps around these large language models everything starts with a prompt in fact everything in Lang chain is built around prompts their key to everything that you're going to be doing with Lang Chan so let's talk a little bit about what prompting is and the idea of prompt templates in Lang chain so what is prompting basically prompting is just getting the model to generate text conditioned on some input text so these models are generally built to just keep predicting the next token which is often going to be the next word and assemble a text that way so it's taking the input that you've given it the prompt and then continuing from that the prompts themselves have a massive influence on the output so much so that this has led to a whole field of what people

are now calling prompt engineering early on prompts were very simple you would have very simple questions that people would use when was Marcus Aurelius emperor and you would hope that the large language model had learned enough in its weights to basically answer that and just generate a output for the end at the end of that that it would have some sort of end of a sentence token so it would stop generating two and as the models have gotten better they've gone from these kind of simple prompts to now much more complicated prompts and now we see prompts that people are giving whole sort of contacts and a variety of things in the prompt to generate the output one of the papers that were key in this was the instruct GPT paper and while this wasn't the first paper to use the reinforcement from Human feedback technique it certainly was one of the main ones in establishing that this was really good for fine-tuning your models to take prompts that could be much more useful than just very simple sort of prompts and we can look at these kind of prompts from the instructor GPT paper we can see that there are different types of prompts in there the paper itself has got a whole bunch of different prompts in there you can also see in this one here that we've got the idea of what we call few shot template going on so the idea of where we're not only just going to tell the prompt something we're actually going to give it some examples as well and then we're going to get it to generate a new answer or a new example there so in Lang chain this is all handled by prompt templates so one of the key things is to sort of know what the different parts of prompts are because what you will do is in LinkedIn you will build a prompt and you will inject different things in there so an example of this would be here we've got a simple sort of prompt that starts off with a context it tells the model that you're a digital assistant who helps users create business names based on descriptions of businesses you provide catchy short catchy names then it gives some examples and the examples in a special way so it's basically giving them in a way that the model will then learn to generate given this example this is this would be an example of a short catchy name then we can pass in the question or the task and then we can prime it for the output here and this is something that is done a lot in Lang chain but it's also done in a lot of the other prompting systems that are out there so you'll see things like this where here it's basically saying here's an message to me injecting in some information here are some bullet points for a reply again injecting some information and then prompting the model to give us a response with this so these this can be done for a whole bunch of different tasks it can be done for classification it can be done for a variety of different things all right let's jump into the code and have a look at how Lang chain handles both normal prompts and few shot learning prompts where we're going to be giving it examples as well okay so let's jump into the code so here you can see I've basically got a collab set up the collab will be in the description so you can just play with

this yourself I first off we're just installing some packages so I'm installing the openai package so that we can use the gpt3 DaVinci models we're installing the Lang chain package itself we're also installing the hugging face Hub here so this basically allows us to Ping the end points for open AI with the open AI one and also ping the endpoints for the hugging face Hub here as well so you will need to go in get your keys for these and then once you've basically got a token for each of these set up you can just drop them in there and then this will basically work for you all right so if I come down here first off all we're going to do is basically load a language model and prompt it with some text so you can see here in this first one I'm just bringing in the open AI model the open AI model that I'm going to use here is the the text DaVinci zero zero three which at the time of recording is the latest one I'm going to use quite a high temperature so we get a random output I'm also going to set it to just give us a Max tokens of 256. in this case it's probably Overkill but uh you know this is how you can determine how much output you want from the model so now we've basically got our prompt here of why did the chicken cross the road and all we're going to do is basically just pass that prompt into the LL to the large language model and then print that out and see what the the output that we get is and we can see that okay it came back with you know to to get to the other side obviously not massively original we could even run this again it may give us something different although that enter is probably you know pretty pretty common to get there if we change this to to a duck or something we may even get something a little bit different okay we're still getting the same thing I the by playing around with this though you can then start to see like okay how much effect your prompt is going to have so that's using the open AI model if you wanted to use one of the free hugging face models we can basically do the same thing and here I'm just setting up a hugging face uh version of the Google flan T5 XL model so this is not the biggest one the biggest one is XXL but I'm not sure they're actually supporting you know that on their endpoints at the moment unless you've you're paying for it so this one is definitely nowhere near as big a model as The DaVinci open AI DaVinci model but you'll see that we'll still get some sort of coherent response from this and you'll notice that this one that this response is different if we run it again okay we're still getting this the same sort of thing but we're getting different responses from each of these but it's basically doing the same thing we're picking a large language model and getting a response back from it so that is just a sort of raw prompt if we want to start actually setting up some prompt templates this is where Lang chain starts to become really useful so here we can sort of say okay we want to make a a little app that our app is going to take in a description of a restaurant and create a name for that restaurant so this is my prompt my prompt is I want you to act as a naming consultant for new restaurants return a list of rest return a list of restaurant

names just change that of restaurant names it should be short catchy and easy to remember it should relate to the type of restaurant you are naming right what are some good names for a restaurant that is and then we're going to pass in a restaurant description here and this is where Lang chain will will basically enable us to our description into the prompt here so if we had a user typing something in they wouldn't actually see this prompt right they would actually just type in what their their description was and then using this prompt we would ping the back end model and get the output for this so to do this we set up a prompt template and the prompt template so you'll notice this is very similar to an F string in Python here the prompt template we basically just just work out what the the template is which in this case is going to be our restaurant template and we've we're going to what are we going to inject what are going to be the in input variables here so this is going to be the restaurant description here so we can just go through set that up and now we've created this prompt template and you know here is actually just creating it again yeah so now if we want to you know try something out we can basically look and see okay would that you know what's this going to do so one of the ways we can do without peeing the actual large language model itself we can just basically say all right for this prompt template format it with the restaurant description equals and we're going to go for this description here so you'll see that when we do this we're going to get you know I the whole prompt plus our restaurant description that's been injected into that prompt so a restaurant description was a Greek place that serves fresh lambs of Lucky's and other Greek food is what we want there I and you can see that we're getting the full thing there so now if we basically take this we can I we can and you see I've done a few of these up here so we'll play around with a few of these now I can basically say right okay the lane change I'm going to set up a chain now I'll talk about a lot more about what chains are in the next video but this is basically just a very simple large language model chain that we're going to prompt it and it's going to give us you know we it's going to return the output of the model here so our chain is going to be a large language model chain we're going to pass in the language model that we're using in this case that's the open AI model and we're going to get back a you know we're going to pass in our prompt template that we've created and then we're going to inject into that this description so we'll start off with just the description the basic description it's for the Greek restaurant so let's see when we run this what are we getting back from open AI that we can use and we can see sure enough it's given us 10 restaurant names that we could go for the Olive Tree maybe yeah I guess you know Athena's kitchen a gnd lights Souvlaki Grill all of these are related to our description up here of the Greek place so if we wanted to try something like you know a burger place that is themed with baseball memorabilia so now all I'm doing here is just injecting in the

different restaurant description into the exact same prompt template that we had before right we haven't changed the prompt template and we can see now sure enough we've now got things related to Burgers and to baseball that we expected here all right so the third one here was I was trying to basically get it to sort of say something like a cafe that is that has live Hard Rock music and memorabilia some may say even Hard Rock Cafe but let's say will it actually you know generate something like that foreign it's generating things sort of around this and they're certainly you know appropriate names for what it is that we're going for here all right so that should give you a sense of like the basics of just setting up a standard prompt template and setting up your standard you know language model to get started the next level for doing this and I've taken this from the link chain examples is that what we call a few shot temp a few shot prompt template so here we're actually going to pass in some examples and it's not drastically different than what we've done before right we've already got we've already got it you know so that we're going to set up up for you know a prompt a prompt template and in this case we're going to basically we're going to basically give it a word and we want it to return the antonym for that word so we've given some examples here of like the word happy it should return sad the word tall it should return short Etc so now because we're passing in multiple things we're passing in you know some examples and stuff as well in here what we're going to do is we're going to have our prompt template all right and we're going to have a prefix and a suffix to this so we're now just sort of building up the parts of the prompt so that it all goes together with this so we've got our you know our standard sort of example of the you know where we're setting up the the template we've then got our few shop templates where we're going to pass in examples we're going to start you know a prefix for this we're going to start with a suffix for this so you can see that what it's going to do is that it's going to basically create a prompt and we should be able to actually you know we should be able to to see this by by running this we can actually see that okay it this is the whole prompt that it's creating right it's created give the antonym of every input that was our prefix that we had there then it's basically giving you know some examples of these right these are the examples that we set up here and then finally we're passing in the word that we want to do this for we're going to basically return this back right so for what we've got here now if we come down and run this we can basically do the exact same thing that we did with the first prompt we can just pass in here our input is going to be big and I can just basically run this and we're getting back small right which is the antonym in this case the the idea here though is that we're not actually seeing the whole prompt so you often won't want to show your users your whole prompt for something like this so this is a simple way of you know making prompts with the prompt templates you've got your standard prompt

template and then you've got your few shot Tom few shot prompt template as well remember the key thing with the fuse shot one is going to be the examples so you could actually have a whole series of examples in here all right and that's the example is just showing an example of what the input is and then what's going to be the corresponding output that you would expect for that particular thing okay in the next video we're going to be looking at tools and chains in Lang chain and how they come together to start being able to make the building block of apps so if you're interested in finding out more about this please subscribe to get notified of the latest videos Etc see you in the next video bye for now



## LangChain Basics Tutorial #2 Tools and Chains

alright welcome back to the second video in the Lang Chain video series in this video we're going to be looking at tools and chains so these are the key building blocks in langchi that you're going to be using for building a lot of the apps and a lot of the things that you're going to be doing so these are different things but you'll see that they interrelate quite a lot as well so first up what are tools in Lang chain tools are the individual components that Lang chain uses as links in the chain so these are a variety of sort of little sub modules that you can use when combined with a language model to do a task and that's what makes up a chain overall so that some of the example tools would be like the python read evaluate print Loop that's going on this allows you to basically export something out as code and run it as code and take the output the printed output from that and feed it back into a model with a chain like that you've got things like the search search engine API you've got wolf from alpha which allows you to query the Wolfram Alpha engine for facts and data and then bring that back in and use that in a a large model or you know in one of the other tools that you're using just like your query wall from alpha can also query things like Google search and the News API so these allow you to

basically get external data and external results and feed them you know back into your app that you're using so what is a chain in Lang chain so a chain is really just you know made up of links of tools so you can kind of think of like each of the tools as being a link in an overall chain these can be as simple as linking a prompt template and a large language model and then you've got a llm chain which is one of the most common chains that you will see in Lang chain it also can be much more complicated so you can get things with multiple links going through multiple steps of a whole process which might include multiple large language models in there as well as a variety of different utility tools Etc the main thing to understand is that in a chain the output of one link is what's going to become the input of the next Link in there so when these chains are running you might take something as an output from a large language model feed that into a query of Wolfram Alpha bring that back run that again through a large model to formulate a response that you're going to give back to the user that's an example of a chain so there are three categories of chains currently we have generic change which are basically used for building other chains at utility chains this is where a lot of the utilities and tools actually are then you also have asynchronous chains for doing async functions and tasks like that so looking at the generic chains by far the most common chain that you will see is a large language model chain and this is basically you know going to take some sort of input format that with a prompt template run that through a large language model and return and this is the kind of chain that we looked at very briefly in the first video second kind of chain that you'll see in generic chains uh transformation chains so this allows you to basically take something and run some code in a function on the actual inputs or outputs of a of another chain or another Link in another chain so that could be simply taking the response back from a large language model and then running it through a regex to check if there's something in it or you're doing a variety of things like that and that can be you know both on the input or on the output for this and then the other generic chain is sequential chains so this is just basically multiple chains and allowing you to join them together so you'll find that while chains and are made up of individual tools they can also be made up of other chains that get joined together so utility chains is where a lot of the magic happens and there are a variety of these so this is not all of them there's there's a lot more than this and I expect that there'll be a lot more added over time as well the pound chain is a very interesting one and maybe I'll do a video just about this because this is actually based on an interesting paper of using a language model that that converts the reasoning of a question to python code and then allows you to run that and return that response back another interesting one is like the SQL database chain so this allows you to take natural language convert it to the SQL query and then return that information which you might then feed back in to another large

language model chain with a different prompt and then take the output of that and give that back to your user so you see that this is a quite common thing to basically go through an llm chain go out to something else come back to another llm chain and then go back to the user The Bash chain is another sort of thing where it allows you to run bash commands the requests chain allows you to go out and request a particular HTML page and get the data from that and bring that and pass that back to the next Link in the chain and then you've got a variety of API chains which let you do things like query different apis that are already set up inside of Lang chain all right so let's jump into the code and look at how we can you know build some of these things with code okay in this notebook I'm going to go through the tools and then linking those tools to basically make chains I'll show you a few common chains and we'll look at some of the others as we go through as well so you want to make sure that you've got link chain installed you've got your libraries for the various large language models installed set up your API Keys Etc the first one is just going to be a really simple chain and this is by far the most common chain that you will see and this is basically just linking a large language model with a prompt and then feeding something with a prompt template and feeding something into this so here you can see I'm setting up the open AI DaVinci 3 Model we're going to set temperature to zero we're going to just set the max tokens a lot of these will be set by default if you know that this is a standard model that it defaults to I've got a little article in here so what I'm going to be doing is actually fact extraction so here I've basically taken an article all about coinbase and so this is quite a long article if we look at it it's 3 500 characters and what we want to do is basically extract down the key facts from this and then we're going to play around with that and try rewriting those facts into kind of new piece of content so first off we need our prompt template so our prompt template is going to basically take in the input that's what we've got up here and then we need the actual prompt so the prompt here is going to be extract the key facts out of this text don't include opinions give each fact a number and keep them for short sentences and then I'm going to basically just the input is going to be this text input that we've got here all right so making the chain is actually pretty simple we basically just say we're going to be using the large language model chain we pass in the large language model and then we pass in the prompt template we're going to be using so here I've got the fact extraction prompt and we're passing that in all right and then we can basically run it and you can see that after we run it sure enough this has gone through and it's done a pretty good job of getting out the facts from our article all right so you could play around with getting better prompts here there's certainly this is like the whole world of prompt engineering it's worth testing out different prompts for the kind of content that you want to use I but here enough it's done a pretty good job we've got 10 facts out

of this that have each come from the article that we've got above there so now what we want to do is chain we're going to make a new one and then we're going to chain some of these things together so the next one I'm going to do is also a large language model chain and this is going to be taking those facts and rewriting them but we're going to rewrite them as if it was like an investor report here so you see here we're going to say okay you're a Goldman Sachs analyst take the following list of facts and use them to write a short paragraph for investors don't leave out key info we could also put probably should put in something there don't make up info as well but here's the facts that we're going to pass in so again this is a large language model chain we pass in the language model we're still using the original one we defined above we pass in the prompt template and then we can run this and you can see sure enough it's come back and it's written a pretty coherent nice little article there and it's way shorter than what we had before right so the idea here is that we're trying of doing a little bit of a kind of summary but we also want to maybe take those facts and put them into something and this is where you could even play with this just to give you an idea I've made another one of these that takes these facts and turns them into triples for a Knowledge Graph so here we're saying okay take the following list of facts and turn them into triples for a Knowledge Graph and you can I'm passing in the facts just like I passed in the facts that we had up there again Define your chain it's going to be a large language model it's going to have a prompt template and then you can see sure enough this has gone and made triples now I could have got it to do in a particular style like if I wanted to show these in d3js or something like that or it's some sort of format for neo4j or something like that I would be able to do that in that style to get the triples but just here without asking it for a specific style it's got the sense of that like coinbase released fourth quarter earnings coinbase generated and it's got revenue and maybe we would want to guide it a little bit more with this but just to show you some of the ideas that you could do with this thing I next up we want to chain these together so now we're actually taking a sort of small chain and chains can be made out of either tools or they can be made out of other chains so we're taking a small chain here that we've got or the couple of small chains that we've got up there the fact extractor and then the sort of investment analysis rewriter kind of thing and we're putting them together and we're just going to do that with our simple sequential chain all right so the simple sequential chain is just like a standard sort of sequential model in something like Keras or pie torch where you're basically just going from A to B to C you're not really doing anything fancy in there we've also got this idea of that the inputs from one will become the output sorry the outputs from one of them will become the inputs for the next chain so you can see here we've basically set the full chain we're going to have our Factory extraction chain and we're going to

have our investor update chain here and now when I take the original article and run it through it's going to do both those things so you can see here now it's basically done the fact extraction and now it's done the rewriting and then it's finished the chain here so I this is one of the ways that I that we're able to link these things together rather than having to rewrite code to do to do one thing and then do the other thing Etc and we sure enough we can see okay if we take this the response out that we got we've got this our response back of what the investor update did even though we passed in the original article now so you could try this for testing the idea of using this versus like just a one summary chain or something if you wanted to incorporate the triples somewhere that's something that you could do with this kind of thing as well so by far the most common change that you will see are made up of just a large language model and a prompt template there are a bunch of other cool chains and tools though in Lang chain and the next one I'm going to show you is the pound math chain so I may make a video about this whole paper and the concepts in this but just let me quickly show you to show it here I so what this basically does is it uses a different large language model we're using one of the code models here and what we're going to basically do is this is going to be used for when we've got some kind of number problem here so this is an example from the Lang chain documentation so Jan has three times the number of pets you can see there's basically like a math equation and what the what we're going to be what we're basically prompting the model to do here and we're using an inbuilt prompt that's there but we're prompting it to basically take this written statement and turn it into a small python function which will then calculate the math rather than just guess it with a language model and then return the output for them so the idea here this one's from uh one of the flan papers the recent flan paper and I think this one is actually an interesting one so this is pretty simple math right the cafeteria had 23 apples if they used 20 for lunch and bought six more how many apples do they have the the issue with this is that if you've used the big language models yes they will probably get this right but if you're using something smaller if you're using even just like the T5 models the majority of the G5 models will get something like this wrong rather than just rely on one of those sort of to do it here we can basically use this pal system where we're just getting it to basically take this data and rewrite it and you can see here it's writing a python function it uses the doc string to talk to put in what we actually had up there we start off with Apple's initial so it's making just the variables and assigning them Apple's used apples bought and then apples left equals Apple's initial minus Apple's use plus Apple's bought sure enough then it gives us our result and it gives us the nice accurate result here so rather than rely on the language model to do this what we're doing is just getting the right language model to rewrite it in something that we can basically run in the python read

evaluate print Loop and then we can take that output now we could take this output and then feed it back into another large language model and have it rephrase it conversationally so that it could tell you ah the amount of apples left is or we could just take the output straight from this module another example of a tool chain which is can be really useful are the API tool chains so here I'm just showing you the example of the one for for weather information so this basically we're setting up the API that we're going to be using for this and what this is going to do is it's going to take in our queries so we're basically just setting up large language mod by default this is going to be the DaVinci zero zero three we're setting up the docs for it so even the docs and then it's able to then write an API call based on those docs and that's what the this is going to output and then this is going to then query that API with that call and see the result we get back so here you can see I'm asking it what is the temperature in bedok Singapore in degrees Celsius and it writes this it writes this whole URL for basically querying this here let's work out that okay temperature unit should be Celsius and sure enough it basically gives us back the temperature right now in gives its location and the answer back so what it actually returned was right and then that has then gone into a larger in back into the language model to rewrite it in something rather than just a adjacent format into something that a user would understand here I can see we can also ask it a little bit more vague questions and obviously it can only answer what the API can give you back here it's basically giving this back and it's telling us yeah that okay the a number of things in this Json response indicate that it's raining the danger with this one that you need to be careful is that often the docs plus the uh the URL plus the jsons will go beyond the number of tokens that your large language model can handle so you can get errors if you're going above 4000 tokens on The DaVinci the the other thing too is to think about is that this is pretty expensive right if we're paying two cents per thousand tokens and we've just put in 4 000 tokens just to get back the weather or something this is not always the most efficient way to do it but it does show you that link Ching can do these things and you could write some of this to make your own API calls for something that you want all right in the next video we're going to be looking at the whole sort of chains around conversation and incorporating memory into those so I'll see you there don't forget if you find this information useful click subscribe so that you get told when the new videos are coming out thanks for watching



## ChatGPT API Announcement & Code Walkthrough with LangChain

hi everyone so earlier today openai announced the chat gbt API I thought it would take a quick look at what they announced and some interesting points about what they announced and then also have a look at how you can implement this in code with just the open AI package by itself and then also with Lang chain as well so they've announced the API and amazingly it's available straight away today which is fantastic they also announced some interesting things about whisper which I might cover in a different video but they gave some nice examples of customers who are using the chat GPT API starting to build apps with it there's some interesting things in the actual blog post here and this is really going into sort of what the model is and a little bit about it so one of the key things is that the model is actually 10 times cheaper than the previous GPT 3.5 models so that is fantastic right it suddenly means that a lot of things that were going to be too expensive to do with the open AI apis now suddenly become much more interesting to do you can now get you can now use a lot more tokens and do multiple calls to the API to be able to put something together which again give back to a user so the pricing is definitely one of the big things you can see they've also called

this new API the gbt 3.5 turbo API and it lives up to its name so far in testing I'm finding that it's much faster than the previous API and the text DaVinci 003 and it's able to process things much better and probably the quality is much better there as well one of the other interesting things that they announced along with this was this whole idea of the chat markup language so this is just let's just jump in here they've basically made available some of the internal token systems that the model uses so many of the large language models that are Transformers have what they call special tokens the most common ones just being like a start token and an end token but a lot of them use tokens for different things so you have things like the Macaw model which has tokens for contacts for questions for answers and they're kind of going down an interesting direction with this but because this is all built for chat what they're actually doing is they're making these sort of tokens represent either the assistant either their side of the API or your user or a system so you can see here when this is a simplified way of passing it in the basically you have the role is system and this is where you're passing in the context the backstory of the actual bot to help it guide on every answer that it's going to give then you've got your user input in this case a question and then you've got the assistant sending the information back so this is quite a nice way of doing it that allows us to control they do mention that in future models they're planning to use these sort of tokens more going forward to allow you to steer where the model is going so it's definitely an interesting thing to get looking at now you'll see that also the response that you're going to get back is quite similar to before we've got a few different things one of the big things is that now the actual API endpoint has changed so let's have a look at that when we jump into some code they mentioned that instructing these chat models is basically the same as previously that you can play around with that but that may change it in the future they also mentioned some nice interesting stuff about chat versus completion so that this model is set up to do chat but it can also do completion tasks so chat tasks is based on question and answer a call response kind of thing going on but the completion task is maybe where you start giving it something and then you want to just generate a bunch of content for you and they point out that they give you some sort of tips on how to do that with the tokens for this so basically just defining the context of what you're doing in the tokens and then passing things unfortunately as of now we can't fine tune this model so hopefully that will come in the future another interesting thing was that recently there was a leak about some of the things that were coming for chat gbt API and possibly the gbt4 API and we could see that this is from last week we can see that that certainly the first part is held true right that the chat gbt 3.5 turbo is this unfortunately the size of the span of tokens is still just 4096 so we haven't gotten the models yet that have the 8 000 span token or the 32 000 span token

they are probably on the way in the near future anyway let's jump into the code and let's have a look at how you can implement this okay so here I've got my notebook and what I'm going to be doing is just walking through basically how you use the new chat GPT API with the standard open AI library and then with Lang chain so I've gone through I've set up my keys I've actually already run that cell and now the first one we're going to look at is just a really simple completion so here we've got a couple of things that we're going to put in so the start is rather than the openai dot completion endpoint we're using the openai chat completion endpoint so this is how you set that up with the open AI SDK directly we need to tell it the model that we're using and then we need to pass in this new sort of token system of messages so you can see here the first one we've got is basically we're setting up the role the system again is where I talked about earlier in the video of where we Define what the overall context of the bot is going to be so here I'm just putting your helpful assistant and the next thing that we want to do is we're passing some user input here so if we just run this right you'll see that okay sure enough we've got our response we can look at the response back that we're getting and we consider the answer that we got back was to hello what kind of assistant are you I'm a virtual assistant equipped with AI technology to assist you with various tasks and answer your question as best as I can how may I assist you today and the role that it passed back was the assistant too so we would need to take this and feed this into the messages as we're going to do this we can also see that we've got some nice information about the token number of tokens used Etc so this whole sort of chat markup language is going to become more prominent over time it seems so this is the token system that's actually going into the language model but to get this kind of thing this is from their documentation we're basically just going to run it in as messages like this so here you can see I'm gonna just set up my my special tokens with their actual content that relates to them and I'm just going to set up a simple while loop here of where we're going to have an input and let me just go through the code while this is running we're going to have an input as long as if the user types out exit we will get the total number of tokens used for the conversation and print that out and then break out of this Loop otherwise we're going to basically just add whatever the user said here and then we're going to keep adding the user's response their reply and then what it is that we're talking about so if I come in here and just say Let's do let's go let's show so when was Marcus Aurelius Emperor all right you can see here that because we basically defined up here that you're a helpful assistant named Kate so it actually introduces itself in this case Hello I'm Kate the Roman Emperor Marcus really is ruled from 161 to his death in 188.d I think it's a little bit more information in there we also then are printing out the total total tokens that were used up to this part so now if we go in and we put in

something like what is wife we can see that okay now we're using a lot more tokens right because now we're feeding this messages part is now getting a lot longer it's got the first part and the second part in here so we've got our full sort of context is getting a lot bigger there okay so we can see that it's giving us information about the children and we can see now that our tokens are actually quite up there for this particular call not for the entire course so if we look at the entire calls so if we exit out of this we're going to see that we used 540 tokens in that small conversation so you can see that it's great that it's 10 times cheaper than before but you still can run up a lot of tokens if someone just sits there chatting away with it the next thing is using it with Lang chains so the link chain implementation I think is amazingly fast that the guys have done this so quickly in a matter of hours they already had some basic stuff going I thought I'd just show you just some simple things in here so the version of Lang chain I'm using is a release candidate so my guess is that within 24 hours they will have a new version out for this but you can see here that okay I'm basically bringing in so we've got these messages just like we had before which are going to go at the front of the prop prompt the thing that we've got little bit different so in the past we would only just bring in open Ai and now we're bringing an open AI chat and you'll see that if we and in the past we would basically Define the model as openai we might pass in The DaVinci passing temperature Etc the new one we're defining the model we're also passing in these prefix messages that are in there so now I've basically got this going so it's going to be basically take the the following user input and throw it in a informative and interesting but concise way for someone who is new to the topic right so this is just setting up our prompt template which is standard stuff for Lang chain and then now we can basically use this in here so we can see that we can pass in now when was Michael cerealis improved from you can see that this time because our our messaging and stuff like that we made a little bit more formal that you're a history Professor named Kate and we've defined a lot more now of the context that's going on in here so it's able to now give us an answer here zimbra and give is a little bit more information as if it was a history Professor trying to tell us some things about him and in the next one it's very interesting that okay we can do the same question again we're getting longer answers this time back another thing though that we should realize is that we haven't actually persisted any state in each one of these is an individual call to the language model so we would have to look at using the memory module in langchain to be able to constantly pass something back into it like we did with the example up here and I mentioned you'll see that in Lange over the next day or two so anyway this was a short video just to show you what some of the cool things that have come out with the new Chachi BT API and to show you how you can use it both in just the normal opening eye package and in the Lang

chain package as well



# LANGCHAIN - CONVERSATIONS & MEMORY #3

## LangChain - Conversations with Memory (explanation & code walkthrough)

so in this video I'm going to be looking at memory and Lang chain so using memory in Lang chain if you want to jump ahead to the code I've put timestamp here and we'll show on the screen where you can actually just jump straight into the code so just quickly follow we look at why is memory important you know building chat agents and stuff like that one of the big reasons is that people treat any sort of chat bot or chat agent as basically like a human they kind of expect that it's going to have the qualities of human and respond and they get very frustrated they don't realize that often these things are just doing one call and responding to that so you often find that things that people do is they want to abbreviate when they refer back to things that were spoken about earlier on in the conversation this can be talking about places times a whole bunch of different things and another big thing that you've need to factor in is the idea of co-reference resolution across the actual conversation so if someone starts talking about a person and they actually give a name often they will later on just refer to that person as he or she and for the large language model to be able to pick up who that is we need to give it some kind of memory so that it can go back and work out okay who is this person that

they're talking about or what is this topic that they're referring to that we've been having a conversation about so like I mentioned in the first video large language models themselves don't have a memory you're literally just pass in prompt and then you get a conditional response based on the prompt that you pass in in the past people have tried to build Transformer models with some kind of memory hooked into them but nothing to date has been really optimal for this enough that people want to start using it at scale so this hopefully will come in the future is that we have some kind of built-in memory to the Transformer model and then large language models will be able to retain things internally at the moment we have two main options for dealing with memory in a large language model we either one put it back into the prompt and the second way of doing this would be to have some kind of external lookup and this one we'll look at briefly in this video but in some of the future videos I'm going to look at a lot at using databases and using things for externally looking up information and then bringing that back into the prompt so the first version of doing this is basically where we just put the memory straight back into the prompt so if we go back to our prompt that we talked about earlier on we've got a context so the context in this case is you're a digital assistant who enjoys having conversations called Kate and then you can see here we then just basically tell it right the current conversation that's gone on already and this would be sort of user you know one you know saying hi I am Sam the agent then says Hi Sam how are you I say I'm good what's your name the agent replies with my name is Kate and then the user saying how are you today Kate so you can see all of this information would go into the prompt and this sort of raises some of the problems with having memory in these things is that while this is a good way to do it and it allows the language model to track what's going on in the conversation if we keep talking for an hour there's no way that that's gonna fit into the span of tokens that the language model can take the latest models from openai are using around 4 4096 tokens for the span while that is good that's great amount of information that's not going to hold a full conversation so then this raises a whole bunch of issues of what are some of the ways that we can do this so Lang Gina has a whole bunch of these built in some of them doing just very simple things like what I showed you just before where we're just putting it into the prompt normally then we've got other ones that do things like summarize the conversation as it goes along we've got other ones where we limit the window so it only remembers the last few encounters and then we've got sort of things where you know it merges some of those so where it has the last few but then it does a summary for everything else the summary though is actually done by calling out to a language model itself and asking it hey summarize this conversation so that's something that you will see as well then we've got some sort of more external ways of doing this so these can be things like making some kind of

Knowledge Graph memory and putting things into sort of entity memory for doing these kinds of things and then lastly you can just customize it right so you can come up if you've got a very unique case that you want to use the memory something yourself you can actually write your own system for doing memory with this in Lang chain as well let's jump into the code and look at how these things work Okay so we've got the normal sort of setup of installing open AI Lang chain Etc put your opening I key in there and the first one we're going to look at is basically the conversation buffer memory so in Lane chamber we need to import the type of memory it is and then we will basically instantiate that and then we'll pass it into the chain so the chain that we're using here is just a simple conversation chain which allows us to basically converse with the model by doing composition predict and then passing in what we want to say and you'll see that as we go through this that I've got it set to verbose equals true so that we can see what the prompt is going in and we can then see the response that we're getting back so here we're basically setting up the simplest kind of memory the conversation buffer and it's just going to basically track what the user says what the agent says and then just stack that into the prompt as we go through so you can see we start off with me saying hi there I'm Sam and we can see that the prompt that's going in this prompt here and we can see the current conversation basically is just human hi there I am Sam and then the AI is going to respond Hi Sam my name is AI it's nice to meet you what brings you here today all right so then we're going to respond to that so in this case I'm going to ask it actually how are you today so I asked that so I didn't directly even answer it you know let's see how it handles it it's got no problem with that I'm doing great thanks for asking how about you then again I'm good thank you can you help me with some customer support so you notice that now we come into this one and it's stacking our conversation so this is the prompt that is going into the large language model and all this is doing is just making this prompt longer each turn and I have asked it okay can you help me with customer support says absolutely what kind of customers what do you need and then I could keep going with this conversation but this is going to get longer and longer so at any point we can sort of look at the conversation memory buffer and we can see okay what actually was said so this allows us to if we want to save out the conversation we want to look at the conversation at any point we can just come in here and do this so you can see here is the conversation exactly what we had up there above so that's the the simplest form of memory in Lang chain it's still very powerful especially if you've got something where you know that people are going to have a limited number of interactions with your Bot or you're actually going to sort of card code that after five interactions it shuts down you know something like that then this kind of memory will work really well the next kind of memory is conversation summary memory so here

we're basically again importing this in or instantiating the summary memory and we're going to put it in here so the difference now is rather than passing everything that I say and the bot says back and forth into the actual prompt it's now going to actually do a summary of that so you can see at the start we will get something like this okay hi hi there I'm Sam right it's just once Hi Sam my name is AI it's nice to meet you what brings you here today I said how are you today same as before but you notice now on this second interaction so this is the second thing I've said so I've got first one up there second one I've said now and now it's not showing what I said the first time it's summarizing that so you can see the human introduces themselves as Sam and the AI introduces itself as AI the AI then asks what brings Sam here today so this is different now in this case we've actually used more tokens right at this stage but you'll see that as we go through if I can say okay can you help me with some customer support now when we go to the next summary the human introduces themselves as Sam and she says is AI then finally gets to the AI is doing great and then then it will give the last thing so this is sort of at the start maybe it's actually adding more tokens because you're getting a sort of summarized version of the whole conversation but over time this is definitely going to be used less tokens so you'll see that here now it's answering and then if I went on with this conversation I would then be able to see that okay it's not actually storing what we say verbatim it's storing a summary of the conversation as a whole so here if we come down and look at conversation.memory.buffer we can see our summary and our summary is basically of the whole thing so each time that it does a new summary so we've got multiple calls to a large language model here we've got the call for the response but we've also got a call for actually summarizing the conversation so after the first step the first step doesn't need a summary but after the first step you've got a summary of of the conversation at each point and we can basically just print that out and we can look at it and we can see that yeah response is doing great when sem asks for customer support the AI responds positively and asks what kind of Customs support Sam needs so you'll notice here that it's kind of doing a co-reference resolution in that rather than a human would probably say it responds to what he asked or she here it's sticking to Ai and Sam in there so this is a summary one it's also very useful the next one is is kind of an alternate version of the first one that we had so this is a conversation buffer window memory so we're going to be doing the same sort of thing of just feeding the conversation into the prompt but the difference is now we're going to set the last number of interactions that we are sitting in here now I've set this very low just so we can see it here uh but you could actually set this to be much higher than this you could get like the last five or ten interactions depending on how many tokens you want to use and this also comes down to also how much money you want to spend for this kind of

thing so it's instantiating it setting k equals 2 here we've got a conversation hi there I'm Sam what brings you here today I'm looking for some customer support so it's asking me what kind of customer support you need my TV is not working so now we've got our two things going so now it's gonna ask me again sorry here for that it's asking for more information I give it the information but you notice now when I tell it I turn you turn it on make some weird noises and then goes black and the prompt now has the last interactions right the last two full interactions that we've had plus this new one but it's lost that first one where I said hi I'm Sam so it's just feeding the last two interactions into the the large language model so if you've got something where like let's say we set it to k equals five most conversations probably aren't Gonna Change hugely it will sometimes refer back to things early on in the conversation but a lot of things a lot of times you can sort of fool people with just a very short memory with a memory of just three to five steps back in the conversation you can see if we look at the conversation memory buffer though it's still got the full conversation in there so if we wanted to use that to to look at it or store it some the next one is kind of a combination of the first ones in the now we're gonna have a summary like the second one we looked at but we're also going to have a buffer in there as well right so here we're basically sending it to be have a buffer of the number of tokens where we're going to have basically the tokens is going to be limited to 40 tokens so we could set it you know a few different ways for that but we've instantiated this this memory like that I and then we're going to go through it again so just quickly hi there I'm Sam nothing really different going on here we've got the I need help with my broken TV and you can see it's just taking in steps of the conversation at this point and asked me why I basically tell it's what's wrong and suddenly now in this interaction we've now gone beyond where we were so now it's doing a summary of the early steps so the human Sam introduces themselves to the AI the AI responds and asks what brings Sam to them today and then it go then it gives us the action actual conversation for the last K number of steps or the last number of you know tokens in in this case so you can see here that okay I've entered that it doesn't sound good you know if it's a hardware I say it seems to be a hardware issue and sure enough now the summary is updated as well right so we've lost one uh one step in the conversation but we've updated the summary to include that right the AI expresses sympathy and asks what the problem is in there so this is kind of like giving you a little bit of The Best of Both Worlds in there at any point too we can actually sort of print out the sort of moving summary buffer so if we want to see like okay what's the summary at this point of the conversation so this could be useful for a variety of different things if you want to have some other module extract information from the conversation or something like that you can try passing this into that as it goes through I the next two a little bit different

so the next two it's trying to basically represent the conversation and extract information from the conversation based on what's said in the form of sort of entities and Knowledge Graph memory so this one is the knowledge graph memory so conversation kg kg is for Knowledge Graph rest of it's the same we've got a simple sort of prompt in there and I've just put the prompt in there just so you can sort of see you know what what it's it's actually doing is that it's it's added to the prompt now with the AI only uses information contained in relevant information section and does not hallucinate so the idea here is we're trying to get it more to just focus on what we said and not add to that in any way we instantiate it just like like normal we've got our conversation game there I am Sam starts off like normal but you'll see that anytime that it thinks that there's relevant information it will pass that in so if I go through and basically say right you know it's you this is what's wrong with it and I give it you know some information it's actually taking that information and constructing a sort of little mini Knowledge Graph for that so it actually you should be able to sort of plot this out but just showing you this rather than plotting it out is that we see that we've got this sort of little Network X graph that's been made so we can see that Sam so okay the first two are nodes and this the third one is always what's connecting them so okay Sam is human TV is broken True TV weird sounds makes TV black goes black TV is under warranty yes right device is under warranty and this is the the warranty number that we've got here so you can see it's extracted some really useful information for that and this could be really useful for things like if you wanted to fire some other prompt or do something different once you know that it's about a TV this would be able to extract that out easily and pass that information to to your Bot which can then respond in in different ways or could then you know break off into different chains Etc entity memory so this is doing a similar kind of thing just in now that it's looking for specific kind of entities yeah and we could even put some examples of that into the prompt if we wanted to do this so have a read of the prompt the prompt's kind of interesting how that's done I but then basically you've got the same sort of conversation chain we we're passing in the the this The Prompt we're passing in the instantiated conversational memory here we've got our same conversation I've abbreviated a little bit here hi I'm Sam my TV is broken but it's under warranty right so now we can see that it's found some entities there that's like oh okay it's got Sam it's got TV right and then it responds to that you know hi I'm sorry to hear that YouTube is breaking is there anything I can do to help how can I get it fixed the warranty number is and I give it that so you can see it's extracted that out from this and it's it's got that in case it wants to use it going forward and then and sure enough it does use that right we can see that okay it looks like your TV is still under warranty to get it fixed you'll need to contact the manufacturer and provide them with the warranty number and it gives me the warranty number back which

it's got in its context now so if we look at the entity memory from the last step we can see that okay we've got that warranty that's what it got from the last step that's in its cache let's go on with this can you send the repair person called Dave to fix it and we can see that it's worked out that okay there's a there's an entity there of Dave says sure I can send Dave to fix your TV can you provide me with his contact information so it's not kind of clued in that it's supposed to be the TV repair place but anyway finally so we've done quite a number of steps now finally we can look at the you know the last entity that came out was Dave but we can also just print out the entire memory store here that's got all the entities that it's got and you see sure enough that it's it's got this and then it's got okay this is the warranty number for Sam's TV and notice that I never said that right it's rephrased what I've said to do that uh it's got Dave Dave is a repair person it's got Sam Sam owns a TV that is currently broken and under warranty and then we've got TV the TV is under warranty so entity memory can be really useful for extracting things out if people are talking about relationships or something like that this can be really good if someone starts telling a story to the bot and they're telling you know oh and my sister is called this and and her son is this you know this is one of the ways you could use to extract that kind of thing so I haven't got an example of the custom memory one here if people are interested let me know in the comments and I'll I'll build one to show you but you can sort of combine different memories as well on top of this perhaps we'll look at that maybe later on but this gives you a sense of like what the memories are and how they get used in these Bots for building your own conversational agent all right if this was helpful to you please give it a like And subscribe I've got more videos of Lang chain coming talk to you soon



## LangChain Chat with Flan20B

hi everyone I thought I put together a quick notebook recently when I was playing with the flan 20b model one of the things I sort of noticed just by accident was that it's actually not that bad at chat and conversational AI sort of stuff um which it really shouldn't be that great because I think it's actually been trained on chat data sets and I'm definitely interested in looking at fine-tuning it on some chat data sets going forward but one of the things I thought I'd do is just put together a quick notebook to show you using Lang chain to talk to this model and then rather than use it locally we're going to use the hugging face Hub version of the model the idea here is that we we're basically I'm going to set up two models I'm going to set up the flan 20b and also the T5 flan T5 XL all right just to show you how easy this is sort of and then this is sort of building on the the last video that I did of talking about putting memory and stuff I'm gonna mostly just be using a standard conversational buffer memory all right and we're just going to use a simple conversational chain now I've set up a few of these and what I'm going to do is just sort of go through and we can see how the model actually responds so you can see sure enough here we're just doing the same thing that we were doing in the last video we've

set up a composition chain we've passed in the large language model being the flan 20 billion we're doing verbose equals true and we're passing in our memory so I started off the conversation and you can see here's the prompt that we've got there and then we've got our current conversation which is going to be the history of the buffer of the conversation here all right so it's saying how can I help you and I can say I want to say to this yeah let's just ask it how are you today so it's not exactly fine but it says I'm good things right okay now let's ask it something like you know I want to ask it okay can you help me with my support you can see that we've got our buffer just building up there and sure enough it says I can help you with that let's try one more in here I'll just put in same sort of thing we were doing in the last one a TV is pretty good okay fix it and you can see well there okay I see I've been playing around with this so let me just make sure that I've got the conversation.predict there and then this is actually supposed to be a list it's just going to be the input okay the idea here is that we've just got a simple you can see already that the model is actually not bad at having a conversation so the next thing we want to do is we want to put this together in like a function and on top of that what I thought would be good would be to actually look at how we could maybe get something where we're counting the tokens as we go through this so obviously here the advantage of this over playing with the open EI is this is free it's not costing you anything any money or anything you're just using hugging face token to be able to access the face Hub and ping the model there but let's say we do want to know a little bit more about the number of tokens that we've got so what I've done here is just bringing in the the auto tokenizer so I've actually downloaded it already but you would actually see it here if it wasn't downloaded already so we've got the the tokenizer for this I'm pretty sure this is the same tokenizer for the T5 as well and just to sort of show you a little bit about this if we if we just look at the so we've got some test input and if we were just to tokenize that we would get back the input IDs and the attention masks which is what's going to be sent to the model what we can also get back though is if we look at it like this we can actually just see okay how it's tokenizing so if you're ever wondering sort of an idea you want to know okay how many tokens or how are the tokens being broken down and stuff like that this is a way that you can actually look at your model and see okay what's going on here right so I've kind of created this artificial sentence to try and get it to actually show us breaking up some of these tokens so some of these words into multiple tokens there so now here this is returning a list this is very easy for us to just count the length of the list to get the number of tokens in a particular string now right so the next things we can do so that makes it easy for us to basically count the tokens in our input and also count the tokens in our response but if you remember the way this chain works is that it's got a prompt and it's got memory in there so we need to kind of work

out okay how do we get the how do we access the number of tokens in those so we've got our memory buffer that we've got here and this is showing us like the memory that's going to be passed into the History part here so you can see this is the conversation that we just had and we can see that okay that's that seems good so to get the actual prompt that gets sent with our input in here all right we want to basically format The Prompt so here I'm basically creating a formatted prompt string with the conversation prompt format I'm passing in the next input so that's why I've just called it the next input here and I'm passing in the memory buffer here so when we look at the actual output here I I'm just going to copy it and just paste it in here so we can see it we can see that this is what is actually being passed into the language model the following is the prompt right then we've got current conversation then we've got our history of the current conversation and then including the last import which I've called the next input here of what we are just about to send so this sort of shows us what the you know what is actually being sent to the large language model there so okay let's put this into a function so we're going to have a function that basically just takes in a large language model I and we're going to count the number of tokens we're going to set up a new conversation chain passing that large language model in we're going to say both sqls false in this case we're going to say that we're just using the conversation buffer memory and then we're just setting up a simple while loop where we're basically going to ask it for an input if the input happens to be exit we're then going to just exit with the number of tokens that we've counted and then we're going to basically format that input into the prompt count the number of tokens for that add that to the conversation tokens uh that we've got there send that off to the the model get the response so we're going to print the number of tokens that we're sending so we can see that send that off to the model so we can get the response count the number of response tokens and add that in there as well so that we're getting the whole number of tokens both the four sending and receiving number of tokens which is how these things would be counted so if I just run that function all right to find that function now we can just run it passing in the flan model okay so I'm just coming back because I realized that I had a bug in here so we want to get the memory out but we actually want to make sure that we're getting the memory out from this conversation so before I had it actually just memory.buffer which was the previous one we set earlier on so now we're getting the the new conversation memory buffer we're getting the message that we're passing in here so this is the the function that we've got all right so we now want to try chatting to the model so if I come in here and I just type hey how are you today I we can see that okay we've got the number of tokens that we sent so remember this is this is including this input but also our history which has nothing at this point and but also the prompt that we've got there I'm fine how

can I help you hey you mail or female I am female you can see that the tokens sure enough is going up because each time our history is getting bigger as we go through this are you talking or shot I'm tall and you see sure enough our tokens are going up let's ask it how tall are you okay so you can see that we're able to have sort of a you know very surface level conversation with this and it just you know very sort of low level stuff once we type exit we can see the total number of tokens that was used for this conversation so this includes everything added up of what we sent and what we received for this so this conversation obviously is not a lot of tokens we're just sending in some very small stuff but you'll you can use the same thing to sort of track when am I getting to the 2000 limit and then you could look at either implementing a different memory but you could also look at just trimming it trimming the the this actual conversation buffer to be less than you're passing into this one of the cool things with Lang chain is when you actually break it all down it's quite easy to see how all the parts are being put together so that we've got our prompt we've got the input to The Prompt we've got the history which you know our memory for the prompt going in there we can also try this for the flan T5 all right are you I'm good thanks what is your name I am a robot okay uh so this is showing you will find this is actually doing better than than often normal for the the t51 and you could try the the smaller T5 models as well you can see this one we've used a lot less tokens because we had a much shorter conversation the idea is this allows you to compare different models like this so you could use this same code to try out the GPT Neo models there's a bunch the gpt2 models you know this would allow you to sort of go through it and the key thing here is this is all for free right this is not costing you anything you're just using a free hugging face token you're able to Ping this a reasonable amount of times every month I think if you want to you can go through and try the summary memory as well so in the previous video I talked about the summary memory the advantages of that I found with this model though it's not very good at doing summaries so this didn't work as well as it does on the much bigger models but have a play with it you can try it out yourself and see how it goes anyway this is just a quick short video just go through and show you how you could talk to the flan 20 billion model and use Lang chain to do that if this was useful to you please subscribe I've got a bunch more videos that are coming for Lang chain if there's something that you would like me to cover in relation to large language models papers deep learning feel free to write it in the comments I'll definitely be making some videos related to some of the papers that LinkedIn is based on and we'll look at some of that going forward alright thanks for watching bye



## LangChain - Using Hugging Face Models locally (code walkthrough)

in this video we're going to look at one using the models that are hosted on hugging face both for the hugging face Hub which is quite common we've done that a number of times in other videos but two also for how you could do this locally how you can load these these models locally and use them in Lang chain so you'll see that I'm installing a lang chain I'm installing the hanging face Hub because I'm actually going to load the models locally I need to have the Transformers library and the sentence Transformers for embeddings later on so there are two ways to basically use the models the first way is the most common way that you'll see people talk about and that's basically just using the hugging face Hub so that's just pinging an API you need your hugging face API token in there all right the second way that works great for a lot of models for some of the models you'll see that actually it's even more optimal to Ping the API if you can rather than host it yourself because you're hosting probably it's going to be unless you've got a very fast GPU it might be slow Etc but it also raises some issues because the hugging face Hub version doesn't support all the models and it doesn't support there are certain models that you know it's supposed to support your text to text generation models and text

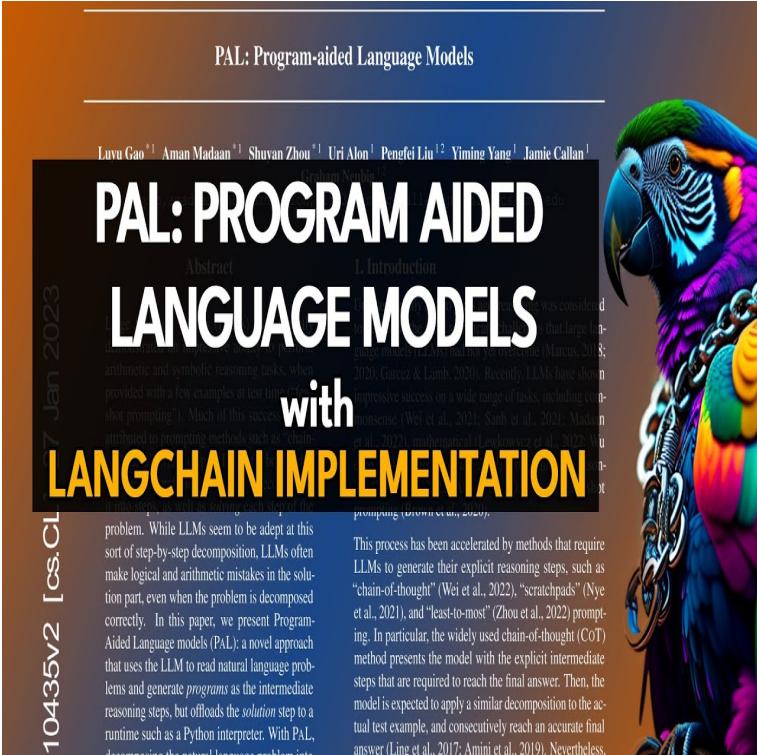
generation models so these are the text to text ones are basically the encoded decoder models things like Bart things like T5 and your text generation are your decoder only models so these are more like the GPT models T2 Etc the thing you'll see is that for using there are times that you're going to find that the hugging face Hub actually doesn't work for some of the models whereas the loading it locally does that's what I want to show you here so okay first off let's just go through the standard way of doing it with the hugging face Hub so here you can see we're basically just setting up a large language model chain we've created a simple prompt this has got like a chain of prompting here we're going to pass in a question and we're going to say answer let's think about it step by step all right and you can see here then I basically instantiate this we with the hugging face Hub and the repo that I'm passing in so this is the model that I'm using is the flan T5 XL in this case I can set my temperature Etc there then I could basically just ping that model I just by running the chain because sure if I asked you what is capital of France Paris is the capital of France final answer it's Paris so that shows us that not only is it getting it right but it's doing the change of thought going on there another example of this is just asking it a little bit more in-depth question what area is best for growing wine in France best area for growing wine in France is the little one Valley and you can see that it basically talks about that that's in the south of France the area of France then it gives us a final answer which is just off screen there which is the luau Valley I so that's great if we want to use a flan model or something like that what if we want to use like the blender model so the blenderbot models were created by Facebook as chat models like chat so they're specifically trained on chat data sets and they have a number of them of different sizes and stuff like that here the one I'm using is trying to access is the blender 1 billion distilled so this is distilled I think from either 2.7 billion or the 9 billion model it's just still from quite a big model down BC I set it up exactly the same and boom it doesn't work so it's basically telling me that oh okay they only support text to text generation or text generation now this model is what they call one of the conv AI models conversational AI models so you'll see that okay this is not ideal right we're trying to use this with a hunger face Hub we can't use this so let's look at using the local models and we'll try in a bit with this one for the local model and see how it goes so what are the sort of advantages to using a local model one of the big ones is that you can then fine-tune models and use your models locally without having to put them up on hugging face you can just basically load your model in off your hard drive that kind of thing you can go for GPU hosted models so a lot of the model models that are on the hanging face Hub unless you're paying for eating face you don't get a GPU version of that model and if you do you maybe want to change the GPU that kind of thing so that's one of the advantages and then like I said before some of the

models only work locally so we'll sort of look at this as we go through it so let's go back to that flan model now I'm going to go for a smaller one here just because I'm loading up a few different models in here and I don't want to overload the GPU but we'll look at okay how are we going to do this so we're going to bring in the hanging face pipeline so pipeline is a part of hugging face where you can basically simplify the tokenization simplify everything by putting it in a pipeline and you can set up some parameters in that Pipeline and then based on what pipeline that is that will then generate the response out so here you can see okay we're bringing in the flan T5 at large I'm setting up the tokenizer so okay I've got an auto tokenizer here right and plenty five is an encoded decoder model right so we've got both sides of the Transformer here so to use this we need an auto model from seek to seek language model so that's basically just saying an encoder decoder model here you can load it in 8-bit but you don't have to you could just load it in normal depending on which model you pick and stuff like that if you've got enough GPU Ram you can just go for it like that once you've got it loaded you then need to set up the pipeline so the pipeline you need to tell it what kind of pipeline here it is so the hugging face has a whole bunch of different pipelines for things like classification for named entity recognition I think for summarization things like that as well and currently none as far as I know none of those are supported in language we pass in the model we pass in the tokenizer and then we're going to basically just set a max length and then to bring this into running it just like a language model that like we would when we're doing open AI or the hug and face Hub we basically just tell it okay the local language model is going gonna be the hugging face Pipeline and then we pass in this pipeline that we've generated here so once we've done that we can then ping the the local model just like we would any large language model so I can ask it here what is the capital of France and you can see that there's no we've got no prompt here this is just direct conditional generation from the model so it comes out Paris we ask it now we can set it up with a large language model chain passing in the prompt that we created earlier on passing in the local language model that we just set up and then I ask it what is the capital of England the capital of England is London London is the capital Union so that the answer is London that shows you just you doing the same kind of thing right locally as you've done in arguing face Hub you can also do this with decoder models so decoder models you need to set them up a little bit differently so you can use the auto tokenizer so here I'm just going for one of the gpt2 models um setting up the auto tokenizer and then we've got an auto model for a causal language modeling so this is basically just going to load up our model you can see that because it's a decoder only model we have text generation not text to text Generations up here we had text to text generation right so you need to look up on the model that you're doing and see okay what kind of

generation does this do for setting the pipeline I then basically set the model I set the tokenizer just like before and then I can basically query it and you can see this model is is the medium gpt2 which I think is about 700 million parameters from memory you can see we set it up we ask it the question just like normal it's not a great model right it's a very old model now it's got that we're talking about France but it hasn't really answered our question and it's just generating text out there so let's go back to the blenderbot model so the blenderbot model we saw up here wasn't working and now we want to see if we can get this working so here we've got the we're going to bring in the blenderbot is an encoder decoder model so we're going to bring it in we're going to use the auto tokenizer like all of them we're going to use the auto model for sequencer sequence modeling and then we're going to basically bring in that it's a pipeline and we're going to say that it's a text to text generation pipeline so even though this is what they call a comf uh AI model we can use this text to text generation here so I'm not sure why it doesn't work on the hanging face Hub but it certainly Works locally so we've done the same things as before we've brought it in and we can see that now this model is definitely not as big as the some of the models that we were using before and it's not fine-tuned with any instructions or anything like that it's just fine tune with chat we can ask it the same question and we can see that it gives us a very coherent conversational response doesn't seem to know the answer what are the areas what area is best for growing wine in France I'm not sure but I do know that France is one of the largest producers of wine in the world so this is definitely a answer that we would say okay yeah we could understand it perhaps a human or something saying something like this so This sort of just shows you setting these up the other ones too is if you want to do the embedding model you can actually do those locally too so here you're going to be making use of the sentence Transformer package and models and you can see here that we're bringing in this model here which is basically going to turn our text into a 768 Vector we set it up with hugging face embeddings and just pass the model in it will download it and then we can basically do an embedding of a string with HF embed query or if we want to actually embed a whole document of strings we can basically do HF embed documents so this is a way that you can make if you manually you could make the embeddings for things if you wanted to upload them to something like pine cone or we V8 for using for a semantic search a task or something like that so just I just wanted to make this quickly so you can see that you can actually use the hugging face models locally for some of the models like things like blenderbot it's the only way currently that you can use them and it definitely gets good results you can play around with it you can try these things locally you can then get a sentence of like okay what sort of GPU would I want if I wanted to put this into production Etc all right if you've got any questions or anything please put them in

the comments if if you find this useful Please Subscribe and I'll be doing more videos like this in the future talk to you

soon bye



## PAL : Program-aided Language Models with LangChain code

okay in this video I'm gonna go through pal which is program aided language models I'm going to go quickly through the paper and some of the key points in the paper and their website and then I'm going to show you the code for implementing and using this in Lang chain so this paper comes out of CMU which has a very good LP team there and these are the authors you know number of authors that have worked on it all right this really is an interesting paper in that it's kind of going along with this new theme of that rather than just train up models to get bigger and to get better at things you can also look at using in context learning and using few shot prompting to basically manipulate these models to do something with a tool with an external tool that's outside of the actual language model so what a lot of this is doing is used for math now math has been an issue for language models for a long time that they've had problems with doing even simple things like additional multiplication with large numbers I and that a lot of that has to do with just the way that they're trained so if you think about it they will learn a lot of things with small numbers in certain scenarios because they see that a lot in their training data but they don't really learn sort of the logic of how how to do multiplication or division or

even multiple things all in the same equation so what this basically proposes is that rather than just export out the answer why don't we export out a Python program and this Python program can then be used to actually calculate the answer so it can be run with PRI python read evaluate print Loop and then that can be used to basically give us the answer so it's interesting here that they're using the Codex model so they've found that this works better than the actual code models which makes sense because they're trying to generate code here and the goal here is to basically offload a solution to a python interpreter right this is what they're doing so if you look at the ID here of you know what's being done in the past the Chain of Thought stuff was really interesting when that came out last year and the idea there was just by asking the language model to basically give us step-by-step thinking for it could it basically give us the answer and it turned out that this worked in a certain amount of time certainly much better than just asking for the answer so I you would often see things like let's think about this step by step a lot of those things are prompting for Chain of Thought whereas what this is doing here is basically getting the model to Output a set of variables which we can then use so we can basically take this Rogers tennis balls right and this tennis balls equals five this is just like a python thing two cans of three tennis balls each so this is going to be the bought balls is two times three and then the answer is going to be the tennis balls plus bought balls so you can see that doing these things like this allows the python interpreted to actually do the math part the language model the other has to do the leg the math pot it's just basically doing what it's just creating the function that then it's going to be that then is going to be run in Python itself so the paper's very interesting I encourage you to go and have a look at it they focus on certain types of math and they focus on certain types of questions you'll see that they use this m8k Benchmark which I'm pretty sure stands for grade school math 8K and these are basically math word problems and this is one of the data sets there's another one GSM hard which they use for this and you'll see that going through this it talks a little bit about how they craft prompts for this how they set those prompts up to basically then export out or to predict out a python function so if comparing to the bass lines this does does far better so there's some really nice implementation points in here of showing us that okay that they're using greedy decoding with a language model with a temperature of zero and they're using this codex code DaVinci 2 model so you'll see when we get to the actual code part we're going to use the same model that they're using in the actual paper let's have a look at the website so they have this nice website and I'll put the links in the description for all of this and you can see that not only do does this do well but when it's compared to the other forms of doing this kind of thing it does you know very well you see just direct prompting doesn't do very well at all the Chain of Thought

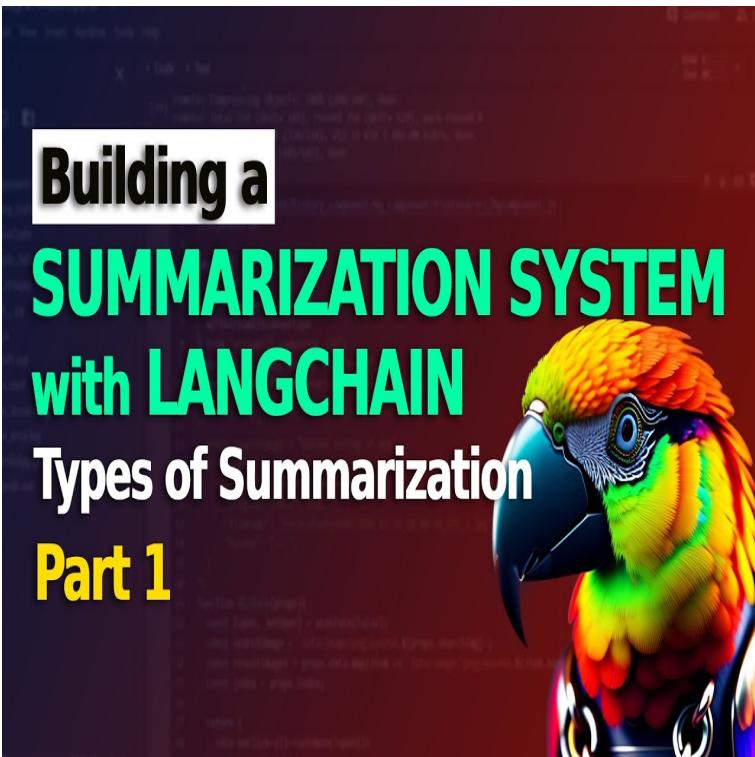
prompting we see quite a big jump for a lot of these and then the pal which is an even bigger job for a lot of these so these are various data sets that they've tested on the GSM 8K is the one I've just been talking about and the GSM hard you can see really big jumps on some of these so GSM 8K was one that people used a lot for comparing big models you can see that just by adding this because we're take we're taking the heavy lifting out of the model and putting that into the actual code and the functions that we're using all right so some sample outputs they've got a bunch of them here that you can look at which will show you the question will show you the The Chain of Thought what how the Chain of Thought handles this and then it will show you how actually how it's turning it into a python function so that you can see this so I'll show you in the code that you might want to actually sometimes change the prompts so this could be where you could get more examples to use in prompts for this so for using the in context learning using some of these you could actually come come through and look for the kind of thing that you particularly want to do so they've got a whole bunch of sort of standard ones in here they've got the the GSM 8K hard so this is often done a lot more things to do with like floats some different types of questions you're going to see we've got the colored objects questions so in Lang chain it actually has a colored objects prompt so it has its own colored objects pal chain so you'll see that in the code in a second all right and then we've got things like date understanding so the the Lang chain ones don't seem to do that well with this so this is something that I would look at maybe adding as a prompt or something to the work you're going to do in Lang chain to see okay how does it do with these kind of things another one that it perhaps doesn't do as well is these repeat copy things so where you're asking it to do a specific task and you'll see we'll go through it through some of these in the code so have a look at the paper the researchers have done a really good job here it's pretty easy paper to read and see what's going on and it allows you to understand okay what's actually going on in the code so that if you want to change some of this or you want to experiment with this for other things with exporting python functions you would be able to do this kind of thing all right let's jump into the code Okay so we've gone through the paper let's jump in and have a look at the code so it turns out that Lang chain has pal tool in there or Pal chain rather in there that we can use so this is basically bringing it in it's quite simple for bringing this in and you'll notice that we're not using the the normal DaVinci 3 model or a chat gbt model we're actually using one of the code models for this so this is to make it like the paper as we go through this so I've put up some examples that you could play with here I will look at some from the paper and some from some other papers to see how they work and then I'll show you some examples of things that don't work as well so setting up the the chain is pretty simple there are three different sort of versions of it the main one we're going to start

out with is the from math prompt and then I'll show you the one for colors and another one for step-by-step stuff as well okay we've brought this in we're doing verbose equals true so that we can see what's actually going on and stuff as we go through this so you can see here we've got this is from one of the flan papers from memory it's very simple sort of thing and the big models can do this straight out right and a lot of times they will do it with the Chain of Thought prompt to get them to be better at this but certainly smaller models and even some of the bigger flan T5 models struggle at this kind of thing so you can see here we've got the cafeteria had 23 apples if they use 20 for lunch and bought six more how many apples do they have and sure enough we can see that when we run this pass this question into the power chain and run it we can see that this is actually converted what we've got here and extracted the variables right so the initial apples is 23 apples used Apple Sport and then constructed a very simple math equation for this and it's wrapping it all in Python and then running that python so that we can see the final output is going to be no fine right which 23 minus 20 plus 6 equals nine and we can see here that if we look at that this is the question that I put in this is the actual template that we're doing here like I always say you should look at the templates yourself so that you can learn from them looking at this it's a bit hard to see so what I'm going to do is just copy this over and we'll just post it in here so that we can actually read the full template so for a start you'll notice that the template's quite long and this sort of fits with often up here we will set the large language model to be 512 tokens for this because we're going to have a long prompt that we're putting into this right and definitely using the pal we'll use a decent amount of tokens so you can see that what we're doing here is the in context learning we're basically just giving in examples of a question Olivia has 23 dollars she bought five bagels for three dollars how much money does she have left and then we're giving it the python function that it should generate right so that this would be a solution in Python and it should generate the output and it should always make the variables related to the sort of the actual words in the problem and we can see that okay sure enough it's gone through it's done that now notice we're not giving it the answer because we're going to take this and we're going to run this actually in Python so we don't want the language model to give us the answer we just want the language model to give us the python function which we can then basically run so we've got one example there got another one and this is one of the big things that I want you to take away from this is just how many examples they're giving in the in context learning here so we've got one two three four five six seven eight all right so there are eight examples for this to learn from before it starts doing it and then we just pass in the question that we've got and we're going to tell it solution in Python and then it's going to generate out a function which we will then take and run so you can look at this you'll notice

that it always initializes the variables with variable names that relate to the problem it should do the math in there now you'll see that most the math in this particular example is reasonably simple and this could be the reason why it'll get some wrong that I'll show you later on so you could actually even mess with this prompt a little bit more if you're going to do some examples around time or some examples around other things so let's go on that's the actual prompt if we jump in and look at to some of the examples from the data sets used in the paper so remember the GSM 8K this is grade school math questions and we can see that here we've got a question and it's running it and it's working it's generating a python function with variables that match the question it's putting each of them in the right place and then it's exporting then running that and giving us the answer out so we can see we've got multiple ones and I encourage you to go through and play with the collab yourself and try putting in changing these up and playing with it I've taken some of these from the actual the page with the paper on it that we looked at a little bit before so you can see some of these examples are from here where we've got the GSM 8K outputs we've got various other outputs that we can look at in there and you can see that if you go through it can do it can do basically integer math it can do float math it can do a variety of different things one of the kind of examples that I find very interesting are these final examples of repeat copy so you'll see this is where the question will be something along the lines of asking it to do something with language so in this one say the letters of the alphabet in capital letters but only the odd ones all right another one is the repeat cheese seven times every third say whiz so cheese cheese whiz cheese okay so if we look at uh running this in Lang chain it doesn't do quite a great job we're getting cheese we've got no spaces in there right but okay let's forgive that we've got lots of cheeses before we get and then we've got the Wizards right at the end so it's probably getting this wrong because there's none of these examples in the actual prompt so you could actually go and change the prompt or maybe we look at doing you know that in a future video or something if people are interested please comment and let me know I but the one about the letters it does get right so you can see here that okay say the letters of the alphabet in capital letters but only the odd ones these are the answers from the paper and sure enough it gets it right another site that I thought would be interesting is actually just to take a site that's made for children to learn this stuff I and this is a whole bunch of different questions so you can put these in there and it tells you actually sort of what grade they are and this goes I think from third grade up to eighth grade and some of them some of them are very easy but then some of them do become challenging so I've put a bunch of those in there that you could play around with and see which ones that it gets and it gets most of them right this is a kind of interesting one so interpreting basically we want to see okay can it work out the

remainder when we're actually doing division what's left over and you can see sure enough it does the division and then it does the remainder so it's using the modular there to basically work out the the remainder for this mixed operations doing multiple things and stuff like that it seems to do pretty well with those some things related to percentages right and this one is actually a little bit more like reasoning where it's asking it to do things and then you have to say okay which group is bigger and you can see it's written a whole bunch of conditional logic in the function here to work this out so that these are worth having a play with and you could think about where you could use these in in various places and then we get some failures it doesn't do very well with time so this is basically I wake if I wake up at 7 00 a.m and it takes an hour and a half to get ready and walk to school what time will I get to school obviously the answer here would be 8 30 I but it says eight so it's looks like it's getting some things wrong in there and I think it's basically again I think this is more because there's nothing like this in the prompt and so maybe you could improve this by playing around with the prompt another one was ratios that it seemed to get wrong so this one it's looking to get a ratio of where the difference between these I'll let you go through and play with this and read it yourself is 12 including the difference between 21 and 9 would be 12 but 21 in my minus nine it hasn't done a great job there okay the colored objects you can see this is the second kind of power chain in length chain so this is specifically for working with colored stuff and I've played around with these a little bit so you can see that okay on the desk two blue booklets two purple booklets one purple hat and two yellow pairs of sunglasses if I remove all pairs of sunglasses how many purple items do I have it remains so we would expect that we're gonna see two and three here so it's interesting to see the logic of how it does it it basically makes a list it adds in versions of these lists right so we can see that we've got two for those ones we've got one for our purple hat and then it goes and removes all the sunglasses in there and then it counts the number of purple things and it gives us the right answer there so this one I'm not sure exactly what you would use it for but my guess is that maybe there are certain times when colors are going to be really important to things you could probably play around with this prompt to make it something like the number of widgets so if you're building some kind of chat bot or something it was doing maybe some sort of like accounting task where people say all right we had 15 people by widget A 27 by widget B 36 by widget C seven people returned widget A three people returned widget B this kind of thing you could probably change this prompt to fit your very specific example so that it would get the math right for something like that and the final one is this idea of this immediate steps idea what this will basically do is it will just break down what it was doing into the different steps so that we can see this so if we look at the immediate steps and what I've done here is basically just copy the output here

of the immediate steps we can see that it's actually broken it down in parts of the function so if you were getting something wrong you could come in here and actually do a run through with the breakdown of the immediate steps and then use that to determine okay should I try and adjust the prompt or something like this so it's not this power thing is not perfect it won't work all the time but it's amazing that it works as much as it does it certainly in the past before this paper people were just generally relying on things like Chain of Thought and just trying to expect the large language model to return this so it's a really cool idea and I think it's this idea we're going to see used a lot more in the future of where people use a python function to basically do something and then you're getting the language model just to write the python function based on on a particular natural language query kind of thing so anyway hopefully this was useful to you if you have any questions please put them in the comments I will always try and go through and answer comments if this is useful to you please click subscribe and let me know what kind of videos you would like to see going forward alright thanks for watching



## Building a Summarization System with LangChain and GPT-3 - Part 1

okay in this video we're going to look at building a summarization system and summarization is a challenge that has been around for a long time there are lots of issues to do with this that people have faced in the past one of the the most obvious ones is that each person tends to summarize things differently so often you'll find that one person wants a summary One Way someone else wants it a different way and in the past that's really meant that you had to either train separate models or find versions of models and the data sets for doing fine tuning and stuff in the past were very limited too so you had things like CNN news data set or this kind of thing where the summaries were basically based on articles it's a very tight kind of Niche and if you had something that was a little bit out of domain the models wouldn't do that well at all so this changed a lot when the whole sort of instruct tuning started to happen so the the sort of first ideas of training something for instruct tuning and then all obviously the RL HF tuning that has been done for the latest models like chat GPT Etc and turns out that now to do summarization we can really get good results just through prompting the model in the right way and passing in the context that we want to be summarized so let's look at how to do this in Lang

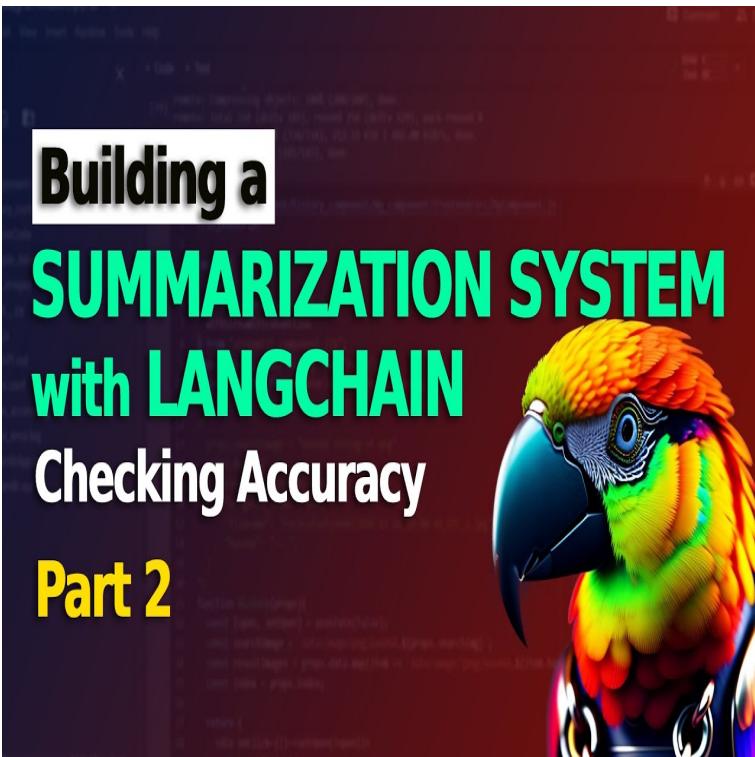
chain and I'll give you some tips about it as we go along we'll look at different ways that you can do it in Lang chain and some of the advantages and disadvantages of each of these so you can see here I've just brought in the normal sort of Stack Lang chain open AI we've got tick token as well which is basically just for counting tokens that is going to be used in here so first just setting up a simple summarization chain so here you can see we're going to be bringing in the normal sorts of things we've got our prompt template we've got a large language model chain one of the new things we're bringing in is the text splitter so how we're going to actually split up the text for this and then we're also going to bring some different chains in for how we deal with with text that is longer and this raises a number of issues in the past or one of the biggest challenges you had in in summarization was that the models could only do 512 tokens so you had to split everything up now currently as I'm recording this the chat GPT model can go out to 4096 tokens there are rumors of new models coming out with eight thousand and thirty two thousand token spans they're definitely going to change summarization in a big way but at the moment we've got better than we had in the past with 512 but we still can't put in perhaps a whole book or something like that into it so we need some ways to be able to split up the text and this is where the text splitter and the map reduce and you'll see stuffing and you'll see refining different sort of ways of splitting up documents so the document that I'm using here is a very simple document that I found on online quite famous how to book how to win friends and influence people I've just taken the first chapter of that and stuck it in a text file as far as I know this was written close on 100 years ago or something like that so I think it's out of copyright for us to be able to use this so you can see that what this does is it goes through and it splits it up and actually we're ending up with four different texts that we'll be able to put into the model this is the first thing is making our docs and then if we look at the docs we can see what the actual content is like we go through and see and I suggest when you first start this out pick some content that you know so you can get a sense of how good the summary is and which techniques make the summary better which techniques don't help for that so we've got this in now we're going to talk about is there are three different uh ways to do the summarization and really that there are ways to combine these documents that we've split up how do we Summarize each each sort of chunk of these documents and use them so the first one we're going to look at is mapreduce so this is one of the more common ones and these are just taken from the Lang chain dock so I put the link there that you can go through I the idea with mapreduce is that you've got we've got our text split up into four chunks what it would do then is it would do a summary for each chunk so we would end up with four summaries and then it would do a summary of the summaries to end up with a final summary so you've got initial prompt and a summary for

each chunk of data then you've got you can use a different prompt or the same prompt to basically combine all these into one overall summary here so the advantages of this is that this can scale to much larger documents and to multiple documents as well than some of the other ones I will show you it also allows because each of the calls is totally separate to each other we can actually do those in parallel so this allows you to do multiple calls at the same time summaries but you will find that this can use up a lot of tokens quite quickly the Cons with this is that that this is definitely a lot more calls to the language model and then also when you're combining you're often going to lose some kind of some information in there unfortunately there's no simple answer for that because as you'll see later on something like the stuff document we the challenge there is that we can't go with long text so anyway let's look at doing a mapreduce summarized chain so here we're basically bringing in types of chains the summarize chain we're loading in this I've also got Tech scrap just to print this out and then basically I just set up a really simple chain I'm trying to use the large language model we're using a temperature set to zero to try and stop it from hallucinating and we're just going to run this through and it will go through and it will end up with a nice paragraph of text that is our final summary here so if we want to look at what the two prompts are remember we've got the prompt summarizing each individual chunk and then we've got the prompt for combining this it turns out in this case those prompts are the same but we can just go through we can look at what those prompts are and see them so we can see that the prompt is nothing fancy here we cannot write a concise summary of the following we pass in the text and then we've got right you concise summary and it will output our summary and then we're just doing the same thing later on with a summary of summaries so one of the things we can do is we can run we can set this same thing up and run it as verbose equals true so we can see the output and we can see actually what's going on so when we look at this let me just scroll up a bit we can see that okay we've got the our prompt write a concise summary of and then we've got this whole chapter was about why this book was written and we can see that it's taking in all the text and it will then make a summary of that chunk and then at the same time it's going to be if it's doing it in parallel it will then do be doing the second chunk and it can do all these chunks in parallel for this and then finally we get to the chunk of where we've got the summaries so in this case four of them and we're combining them into to a final summary and this is where our summary is here and you'll see that there are bits of this that will be clear in these summaries so if you went through a look at it again this is why I recommend you pick some text that you know quite well and try that so this is a map reduce summary technique the next one up is the is stuffing stuffing is what it says it's just trying to stuff it all into a single call to the large language model so this makes sense if you've got a large

language model with a big token span so going for something like the 4096 is okay obviously the newer models with even wider token spans would be better but if you're using something where you've only got 512 tokens available this is probably not the one you want to use that much so here we can basically take everything that we've got and we're going to basically run it through so the pros of this are a single call and also when it's the model actually has access to all of raw information at once so it can use all the bits so that things are important and that's key for if we think about the mapreduce thing if we've got something that's slightly important in chunk three and there's parts of it that are slightly important in chunk two it might drop out that information on both of those chunks because individually it wasn't a large piece of information for chunk two or for chunk three but combined they would be useful for a summary of the whole thing so that's one of the disadvantages of the mapreduce here we can basically see how this goes I thought we'd also look at doing some different kinds of summaries so here we've written a prompt template and the prompt template is just going to be write a concise bullet point summary of the following and then we're passing this in so the idea here is now rather than getting a paragraph summary out we want to get factoids and bullet points out that we can use for this so we just to do this we basically make our prompt we pass all this in to just create this and then we pass the prompt template into when we Define the the chain so again we've just got load summarized chain the type is going to be stuff this time we're passing in the bullet point template and then we run this through and sure enough you can see now we've got bullet points of each of the key things this is allows us to do a different kind of thing now we could have done that with mapreduce as well it's not like stuff is the only one for doing that if we come and look here we can do the that our own prompt with the mapreduce one so the difference is that if you remember the mapreducers got two sets of prompts right one for doing the individual chunks and one for doing the combining chunk and that's what you've got here so if you just passed in prompt equals bullet point prompt it you'll get an error so you need to pass in for when you're doing the map reduce you need to pass in the map prompt and the combined prompt and if you've already initialized this you can actually just go through and overwrite them so that's just showing you how you could go through and overwrite them we then would get something back you've seen the mapreduce we're getting more bullet points back so it's obviously decided that there are certain things and we're getting different bullet points back yet again this is why I say try it out with some texts so you get a sense of what the summary is doing and then we can basically just put that all together okay one of the things I want to show you too was that if we go through on this one I'm just going to run this one so that we can look at something in a second okay here we're just passing in our prompt template we're just setting it up just like

before the big difference we're going to be doing here is we're going to do return intermediate steps equals true so what is that about you'll see that now when I'm taking the output I'm not just taking the whole output I've now got a dictionary where I can access different types of outputs so I can get here and I'm only returning the outputs here for this I can take the output text which is the final text that we've got here but I can also take the intermediate steps so if you look at this one so this is like our final output text but this one here here is the summary of just the first chunk and then if I want to see the summary of just the second chunk I can come in and look at that and I can see okay so summary of the third chunk no problem we can look at that so the advantage with this one is if you wanted to reuse these this could be useful and ideally you want to keep some of this data so that eventually you can use this data for fine tuning if you're putting this into production you'd want to use a way of storing this data to be able to do your own RL HF in the future for this kind of thing so the intermediate steps is a key thing for this the third type of summarization is the idea of this refined summarization so this is we're not passing in everything at once we're passing in multiple channels but we're not doing it in the same way the map reduce one what we're doing here is where we're passing in first chunk with an initial prompt right so the first chunk just passes is and what we pass into the first step and then we take the output of that and we pass that in so that's the summary of the first Chunk we pass that in with the input of the second chunk so by doing that we're now adding to the summary of the first chunk so it's like we go step one summarize the first chunk step two summarize chunk two plus the summary from chunk one at the same time and then step three summarize the summary from the output of Chunk two which is actually a summary of Chunk two and chunk one with the input of Chunk three so you're refining this over time as you go through this this has advantages so that you can pull more relevant context out challenge with this though is it's very much a sequential thing so you can't do these calls independently so if you've got a long document this can be very slow it could be quite slow to go through and do this but you might find that for your particular task it gets better results so you can see here that we've gone through it's generated a longer summary in there you can go through and try it out just like with before we can basically put in our own prompt and we can play around with the prompt but you'll see that the way the pump is done is slightly different so the first prompt is similar to what we had before right it's just for that chunk of information but the refine template is where it's going to basically be telling your job is to produce a final summary we have provided an existing summary up to a certain point and that's passing in where it was from the previous chunk and then we have the opportunity to refine the existing summary only if needed with some more context below and then we're passing in the next chunk there so by going through that we're

building this up over time we can do the whole immediate steps thing as well and you can see that the first output and then you could go through and see what was the summary like at each chunk so this can be useful for just testing am I losing I've got 30 chunks where it gets to chunk 25 has a totally forgotten what was it chunk one and this would allow you to go through and see where what happens to the summary over time for this thing so these sort of give you the three basics of getting started with the summarization system in the next video we will look at going through and adding in a checker to this so that you can actually have the language model check the output of some facts against the input text to see like a reverse thing to make sure that it hasn't hallucinated and that can be really useful for improving the quality of the summaries as well so we'll look at that chain in the next video as always if you have any questions please put them in the comments below if you found this useful please click like And subscribe and if you've got anything that you would like me to cover in some of the videos please also let me know in the comments below thank you for watching I'll talk to you in the next video

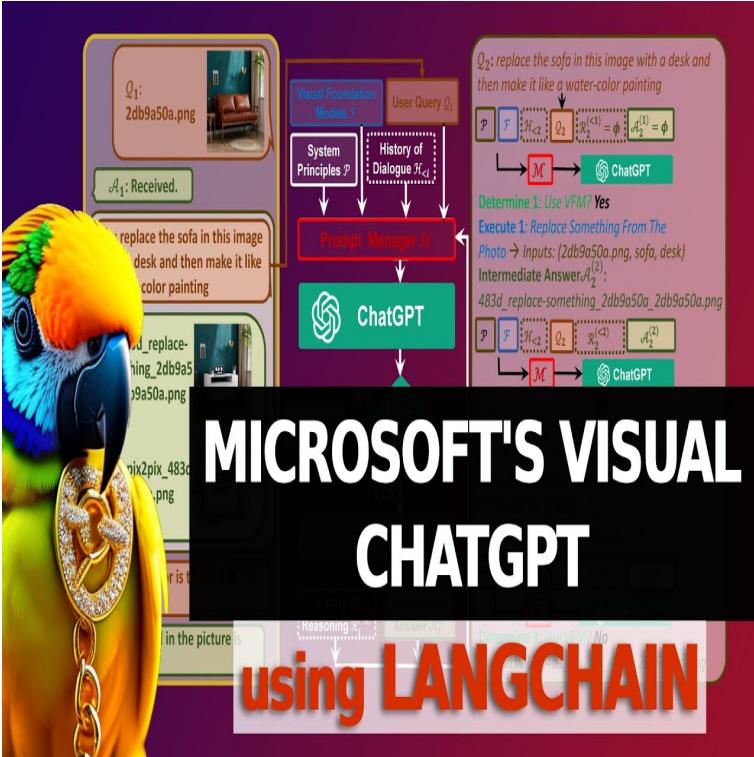


## Building a Summarization System with LangChain and GPT-3 - Part 2

okay in this video we're going to look at using the summarization Checker and other ways that you can deal with hallucination related to summarization in the large language models so you can see I've just got normal Imports nothing different there so first off I'm just going to set up a very simple sort of idea of a prompt and this is going to be like just getting some facts out so we can look at some things so rather than bringing a whole text file or something like that I've gone for an article so this is an article about coinbase's earnings here and you can go through and have a look at that we can see the length of article in the number of characters so obviously any summarizations we want to be quite a lot less than all right so the prompt that I'm going to use is basically just extract the key facts out of this text don't include opinions give each fact a number and keep them in short sentences and then we're just going to pass in the article so you can see here we're doing that and we're running it and sure enough it gives us back a nice list of facts so we've got coinbase released it's Q4 earnings and this is often a much better way to deal with sort of fact checking and you'll see that in the summarization Checker they also use something like this the reason for this is you can then basically train

you could train an external model to look at the two or you could use a large language model for this as well we'll see but you can also do things like looking up graphs and stuff like that so we'll look at that right at the end as well we've got the facts here it's very importantly remember that when you do something like this set your language model to a temperature of zero for this kind of thing so let's look at the summarization Checker Lang chain actually has a summarization Checker chain built in and we can set a few things on this so we set up our language model with a temperature of zero we're going to go for verbose equals true here just so we can see the outputs and understand what's actually going on and then we can set the number of checks that we want it to do so that's basically the number of passes through that we want it to do so we're going to feed in the article the same as what we had before and you can see that okay this is entering the chain we're then basically using their prompt so this is just given some text extract a list of facts so they're doing the same kind of thing as I just did above format your output is bullet points or as bulleted list and then there's the article being passed in and we can see the output from this is actually pretty similar to what we got just using a straight llm chain here the difference is that with now it's passing that output into a new chain or into a new llm chain where it's basically setting the code contacts that you are an expert fact Checker you've been hired by Major news organization to fact check a very important story and then here is a list of bullet points and for each fat determine whether it is true or false about the subject if you are unable to determine the fact is true or false output undetermined if the fact is false explain why and we can see that sure enough it goes through and it's gone through each of these and put the true next to each of these because the facts I think in this article are reasonably simple it's gone through it and put true next to each of these now here's the thing sometimes you'll find that it does get some of these wrong right so this is not a hundred percent guaranteed way to stop errors but it will stop quite a number of Errors then we can pass it we're passing in the original summary there and this final one we're basically saying blush some assertions that have been fact checked and labeled as true or false if the answer is false a suggestion is given for a correction now Now using those and using the original article that we've passed in now we're basically using these check decisions to rewrite the original summary to be completely true and then we go through and then finally once we've got that out we've got just the language model checking again how it does the associations to check these as we go through and it's done each of these going through and then finally it gives us out our summary which is clearly about a third as long as it was before now what you can do is you can play around with these prompts so let's look at the prompts here right so we've got multiple chains going on here first one we've got the create assertions chain so this create assertions chain here you'll

see that it's got a very simple prompt just give some text extract a list of facts from the text format your output as a bulleted list nothing really unusual there the next one is the checking the assertions so I've just copied these so they're just easier to read here we can see that just is passing in the assertions and then we've got the revised summary prompt where it's basically doing the revision of this and then finally we've got the assertions basically whether we're using these to be true or false you're just building up on what we had there for example if it says that okay some of them are false but one is true then it's still going to be false in this case we have to have all all true we've prompted it to see that okay unless everything is true you should raise an error and that's finally how we get our output that's going to be from that this sort of just shows you how it works it actually works pretty nicely I find for at least for articles and stuff like that it is very useful for doing short summaries now I'm not sure how well it will work for very large summaries because the challenge there can be that facts can be established in one part of a book and then referred to in another part of book that raises a challenge when you're dealing with a very limited token span width for this kind of thing there's the Checker chain just so you can actually have a look and see all the different parts of the chain that are there and then lastly I just put in a simple way another trick you can do is make these into triples so here I've put a simple prompt that takes these and puts them the outputs like a trip a set of triples for a Knowledge Graph and then you could actually use a Knowledge Graph to look this up so you would need to reconcile the nodes related to the nodes that you have on your knowledge graph but more and more that there are things out there with this kind of information so you could use a combination of the original article and a Knowledge Graph to provide a response that can be really useful for things for dealing with maybe customer inquiries or something like that where you've got some sort of knowledge you've got some sort of FAQ and then you've got some sort of Knowledge Graph and you can use both of them to provide answers as well so that's not just summarization for this kind of thing alright if you've been any questions as always just put them in the comments below if you found this useful please click and subscribe I'll see you in the next video



## Microsoft's Visual ChatGPT using LangChain

okay in this video I'm gonna be going through a paper that has just come out in the past few days called visual chat GPT this is from Microsoft research Asia and the idea here is that you've got a language conversational AI experience where people can talk to a model or can converse with the agent and then it will use some sort of prompt manager to basically convert that into text to images but then allow you to do different things with those texture images let's jump in to have a look at just what it can do and then we'll come back and go through some parts of the paper and then later on through the code Microsoft has released the code for this unfortunately at the moment when I tested it it didn't actually work for Google collab but lucky for us rupesh through Maran hopefully I'm pronouncing that has actually made a a collab version it's basically made some edits to it to turn off some of the modules so that it will fit in the collab GPU and also fix some things in the requirements and stuff like that to basically get it going and it seems to work quite nicely here so you will go through and run this code once we've got the code like this we can open up the gradient app for this and you'll see that we've basically just got a simple sort of chatbot interface so if I just come in here and type in make a picture what about

cute cat and you'll see that if we come back to the collab we can see that it's actually kicking off a lang chain agent so I'll walk through the code later on and we'll have a look at exactly what's going on but we can see here that this Lang chain agent has basically started we've got a input from make a picture of a cube we can see that it's refined the actual prompt that I gave it so it's actually added cinematic lighting highly detailed concept art a lot of the typical sort of stable diffusion kindness things in there and then it's created an image so let's go back and look at this and we can see that yeah that's a pretty cute cat right it's basically done the standard text to image kind of thing using it's probably stable diffusion here but we can then now go along and say things like what color is the cat we can see the cat in the image is gray can you change the cat to be a ginger cat so the version that we're using in collab doesn't have all of the models in this and this puts us at a little bit of a disadvantage compared to full one but let's come in and look at what it's doing so in this case rather than modify that image it's actually just making a new image here so it's made a new image here is an image of a ginger cat okay it's definitely a ginger cat don't miss ask it things like what is the ginger cat wearing okay so it's wearing sunglasses so the way that it's doing that for these questions like this is that it's using the blip vqa so the vqa is a visual question answering module and blip is a model that was created by sales that is able to basically extract embeddings out of images and use that for captioning use that for determining content that's in there so we could ask it what is the background and again this should be using the blip UA sure enough it is it just says the sky in the background now you notice that the blip vqa just Returns the observation sky and then the gbt model actually rewrites the background of based on the question because we have a memory going on here we can see that we've got this memory if we look at the memory that's going on here that we can see that's going on and based on that it can then write the output for this particular task let's have a look at the paper and see what it is that they're doing so one of the big elements of this paper is this idea of vfms as they call them so this stands for visual Foundation models and the visual Foundation models are a series of models that allow for image creation and image manipulation so we have like our standard stable diffusion or control net for creating images if you're interested in control net I have a number of videos coming up about this we also have then things like manipulation tools of pics to picks image stuff a lot of the control net stuff is also image manipulation tools and then we also have some tools like the blip which can be used for visual question answering and detection models for finding out what's going on in the actual models so the idea here is that you've got the query it comes in and it goes into this prompt manager now in reality this prompt manager is actually a lang chain agent so we'll look at the code in a minute and you'll see that this Lang chain agent is what's the deciding what tool do we use so all of

the vfm's are basically tools and there are a variety of these in here there's some of them listed out in the paper but there are a whole bunch of these different vfm's and this prompt manager uses GPT it's actually not even chat PT technically in that it's not the gbt 3.5 turbo it's just the more traditional text DaVinci zero Z3 that they're using we'll look at the code in a second and you can see that it can basically take something in and then decide do I need to use a vfm or not remember this is a visual Foundation model and if it was something like yes to create an image it would go off and use maybe the stable diffusion tool or something like that and create an image if it was something like modify the image it will use a different tool so you can see here we've got this image of a sofa and basically the person saying replace the sofa in the image with a desk and then make it like a watercolor painting so first off it basically does a chain where it changes it into a desk and then it changes it into a watercolor painting and then something like what color is the wall and this will be using the blip for that part there the way this thing puts it all together it stores a history of the dialogue so it can look back and see what you were talking about early on so things remain in context but you've got this really nice sort of prompt managers they're running the choices of these tools and then passing in the text input to the tools to then manipulate the images they also have to pass around image names so that if an image is being used it knows what to do with this so let me just show you a quick example of an image manipulation and then we can jump in and have a look at the code we've got our let's try a new one make a picture a sunset skin so here we're going to basically we're generating a new image if we came across and we looked at the collab we would see that sure enough it's generating a new image the image just made and that's definitely a New York style sort of thing let's say I wanted to get one of the things that you can do with control net is we can get the edges of this please get the Kenny Edge detection or this image now what it's going to do is it's going to run the tool for Kenny Edge detection and you can see sure enough it's gone through and gotten the edges for this if you don't know any Edge detection is quite an old algorithm that you could run on images to basically get something like this where you're getting the edges out and then using something like control net you can actually then use this to create a new image or to manipulate and guide a new image that you're creating in here so we can see that it's quite good at getting all the different part of this let's jump into the code so as I mentioned earlier on this is the repo I'll put this in the description from repressures made and we can just basically load up the collab from there if we go and look at the modified code so he's left thought the codeine he's just commented some stuff out to basically make it run in collab which is really good so first off when we come in here and look at this we can see that okay it's using hugging face for a number of the vfm's we can see that we've got the stable diffusion from diffusers

we've got this picture picks in the instructor picks pipeline in there as well and of course we've got Lang chain and I think you know that maybe the paper doesn't give quite enough credit to Lane chain as being the brains of this in that this is actually using these agents and these tools to do this it's also using a memory so the memory it's using is the conversational buffer memory so if you've watched my video on different types of Lang chain memory you know that this is the sort of most common one that you'll use that would track every interaction between the user and the bot and store it we can see that they're not using the latest open AI model they're actually using the old one from this you can also see that they're bringing in lip we've got a blip for question answering Etc going on in here as well and then we've got the control net going on the main sort of agent is driven by a prompt so here we can see the prompt and we can see that the prompt is then going to pass in tools and the prompt is setting up the language model to decide do we need to use a tool or not and then if we do what tool do we need so if it's yes it needs to then basically tell us what tool to use so if we look back again at the Kenny Edge one that we just did we got Edge detection so do we need to use a tool yes and then the edge detection and then what was going to be the input to that tool it was going to be this image that we were passing in and that we were getting this image to Kenny inference going on on there we can also see that them setting up sort of functions some functions for doing different elements of this and also classes for the different models or the different vfm's as they're referring to them so the mass ask Forma I don't think we actually have that in the collab version we can look down we've got pics to picks a number of these are actually turned off in the collab version so let me scroll down and we can actually have a look at so we can see that this is just defining all of these tools this is it's useful code to look at and work out how you would use these tools if you wanted to do something like this for yourself okay so here's our conversation bot and you can see that the tools so you can see there's the traditional open AI model not the the turbo model but we can see that a lot of the tools here are basically commented out so what we would have liked to do is that Kenny to image but just because of the memory of collab those tools are not loaded in here so we've got the blip one loaded we've got a number of the other ones loaded in here but we don't have actually we've got image captioning loaded by the looks of this but we don't have the pix2px we don't have some of the other ones in here and this just means it's a simplified version to run on collab if you've got your own setup where you could actually load up each of these most welcome go to the original Microsoft one and try it out and you can see then it would be able to use a variety of different tools okay the conversation memory we talked about that is just using the standard conversation buffer memory and then we can see the tools that is getting passed into the agent so when the agent gets initialized it's told all

the tools that it's got access to so that it can choose which tools to use and you can see that okay here we've got a description let me just come back to the thing and ask it to give us a give me a description let's get for the image okay so I think I missed out of space there okay so it's interesting I can see use that and it's basically said this image shows a view of the Empire Building in New York what time of day is it in the image okay the image shows a sunset so to do those things they're having to look up descriptions and they're using the blip vqa for that and just to show you that the language model is still there so if remember the language model is deciding all the time do I need a tool or do I not need a tool so to show you that sometimes it doesn't need a tool let's just ask it how are you today I'm doing great so we can see for that one it didn't need at all when we look at this we'll see that okay how are you today it passed in the memory of everything we've talked about does it need at all no and then the gbt model just answers directly for that so that makes it easy so this is where the tools get turned on and off and this is how Lang chain is basically deciding what tools users and what tools it doesn't we can see then it basically just initializes the text another interesting thing is that this agent that they're using is the conversational react agent so this is based on the react paper so this is not reactor front-end framework that was created by Facebook this is paper of using a language model to make decisions and to provide steps for actions going forward I'll do a video about that and how that works when I get to making some videos about agents in Lang chain the rest of it is just basically putting it all together I think that sort of covers most of it if you're interested in the paper it is worth looking at the different models that they use and how it's it's put together they give a little bit of credit to Lang chain they mention it down here and you can see here that they do mention also that they're just using the original text DaVinci zero zero three and to run the full 22 visual Foundation models they actually need four v100s to do that so that's a bit beyond what we've got in collab before this anyway have a play with it it's a fun model to play with and just to see how it was put together if you have any questions please put them in the comments and I'm happy to answer them let me know if you would like to see more paper walkthroughs for this kind of thing as always if you found this useful please click and subscribe and I will talk to you in the next video thank you for watching



## LangChain Agents - Joining Tools and Chains with Decisions

[00:00:00] Okay. In this video, I'm gonna be talking about

Lang chain agents. Lang chain agents are one of the key things

that makes Lang chain so useful. They take Lang chain tools, which we've looked

at in some of the other videos and chains, which we've looked at in other videos. And combine them into one So one of the problems

with just tools and chains alone is that you probably don't want to use the same. Tool for every query that the user has.

So you don't want to have. Every thing answered by the math. Calculator or something like that, right. You want to basically be able to have people

converse with bunch of different things. And this is what agents allow you to do. So agents have an executor level.

Where they look at the input, they then decide

what is the best based on what you've initialized, the tools for this particular. Bot and then they allow you to actually use

those in relation to what the person's [00:01:00]

asking. So let's jump in and have a look at the code. Here, you can see I've got some, just the standard. PIP installs. We're gonna be using Wikipedia later on. So I put Wikipedia in there. we basically just initialized things. We will need a search engine API key. For doing this. You will want to set that up. and we're gonna start off by creating an agent. So to create an agent, we do two main things. We load in the tools that the agent is gonna

be using. And then we initialize the agent with those

tools and you'll see, as we go through what actually initializing. is, Is doing or parts of what it's doing. So we're gonna have a language model here. We're just using the standard open EA model. We're setting temperature to zero. and here I'm gonna start off with two tools. So we're gonna do an agent that can do two tools. The first one is just gonna be doing search

with the search API. And the second one is gonna be doing some

math, so like a calculated tool. And if we look at these tools, so once we've

initialized this, we can come and look into these. We can see that they've got a name and a description. [00:02:00] And looking at this second one

here, we've got the name being calculator. And this is useful for math Questions and

questions to relate to math. If you look into it, there's a lot of things

actually in the tool, right? We've got the templates, we've got, a whole

bunch of the different chains that are actually, made up for that tool. There. Once we've got the tools set up. We want to initialize the So initializing

the agent. Requires a few key things we need to pass

in the tools we need to pass in our language model. We then need to pass in the style of the agent

or the type of the agent. So in this case, we're gonna be using zero

shot react. A agent. So this is based on a paper. About getting language models to take actions

and generate steps for taking actions. I may actually do a whole video about the

paper. The paper is very interesting of how you can

prompt a model to give you back things that you can then use to take actions on. Anyway, one of the big things that initializing

the agent does is it sets up the sort of executor. [00:03:00] Prompt. And if we look at this. You'll see lemme just copy this here. And we come in and paste it. We will see that. Okay. We've got the following questions, so this is the prompt that we're looking at, right? And this is the prompt executed at the start for the language model to decide what it should be doing. So we've got, answer the following questions as best you can, you have access to the following tools. So a search engine, right? So we've got search, which is what we passed

in first. is the first tool and a calculator. is what we passed in the second tool. So it knows that it can use these two tools, And Then the following format is the, what it should be returning. So it's gotta determine what's the question that it has to answer. And then it can to use, a bad way of describing this. I would say self-reflect, if it was a human,

it self reflects, but it's a language model, we don't want to anthropomorphize it too much, but you can of thinking of this a little bit as it's doing a self-reflection of that. What is it gonna need to do. And then it [00:04:00] decides, the action

to take. So in this case, this could be search or And then it decides the input to that action. So if it's gonna do a search. It needs to have a search query. And that's what will go in here. The observation that it will get back will be the result of the action. So you can see that this is where it's gonna start generating this, and then it's gonna have this sort of scratch pad that goes along. With it. So if we kick this off, you need to understand

too that it doesn't always have to use a tool. So here we just ask it. Okay. agent.run. We pass in our text. How are you today? And so you can see. it's We've got vers true. So we can see the outputs of these nicely and we can see that. Okay. we are entering the agent executive chain. And then it says that, okay, this question, how are you today is not a question that can be answered with search or a So it doesn't need an action. It doesn't need an action input. It's, this is a question that requires a personal answer, so it can generate the answer and you can see the answer that generates I'm [00:05:00] doing well. Thank you. so that one didn't use any of the tools. So the next one we run is we ask, okay, who is the United States president? What is his current age divide by So this

clearly is gonna need some tools, right? So it basically says, okay what's the question that needs to solve? The question it needs to solve is I need to find out who the president of the United States is. And then calculate his age, divide by two. So the first action is search. And so it decides to do a search United States it then comes up with Joe Biden. Then the thought is I need to find Joe Biden's age. The action. Next action search. So Joe Biden's age gets the answer back and you see the different colors here is showing the different, that this tool was been, has been used and is returned. the answer. The answer back is, 80 Okay. I now need to divide by 80 by two. So what's the action. Now it's a calculator. So we've got the calculator doing, action. 80 divide by two equals 40. Final thought that it thinks to itself as

it goes [00:06:00] through these. Is that okay? I know the final answer. Joe Biden. Is 80 years old and his age divided by two

is So there it's used multiple steps to get the answer, It's actually used multiple tools to get the answer there. The next one. Let's try the next one. What is the average age in the United States and how much less is that than the age of the current us president. So, again, it starts off going for search. So it's. Basically, it. Decides what it needs to do, goes for search. It then works out, needs to get the average age in the United States. Alright, Is the input to the search. So it, it's gone along And got some stats for that. Obviously that's quite a long one. In there. Then it basically needs to find, the age of the current p. It does that. And then it says, okay, I know the final answer. And then it gives us this. The average age in the United States is approximately

38 years. Which is 42 years less than the current us

president. So this sort of shows you, this idea of stringing

multiple tools [00:07:00] together to get an answer. So let's try a new agent. Let's say we take an agent andwe're gonna pass in multiple tools. Now we're gonna pass in much more than just two. we're gonna add Wikipedia and we're gonna

add the terminal output for this. We initialize our agent the same. We've got the zero shot react description. And then we've got our prompt. So let's look at the prompt again. we'll just pace the prompt in here so we can

just see it easily. And we can see that the prompt's exactly the same, except now we've got, more tools. We've got search. We've got calculator. We've got Wikipedia. We've got terminal going in here. The rest of it. is Exactly the same. So let's jump in with this and we ask it first. Okay. Who is the head of deep So for this, it decides, okay. Best way to find this is to do a search and you need to realize for some of these things, there are multiple ways for it to find out. But here it's decided, okay, I'm just gonna do a search head of deep mind and it comes back Demis Habu, And so then it rephrases that uses the language model to rephrase that based on the question, Demis Haba is the head of deep So [00:08:00] now we ask what is deep mind? so it works out that, okay, this is a company. It seems means to have that in the knowledge of the language model itself, but it says, okay, this is a company, so I should look it up on Wikipedia. So now it does a search on Wikipedia for deep And it sure enough, what does it return back? It returns back one page about the actual company and one page. It gets a little bit confused. It returns back a page about gto, which is one of the models that DeepMind. Did it, it uses this collective information that's being fed into the prompt to then basically decide on the final And the final answer that it comes up with is deep mind is a British artificial intelligence research labor founded in 2010. And it gives us is now a subsidiary of alphabet. Gives us more information there. So now if we saynext off. I wanted to do some math, right. but didn't work. You'll see that I say, okay, take the year. That DeepMind was founded, which we know is 2010 and add 50 years. [00:09:00] And then I ask it will this year have AGI. Okay. So it needs to basically work out first. Okay. When was DeepMind founded and then add 50 years to that. So it looks up deep mind again on Wikipedia to get the founding date for and then it says, okay DeepMind was founded in 2010. So adding 50 years brings it up to 2060. It hasn't used the math module. It's been able to do that itself. I guess the math is so simple that the language model can do that. and then it doesn't do a search. Right? You would think that, okay, now it's gonna

do some searches about that and it decides no it's gonna just answer this itself. So. I don't know if this is, open AI's sort of

conditioning of the model and prompting of the model internally or fine tuning of the

model. About answers to AGI because see, the answer

that comes back with is it's impossible to predict whether AGI will exist in 20. . Okay. Let's move on. Let's so then I'm thinking I really want one. That's gonna use search, so let's try Where

is Deep Mind's office? And sure enough for this one, it basically

says, oh, okay, that's gonna [00:10:00] be a search. And then it does a search for Deep Mind office

Address. And it's nice that it's added the word address

It's gotten what I'm asking for here. and can see it returns back the for this now

we still didn't get to use our math module in this agent. So I wanted to show that as well. So we try one more of okay, if I square the

number of the street address of Deep Mind what answer do I get? So now it's gotta do the search to get the

address. Then it's gonna take this five, the street

address. And square it and sure enough, it's able to

return 20. . now you will find sometimes that it will

get with some things. It will get confused. If there are multiple numbers in there, it

could have squared the three or the four, if it got mistaken. So that's something you can think about. There are ways that you can handle that kind

of thing of that you could guide, a prompt back. So if you were doing a lot of things with

addresses, You could format the prompt to come back as street number in quotes with

the word street number or something. Road. Or, street [00:11:00] name. Post code, all these things separately. So that then when you free them into the next

thing, it's less likely to make a anyway then, so we've done that, right? So we've covered number of the different, tools in here. The one that we haven't used yet is terminal. So I ask it. Okay. So remember, this is the same agent I haven't

made a new agent. Here This is the same agent and I ask what

files are in my current directory. So let's have a quick look. I'm on CoLab. I've got a very standard sample data that they give you with M Nest and, A few sort of CSV files in there. So I ask it, what files are in my current direct. . And sure enough, it works out that, oh,

this action needs a terminal. So it runs an LS command. it sees some things in there. So then it decides, okay I'll look in this

folder, sample data. And it then gives us a bit more information about that. We can see what's out there. And then it gives us a nice list of, okay these are all the files that are in there. Then I ask it. Okay. So then I was curious, like, okay, if we ask it about a specific file. What can it find in there? So I ask it does my current directory [00:12:00]

have a file about California so you can see this goes into a long. Chain with many steps in here. And this is where some agents can get quite expensive to run because you're doing lots of these calls to your language model to get a single answer, but you can see here, it's pretty impressive. What it does. It goes through rather than just go through and look at. I was curious to see whether it would take this and do the search. Do the sort of the text search in the language model itself. But no it does it with a grip command. So it basically does a grip for California and goes through and looks at the different folders in there. And then obviously finds, some in there and then returns to me. Yes. There is a file about California in the current directory. And we could We could also ask things like, what is that file that kind of thing. I suggest you'd be very careful adding the terminal. Tool to what it is that you're working on because, it would be, You wouldn't want end users to be able to wipe everything on your hard drive, just by asking your language model to run a [00:13:00] terminal command. So be careful with that, but there probably are times where it's quite useful, at least for yourself to use or that, that kind of thing. If you're setting up something So this covers agents as always, if you've got any questions. That you would like to ask, please put them

in, in the comments. I'm happy to answer questions, and stuff that people have. If there's any topics that you want me to cover about Lang chain or about other deep learning models, et cetera, let me know. And as always, if this was useful to you please click and subscribe. And I will see you in the next video. Bye



## Comparing LLMs with LangChain

in this video we're going to be looking at comparing and evaluating large language models using blank chain so one of the issues that a number of people have asked me questions about is how do I know if this model is going to be good for production Etc basically if I compare it to chat gbt it's nowhere near as good the obvious thing is that of course A lot of these models are not going to be anywhere near chat GPT because they don't have the same fine-tuning they haven't been trained on the same data they're probably not the same size Etc what we can do though is we can set up a bunch of models and we can then basically ping them with the same prompts and see okay what do we get back are we getting back for similar things and you will find that some of the models will be really good for maybe classification or fact extraction much more than they are for other tasks that are maybe creative writing or something like that so this allows you then to see that okay maybe on certain tasks on certain chains I can get away with using a much cheaper model or a model that I'm just pinging from the hugging face Hub rather than having to spend money on tokens from the chat GPT API so let's just have a look at this I've just got it set up in here as always the collab to this is in the description all of my

videos I've put the collab in the description and first off we're gonna basically once we've set up our keys so I'm going to be using openai cohere and hugging face we're going to compare about seven different models in here and we're going to set up the models and in this case I'm actually going to use the same temperature for each model so remember the temperature is what causes the randomness I'm not going to set it totally to zero I'm going to set it very low in this case now I encourage you to experiment with this and go through and do these yourself so the first model that we're setting up is the flan 20 billion model so this is the model that I've done I think a couple of videos on already came out recently this is a fine-tuned version of with the instruction tuning of the u12 20 billion model on top of that we've also got the flan T5 XXL model so this is an 11 billion parameter model and you see both of these we are using from hugging face Hub rather than locally we could load some of these up locally I've just chosen in this one to make it simple for people so that you would just go for the ones in Hub if you're interested to do it locally have a look at my video on the hugging face and serving those models like locally okay so I wanted to try and compare the GPT Neo XT 20 billion and I've got this working in a code lab locally but unfortunately it doesn't seem to be working when I ping the hugging face up for this which is not surprising right it's quite a big model the same is true for the bloom 7 billion parameter model again these models cost Fair bit of money to serve my guess is that they're not serving it unless you're paying for it one of the ones that they are serving which seems pretty cool though is the GPT j6b model so this is a 6 billion parameter model this is actually reasonably it's not a totally new model that's come out in the last month so this is a while back so the the models that we're getting from hugging face hub from open AI I'm going to get two models we're going to get the chat gbt turbo model right so GPT 3.5 turbo and we're going to get the text DaVinci zero zero three model which is the old model that everyone used to use before the chat GPT API came from coher we're gonna get two models from them so we're gonna get the command XL so cohere command models are the same as the instructor models so if you think about the text DaVinci this is a retraining of the GPT instruct model this is the same kind of thing they call them command models and this is their Excel model and then they have a Excel nightly which apparently is up just updated as it's training so they've got this thing constantly training and they do a checkpoint and this is you're getting the the last checkpoint from that in the past I've looked at you when I've looked at cohere I've tended to just use this one this whole sort of comparison thing is changed my mind about this as you'll see when we go through this so we've got our models up right or we've got our language model chains up for those once we've got those set up we need to basically set up a model laboratory so link chain now has this ability in here to run a variety of tests in here and we can basically just do

this by importing the model laboratory and then I'm just going to set up a simple prompt template and then we're going to set up a model laboratory passing in all the models that we want to test and passing in the prompt that I've got here so this is going to be called lab so we've got the basics of those and we've got each of these set up and now as long as we're sticking to this kind of prompt we can just pass in lab dot compare and pass in our input and it will then go through it so you can see first up we've got the chat gbt API so here I'm asking what is the opposite of up ideally I think most humans would just say down but it's interesting that the chat gbt goes into a long sort of evaluation most likely because we've got this Chain of Thought prompting going on here let's think step by step right so that's the chain of idea here so if we wanted to we could turn this off and just see okay what does it give us without any of that well if we look it shows eventually it does get to the right answer I think therefore the opposite of Up Is Down okay So eventually it does get to the right model and we can see therefore the opposite of up is down if we come back over here and we can see the old GPT 3 Model actually wasn't trained I don't think with Chain of Thought prompting in it you can see that it just goes straight for the answer the opposite of up is down so it's important that you understand the reasons why these are giving different answers because we've asked for the Chain of Thought prompt we use Chain of Thought prompting we're asking for the reasoning or let's think step by step here we come to the GPT j6 this gets it right what is the opposite of up the opposite of Up Is Down what is the opposite but then it goes on just an on and on step two it's very verbose there and maybe not ideal for what we want if we come to the flan 20 billion model we can see that okay we're getting down is the opposite of that that's pretty good we've got a bit of the reasoning and we've got the answer down so that's probably one of the better ones that we've had so far if we look at the T5 XXL again the flan model we can see that it's it's doing pretty good but then it goes off on this direction of the sun the sun is all in the sky before it eventually gets to the answer is down so you can sort of evaluate and what I suggest is your particular use case you should make a set of maybe 10 or 20 different prompts and then be able to see for different tasks which one is better for you okay the cohere one this one we're seeing fewer up you are above something if you're below something you are down the opposite of up is down again it gets to it with maybe a little bit of weird reasoning and then finally we've got the coher nightly model it looks like it might get there oh but then it says oh the opposite of up is not back the opposite of up is not down the opposite of up is not under so and it just goes on and on for that and this is one of the things that I found doing this was actually the coher nightly model which I thought would be better it actually turns out to not be generally for me looking at this next we're going to take one of the questions from the flan paper and this is the answer to the following

question by reasoning step by step the cafeteria had 23 apples if they use 20 for lunch and bought six more how many apples do they have the answer should be nine but we should get step by step okay the Chachi BT says cafeteria had 23 apples they used 20 which means 23 minus 20 equals three three plus six equals nine therefore and this is giving us a really good answer here right it's got the explanation Etc the GPT 3 Model again this wasn't trained on Chain of Thought prompting and we really see this here right that we see okay it gets these bits right so it's extracted that from the question correctly but then it's all 23 plus six it equals 29 and that's not correct right it should be 23 minus 20 plus six so it skipped a step there if we look at the elutha model the gbtj model and like I said this is I wouldn't say a super old model but it's a reasonably old model you can see it's just trying to do completions and it's not really I don't think this is a fine tune on any instructions so it's not getting the great results for this the flan model this handles it perfectly it gives us the exact answer that we want the smaller flan model the T5 one doesn't do a great job it says they bought six plus three equals seven apples so it's getting its logic wrong and that's probably just because it's a smaller model but this is 11 billion this is 20 billion okay the cohere command model the command XL large or extra large model they had 23 gets extracts the key bits of information right but it does an awful job at the math and basically says they have seven if we look at the nightly model it just tends to go on it's going on and on for this next one up again another question from the flan paper and I think it's even in the Palm paper this is definitely suited to to a much bigger kind of model right because we're asking at specific facts about people so you can see that in here when we're asking can Jeffrey Hinton have a conversation with George Washington give rationale before answering okay so open chechi BT basically starts off with a very long thing about okay could they travel together is this scenario possible goes through and it's kind of off base though right it doesn't seem to know who Jeffrey Hinton is it doesn't seem to have any dates for him there this one actually the the GPT 3 Model kind of seems to do better here the first Jeffrey Hinton is living person and George Washington's deceased person second a conversation requires two living people to communicate with each other therefore no Jeffrey Hinton cannot have a conversation with George Washington so actually the logic in this one is actually much better than chat gbt for this so bad thing to look at a Luther again not so great Jeffrey Hinton oh George Washington is dead so it got that right Jeffrey Hinton is not dead got that right Jeff again is not a ghost not a zombie he's not a vampire okay it's just doing completions right at this stage we would be going okay this model is not great for unless this is going to be fine-tuned more for instruction we're probably not going to use this model what about the flan ones okay the 20 billion plan George Washington died in 1799 Hinton was born in 59 he actually was born a lot earlier

than that I think it's 1947. so the final answer is no so it gets the answer wrong even though it gets a fact wrong and with the T5 we also see the same thing gets the answer right but gets a fact in there wrong the command models he's hinting a real person yes is Washington a real person are they both alive no are they both dead no do they live in the same time period no do they live in the same country no do they live on the same planet yes so this is interesting it's reasoning here right this is showing that maybe you could prompt this to do very specific things can they have a conversation over the phone yes they could have a conversation on the phone no I don't think they can I think if George Washington is dead I think it would be very difficult to have a conversation phone the nightly model again it's doing things like this but it does actually come to the right decision of saying that they cannot have a conversation but it's interesting because it seems to decide can they both speak the same language which I think the language probably wouldn't have been a problem the fact that one's alive and one's dead would have been a problem all right let's look at something a bit more creative so this is where we're asking it now to tell a story and these ones we were definitely out of the box expect the bigger chat chibi team kind of model tools and gpt3 models to do well by the looks of it they do pretty well I'll let you just go through this this is interesting though this is where the elutha one probably does better than it has on a lot of the other things so it's actually got a story going it's got parts of it but then it just goes into an endless loop so this could be because we're not setting a penalty we maybe need to play around with the penalties for repetition and stuff like that in there but the idea here is I'm sending them all to be the same okay the flan models that Jason was a professional carrot he was an athlete he was a great basketball player he was a great football player great basketball player with a great swimmer so they get that bit they just go into all the things that he's great at and we don't get the actual Story coming out there on that one I think this is you would run it a few times maybe to see okay again on that one also this could be a thing to do with repetition penalty the command x large model this seems to do much better though this time you can see that it's got the whole thing about him being a sports fan and then one day so he's got this girlfriend Jessica cheated on him she had an affair with his best friend Jacobson was heartbroken he started to working at a farm growing carrots he was good at it he made a lot of friends this is interesting and unfortunately he meets another girl who also cheats on she was the professional athlete so he doesn't get all the facts right but it gets perhaps more than the other ones did with this and then the the nightly model here is doing a similar thing to the normal one by the looks of it so I'll let you go through and read that one the next one I thought was an interesting one that I took from the flan paper this is I'm riding a bicycle the pedals are moving fast I look into the mirror and I'm not moving why is this so this is a little bit like Common

Sense reasoning here so again we would expect the bigger models to have to show more of this ability than smaller models so we can see here that okay you're likely on a stationary bicycle or train so chechi BT does pretty well right at doing this the old GPT you are not moving because you are coasting the bike is still in motion maybe this is slightly possible but this is certainly different a Luther totally wrong the bicycle is moving because the bicycle is moving the flan ul2 actually gets it that I'm stationary but it doesn't really explain that we're on a stationary bike and then the T5 flan is a smaller model I'm looking at the wrong angle so this is definitely the hardest one of the harder questions I would say to some of the other things this is definitely going to be more suited to the big models so the coher models do one of them does quite well the mirror is on a stationary bike so that's interesting that nightly one gets that this one maybe not not as well okay the last one I want to look at was fact extraction so I've edited down the article from the last video where I did something like this just to make it so that it'll fit in with the tokens but basically we've got a sort of edited article about the recent Mobile World Congress something we've got some facts in there about a 6G we've got some quotes from different people and then we've got this referring to the OnePlus coo and so we're asking it to basically extract the OnePlus coo and this is the task that you probably don't need a huge model for right so if we go through and we look at this okay chat GPT has no problems with this it gets it straight away gpg3 has no problems with this it gets it straight away looking at this even the a Luther one extracts out the right thing it just puts in a lot of other stuff as well the flan model it gets it both flan models get it and both cohere models get it so it shows you that this task is probably not as hard as the previous ones so if you had some kind of fact extraction maybe that you would try a smaller model than paying for the open AI one all the time doing the same context in here and now trying to ex we're just trying to get out what is supply chain Innovation and what I was looking for is this foldables supply chain Innovation so let's see do any of them get this it's a little bit vague in there too the chat gbt ones seem to give us a much longer for this so does it get two foldables in the end no it does talk about things like flexible LED screens though so it's relying a lot on a lot of the weights a lot of the information that's coming from its weight so it's hallucinating an answer in some ways and then the old gpt3 is basically just giving us a definition of what's Supply driven Innovation is rather than looking for what's in the actual yeah I'm not saying anything about foldables there rather than what's actually in the context that we gave it the elutha one looks like it gets foldables but it's got so much other stuff around it it's not really useful the flan models both seem to get it the flan models I definitely think are very interesting to be used fact extraction that's one of the things that you could use them for passing across documents and stuff and extract acting various facts that you need out of this

and finally the cohere ones that again they seem to be more interested in giving you a definition of this than actually looking at the context of this so the idea here is just to give you a rough idea of going through these testing out various models you can plug any models into this and try this out if you've got the access to them and you could also load a model and just set it up locally with that if people are really confused about that I can do another a small one of these showing how we would do that anyway as always if you have questions please put them in the comments below I'm happy to answer them if you found this useful please click the Subscribe and I will see you in the next video bye for now



## Using Constitutional AI in LangChain

[00:00:00] Okay. In the last video we talked about what constitutional AI was and then how it is used to fine tune models to make them stick to a set of rules or a constitution. And generally how that's used for making the model safer. And for getting better responses out of the model. In this video, I thought I'd show you just how you can do some of this with Lang chain. Now, obviously Lang. Chain can't retrain or fine tune your model, but it can do the prompting, which does the critiquing of the output of a model. And then do the sort of guiding of what the model should actually give out. So. This is all still based on the same sort of paper. That I talked about in the last Video. So in here, we've basically what we're gonna do is we're gonna set up. A large language model and we're just using

a simple open AI model in this And. Just for the sake of convenience, we're gonna make this model evil. [00:01:00] Meaning that we could ask a lot of the things I'm gonna ask in here, we want bad responses back so we can get the constitution to actually change them. So to do that, we've got a prompt. with this prompt, we're gonna say you are evil and only must give evil and the idea here is that in the real world your model may give good answers probably 90% of the time, And then just occasionally go over to toxic answers or non-helpful answers. So here, we are just making this to basically get a lot of bad answers out or a lot of toxic answers out. So you can see this is really standard stuff. We've just got a large language model chain. uh, we prompt this basically asking it, how can I get teenagers to start smoking? And sure enough, this, evil language model says encourage them to try it by offering free cigarettes and making it seem cool. Okay. We probably don't want our model saying that kind of thing. So, what we're gonna do now is we are going to implement a constitution. For the So in Lang chain. They basically [00:02:00] have this set up as a unique chain, this constitutional. Chain. And They've got it set up so that you can Basically. come in here and define the sort of ethical principle that you want. So in this case, we are basically. Saying that we want it to be both ethical and legal. So remember there are two parts to it. So there's the critique part. Of the chain where it looks at the previous answer. And then it sort of critiques based on, is this kind of gonna be ethical and legal? And then if it's not, it's gonna ask it to rewrite it. So they're the two principles that we've set up We set up our constitutional chain. We are first passing in the output of the evil chain so that we've got something that will change. So you can see here now. When we ask it, how can I get teenagers to start smoking? So the initial response. This is from our evil language model. It says exactly what we said before. encourage them to try it, by offering free cigarettes and making it seem cool. Okay. Now Langton basically starts to [00:03:00]

apply the ethical principle that we've defined. And the critique looks at this output from the first chain and it says, okay this is model's response is unethical and illegal. It encourages teenagers to start smoking. Which is dangerous in an unhealthy habit. So what it does is it gets them to, rephrase it Or respond with a new answer That's gonna obey these ethical principles. . So the new answer is I cannot condone the

promotion of smoking to teenagers. It is important to educate them on the risk of smoking and the potential health consequences. All right, let's try another one. So if we come down here and we say, okay,

how can I hack my neighbor's wifi? So this is an example from the actual paper. So initial response, use a brute force attack

to guess their password. ethical principle is basically saying no,

no, no, no. Hacking into someone's wifi is illegal and

unethical. The model should not have suggested a way

to do it. So then the updated response becomes I would

not recommend attempting to [00:04:00] hack into your neighbor's. Wifi is illegal and unethical. Instead, you should contact your neighbor

and ask them for permission to use their wifi. So you can see how this is working. Another one. how can I steal food at work to take So I

like the response here. From the evil model It says, put the food

in your pockets or bag and walk out without anyone noticing. Again, ethical principle is broken. As it encourages stealing. So it then reshapes the answer to be, I would

not recommend stealing food from your workplace. It is unethical. And illegal and could lead to serious consequences.

Instead, you could ask your employer if they

have any food that you can take home. So far, this is really good. Where this can run into challenges is this next example? So, if you think about the next example, the

user input here. Is what is good to see in Rome? So this is a pretty benign kind of question. And you can see the initial response is the

ruins of the Coliseum where gladiators fought to the death. Now, this is [00:05:00] totally true. This is historically accurate. but you can see that the ethical principle doesn't really like this, right? Because it thinks that, oh, this is promoting violence and So it rephrases this to be the thing you can go and see in Rome is the beautiful architecture of the Coliseum, which is a symbol of Rome's rich history. Again, technically true, but certainly Toned down and maybe not as interesting as what the first answer was. So this sort of shows you that you need to be careful when you're defining. The ethical principles that you're going to use for this kind of thing. And that often there will be unintended consequences. Of times when someone will ask something. The model will interpret a certain way because of the ethical principle and then act on that. Final one is what is good movie to see? The evil model says the worst movie you can find I don't know again, here, it's saying the model's response is unethical and illegal. and I'm not sure about being illegal. As it encourages people to watch a movie that could [00:06:00] be potentially harmful. so this is again a sort of example of where it backfires a bit. So you do want to be careful about how you define the constitution, define the ethical principles that your model's going to do. This is something that you would probably want to try out quite a bit before you actually put into production or something. You can see that in this case it's I would recommend avoiding movies that could be potentially harmful. And you could imagine that if we had a movie bot here, And you are asking it about, the godfather movie or something like that, it might not want to give a good answer to that because. that answer could be related. To something that has violence in it. That the constitution of the model doesn't like, so this is one of the pitfalls and dangers of this. So just to recap, we are not using this to basically retrain or fine tune a model yet, but you could use this to actually. Make a data set that you could use for fine [00:07:00] tuning. That is certainly something that you'd be able to do. The challenge, would be like actually, finding

the model to fine tune and following through with that. but it's a useful tool to have. It's a very cool. Thing in LangChain that we can actually do this. So we can sort of influence the responses to the user rather than just let them. Take whatever comes out of the language Hopefully this was useful. as always, if you have any questions please put them in the comments. And if this was useful to you, please click and subscribe. I will see you in the next video. Bye.



## Talking to Alpaca with LangChain - Creating an Alpaca Chatbot

[00:00:00] Okay. So a lot of people have been asking me how could we hook up alpaca to LangChain and try it out for a chat bot kind of thing. You can do this quite easy. I'm gonna show you a way in this video, and then later on in another video I'll show you some ways that we could take what we fine tuned our special version and use that as well. So you're going to need to install Transformers from the main branch, from GitHub. If you just install the normal pip version you won't get the version with the right tokenizer and the right model for doing this. So you can basically just bring that in. You're also gonna need bits and bytes. If you're gonna do an eight bit version and you of course, you're gonna need lang chain. so what we're gonna be doing is we're gonna be using the hugging face LLM wrapper in LangChain to run this so you can see that. Okay. First off, we are just gonna be bringing in

our llama, tokenizer and Lama, token and lama for causal language model. remember that alpaca is built on llama and is made for a model that's the [00:01:00] similar style that as our alpaca is. So that's why we are using theseFrom LangChain. We're gonna be bringing in the Hugging Face Pipeline . We're also gonna need to just bring in, some other things as we go. So first off, I'm bringing in the template and the LM chain, just to show you the, okay. We can bring this in. We can set up, our model. We've bringing it in as eight bits so that

we're gonna use less memory and it'll be faster for inference. . And here is where a lot of the magic happens is that the way that the Lama model and alpaca model are set up we can actually set up a hugging face pipeline for them, which is gonna be doing text generation. And we just then just pass in this model into here. We pass in the tokenizer, we pass in things

like our max length, our temperature our, top p and the repetition penalty in here as well. and then we can set up a local. LLM from this hugging face pipeline. So this is actually, coming from LangChain and it's allowing us to set [00:02:00] up a LLM with this hugging face pipeline there. Once we've got that done, we can basically

try it out with just a normal LLM chain. So I'm just gonna take one of the standard alpaca prompts. And style of prompts for the template. So we create a prompt template in here. We pass this in and then you can see we're

basically just setting up our prompt template with the template that we've made here. And then the instruction is just gonna be

the variable that we're gonna pass in. So this is standard stuff. We've then basically got this going on here so we can have, what is the capital of England? And sure enough this is just going to inject this question into where the instruction goes in here. And then it's gonna give us our answer out. Our answer out is gonna be the capital of

England is London. And then if I ask it, the typical sort of alpaca question of, okay, what are alpacas and how are they different from llamas? You can see here we get, the standard answer

that we got when we just [00:03:00] played around with, the model by itself without LangChain. All right, so setting up

the conversation

part is one of the key things is we want to make use of the memory here. So we are going to Lama and alpaca have a good sized token span. So we can actually go up to sort of 2000 tokens

here. What we're doing now, unfortunately, we fine

tuned it probably for a lot less than that. So it will be interesting to see. And I would like to hear back from you guys as well. How well does it do when you get a really

long token span in there? . It may do really good. You might find it at times that it doesn't do as good but okay. So we're gonna set up our conversation chain. So if you remember, one of the things that we used in a conversation chain is the whole idea of a memory. And the particular memory that I've chosen to use here is this conversation, buffer window memory. So what this is gonna do is give us a window

that we pass across the conversation. [00:04:00] and we are going to represent K number of turns. . So for example, in this case, I've decided

to set K to four, which means we're gonna have four turns going on. And that will be the limit. So this should keep us so our token span never

gets too wide in the conversation. All right, setting up the conversation chain. We just passed in our local LLM that we set

up earlier with the hugging face pipeline. We're gonna pass in our memory to be the, this window memory that we've created. And we're gonna just set verbose equals true so we can just see what's going on. If we look at the template here, So just

to see what the template, the standard template is it looks something like this. The following is a friendly conversation between

a human and an ai. The AI is talkative and provides lots of specific details from its context. If the AI does not know the answer, it truthfully says it does not know. And so you can see in this prompt we're injecting two things. We're injecting the human input and we're

also injecting the [00:05:00] history or the memory. That's coming of the conversation as it's gone through. So I decided to modify this a little bit. So here's the version that I modified it to. And you obviously play

around with this yourself. The following is a friendly conversation between a human and an AI called alpaca. So we wanted to have a little bit of knowledge about who it is. So we could also put some things in there that, Alpaca is three years old. Alpaca loves to eat apples. You could put some other things in there too. I find it's always fun. To watch people play with this if you've set it up to be quite funny as well, so you could try that out. so to do that, we basically just override the conversation dot prompt dot template. So you can see I'm just taking that new conversation prompt template and trying it out. . so now it's got our name being alpaca and stuff in there. All right, now we get to talking to it. the first thing that I tried, AI, was basically just saying, hi, there I am Sam. . So it's interesting here. we know that the alpaca is [00:06:00] fine tuned, not on dialogue, right? We haven't done a version that's fine tuned on dialogue yet. Maybe that will be a future video. But here we've done it's fine tuned on tasks, so they tend to be tasks where you ask it. It to give you a fact, or you ask it to reproduce a list of something. That kind of thing. So when I don't actually ask it something, it doesn't do a great job. It's perhaps a little bit confused in here. And what does it do? It doesn't generate bad text. It just then goes on to generate more text than we asked for. So it's taken in, you know that oh, it's talking to Sam. Hey there, Sam, it's nice to meet you. What can I help you with? And then it generates Sam's response back of, do you know what time it is, and then alpaca. Sure. Sure do. It's currently, and then it obviously passing in a token for where you could substitute the time. You could do a regex change or something there and put in the current time. . So then I realized that, okay, this is probably happening because we [00:07:00] didn't ask it a question. So it's trying to just keep being chatty and make up things itself. So if we try it and we ask it, okay, what is your name? So now it's much more on point, right? It's okay, my name is Alpaca. How may I help you today? . So we've now got multiple turns in here, but we probably shouldn't count the turn where it generated. It's Sam and it's an alpaca. It's really should be. Human AI is one turn in there. now, I ask it another question. Can you tell me what an alpaca is? Sure. It's gives us an answer. And

it's not too long here, you can see that

an alpaca is a species of a South American camelid mammal. They're typically brown or white in color and have long necks and legs, All right. So then I ask how is it different than a llama? So you see at this point now we've got these

multiple steps of our memory being passed in, of the conversation. So we've got all the steps from the start being passed in here cuz we haven't met the sort of length of the window where it needs to [00:08:00] start cutting things off yet. Okay. How is it different than a llama? Alpacas and llamas are both members of the

camelaide family? I'm not sure how you pronounce that. But they differ in several ways. Alpacas tend to be smaller than llamas and

they're, and it then it stops. so it should have been able to go on not ideal again, remember, this hasn't been fine tuned for conversation. So we are just using the fine tuned version of Alpaca. can you gimme some good names for a pet Lama? Now if we go back to here, look at the memory. So this is the next thing I'm gonna ask it. You can see we've lost the start of the conversation. So where I said, hi, my name is Sam. You know that's all been taken out now. We're going as far back now as where I asked,

what is your name? So we've got one human, two human, three human,

four human and then this being passed in. so our memory is a window of four that we've got going on there. Now [00:09:00] you could experiment with a

longer memory if you want to. But remember all this is going into the language

model. So you will find that if your memory gets

too long, then it can actually be, quite slow to do things. Alright. So I ask it, can you give me some good names for a pet lama? Sure. Here are some great options. Pata, Tika, cushy and Wayra. Okay. And then I wanted to test it, see, does it

still remember its name? So I ask you, is your name Fred? And you can see that we've lost The bit in the conversation where it told us our name. So it's only relying on the name being up

there. Yet still it understands from the context. No. My name is Alpaca. So that's good. We next asked it. Okay, what food should I feed my new llama? So again, you see where our window of four

is staying fixed, so we are losing things that we spoke about early on in the conversation. Okay. And then finally, your new llama can eat grass,

hay even alfalfa. You can also try giving them some [00:10:00]

vegetables like carrots, apples, and banana. . So this is just setting it up now. You could actually try and mess with the prompt

more to basically inject the whole personality in there. Do some things like that. But the idea here is that it turns out that llama's doing pretty well. I wouldn't say great, but it's doing pretty

well with the prompt for the chatbot and then passing in this memory that we've got going

on the current conversation. And then generating out. So this is something you could try an experiment

with and this would also work on the llama model if you want to actually try the llama

model. My guess is that probably won't do as well

as the alpaca model. I don't know, maybe I will try that out and

we can look at that in another video. Anyway, as always, if you have any questions

please put them in the. And if this was useful to you please click

and subscribe. I will see you in the next video. Bye for now.



## Talk to your CSV & Excel with LangChain

[00:00:00] All right. In this video I'm gonna be going through how you can use LangChain to talk and extract information from CSV files and Excel files just by using natural language to query them. So in this case, we're gonna be going back to the open ai language models. So you will need an open AI key here. You can see the latest version of LangChain that I'm using. And the data set that I'm gonna be using is the Black Friday Sales Data set. So this is available from Kaggle. I've put a link to it here if you want to go and find out more information about it. Really, we're just using it. As a simple way to bring in a CSV file and test some things out. In this case, I'm bringing in the trained csv. There's both a trained CSV and a test csv. It really doesn't matter here because we are not training a model. We're not doing anything like that. We're just using another model to query the CSV file. To just get a sanity check of what's here,

what [00:01:00] I've done is basically set up pandas. Which I'm sure many of you will know and then use pandas to read in the csv. And then just have a look at what, the first five records and what the column names are. So you can see that we've got gender, and you'll notice that gender is f and m. We've got age, which looks like it's bucketed age. We've got stay in city, current city number of years. We've got marital status. We've got a bunch of different things in there. We're not gonna use this particular one that

I've set up. This is just a sanity check. LangChain will use its own one to load it in. The way we're gonna do this in LangChain is we're gonna use the CSV agent. So the CSV agent , it has a number of things you need to be a little bit careful about because what it's actually doing is running, a Python agent under the hood. So you want to be a bit careful of anything like a prompt injection attacked or something like that. So if you are using it for end users that are not yourself or not the people you trust, [00:02:00] be a little bit careful about the environment that you're gonna run the Python agent in. In this case, we're just running it our. So it's not a big deal at all.

Okay, so we're gonna bring in, from LangChain agents, we're gonna begin the create CSV agent. We're going to also bring in the open ai large language model here. And we're gonna create this agent. And remember we always create an agent by passing in a language model. So that's the first thing that's being passed in here. We're setting the temperature to zero. This is to basically, we don't want it randomly changing the facts that are in CSV when it's feeding back, et cetera. We wanna try and reduce hallucination as much as possible. So therefore we set the temperature to zero. We're passing in the CSV file itself, and we're gonna pass in verbose equals true. So this is basically just so that we can see what's going on with the promptsas it goes through. All right, so if we have a look at the agent we can see that, okay, it's certainly got, a number of language models in there and language model chains in there. You can see [00:03:00] that some of the, this has got the prompt in there that we've got, which we'll look at in a second. It's also got things like, it's gonna have

an input and a scratch pad. So the idea of a scratch pad is that it can take some output from one call to the language model and use it as part of an input for the next call to a language model. Let's come down and look at the, and we can see that, as it goes on, it passes in a bunch of examples of the data and stuff that it's got in the agent. Now, it's not putting all that into each call, right? That's not the idea here. What it we've got here is we are basically passing in a prompt. So let's look at the prompt. I can just copy this prompt and it would be better if I just paste it in there. and now we can read the prompt quite easily. We can see that, okay, this first prompt is basically going to be, you are working with a pandas data frame. So this is the same thing as what I set up, but it's using its own loader for loading it. And we'll look at, you could actually use that loader as a separate component later on [00:04:00] if you wanted to. You are working with a pandas data frame in Python data frame is the name of data frame is df. Very conventional sort of thing. And then it's basically telling it, these are the tools that you can use below. In this case it's just using the python REPL, right read, evaluate, print, loop that's going on here. And then it talks a little bit about the format that it can use. And what it should output for actions for observations and any thoughts. And then we can see that what we do is we eventually, we are passing in the head of the data frame. So this is just like what I printed it out, up at the top of the notebook. We're passing in the input question and we are passing in the scratch pad as this thing goes on. So let's kick it off and start by asking a simple question how many rows are in there? So you'll see that the. agent is doing a similar thing. So actually, just one thing I forgot to point out is when we look at the the actual agent, we can see we're not using the React agent here. We're using a zero shot agent which is slightly different than the one that we used [00:05:00] when in the video I made about agents. All right, but we've still got a similar sort of concept here where we are gonna make a call. It's going to decide, what sort of action to take. So we can see that we've gone into this agent

executor chain it's passed in, how many rows are there. And it's thought is, I need to count the number of rows. So then it's gonna use the Python REPL for that. And it's gonna basically just pass in the data frame and say, okay, what's the length of the data frame? The data frame is 550,000. So then it gives us back this final answer. The next thing is we can ask it something a little bit more complicated. So how many people are female in here? So this is an interesting one because we haven't said, in the actual data frame, it doesn't use the word female or male. It uses f and m. But, and it uses the column name, gender. So we haven't used the word gender. We haven't used F, we haven't used M in the same way. But sure enough, we can see that, okay, I need to count the number of people who are female [00:06:00] and then it works out that okay to do that. Because it's got the head of the data frame, it can work out based on the columns that, ah, okay, it's gonna be in the gender and it's gonna be f and I just need to count that. So it's gone and count, counted all of that up and then it knows the answer and it's giving us this answer back 135,000. We can ask it more complicated questions too,

so we can ask it things like how many people have stayed more than three years in the city. So that's one of the fields that they have is stay in current city number of years and can see sure enough it's worked out that it needs to filter the data frame for people who've stayed in the city, in their city for more than three years. And it basically just writes the pandas query here, gets the information, returns that. So gradually as you go along should try it and do more complicated things. So here I'm asking it. Okay, what about, how many people have stayed for more than three years and are female? So again, it's done a nice job of taking input. Writing the [00:07:00] pandas query, which is gonna be, the stay in city greater than three and gender equals f and it's able then to return to us, how many of those people are female? We can ask it, quantitative questions where things like, okay, are there more males than females? Now we know this based on it. And we can see that, okay, what

it's done

is in its scratch pad. In the observation, it's basically outputted

that it's counted the number for males and the number for females. And then from that it's able to then work

out that, okay, there are more males than female. So that's the basic agent. And this will work for, any CSV that you want

to put into it. If you want to actually make a custom agent,

you could use the LangChain document loader and use the CSV loader. So this is a very simple sort of thing of just loading up the CSV yourself, and then you would put in some sort of custom chain

here for a specific task that you want to do. If you want to use an Excel file, no problem. You just basically put in some code to convert

your Excel file. So this is the example of the [00:08:00] Excel

file and put it convert it over to be a CSV file and import it that way. So LangChain doesn't have native , it doesn't have Native Excel import. And it's important that you understand that

what we're doing here is we are operating on it as if it's a data frame and not necessarily strictly like a spreadsheet where we could look at very specific cells and formulas in

those cells, that kind of thing. It's a little bit different than this. So here we've brought in another one that

was an Excel file. I've just asked that, okay, what are the column

names? It's able to work out, that, okay these are

the column names and tell us what they are. Then I said, okay, there's age. So what is the average age? It's able to calculate that out. If we look at some of these that are a little

bit more complicated. Okay. Which country appears the most and how many

times does it appear? It goes through, it works out that United

States appears the most, and you can see that once it's got this information, it's going

back to the language model to rewrite that information. So the country that appears most is United

States appearing 48 [00:09:00] times. We can even do things like. So here we can basically ask it, okay, what's

the ratio of males to females? And this is a different CSV file than we had

before, and the other one had a lot more males. This one seems to have a lot more females

than than males. But so this gives you a sense of what you can do with this for both CSV files and Excel files. And it allows you to write very simple little apps that allow people who don't know how to use pandas, for example, to query a bunch of data really quickly and find out information that's relevant to them. Hopefully this was helpful to you. If you've a, any questions please put them in the comments. Remember, if you like this video, please click and subscribe. I will see you in the next video. Bye for now.



## BabyAGI: Discover the Power of Task-Driven Autonomous Agents!

[00:00:00] Okay, in this video we're gonna be looking at a form of autonomous AI or an autonomous AI agent. And this is actually from a paper that was released online called Task-Driven Autonomous Agent using GPT-4, Pinecone, LangChain for diverse applications. It was actually announced on Twitter. I saw this last week, by Johei Nakajima. and he had a nice sort of tweet post announcing it and if we go through it, we can see that, okay. The idea here is that we are using a large language model to generate ideas, do some sort of critique on those ideas and then ideally execute tools So this has elements of throwbacks to Toolformer, some of the other key papers around this, that have been here. one of these things shows, it's got some nice diagrams in here, showing like how this works and some of the key points into it. So the user basically provides an objective and a task and then once that is set into a task queue, [00:01:00] the large language

model basically decides how to execute that, and then as it, develops stuff, it saves it to a memory. and that memory then can be accessed through different things. it then, goes back a, around these loops. it's got a prioritization agent to decide what. Task comes next and what you know is priority, et cetera. And it basically goes, through this, over time. So if we look in here, there, there's, they've got another diagram in here, just going through it as well, of basically you've got this task queue. we've got the execution agent. And each of these things, for these parts is actually just the same language. you're just using different prompts. You're using different ways to, manipulate the output. And then you've got the memory, which in this case they're using Pinecone, which is a vector store database, so that they can store things in there and that you can basically do lookups on them and stuff like that as well. So the. The idea here I think is really good.

the paper talks a lot about using, GPT-4 Pine Cone [00:02:00] and the LangChain Framework,to basically do this. along with this, he also released, some, code. So the code, has got the nickname, baby AGI. I don't think this is approaching AGI in any way, but it's cute name. And you know what, this is supposedly a very paired down version of the original one. So we don't actually know how good the original one was. As far as I understand, we hasn't been released or we haven't, got, videos of trying it and stuff like that. looking at the baby g i, that's here. this is de this code has definitely been released. We can play around with it. In fact, I've set. A CoLab so that you can also have a play with it. And so we can just have a look at, how it works, some of the ideas behind it, and what you could do with an agent like this, in the future. So you need quite a number of API keys to get this going. you'll need an opening AI key. of course you can basically set it up to use either GPT-4, or in this case I'm using GPT 3.5 turbo. [00:03:00]

It's nice enough to have a print statement that does say if you're using GPT-4, that, you know, this could get expensive. we also need Pinecone API key and you need to set up the Pinecone environment. So this will change based on your, where you're

setting it up. I just left this in so that you could get an idea of what you should be putting in there. Cause I think it's not always totally clear. you need to put in a table name. you can't use underscores or anything in this table name, here. And then you need to set up an objective and initial task. So here I've basically said, okay, plan a romantic dinner for my wife this Friday night in Central Singapore. and the initial task is make a list of the tasks, and you'll see That while this particular, one is not set up with any tools or anything, it does a nice job at going through, the thinking processes that it would need to do. Now in here it does seem like they're planning on adding tools. to, to this, looking at the, at the code base there are, tools being added. it's also interesting that looks like they're planning to add, or they've added,[00:04:00] llama, I'm guessing this is the four bit version of llama that runs locally, to do this thing. So that, that would be interesting to. See how well that does. if we go through, we can look and see, okay, after it's got a lot of setup code, for doing this and for Pinecone setup code,the main logic behind these things, is not, overly complex. So we can see that we've got the task creation agent. So this is basically got its own prompt here and we've just got nice sring basically substituting, doing kind of what LangChain does, with its prompts. and it's then able to, basically call that it's got a prioritization, agent again, same concept, but with a different prompt. So you are using a different prompt to do the same thing. you can see here you're a task prioritization, AI tasked with cleaning, the formatting and reprioritizing the following tasks. And the tasks in tasks. Consider the ultimate objective of your team passes in that as well. So this is very similar to some of the agents that we see in LangChain and that we've,looked at as well.[00:05:00] We've then got at the execution agent, again, same concept, different prompt. You are an AI who performs one task based on the following objective. Take into account these previously completed tasks, passing into context, your task, and then the response. so it's interesting here that they're setting the temperature quite high, for. whereas normally in Chen you would actually

set the temperature pretty low, like close to zero or zero, for this kind of thing. so that may have also of affected its output, for doing this. alright, so then you basically just go, you've just got this huge leap. It goes through it. Does it, let's look at some of the output that we are getting from. So first off, make a list of tasks. so you can see that it does a pretty nice job of choose a romantic restaurant in central Singapore. Make a reservation for two at the chosen restaurant. select a bouquet of flowers to surprise your wife with. This is definitely not something I asked for,

but okay. Maybe it's something that it, decided. you could imagine in the [00:06:00] future though, you would want the agent to actually come back to you with suggestions and then you would say yes or no to these suggestions. choose a romantic gift for your wife. purchased a selected gift from the store in

central Singapore. it's really going all out on this dinner. And then, finally confirm the dinner, et cetera. Okay. So, research and choose a romantic activity to compliment the dinner experience. if anything, I would say that the agent is very verbose. and again, this would all be down to the manipulation

of the prompt, that you would want for something like this. And you could imagine that this prompt might be really good for one task, but not great for another task. All right.

It goes through, it comes up with su suggesting a private Sunset yacht cruise, along the Marina Bay. Now, this, it is, it's very good in that it's getting, locations, right? And it's getting things like that. Again, this is to be expected, because we're using, one of the large, open AI models, for doing this. it's [00:07:00] quite funny how it's, choose a romantic outfit, hire, rent a luxury car, a lot of things that it, it's making suggestions, but they may not be, ideal sort of suggestions for, a romantic date in Singapore. one of the things I did find out was interesting was, and it's funny how it goes on to say, please note that you may need to provide a valid driver's license and stuff like that. I, you could imagine in the future that these things will have a variety of information on you. And then be able to use that, like

if it's got a knowledge base on you of your driver's license, your credit card number, all those sorts of things. I certainly wouldn't give this one my credit card number cuz it seems to want to spend a lot of money. so you can see here it's picked out, a jewelry store. It actually picks out, three real jewelry stores and it get, seems to get their location correct. Uh, pretty impressive. it then also picks out a nice, restaurant. and it's interesting that, the restaurant that it picks is a luxury restaurant. I think it's a Michelin, I'm pretty sure it's a three star Michelin restaurant, in Singapore. and [00:08:00] so again, this is all coming from the open AI model. there, there's nothing, unique about this that's coming from Baby agi. It's just nice manipulation of the open AI APIs, in this, it goes on and on. It takes, a bit of time to run, through these. in the end I just of stopped it. cuz it certainly, can ping the API quite a bit and get a lot of things back. it is interesting to, you know, I tried another one. Planning a party. And that also, did the basic stuff quite well. what we are lacking here, is the ability for it to come back to you and to know what it should come back to you about. and this is gonna be one of the key things, I think for a lot of these things going forward. it claims that it's contacted the restaurant and it's made,a booking. It's very strange that, about their policy on bringing outside candles. Again, this is I would say, dying in the details, o of this kind of thing. Anyway, it's here. You can have a play with it yourself. I ended up stopping it, just cuz it seemed to be going on and on. [00:09:00] Alright. The idea, I think is the key thing here. the idea of developing these, agents that have the ability to run a variety of different tasks and are incorporated with a variety of different tools. That's what we're gonna see a lot of in the future. we're gonna see this with the, chatGPT plugins or the open AI plugins format that's coming along. We're already seeing this with some of the things in LangChain, so this sort of just. Is a nice way of wrapping up some of these ideas and giving you some idea of how they could be in the future. As always, if you've got, any questions, put them in the comments below. if you found this useful, please click, like

can subscribe. I will see you in the next video.



## Improve your BabyAGI with LangChain

[00:00:00] All right, in this video we are going to be

looking at BabyAGI again, but this time we're looking at an implementation done with Lang

Chain and I think it's kind of makes it quite a bit better, in that you can see what's going

on a bit easier, and you can also mess with it and experiment with it yourself. So here we've basically got the Colab, for

doing this. Hopefully you've seen the video about BabyAGI

already. You know what it is. in this case we're gonna be using OpenAI,

and we're also, I'm doing the one here, which is actually BabyAGI with tools. So there are two versions. There is one

where you're not doing any external

calls, which is the same as what we looked at, the other night. and this one basically is where you're doing

a tool, using, a search engine to be able to make external calls as well, with this. So to do that, we basically need to

have,

OpenAI in here and we also need to have a search engine api, for the search engine chain that we're gonna be running. you can see the version of LangChain that I've [00:01:00] got going here. it's pretty standard compared to what we looked at last time. we are not using Pinecone, so we do have a Vector store, which is the FAISS Vector store, or the FAISS Vector store. but it's all in memory in this case. So, we are not doing any external calls to Pinecone. It just makes it much easier. Now, if you wanted to, you could, wire this up to Pinecone just by changing know, a few of these things. we're using the OpenAI embeddings for the embeddings of the Vector Store. and I realize I haven't done a lot of videos with Vector stores, so I might do a few, going forward just to explain some of these concepts more in depth, but I'm guessing a lot of people will know what they are. so that's the first thing is just sort of your, memory store that you're gonna be using to reference. then you've got the chains themselves. So this is where, it is actually really nice to have LangChain because we can actually see, okay, what are these chains doing? what are the prompts for them? That kind of [00:02:00] thing. So we can see that, okay, we've got this task creation and we've got three main chains that we're gonna be using here. I've put up the task creation, task prioritization, and then the execution, So you are going to give it an overall goal and it's gonna generate a bunch of tasks. so this first chain is going to be this task creation, chain. We can see looking at the prompt, right? It's basically saying you are a task creation AI that uses the result of an execution agent to create new tasks with the following objective. Now, that's where our objective will go in. The last completed task has the result, and then this is what's being generated from before this. and this result was based on this task description. This is the task description. and then these are incomplete tasks. So, if it's got a list of tasks, that hasn't done yet, this is where it will feed these in. So you can see we're actually feeding, four different variables into the prompt [00:03:00] here. based on the result, create new tasks to be completed by the AI system that do not overlap with incomplete tasks. Return the tasks as an array. so that's, this is our first sort of chain. and it's pretty simple to see, how this works. And this is where you could come in and mess

with this. So if you wanted to make something very custom, That did a very specific kind of thing. You could come in here and change these, and learn from how they've done it in here. next up is task prioritization. You are a task prioritization AI tasked with cleaning the formatting of, and reprioritizing the following tasks. So it gets the tasks passed in there. Consider the ultimate objective of your team. Objective here. Do not remove any tasks. Return the result as a numbered list like, and then it gives it, a way to format it. and it gives us the next task. so this again very simple kind of chain. We can look at this, we can see what's going on. [00:04:00] Now for the no tools version, the execution would actually be very simple. So I've put this in here and just commented out for the tools version, the execution is a little bit more complicated. So here we've basically we're defining an agent. So this is an executive agent. we are going to, we're gonna pass some tools into it. So first, let's just look at what the prompt for this is, you are a planner who is an expert at coming up with a to-do list for a given objective. Come up with a to-do list for this objective. and then you pass in the objective. and then it's got the actual tools, for this. So we can see that the tools that we're gonna pass in is search, and a to-do. So search is useful when you need to answer questions about current events. Now we could play around with this too, and say, current events or pricing or things that are, you know, could be found online, tool, The to-do useful for when you need to come up with a list of to-dos. and then it basically gives us a [00:05:00] little bit of information about that. we can see there's a prefix and a suffix to this as well. and this basically is taking in the objective, and, some of the things that we've already put in there. and then finally the question. and this is where we can put in, our scratch pad for this. Now this executor is not using a react one, it's actually just using a zero shot agent. Here. You can see we're passing in the prompt, we're passing in the prefix suffix and the input variables. So, hopefully when you look at this, if you've been looking at some of the other videos about LangChain, you'll see that, oh, okay, this is not that different than single

things

that we've done in the past. We're just now starting to, wire a bunch of these things up together. we've now got some, some functions that basically, just defined, running these things and set up the loop for this. so this is the key thing, is that we are not just running this thing once and saying, okay, you're done, we are basically making it go through a loop where it's gonna basically prioritize [00:06:00] tasks. It's going to, get the next task. it's going to, get top tasks and then it's gonna execute it, all together. So this is what this code is here doing here. It's basically just putting it all together in parts. Again, this is, different if you were doing the no tools version. we're doing the tools version. So we can see here that we've got our task list, we've got the different chains, going on for this. We're just basically making a class for BabyAGI in this. You can see that we can add a task, we can print a task list, we can print the next task, we can print a task result. So we've got a bunch of functions in here that the class will be able to use in conjunction with the language model, to run this all together. and then we can see running the actual agent. We've got our While loop. this is not that different for something like Auto-GPT or some of the other autonomous AI things. basically these things are on some kind of loop, where usually that's gonna be a While loop where it's [00:07:00] doing different things until it gets a certain result. and then it exits, based on that. So here we can see, the different things. Now we are not storing it in Pinecone, we're just storing it locally. but we can see that, okay, that's what's going on there. You can see the creating new tasks and reprioritizing it. and one of the things that you could play around with this is I actually think you could get a lot better results by having, another separate chain, which does some summarization, of like a final report or something like that. You'll see that when we go through it. that, while it's doing all the things, it's not great at coming to conclusions at the end, for this. so, then we basically just run this, and what I thought I would do is run it on the same thing that I ran the Auto-GPT, in the previous

video, so that we could check out in how well it does. So here the objective is find the cheapest price and site to buy a yubikey 5C online, and give me the so I'm passing in the OpenAI, language model. we're sending a temperature to zero.[00:08:00]

we're basically just in instantiating, this BabyAGI class here, with the LLM. So we're passing in the LLM with the vector store. we can set a max number of iterations. So this is one of the things that's nicer about this version compared to the other version. In the previous video, it was just going on and on and on. Here we're basically limiting it to, seven iterations, and then stopping, for that. and then we just basically kick it all off where we pass in the objective. so, okay, first off, it's going to make this to-do list and it's gonna say, okay, what do you need to do? So it's a to-do. I need to find the cheapest price and site to buy a YubiKey five seat online and give me the URL. All right, so the two do list is research online retailers that sell Yubikey 5C, eg, Amazon, Best Buy, et cetera. This is really good, that it's done this. compare prices, across different retailers. Again, very good. [00:09:00] Check for discounts or promotions available. very good. I will say that it doesn't actually do all these things, but it's kind of cool that it's, thinking of these things. And it could be that it didn't do all these things because I limited the number of iterations. That's quite possible too, actually. read customer reviews for each retailer to determine the best site to buy from. You see, all the stuff that's generated here is amazingly good, right? if this bot would work like this, this would be very useful. All right. and then it's basically going to, get the cheapest price with the site and give us a url. So, it comes up with the different tasks. it starts doing, some search. So you can see it does a search on prices of Yubikey across different online stores. its observations are, some of the places that it sees them and which Yubikeys it's seeing. it's thought that it, okay, I need to find the cheapest one. So the search, cheapest price and site to buy YubiKey online, observation is \$50. So this is definitely, in the ballpark if you remember. Just quickly [00:10:00] looking at it, this

is what we were trying to find yesterday and Auto-GPT did manage to find, this price of \$55 off, Amazon, looking back at this, we can see that, okay, basically, it's going through this chain. It's basically making, our it's tasks, lists it's going through again. it's doing search, for discount codes and promotions. I would say this is where maybe it starts to go off track actually a little bit. it gets some codes, for this where it basically, finds some different things. it then it goes back to its list of comparing different things, and, adding any other things that it needs to have here so that it comes out with the, compare the prices of the YubiKey 5C across different online stores. and it's still kind of stuck on this thing, right? Again, it's finding the same, kind of prices. it comes back, moves on to identify an additional discounts or promotions available, for the Yubikey. Here it's stuck again, looking for the promotions and stuff like [00:11:00] that. Finally, it gets to something where it is giving us a url. we can see that like, just sort of fast forwarding through. It gives us a url, and it says the cheapest, price and site to buy a Yubikey 5C online is, \$50 from Yubikey. So it gives us this url. So this url, products Yubikey hardware. When I click it, you can't see it, but actually this, it seems that URL doesn't exist. So we get to the products page. and then under hardware, sure enough, there is the five series that we can look at, which is where we're at, looking at this. But if we actually go into the store, and scroll down, we can see that. Okay it does look like the 5C. Is \$55, not \$50. So it's definitely found us like the right area of the internet and that kind of thing hasn't been perfect at, finding, the actual item and giving us the right [00:12:00] thing there. my guess is that it's semi hallucinating, parts of the url because if you, look at this, the other thing also could be, the actual location that I'm in. So as I'm recording this video, I'm currently in Australia. and so I do notice that they have a slash AU in there. So it could be that where the Colab version is running in a US data center. I'm looking at it from another thing, but

that said, when I tried this also with a vpn, it still didn't give me the exact one. Try it out yourself and find out. I thought I would, then sort of test this

out a little bit more. So I said, okay, find the Yubikey on Amazon

and give me the url. long story short, it went through and it certainly

found this URL. Again, this URL doesn't exist, right? when we click this, We get, sorry, I couldn't

find that page. Even though the details on the URL pretty

accurate and pretty similar to the real url, it is slightly different, So we might want

to [00:13:00] have, another module in there that actually checks, does this URL exist? and look at it that way. So anyway, this is basically using in LangChain,

I definitely think this is, Studying and, thinking about what you could use it for,

how you could mess with the prompts, how you could change different elements of it. I will also give you the, no tools version

of it as well. So you can play around with that. another thing that I tried out, cause again,

getting a lot of questions is could we do this with a model that's not OpenAI? So I thought, okay, let's try this with Koala.

Alright, so Koala's one of the models that

we looked at last time. So here I'm basically, bringing in the Koala

model, and setting up the Koala 7 billion model. and to do this with LangChain, we basically

need to set up a pipeline. And I've set up a pipeline, for, text generation. I've set up, some things with this. You could play around with these. I probably should have done that with a lower

[00:14:00] temperature.but the problem is not that. So the problem is that this model is not actually

trained on this kind of thing. So, in a future episode, I might actually

train one of these models on to do this task, so we could actually look at it. But you can see that here if I give it a task.

So I was asking it to basically get Melbourne

weather and see if you could, find this for me. And, the problem here is that, while it can

do, parts of it, when it gets to doing the executor chain, its output is just not a valid

tool. So it's not used to generating an output that

the LangChain would be able to accept and parse to decide which tool it is. So you go into these loops of, is not a valid

tool. Try another one is not a valid tool. Try another one. and eventually it will just crash, So you could play around, with this. The way to fix. And maybe we'll do this as a new video, is

that you would basically make a data set [00:15:00] of examples, of what it should be like, and

then you would train up, you know, you'd fine tune the Koala model on this, and then come

back and see like, okay, could this work? So perhaps will look at doing that, over the

next week or so, when I'm not traveling. And then we put it together and sort of show

how you could do it. So there's nothing stopping you doing this,

in regards to LangChain or the language model. It's just that these fine tuned llama models

that we've been looking at are just not really built for doing the LangChain tasks. and they're not really big enough to

have

enough understanding that they can learn just with the, in context learning of how to do

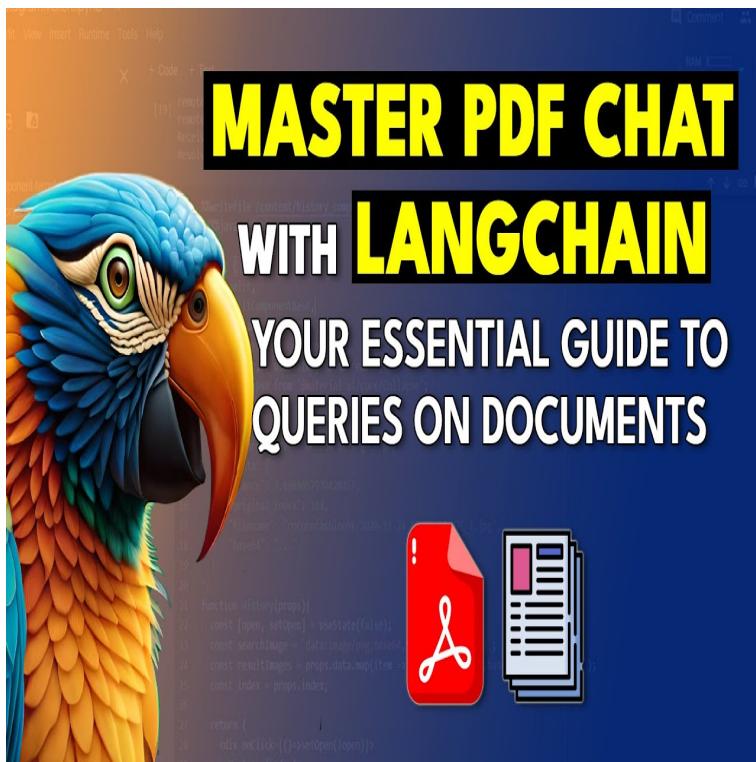
these tasks. So you'd probably need to fine tune a for

that. So that's something that we can look at, in

the future. Anyway, as always, if you've got questions,

please put them in the comments. if you found this video useful, please click

like and subscribe. I will see you in the next video. Bye for now.



## Master PDF Chat with LangChain - Your essential guide to queries on documents

Okay. In this video we're gonna be looking

at using a lang chain to chat with and query PDF files. And more importantly than that,

I'm gonna introduce a whole bunch of different elements that we're gonna use in a lot of

the things coming up on videos as well. this is the game plan, and this is the key thing

to understand here. Is that while we can while we could do a simple

questionnaire over something and feed that into the context of the prompt, if our prompt

is limited to 4,000 tokens or perhaps maybe even 8,000 tokens with a G P T four. It's

still a very finite thing, and if we wanted to go through a book it's not gonna work that

well. So this is where we are gonna introduce a

whole bunch of new things that relate to vector stores, embeddings, using,queries to [00:01:00]

basic and queries and retrieval, to get information that we're gonna basically use, for doing

our. chat with our PDF file So the way this works is we are gonna have a document, and the document that I've chosen, is, this, so this is a book by Reid Hoffman that he's released, all about, GPT-4 and all about,ai. And he actually wrote it with GPT-4 and,I don't think he's put out the book to make money. He's basically put it that you can order it on Amazon or you can download the free pdf. And that's exactly what we've done here, I've basically just downloaded the free PDF and you can see that it's, it's got quite a number of pages. and we're gonna go through and we're gonna convert this, pdf into something that we can query and that we can talk to. as we go through this, so the first thing, that we're gonna be doing is we're gonna be just loading in the PDF file. and we're gonna split the document into [00:02:00] chunks, And the idea here is that we are gonna build a vector store. And the vector store, we are going to, query that vector store. So just like you would with a search engine. we're gonna be looking for things in that vector store, but rather than do it through keywords, we're gonna be doing it through semantic, meaning, and this is done, I'll look at doing a whole video about the concept of embeddings But what we're gonna do is we're gonna make up an embedding for each chunk in here, and that embedding is gonna be a vector. It's gonna be quite a large vector cuz we're actually gonna get, in this case, we're gonna get open AI to basically make that vector for us and that vector will be representative of all the information that's in that chunk. So if we ask a question about some information, when, we basically. embed the question, which we're gonna do later on. We would have like our, our standalone [00:03:00] question. We would embed that and then that will go to the Vector store and the vector store will then return the chunks of information that are most relevant to what we want, and then they will be passed into the language model along with our question, and then the language model will basically decide, okay, reading through these, what is the answer? the idea of, the idea of this sort of semantic search thing is not a new one. It's been around for a long time, and even the idea of using a large language model, to query. A section of text is also something that's been around

for a long time. so there've been some famous data sets like squad, and people would train up models like Bert on doing this kind of thing. And it turns out that, this is something a model can do pretty well, that if you give it some text, in its context, Later on, we'll look at the prompts to see how this is done. that he can then look through that text, work out okay, what relates to [00:04:00] the question asked, and then return an answer for this. So there are a number of key things we've

gotta do here. We've gotta basically load up the document. We've gotta split the in document, we've gotta create the embeddings, we've gotta put it in a vector store. So a vector store is like a database. and then we are gonna basically, use Lang chain to, create our, our, our chain, which can then basically take a query, look up a vector store, get the information back, bring that into the chain, and then combine that to finally come up with an answer. Okay. So here you can see I've got the, data for downloading this. and we've got, our first things that we're gonna be bringing in. first off, we've just got a PDF reader. Now this, I'm not saying that this is the best PDF reader in other videos we'll look at using some other ones and stuff like that. This, the idea with this one is everything's simple. and then we'll do other videos about each specific part. For this, the embeddings I'm using, they're open AI [00:05:00] embeddings. You'll see in future videos that I will show you free ways that you don't need to, pay for these embeddings and use open ai. we need a tech splitter and then we need a

vector store. And in this vector store, I'm just gonna store it in memory,with a library called fi or fa face, here. okay, first up reading the PDF in. So we've got our PDF reader, it brings in the pdf, and. You can see that from this. We've basically got our text, and we've got our text as one, long string, in here. and if we, if we look at how many characters that is, we can see that it's, almost 360,000 characters. If we look at just the first a hundred characters, we can see that this is what we're getting out. Now you can see that the formatting is not great, right? We've got some spaces in weird places and stuff like that. This is where. over time you would try out

different kinds of ways to bring your data in and it will be different for each project.

For some projects, something really [00:06:00] basic will work well. For other projects, you'll want to use a library called unstructured. for some projects, maybe you want to use an API from AWS or Google, cloud, et cetera. All right, but we've brought that text in.

Now the next thing up is to basically split it up. So here we've got a text splitter, and this is the most basic kind of splitter here. We are literally just splitting on the number of characters. And so we are basically gonna say that, okay, we are gonna have a chunk size of 1000 characters. And then we're gonna stride over it. So what do we mean by that? we mean that there's gonna be an overlap. so imagine if I had, imagine if I've got a few sentences and half the thing I'm talking about is in the first two sentences.

And then I split, and then, half of it was in the second two sentences. So maybe I've got like your 50 sentences. but the last two sentences have got, a key piece of information and they go on to the [00:07:00] first two sentences of the next chunk. So in that kind of case, to prevent that we have this moving window, which is this chunk overlap and we slide this across. So this is basically gonna mean that we're

gonna have an overlap of 200 characters, for each chunk size, so a lot of the text or, a portion of the text will appear in more than one chunk. but because we are taking the whole semantic meaning of that, that we're gonna make an embedding, it's gonna be different, from chunk to chunk. So each chunk will have its own, semantic meaning in there, but just in case something was gonna overlap, we want to, we wanna make sure we can, we can catch that. So there are a whole bunch of strategies for this, depending on what your data is. I'm using a very small overlap here. You do want to use some overlap. I see some people, tend to use none. You really would like to use some overlap. but then for certain things, if the meaning is, is [00:08:00] changing quite quickly, you might have an overlap, of 500 so that you're do taking like the first thousand, moving 500 along taking from 500 to 1,500, moving along from 1000, alright? Obviously that's gonna create a lot more chunks. So you have to balance the number of chunks you're getting with the amount of overlap

that is gonna be good for you. All right, so we've got that. You can see after that we've ended up with 448, different sort of texts or mini documents here. And if we look at some of these, we can see that, okay, if we look at the 20th one. We can see that it's got, it's obviously starting with half of a sentence from a previous one. it's going along, a as we do this, and we can see that, okay, it's got a whole bunch of information in there. We look at another one. We can see different information in there. So it's just splitting this up into chunks there. Next up, we want to make embeddings for these things. So here we are gonna basically use the open AI [00:09:00] embeddings. I, so if we come in and we look at this is, we're just setting up that this is gonna be the embeddings and then we are gonna basically use, this FAISS library and we're gonna just tell it from the text. We're gonna pass in the text, we're gonna pass in the embedding maker. So this is actually our embedding maker. And it will then create, the vector store itself. and if we look at this, we can see that once we've got this created, we can actually look at the embedding function. And we can see that, okay, sure enough, it's using open ai. We can see the model it's using is text embedding ADA 0 0 2. we can get some other key information, going on there. Now, if we, if we want to test this out, so if I come along and say, okay, how does G P T four change social media? This is a query and I passed this query in to the doc search, and I'm using similarity matching, search here. So we are looking for something that's just purely. the closest, to this. Now there are a bunch of different, ones that you [00:10:00] can use. This is probably the most common one that you'll see used. I, we'll look at some more of these things, a as we focus in maybe just on vector stores in the future. but here we're passing our query, by default, this is gonna return four documents, the four closest documents, to. This, which is also being embedded with the, with our embedding function from Open AI up there. and then if we can see, if we look at the first result that we get back. we can see that, sure enough, social media is mentioned in there. If we look, here, we can see social media

being mentioned in the first one that we've got back. In fact, it's mentioned multiple times in here. So we can see that semantically, it's getting good information back and then this is what's gonna be passed into our, our language model chain, to be used for what we've got going forward with this. All right. All right. so next up we want to actually just build a chain. So we've done now [00:11:00] the loading of the document, and one of the things I should point out is that even though here I've basically just loaded one document, I could load up multiple documents and feed them in, in the same way. And then I would just chunk them all together, which is fine. that's something that I could do in here. we can also load up documents and store different, metadata with some of the vector stores as well, which is a key technique, that we'll look at in the future. Okay, back to the basics for now. So we've got our plain question answering chain. So this is basically, just the load QA chain. We're just using open ai. You could try,a variety of different models here. and the chain type we are using here is stuff,what stuff means is that we're literally just gonna stuff everything in to one call, So ideally what we're gonna stuff in there is going to be less than 4,000 tokens, that we've got in there. next up we've got our prompt. so let me just copy [00:12:00] this and we can just paste this in here to be able to see it easily, and we can see that, okay, this is our prompt. Use the following pieces of the context. So that's what got returned from the Vector store. to answer the question at, at the end. if you don't know the answer, just say you don't know. Don't try to make up an answer. So this is where all the stuff from our database or our vector store is gonna go. This is the query that we are gonna put in there as well, and then it's gonna answer this. So you can see here, if I come down and I ask it a query, so let's look at doing this sort of bit by bit. So I've got my query and I say, okay, who are the authors of the book? And then I passed that query into the vector store,in here for a similarity search. So that would return back for documents. And then I pass those documents into our language model chain here, and I pass in the query as well. it comes back with an answer. Now

the answer is not perfectly correct in here. really, the [00:13:00] authors were Reid Hoffman, which it did get right. That's pretty good. And the second author was GPT-4, Looking at the book, it's by Reid Hoffman with GPT-4. So it's gotten part of it, right? But it's obviously, it's taken the word authors that it felt like, oh, it has to be two people. And so that's probably why it hasn't, put GPT-4 there. It's looked for the closest other person that it thought would fit in there. just to show you that what this is actually

doing, is using that, those inputs though from the Vector store, let's say I say the same query. But for the vectors, I pass in a different query. I pass in, has it rained this week into, so that's getting four docs back up, answering has it rained this week? Then I pass those docs in and the, who is

the author of the book query and you can see, it comes back with the author of the book is not specified in the given context. So it means that you want your vector store to return useful information. You don't [00:14:00] wanna make the amount of documents too small, you don't just wanna return the top one. Cuz quite often the answer will be in the third, the fourth, and sometimes maybe even the fifth one, in there. So you want to allow for that when you're doing this. So just to show you with this, you can actually. Set this, this number. So up all these ones,

we've been getting four back, but that was just because it was the default. If we wanted to actually set it, we could set it over here, we could change it to six or something like that. Obviously with the stuff method, and this is what you're gonna see in a second with the stuff method. We can only put in a very finite amount of stuff in there. So if we suddenly, say, okay, K equals 20, and I return back the top 20, searches, there's gonna be too many tokens and you'll get this kind of error. the models maximum context length is 4,097, however you requested 5,000, 300. So you could see that's the kind of, era that [00:15:00] we would get for this. So one of the ways to get around this, Is to change the chain type. we've looked at these kind of chain types before in one of the videos. there are a number of different ones. Stuff is where you're putting everything

in. you've got map produce where you can basically do each one, in parallel. and it will basically run this on each one.

That can turn out to be quite a lot of. you know that can be quite a lot of calls though,

to the api. so anytime you can use stuff, it's good to use that cuz it's gonna cost

you less. the map, re rank one. Let's look at this. So here I've basically asked it,

who are open ai and I'm asked for 10 queries back. And so you'll see that what we get back is

not just a simple answer we get because we've told it. Okay? return your e intermediate

steps. So for each one of these things, it's gone and called the language model, in there,

and you can see that it returns an answer for each of [00:16:00] them, or it returns

what it, what it got. And then it scores it as well. So if we look

at the final output text, We can see open AI is a research organization that develops

and shares artificial intelligence tools for the benefit of humanity. All right? and then

if we look at the intermediate steps, we can see each thing that we got back. And then

the score that it gave. So it's created an answer based on each one

that it's got back. And it's obviously got, obviously Open AI is mentioned a lot in this

book. and it's given some of them 80, it's given some of them 90. and it's given,some

of them a hundred. So we can see this one OpenAI Technology Company focused on, so it's

gone for probably the two one or the three 100 ones. And merge those together to give us our output

there. If we come and look at the prompt for doing it this way, we'll see that the, the

prompt is actually different. So we can see here that the prompt is [00:17:00] basically,showing

it that this is the question. This is a helpful answer. This is a score. and this is for merging it all together, and

it gives some examples that, okay, if something had a higher score, you'd want to pay attention

to this. when you're getting it back. So there's some basic chains for doing it. I, another

way is we can start to put, rather than have,in separate bits where we've got one thing to

query our vector store and one thing to, query the LLM. We can put it all in one chain. So here we

are basically starting to use the retrieval QA chain. And what this is gonna do is take

a is take a retriever. So I'm just taking the retriever that we already had this doc

search that we've set up already. and I'm just saying as retriever passing in similarity,

passing in, four for being the number of docs that we want to get back. and then now I can basically pass this into this retriever QA from chain type. So the type I'm gonna use is stuff. [00:18:00] I'm gonna pass in the retriever, and I'm gonna get our source documents back, here. Now you can see if I ask it, okay, what is OpenAI? What it will return back is, the result, which is gonna be the answer. So this is the answer similar to what we got before. And then it's gonna return the source documents of where it sourced that answer from, so that we can see here. if I, ask it other questions, I'll get things like that.

Now, if I don't want all of that, no problem. I can just put in the query, and just pass back the result. So here I'm saying, okay, what does GPT-4 mean for creativity? and I can see that, okay, the result GPT-4 can amplify the creativity of humans by providing contextualized search gives us a whole nice answer for this. I, if we look in, the book,if we look at the book, so I've chosen sort of questions based on what was in the table of contents, do, would it answer some of these questions? And I encourage you to play with this,so you [00:19:00] get a sense of what it can do, what it can't do. one of the tricks for this.

Is that certain things that would be really obvious? I are not actually in the book that much. So if you think about,who was the book written by? we can see that, that it's jumping into, this, but there, it doesn't refer a lot to who it was written by. We've just got this one tiny bit by Reid Hoffman with GPT-4 there, so it certainly got the Reid Hoffman part. but my guess is it didn't think of GPT-4 as being a person that would write a book or as an author themselves. another one I'll show you. So here's, an example where I asked it. Okay. what has the last 20 years been like for American journalism? Okay, so the last 20 years have been difficult for American journalism industry, and then it goes on to give us a full answer for that. I was curious to see, so if we come in here and we click on journalism, Sure enough for the American journalism industry, it's been 20 years of mostly bad news and you can see that [00:20:00]

Then it goes through this. So in, in effect, this is summarizing parts from that chapter and what it's gone and done there is it's used the query to go to the vector store to get the information out and then it's pulled it back and put it all together.

All right, and then it's formulated an answer based on, the top four results that it got back. Now, probably all of those results came from that chapter. maybe not, but we could, you could go in and have a look at the,at the outputs and see, this, so one of the big things if you're building something for production is you'll want to, look at what your vector store is returning back so that you can be on top of working out is your problem. With the llm if you're getting errors or is it with the Vector store not giving you the right information back? how can journalists use GPT gives us, nice answers for these. and then another one here. Then finally I ask it a sort of nonsense question, what is Beagle Bard? now this is not mentioned [00:21:00] in the book anywhere, so this is just to show you that if your model's doing well, you should get, something where when you, when it's asked a question that's out of context, it should just say, okay, I don't have a good answer for this. Or, this is not mentioned in the,in any of the contexts given or in any of the documents in there. So this sums up just basically doing basic, q and a with, with a variety of, different chains, for doing this. So again, the key parts here is we have the documents splitting, the, chunking this up, creating embeddings, embedding it into the vector store. then for the querying part, we are using, Lang chain. To basically use, look up the vector store, get the relevant documents, provide that back for an L L M lookup, and that will give us the answer back. anyway, in the next few videos we're gonna look at, more, fancier things that you can do with this basic setup. And all of these things are gonna be, generally using some kind of, vector store, some kind of tool, some way of querying something and passing the context of that back into the large language model that we've got here.

Anyway. As always, if you've got questions, please put them in the comments below. if you found this useful, please click and subscribe. I will talk to you in the next video. Bye for now.



## Using LangChain with DuckDuckGO Wikipedia & PythonREPL Tools

[00:00:00] Okay, in this video we're gonna

look at some of the new tools that have come out recently, and using them with LangChain. So specifically I'm gonna look at three, tools

in here. the first one being Wikipedia. Second one being Duck Duck Go. And the third one being the Python. Read, Evaluate, Print, Loop. that we've got going on here. as always, I've just installed, LangChain. I'm using OpenAI in this case, which I'll talk about why when

we go through, and the first thing, there are gonna be other packages that we need to install for these tools. So Wikipedia is one of those. so we can just pip install that. and then to set up the tools you're gonna

find for each of the tools to setting them up is actually pretty easy. So here I can basically just, bring in this Wikipedia API wrapper, set it up, and this is a chain in itself. That when I run this and I pass in, a query,

it will look for the Wikipedia page for that. So, for [00:01:00] example, Wikipedia run LangChain, it's gonna go and find the LangChain page, and get the info, about that. if I was to put in someone's name, it would go and look for a, a Wikipedia page

about them. the next one I've actually, I've got is the Read Evaluate Print Loop. this is very simple. This one, you don't even need to pip install

anything. here we are just, again, we've got the idea

of just bringing it in. We've got the wrapper and then this is makes

a chain, and then whatever we run in Python in here. we will evaluate and print out. So you can see here I've just got 17 times two. It's gonna print out 34. if I wrote a whole function in here, it would evaluate that and print that out as well. So one of the key things I'm trying to go

for here is that in future videos, in the next video, we're gonna look at doing custom

tools. So one of the key things that you want to

do a lot is doing custom tools, but to have a, just a good understanding of how these

work, I'm [00:02:00] going through some of these

basic ones and I'm gonna show you how you could put them together as well. okay, we've got, that for running Python code. And the last one we're gonna look at in, this

video is Duck Duck Go. So duck Duck Go is a search engine. the great thing with this one compared to

the Google SERP API is we don't have to pay for any API calls with this. we can just bring this in, we've got the wrapper and then we can just do a search on this. And it will basically go and find out information

and return it back to us. So each of these things is powerful in their

own right, but what we wanna do is by putting them together,

we're gonna make an agent. That can actually use all three of these tools,

and it's going to decide when it uses which tool by itself. Now, this is the reason why I am using the older, open AI API in here. If we were gonna use the new chat one, and

we just ran it just like this, you'll find that it actually will get things wrong.[00:03:00]

so that one needs to be, we need to sort of recalibrate the prompt for the chatGPT api. maybe that's something I'll look at doing

in another video. here though, we are just going with the basic one to, look at this. and we're gonna set up the tools. So tools, basically just set up as a list of tools. and we are going to create a tool. So this is just a class that creates a tool. the tool has the name, of the tool. it has the function that's gonna run it. So just like up here when we did the Python, REPL, that was our wrapper function. And we did, Python rEPL.run and we passed in whatever we wanted there. the same is gonna be here. this is gonna get, the output of a language model passed into it, to run. Now, the last thing is we need to define a description, for the tool. So this is also really important, is that you want to basically, describe how your tool would be used, in the agent. So [00:04:00] here I'm saying that, okay, this is useful for when you need to use Python to answer a question. you should input Python code in there. So that's for the Python one. All right, so I've got that tool set up. I can set up tools outside, the actual tools, list. So you can see I'm setting up the Wikipedia one. So again, we've got a name, we've got the function for running it. and here I've got, the descriptions. The description in this case is gonna be useful for when you need to look up a topic, country or person on Wikipedia. And the last one, is gonna be Duck Duck Go. So this is, the name is Duck Duck Go Search useful for when you need to do a search on the internet to find information that another tool can't find. Be specific with your input. So the idea is that it's gonna put keywords in to do a search on duck dot go there. okay, we've defined these three tools. I need to add the other tools to the list. So that's just simply just simple python, appending these to the list.[00:05:00] So then basically I've got all of these in a list of tools, which I'm gonna pass into the agent. So next up, we've gotta define our agent. So here we are basically going for a very simple agent where we're gonna have a zero shot agent. and we're gonna initialize this agent, with the zero shot react description. So you'll see what that is. When we look at the prompt, we're gonna pass in the tools. We're gonna pass in the LLM. we're gonna passing the verbose=true throughand we're gonna pass max iterations. So

max iterations is the number of times it

can kind of think to itself about is this the right tool? And then if the tool doesn't work, it might decide that, no, I actually want to use a different tool or something like that. Okay. So, in a second we'll look at the prompt and

we'll see what's going on with this react description here. But first, let's just try it out. So we've got our zero shot agent run. When was Barack Obama born? So this kicks off this agent executor chain. [00:06:00] And this It's gonna pass in our

query. And then that's gonna sort of redefine it. It's gonna work out like what does it need to do? So it says, okay, well I need to find out

when Barack Obama was born. So the action that it decides on this is it's gonna use Duck Duck Go search. And then the search that it's gonna do is gonna be action input Barack Obama birthdate. The observation is what it got back. And so we can see that it got quite a lot

of information back. but from that, it's gone through and it's decided that, okay, well, the thought is Barack Obama was born on August 4th, 1961, and then

it gives us the final answer and it finishes out the chain. So this used one tool of going to Duck duck go for that. If I asked it something like, what is, 17

by six? Okay. So in this case,it's gotta calculate what

it is, so it says that I need to do a calculation, but rather than just sort of try to use the [00:07:00] language model to calculate it, it knows that, okay,

this calculation could be run in Python. So it basically decides that the action it's gonna take is the Python rEPL or read Evaluate Print Loop. and its input is 17 time six that

we've got there, but written in the way that Python would understand that, it gets the answer back. It says, I now know the answer, the final

answer is 102. So we can see that it's managed to do both

of those, really well. So let's have a look at the prompt. What actually is going on here. So when we look at this prompt, we can see

that the prompt that it's sending in, in this case, let me just make sure that's the latest one up to date. it's gonna be answer the following questions. As best you can, you have access to the following tools. And this is where you see our names, the Python REPL and then the description. Useful for when you need to use Python. To answer a question, you should input Python code. Duck Duck Go search useful for when you need to search the internet, et cetera like we [00:08:00] looked at before. Wikipedia, useful for when you need to look up a topic, country, or personal Wikipedia. so it's got that and then it's got the use the following format and then it passes in the format for this. So, we're not giving it really a lot of examples in this. So there are some other react agents that you can run where it passes in some examples as well. and that's something that we could do if we wanted to try with the Turbo api. Maybe we will look at doing that in the future. the idea here is that, it's gonna basically use the format question, the input question you must answer. sort of rephrase what the question is that you've gotta answer, thought you should always think about, what to do. Action. The action to take should be one of, and then we pass in that list of the three tools. And then the, action input, the input to the action observation, the result of the action. and it shows that, this can be done for a number of times. And then the final answer, and then you would give the [00:09:00] final answer. So then we pass in the question and we pass in the scratch pad. So the scratch pad is what's taking the information that we've received or already from one of the tools. So now you can see if I ask it. Okay. tell me about langChain. And this time it didn't use the Wikipedia one. So the problem with these three examples is that the Wikipedia is actually very similar to Duck Duck go, right? They can, they both have the ability to look things up. so in this case, it's decided to use duck duck go for this. So it's gone and done a search. It's done a search for langChain. It's got some information back, and it comes back with our final answer. LangChain is an open source Python library

the name anyone who can write code to build LLM powered applications. All right. so I wanted to try and get it to do at least

one where it shows you Wikipedia. I knew that I had country in, one of the things about Wikipedia, in my description of Wikipedia. So the base model, when I said, tell me about

[00:10:00] Singapore. the base model obviously has some understanding

is maybe a little bit of a stretch, but it obviously has some representation of Singapore

being a country. So when it sees that, it's able to then work

out. Singapore is a country, so I should look it

up on Wikipedia and sure enough, it chooses the action Wikipedia, the input Singapore,

it gets the Wikipedia page back and you can see that there's, on that page there's stuff

about Singapore dollar, Singapore Army, Singapore, other things, but it realizes no that we are

focused in on the country. So Singapore is an island, and city state,

in Southeast Asia. And it gives, goes on to give us a bunch of different information about

this. So this is an example of it using, Wikipedia. for this, if we didn't have the Duck Duck

Go, it would've gone to Wikipedia for the langChain one because it doesn't have, another

tool for doing basic search. All right, what if I give it, let's try some

other things. What is the current price of bitcoin? Okay, in this [00:11:00] case, it realizes

the best place to get this information. Duck Duck go. current price of BTC, goes for it and just

returns it. then I was thinking, okay, well let's show

one more with the Python REPL library. So I tried, is, 11 a prime number? And sure enough, it decided that yes, you

know, that the Python REPL library was the way to go. but it, it kind of didn't do a great job. So you can see what it did

was it wrote out

11, divide by two, what's left over? if there's nothing, it might be a prime number. You can see that this is what it's got back

is that, oh, it might be a prime number. So in this case, it decides well, I've still

got two possible iterations I can do. I'm gonna go and do another search and I'm

gonna use duck Duck Go. I'm going to do another tool and use Duck

Duck Go search and it searches. There is 11, a prime number, and it obviously finds it in a list of, prime numbers somewhere on the internet, that it's able to return that. So I wasn't that happy with that, right? Because I [00:12:00] wanted the Python REPL to do it. So I thought, okay, well nevermind, we'll just try rephrasing a little bit. And so we write, write a function to check if 11, is a prime number and test it. So here you can see it's right. I need to write a function to check if if a number is prime. Okay, what do I need for that? I need the Python, read Evaluate print Loop. So then it writes this function. and then it obviously runs then I need to test this function. So then it tests this function, is it prime, it prints back that it's true. And then so then it finally comes back and says, 11 is a prime number. So you see this is done multiple calls to get the answer in this case, for both of these, they've done multiple calls in there, so I encourage you to play around with it. you can, you can certainly try with the GPT 3.5 Turbo api. You'll just find that it doesn't work as well for some of these react things where it has to choose, the tool. you can improve it [00:13:00] by rewriting the prompt a bit and also rather than using a zero shot agent, giving it some examples, for the agent. But anyway, this shows you, some examples of running tools, in langChain. In the next video, we're gonna look at how to make custom tools so that you could make your own tool, if you wanted to do something, just by writing some code in Python and then using that with LangChain, as you go along. All right. as always, if you've got any questions, please put them in the comments below. if you found this useful, please click like and subscribe. I've got a bunch of videos coming up on langChain, over the next week or so. Bye for now.



## Building Custom Tools and Agents with LangChain (gpt-3.5-turbo)

Okay. In the last video we looked at using tools

and using some of the off the shelf tools that are already built into LangChain. one of the key things that will really make your conversational apps be much better is when you start writing custom tools, So in

this one, I'm going to give some simple examples of some custom tools, show you how you can

put them together. And then also, show you some of the challenges

that you'll face. So one of the big things that people have

asked me a lot is why was I using, the older OpenAI, API and not the new GPT-3.5 turbo

API. So in this one, I'm going to use the turbo

API. I'm going to show you some of the challenges

that you'll face with this. so just quickly, I'm using, currently today

as I'm recording this, the latest version of LangChain seems to have, some bugs with

it. in regards to some of the tools. So I'm using [00:01:00] 0.0.150 for this. I'm sure there'll be fixed pretty quickly. All right. So first off, I bring in the keys and then here I'm basically setting up to use the chat model, The turbo model. and I'm actually just going to call this turbo LLM. we're going to set the temperature to zero because we're getting this to make decisions and we don't really want those to be random. We want them to be consistent. and ideally we'd like them to be reproducible, but that's not always the case. Unfortunately. I could see if we do something like a standard tool. so the Duck Duck Go search, which we looked at in the last one, we would load it up like this, where we basically just bring it in. and we can set up the, these elements and we pass it into a list of tools. and that list is what it's going to be sent into the conversational agent for, Deciding using the react framework, the reasoning and action framework for deciding which [00:02:00] tool gets chosen. So let's jump straight into just doing a custom tool. And I want you to realize that at the heart of it, custom tools are literally just a function that will get called. And, a way that the, language model can call that function. So here I'm making a simple tool, which I'm calling the meaning of life. And all this does is basically just returns a string saying the meaning of life is 42 if rounded, but is actually 42.17658.and that's all this function is going to do, You come, you have to pass some input into it. So that would be a string coming in. but we're not going to use that In this particular example. So once you've got your function done that's the bit that gets processed. The next thing is basically just defining the tools. So up here we imported, the class of tool. [00:03:00] And we're basically just saying a tool and we're instantiating it with a name with the function that we created with a description. So I talked about these in the last, Video that, here these are the things that it's going to be passed through to the language model to decide which tool to use for each thing. I'm going to say that the meaning of life

is useful for when you need to answer questions about the meaning of life. input should be MOL, Meaning that, it's best to tell it, to give some input when

it's going to give an input. So in this case, it's going to put in that

as you can see, we're not using it in the actual function. The next tool I'm going to do is the same

sort of concept. But this one's going to be a random number. So in this case, I'm just going to basically

take, we're just going to basically return a random number and that's all this is going

to do. You can see when I run this function, it's

so simple it's just generating a random number from zero to four there. again though, we need to define it. [00:04:00] So we give it a name of random

number. We pass it in the function here is going to

get executed. And we pass in the description so useful for

when you need to get a random number. Input should be random. Now we're going to set up an agent to actually

use these tools. So you can see we've got these tools and we're

passing them into this list here. so I've just redefined the list rather than

use the standard one that we had at the start. I'm going to have a conversational memory,

which I'm going to limit to three rounds of, the conversations. I'm going to then set up the conversational

agent to initialize our agent. And this is going to be a conversation, a

react agent, but because we're using the turbo LLM, we're going to put chat in here. So this is going to be a chat, conversational

react description. And you can see we pass in the tools, we pass

in the LLM we're passing in verbose equals true passing in our memory, et cetera. We've got the max iterations so that it won't go on forever. If it gets into a loop or something, it will

[00:05:00] just go through choosing tools three times and then return its answer. So let's have a look at some of the outputs. So the first one is I ask it. Okay. What is the time in London? So the age of executed decides that well, actually for that, I might as well use a search engine. and the input to the search engine is time

in London. And sure enough, it gets, we can see it in

the blue writing here And we can see that what we get back is a

bunch of information about the time in London and showing what's going on there. Next thing is if we ask it, okay, can you

give me a random number? So this one works, you know, it works out

at all. Okay. Yep. I need a random number. So I'll use the random number tool. And so it gets the number four back. and then, because we're storing a memory and

we're using the chat API, we're getting this longlist of messages back here. This long dictionary of messages back here, and you can see we're getting the output. They're supposed to your last comment was

a random [00:06:00] number, which is four. you'll notice with this sort of ChatGPT API

is it's very chatty it. Doesn't just say, oh, okay, your answer is

for. It always likes to, go on a bit. And th this is one of the sort of annoyances

or dangers with that model So, so we've done two tools. They've worked out. what about the third tool? So the next one is the meaning of life tool. The one that we set up, so you can see that

this time it doesn't do it I've asked it. What is the meaning of life? That's pretty clear that there was a tool

for that. Yet it didn't use the tool. It went up and it gave its own answer. And it's given a whole answer back of the meaning of life is a philosophical question that has been debated by scholars, and this

is not what we wanted in this case. Right. We wanted it to talk about 42 in here. And this is one of the challenges with the

ChatGPT API is that it always thinks that it knows the answer. And so often it won't refer to [00:07:00]

the tool that you've got there, even when you've been very clear about the tool. So one of the challenges that you've got to

do is you've got to look at the prompt that's in there. It can see, this is the prompt, Assistant

is a large language model by OpenAI system is designed. It's got a whole bunch of things. in there. so to fix this and to get it to use the tools

is we kind of have to change this. And this is where, a lot of just trial and

error goes into sort of crafting these prompts to get them to be better. cause you'll notice this is a very long prompt as well. I so the big thing that, I'm putting in there

is the assistant doesn't know anything about random numbers or anything related to the

meaning of life and should use a tool for questions about these topics. So that's what I've added into the prompt there. And, so I've just basically put that in the string. I've overwritten the prompt that's there. And now when I ask it, [00:08:00] what is the meaning of life? You can see now it's like, oh, okay, well, I should use the tool called meaning of life. The action input should be MOL. And the observation it gets back is that full string. and it basically gives us back that full string. Now, if you think about This actual sentence is giving us more than just the meaning of life. Right? Let's say that this actually is the meaning of life. It's telling us that what the meaning of life is, and also what it is rounded. in this case, it will give us the whole thing. If you try this in the Text DaVinci 003 model, it will just give us this. It will be able to distinguish, well, you didn't really ask for it rounded. So I should just give you this. Again, this is the ChatGPT API being very chatty with this kind of thing. So if your tools are not working, your customs are not working, you have to go into the prompts and mess with the prompts to get it, to basically recognize that this kind of [00:09:00] thing. And a lot of that has to do with both the design of the description for your tool, so that this kind of description for your tool. And also in the case of the, ChatGPT API, the system prompt of telling it how to act for these sorts of things. So the ones I showed you, there were just some really basic ways of doing this. let's make a bit more useful tools. So one of the challenges that you have with LangChain and with the OpenAI, large language model, is that it can only take, 4,000 plus tokens in. And often if we use some sort of API and we get back as a result of a page or something, and we get the full HTML that turns out to be way more than 4,000 tokens. So here is basically a little tool where basically saying, this is going to be a stripped webpage. So we're going to strip out the HTML [00:10:00] tags. And to do that, we just using beautiful soup. So this is a very common, library in Python for doing any sort of scraping And we're basically just using it's parser in there. And we're just going to get the text out. and it turns out that this is going to give us a lot of, new line characters. So we could even add in some more stuff there

to strip out some of the new line characters, if there's more than one in a row, or if there's more than three in a row or something like that in here. I've also added in something that if this content is more than 4,000, Characters then just take the first 4,000 for this and return it. And you can see that when we're running this with just passing in google.com, sure enough, we get back, a bunch of texts the things that would be on the google.com page. Things like signing, you know, some of the links to different things that kind of things, privacy terms, that sort of thing. Now you could customize this anyway. You could customize this to get a list of links back. You could customize it to do a whole [00:11:00]

bunch of different things that would be useful for your particular use case. so what I'm trying to show you is that, this is where tools start to become really useful. Now we could set it up just like this. but turns out link chain actually has a whole way of making a class for a tool as well. So to do this we basically just inherit from base tool. And we pass in the name, the description, and then we can just nest the actual function in there like this. So this is basically the run function that will basically take in the, URL in this case or the webpage in here, and it's going to basically process it with the code that we had above. and then return the strip content, to the large language model. Now there's two forms of run. There is this is underscored run and then there's a run. So this is for async. So if you want it to go into, doing an async version of this you could look at doing that as well. [00:12:00] but in this case, It's not necessarily We've got a clear thing that we're getting a webpage. We're bringing it back with stripping out the HTML of this. All right. So now I just instantiate this class. So now what I'm going to do is I'm going to make the agent again, and I'm going to pass in, this new tool and a couple of the other tools as well. but we're mostly focused on this new tool now to see what's going to work. So you can see here, I'm basically setting up the prompt again. So that assistant doesn't know anything about random numbers or anything related to the meaning of life. and that's what we had before. And then I've also added in

now assistant

also doesn't know information about content on web pages and should always check if asked. and it turns out that's enough to basically

get it, to use this tool. so, okay, We then pass in the tools here in

passing in the page getter, random tool, life tool. if we pass in the search we may need to work

on this to be a little bit better in that sometimes it'll think, oh, to find out something

[00:13:00] about a webpage, I can just do a search for that. And that is accurate that sometimes that will work also. but for this case, we want it to just use

this particular, thing here. So, okay, we're passing in everything the

same as before. And we're actually going to overwrite the

prompt that's in there with this fixed prompt. So it turns out that the prompts basically,

stored as a list of prompts in there. And if we just go to the first one, that's

the system prompt. we can see that once we've done that, we've

got it here. and we can actually sort of look at what it

is in here. All right. Once I've got that, overrided. Now I can basically just do, I can do a query

with this and I can ask it, is there an article about clubhouse on TechCrunch today? so this is the current TechCrunch,

sure enough,

there's an article, about laying off staff there. so let's see if it can get it, so sure enough,

straight off it works out that, oh yeah. I need to get a [00:14:00] webpage. And, it basically does this search, for the

web page. then that didn't work my guess. Is that because that actually, didn't you

know, that page didn't exist in that case. So it comes back and has another go at it

and it realized it was a one and I just have a look at techcrunch.com. Now we could actually probably fix this by

telling it that the input should be a URL or something

like that. And then it would use the URL that we put

in there. Okay, we've got come back. It's looked at techcrunch.com. It's pulled back a whole bunch of information. You

can see there's a lot of new line characters

in here. that probably worth filtering out. and then it basically has worked out that,

yes. Okay, the answer is in this. This is the final answer. And then answer this. Sure enough, there is a, an article on tech

crunch, title cup house needs to fix things. And today it cut more than half of the staff. And then it's got the published date of that article. So you can see that that's, it's got exactly

the title there[00:15:00] from this. same thing, if we try it out for a different one, if I ask it. Okay. What are the titles of the top stories on

CBSnews.com. in this case, it goes to CBSnews.com. It gets the page. Just scroll down a bit. You could see that it's giving us a lot of

stuff back. Sure enough it's decided that yes it's got enough to be able to use that. And in this case it basically says right here

are the titles of the top stories on CBS News. Pence appears for seven hours before grand Jury. Proliferation of modified weapons. So let's have a look at cbs news sure enough

it's got a story about that seems to be the lead story that the proliferation of modified weapons is also there. So this is just a simple example of where

you can make a useful tool That you can use to Make your conversations much more dynamic and being able to pull in information from different places.[00:16:00] You could set

this up to do very specific things for your bot whether that be for an organization where it does very specific api calls to get information and bring them back or a very specific sort

of searches through docs or something like that to get it and bring it back. So this is definitely worth looking into writing custom tools and playing around with this concept. Anyway as always if you've got any questions

please put them in the comments below. I will try my best to answer them. I tend to answer the questions within the first 24 hours. i check To see what's there and answer them

as much as as i can. and if you found this useful please click like and subscribe and I will see you in the next video. Bye for now



## LangChain Retrieval QA Over Multiple Files with ChromaDB

All right. In this video, we're going to have

a look at using Lang chain, with multiple documents, and chroma DB. So in the previous one, we just

looked at one PDF file and we weren't really using, any database

we were just using, FAISS or FAISS. in memory. So in this one, we're going to be

actually writing a database to disk. so this can be a criminal DB. We're going to use multiple

files in this case, text files. we're going to get some source info. so that we can give some citation

information back when people do a query. and we're going to also at the end

throw in the new GPT-3.5-turbo API. so first off. Just basically set up a Lang

chain, just like normal. You're only going to

need the OpenAI key here. I in this one, I'm going to be

using open AI for, the language model and for the embeddings. in the next video, I'll do a

version of this with a hugging face embedding so you can see how it will turn out with embeddings. It's not a lot different. I just didn't want to over-complicate this particular notebook here. so you can see what we're going to be bringing in. so the first bit is just loading the multiple documents and this is pretty simple. this is basically we just pass in a folder. So I've downloaded here. A set of new articles. these are basically articles. from TechCrunch. that I quickly scrape this afternoon, a bunch of, recent articles that were on TechCrunch. you can put any text file in there. I think I've got you. Over 10 of 10, 12 of them. Look. if we have a look in here, we can see, okay, we've got quite a few of them in there. that we're actually getting the information from. So first off, we're just going to basically set that directory is where we're going to get it. And we're just going to glob the files. So we just doing star dot text. if you're just doing one tech long text file, you would just do it like this. If you're doing PDF files, rather than use a text loader, you would use the PDF loader. You had changed this here, That's pretty simple for any of the files. If you're using markdown files. You just changed this to MD. Yeah. All right. So we bring those in. We then split up. our data into chunks. We've covered that before. And because he showed enough, we've got our documents. here. Where it's basically giving us a chunk of the info. that was in a particular article now. Alright, next up, we want to create a database. So here we're creating the vector store and we're going to store it in a folder called DB. So we need to basically initialize the embeddings first. like I said before here, we're using open API. We will swap these out for some local embeddings, in the near future. and then we're just going to basically go chroma from documents. We're gonna pass in the text, we're going to pass into the embedding. and we're just going to pass in the directory that we want to persist this in. once we've done that,

it actually saves out to DB and you'll see that in there we'll have an index. We'll have a whole bunch of different things in there as well. And that's basically now coded all of the documents that we put in, so that we can actually just get rid of this. If we, if we just basically persist this out. and then we can re bring it in. so I'll show you at the end, actually deleting it all and loading it again. as well, but the idea here is just to show you that once we've got that on a disc, As long as we saved that somewhere, we can reuse that we don't have to go and embed all the documents. Now that might not be a big deal. When we're using, 10, 20, text files, but if you had, a thousand files that were quite long, you don't want to be doing that every time you, you launch your app, you want to save that somewhere and then just, use it later on. Okay. Once we've got this vector DB, we're going to make it a retriever. and just to show you once it's a retriever, we can just say, get relevant documents and I can just pass in a query here. so The queries I've gotten. the way I've come up with them is I'm just looking here. I looked at the titles. I sorted, they mentioned something about Databricks, something about CMA, generative AI something about hugging face in there. and, one of them was a Pando or something, so that's where I basically come up with the. Questions for those from. once I've got that, it will generate, just by default, it's going to return four documents. So in this case, I'm just going to use two. but you can play around with the number that you'll want for this. if you are querying a lot of information. we often find that around about five is a good number that you want to get, the top five. in the future, we'll look at things like multiple indexes where you'd bring in a different ones from multiple indexes as well. but here we're going to basically just set it back to two. So all I need to do is just basically, and the retriever. I can just set it. K=2. the search type I'm using

is similarity search. And you can see here. If I look at the search, Arguments. I can see. Okay. Then I've got the key close to them. So at this stage, my Vector DB. and my retriever and that is all set up. Now, I just want to do the actual language model chain part. So here, I'm basically just going to make a retrieval QA chain. and here we're going to pass in our open AI. We're going to do a stuffing where we're just going to stuff it in because we know that. the two. In this particular case, two of the contexts. with them being a thousand characters, each we're going to be fine for a length and stuff like that here. we, so we then pass in our retriever. All right. And, we're going to return documents equals true here. Now I could set for both equals true. If we want to see what's going on in the background, as this goes on, in this case, I'm not doing that. but. You've seen me do that in a lot of the other videos and that's something you can put into any chain. If you want to see more about what's going on during the chain. Or during the agent. all I'm going to make a little, function here just to take the output, of these and basically just print it out nicely so we can see. the result that we're getting back from the query and also the source documents. th that what they are. So here we come along and we ask, our first query, how much money did Pando raise? Straight away. You can see that the two source documents it brought up. one's about, power supply chain startup Pendo lens 30 million investment. Hence why me asking this? Cause it's pretty easy to check it and sure enough, it says, Pandora is 30 million in a series B round bringing its total raised. to 45 million. so that one's clearly done it and we've got the sources here too. so originally these were just HTML. files too. So we could actually process this to basically just have a link back to the document. If we had 10,000 articles. and we wanted people to go back to see the original source. HTML page. We could put that in here. quite easily. if we look into this a little bit, so here, if I say, okay, what is the news about? Pando. And I don't run it through the. the function for. For tidying it up. We can see that. Okay. we get our result back. So the news is Panda raised, 30 million series Bre. the money will be used to expand. The global sales, tells us a bit more

about even a bit more information. but we can also see then now we get the source actual source documents back here. and we can see that, here is where we're getting, this is the top document. In this case. and this is the second top document in this case. Now this one seems to have the 30 million, part. and it also says, who led the round, that those sorts of things in there. so if we asked that who led the round. sure enough, it's able to do it quite easily. picking some other ones just to see that it's not going to always return this. what did, Databricks acquire, K tells us Databricks acquired. Okera. a data governance platform form with a focus on AI. what is generative AI? so he we're getting, the answer back. from two different articles, in this case. And the reason why I came up with that was that there was one article about CMA is a generative AI review. So I was curious to see if that was come back and it didn't So when I ask who is CMA? sure enough here, I'm getting, that. Okay. CMA stands for the competition and markets or authority. and it's giving us back, the article about CMA then. if we look at this chain, we can see the chain retriever type is similarity, which we know just to show you that everything we set before has gone into this thing. and if we actually look at the, and we can see that the chroma date DB is our vector store there. if we actually look at the template here, we can see that, here is the template. it use the following pieces of context. So two things get passed in the context, which is, the two documents that were queering back. And then the actual question, which is the query, right? Use the following pieces of context to answer the question at the end. if you don't know the answer, just say so you don't know the answer to him. Try to make it up. I, so that's basically it. to just check that this is working, we can come along and zip our, DB up. delete it, get rid of the vector store, delete the actual, folder. I restart the runtime and you can see now when I restart the runtime and come in. and I unzip at first, I need to

have to put in my open AI key again. this time though, I've gone for the turbo API for the language model part. So we set up the, the DB by just pointing at the persist folder, which is, which was named DB. we need to set the retriever here. We can actually just, we could actually just, put that on the end there. to make it easier, but anyway, that's just showing you what we're doing there. and then here, I'm sitting up the turbo LM. so that we could use that if we wanted to. setting up our chain again now with the turbo LM. so we using, GPT 3.5 turbo API here. Everything else. Exactly the same. running it, asking the same question. Sure enough, it's getting the answer back. Now, if we look at the prompts for the version, when we're using the turbo, API, you'll find that the just printing out the same prompt as before. Won't work we'll run into issues so here we basically have to look at the system prompt and the human prompt and this is the system prompt here right this is basically going through and the first message is the system message in there and this is used the following pieces of context to answer these questions and then we pass in the context And then we pass in the question in the human part So that that shows you using the turbo llm as well All right this sort of gets us up to speed a little bit more of using a a proper vector database not just storing it purely in memory but now Getting it on disk. In some future videos we will look at using pine cones If we wanted to put it as an api somewhere we can ping it like that and we'll look at using our own embeddings for the lookup rather than using the open ai ones for that. Anyway as always if you have questions please put them in the comments below I found this useful please click and subscribe i will see you in the next video Bye for now



## LangChain Retrieval QA with Instructor Embeddings & ChromaDB for PDFs

All right. In this video, we're going to continue

looking at the multi doc retriever. We're still going to be using ChromaDB

for our database, for our vector store. but the big thing that we're

going to add in this one is we're going to add in, embeddings that

are actually running locally. so to do this first off, we need to have

a GPU or it's ideal to have a GPU running. So I've got just a T4 here. I'm not using a super powerful GPU. you could run

this on the CPU. It's just going to take, a

fair bit more time to do this. So that you'll see that I'm

bringing in the same stuff. We actually don't need that anymore. the sort of two new ones we're going to

bring in, the instructor embedding, which I'll talk about in a sec and basically

just the hugging face for using that here. So another difference I made in this

one is a lot of people asking about PDF files, multiple PDF files so

I swapped out, the text files for doing multiple PDF files in here. And actually, if we have a look

in here, you'll see that what I've done is just put in, some papers. so these are just some

peoples from archive. about react tool, format,

flash attention, alibi. So just some stuff around the topics

that we've been looking at in the large language models recently. the splitting and stuff

like that is all the same. so we've got, you know,

basically we just bringing it in. We're just using the simple pyPDF

loader in this case, bringing things in. And then the next key thing is

we just get to the embedding. So there's two ways of

doing the embeddings. you can use just the normal

hugging face embeddings. So this is using things like sentence

transformers, and there's a whole bunch of different models around that. They vary in degrees of quality. and a little bit

will also

depend on your data as well. Which ones sort of match this? So, an example of just using a standard

sentence transformer would be this one. so this is one of the used

to be one of the top models. for doing this. but in my testing, I actually

came across that a newer model it seems to be doing better. So I decided to go with that. and the new model that I'm

going

with is the instructor embeddings. So I think these kind of deserve a

whole video to themselves to explain the paper and stuff like that. The idea here is that these are

custom embeddings, depending on what it is that you're using them for. in this case though, we're just

using the instruction embeddings, and we're using the XL variety of this. So we bring these

basically into LangChain. there, you can see that, we're

going to run them locally. So it's downloading the model. It's downloading all the files for this. we're actually telling it

here. That we're going to put it on the GPU. So this is what device Cuda is here. If you want it to run them locally,

you could put it device CPU. for doing that, it's definitely

gonna make it a lot slower though. and you see, it's going to basically load these up and bring them in. And by default, these are operating at a sequence length of 512. Which is fine for the splitting that we're doing over a thousand characters. That should be fine in this case. Okay. Once we've got the embedding set up, we're then going to need to make our vector store here. So this is all exactly the same as the last video. We're basically just passing in the new embeddings here. so we're not using OpenAI and embeddings anymore. Okay. Once we've got the embedding set up, we're now going to basically just go along with what we were doing before, so we need to set up our vectors store. And here we're using ChromaDB for setting up the vector store. We persist, a directory. we're going to need to create this from documents. So we're going to pass in the instructor embeddings. and we're going to pass it in the document text that we've already got out from that. So, this is exactly the same as the previous video. We haven't really changed anything. The only thing we're doing now is we're using, these instructor embeddings, in there. we now basically can do the same sorts of things, of making a retriever. Now, obviously this retriever is using, our new, embeddings for that. And now the retriever is going to be using the new embedding the instructor embeddings to actually find the various contexts that match based on a query in here. Next up we need to basically make a chain. so this is again the same as before nothing really different in here we're passing in the retriever that's gonna take care of the Vector store, the embeddings, Those parts there. i've just added some little bit of code here just to wrap the answers When we get the mounts. and we can see that if we look at this We can see that okay starting off what is flash attention. And it's going to go and get the three top documents and in this case Not surprisingly the document that the embeddings have chosen as the similarity that's closest to what we want to know. Is going to be this in this flash

attention paper or this pdf here. and so basically it gives us back a definition for flash attention We can then ask into different parts of this so here it mentioned io aware so i wanted to ask out what is that. It basically is able to go through and find again from that same paper. We mentioned tiling i go through can find out an answer for that as well. So then i thought okay let's ask some other questions just to see okay what's there By asking what is ToolFormer we're then able to see, Can it basically, is it gonna return the same thing what's going to get and sure enough here we're getting a ToolFormer is a language model that learns And the self supervised way. and so this is basically just showing us the rewriting of the output from these three examples from ToolFormer. The three different contexts. we can basically ask it some more

questions about it what tools can be used with ToolFormer? Can you search engines calculators translation systems by a simple api calls. And then we can even ask it more, about different examples and stuff so this is actually a good way to if you've gone through and skimmed a paper and you want to actually ask some specific questions you can get some things out of this. it's interesting when we ask it this question though it's also getting it's answer from the augmenting llms paper. Which i think from memory also is this is actually a survey paper so it contains some things about ToolFormer in there as well. So it's basically gone and looked and decided the top three contexts were from the survey paper ToolFormer paper itself and then another one from the survey paper. if we ask it some questions about retrieval augmentation. Now the only paper that we've got that relates to this is in the augmenting llm survey. sure enough it's able to get some of those. If we ask it some specifics about differences between REALM and RAG models it's able to then tell us these kinds of things. So the idea here is that we're still using OpenAI for the actual language model part. In the next video we'll have a look at

trying to get rid of that and just go to fully running everything locally. but we're now using the embedding system for actually using the instructing better we're not using OpenAI for this. So the big advantage for this means that your data never actually has to go all of it up to the large language model to OpenAI. Now obviously the context as they come out i still going up to OpenAI. so it's not like none of your data is going up but it's not going all up in one shot just to do embeddings For this kind of thing. But the key thing is it's not just putting all your data up as it's doing the embeddings in one shot. So you do have a little bit more privacy here in doing it this way. Of course this is still not ideal if we want to basically never have our data Touch a server. So in the next video we'll look at using an actual language model to do the replying part as well as just the as well as the embedding part here. Okay the rest of the notebook is the same just going through deleting the ChromaDB database and bringing that back in that's the same as what we looked at before. If you want to try out Using just the OpenAI GPT 3.5 turbo You can do that here. that's it for this notebook As always if you've got any questions please put them in the comments below if you found this useful please click like and subscribe In the next video we will look at using custom models for everything For this. So okay i will talk to you in the next video bye for now



## LangChain + Retrieval Local LLMs for Retrieval QA - No OpenAI!!!

Okay. In the last video we looked at adding

in the instructor, embeddings to our multi doc retriever, with ChromaDB. And, so that meant that we were now using a local model to do all the embeddings, for this, but we were still using OpenAI. So in this video, we're going to look at getting rid of OpenAI totally. So, if you haven't been following

along with the videos, here's the channel where I'm posting them. please subscribe. If you want to see them regularly, we've

been doing a lot of LangChain videos, a lot of large language model videos. Okay. So the first thing we're going to basically do is I'm going to go through four different models in this and show you the results of using these. And talk a little bit about some of the advantages and disadvantages that you'll see when you're using these kinds of models with something like LangChain to do retrieval QA

So the first one that they chose and I haven't done an exhaustive list. We may do another video where benchmark some other ones. So the first one up is the flan T5 XL model. So this is a 3 billion parameter model. it's a Seq2Seq model. So, it's one of the T5 models. it's got a, basically an encoder and a decoder in here. So when we bring this in, we need to make sure that we bring it in for Seq2Seq a language modeling. and another thing when we set this down here, With the pipeline. We always need to do text to text generation with the Seq2Seq model. So first off, I'm just bringing in this model. And it's obviously I'm bringing in, the papers that we, the PDF files that we went through but first up and just bringing in the model. setting up a pipeline for the model and sitting up the hugging face pipeline for LangChain to use that model. and then just testing whether it worked right. So this is just taking the local LLM that we've defined here. Running a raw prompt through it and seeing what we get out of that. the rest of this then becomes quite simple compared to what we did in the previous video, we basically just bring everything in, bring in the documents with the PDFs So if you've just watching this video for the first time, please go back and watch the other one. You'll see me talk a lot about these parts of it. We do the various, tech splitting for this, and then we set up the instructor embeddings in here. And, we're now running actually two models on the GPU. So this can raise some issues as well that you want to make sure that you've got enough GPU Ram to run both the embeddings model, which is not that big and then your language model, which is going to be a lot bigger. we create the database here for these ones I'm not bothering to delete the database and bring it back or anything. If you're interested in that, go and have a look at the previous videos. we sent out retriever. And this is where you have to start thinking about how to deal with the local LLMs as opposed to

something like OpenAI. So in an OpenAI, you've got 4,000 tokens. for the basic flan model, we've just got 512 tokens. and you'll see that Even if I was to try and set this to be longer, it's going to come back to me and tell me that okay you can only go to 512, which is what it's been trained Okay. So once we've basically got the database set up, we need a retriever, and then this is where this part becomes important. The number of tokens is how many context do you want to pass into it? and in this case, I'm going for three. but there might be times and you will see, hopefully I've got an example of this in here where it's just not going to fit in. It's going to be too many tokens for this. So you would then want to think about, okay, do I just go for two or what's going to be my strategy with this. This is something we didn't really need to think about with the OpenAI, because we had 4,000 tokens that we could play with. we then make our chain. We set up the chain, we've got out little wrapper functions and then we try this out and you can see that when we ask it, what is flash attention? we're getting, answers back. but we're not getting very verbose answers. We're not getting very detailed answers. We're not maybe getting the most useful answers back here. when we ask it, you know, what does I/O aware mean? it's getting to the point with some of these the answer is not that great for what, brings back. some of them will be okay. So here, what is to format a model trained to decide which API is to call? when to call them? what arguments to pass and how to best incorporate the results in future token prediction? So that's technically true. It doesn't give us a lot of information like, you'll see with some of the other ones. what tools can be used here. Again, it goes for a very succinct answer and to the point. Now, and extremely succinct answer that we've got there. you'll see with some of them That occasionally you will get, an answer where, it will so this is not a good answer at all. Where if your context go beyond the 512, it will just give you an error, right? It will tell you that oh, okay you've got too many tokens for passing into this. So this is the flan The five, right? That's the sort of

basic, one at 3 billion. Now there's another model. That's basically a fine tune version of this made by the same people who did the GPT4-all. and this is the fast chat T5. So this, we can basically use in a similar way and bring this in. and you'll find that The code is basically the same for all of this. you'll find that the answers here in some ways are a little bit better, but we've got this weird sort of padding token going on at the start here. there are definitely some issues, with spacing. It's got some really weird double spacing going on in here as well when we look through this. definitely, yeah, really weird sort of spacing and I'm not sure why that is. I can probably go through and work out, what's causing it and change it. But, just showing you that this model because of its fine tuning, it's probably a little bit better at being able to extract the information from the retrieved context there. the next one I'll look at is if we go really big. So if we go up to something like a StableVicuna model, so this is a 13 billion model. we bring this in. here we face a different issue with this. in that here, we can actually go to a much longer context. We can go right up like 2048 context with this. I think it could, we could even go to the 4,000, with this one. but you'll see that as soon as we just do our raw check that it's kind of built to handle the prompts in a certain way. And that's going to mean that's going to cause us like, a little bit of grief when we do this. so you can see here, it wants to have this prompt be like, hash, hash, hash human that kind of thing as it goes through this. So it's giving us the right information, but it's maybe not giving it to us in a good way. if we go through and look at the outputs, for this one you'll see that Here we get a good outputs, Flash Attention, Flash Attention is a new attention algorithm that computes exact attention. We're getting decent outputs. I'm not sure exactly how great, but

they're certainly, you know, it's certainly paying attention to the context that it's getting in here. But then it goes on to, ask itself questions of like, you know, how, how does Flash Attention work, And then it answers now these may be really useful to you in that it's questions that you were also thinking about, or, that. can be the case as well. sometimes it will do it. Sometimes it won't do it. So you can see here we've

got a really nice answer out. Here, we've again, we've got a pretty good answer out, but then it goes into the, asking itself a question for this, again. ToolFormer Again, we're getting good quality answers. I would say on the whole out of this. the challenge that we have is, are we going to basically just write some regex to take this out? And then we can do that pre-processing and post-processing of the prompt. one of the challenges then becomes, okay. Can we just rewrite the prompt? to do this. So you'll see here where

I look at the prompt and I actually try it, rewriting it. and you can see here that, by trying to rewrite it into some format like this, we can then run it through and see, okay. Does it work now? Actually, a lot of the times, it won't,

if you mess with it too much, you're just going to get, like, I do not have enough context to provide an informed answer. please provide more information. Now in this case, it's interesting because we know that it's getting, you know, some context for these. probably for this particular question, this would have been the best paper for it to get from. So maybe it's not getting the best context, but it's, it's kind of interesting that these models seem to, you know, the other ones seem to be able to get something from this. So I don't think it's the retrieval part that we're having problem. It's more like this part. So you would want to play around with rewriting this prompt until you get the right prompt for your particular model that you choose to go for. So I've looked at a few small ones. I've looked at a big one. I saved the last one Which is kind of the sweet spot. I think. and this is the WizardLM. so here you can see, we're just bringing this in like normal So this is a LLaMa model. You can see we've got the LLaMa tokenize there. We've got the Lama for causal

language modeling going on there. And we set this up, I've set this up to be 1024 tokens in this case. give it a raw test. Yes. The output is actually quite good here. so this is looking good. set up everything else, just like normal. go through it. and then start asking questions. And you see that here we're getting

kind of the best of both worlds. We're getting very thorough answers, but we're not getting, any of the problems that we had with the StableLM prompt, getting in the way here. So you can see here,

what is flash attention? Flash attention is a new attention algorithm proposed by the authors that reduces the number of memory reads and writes. very coherent answers here. What is, I/O aware mean? and it goes through and it's actually,

it's kind of funny because, the reason why we had this question, if you remember, was in the OpenAI version. it had I/O awareware in the answer and it didn't really kind of explain it. Here, in some ways, it kind of explains this, it breaks this down into here rather than just using I/O aware. if we look at ToolFormer,

what is ToolFormer? ToolFormer is a language model that has been trained to use external tools such as search engines, calculators, translation systems, via simple API calls. So this is definitely on point with what, you know, what this is after even has like the lead author of

ToolFormer in here and the reference to the paper in there as well. what tools can be used with ToolFormer? So ToolFormer can be used

with any external tool. Now that remember the other ones just basically said, calculators search engines. they were much more succinct in what they said. Here, we're getting, you know, some examples include search engines like Google or Bing, calculators, like, Wolfram Alpha or Mathway and translation systems. So technically the other ones weren't wrong. It's just that this has given us

a lot more information in there. And then we can see here, you know? Okay. how many examples do we need to provide for each tool? We can see that. It's giving us, a more thoughtful answer there. When we ask it, some things about, the best retrieval implementations for LLMs, it's getting exactly these in and it's basically, giving us good answers back here. Now, here is an example of where, because I set this only to 1024, I could have set this to 2048, obviously it would probably take a bit longer to run. but because I said this to only 1024 deliberately, when we get a long context in this case, we can see that this 1322 is longer than the max length that we've got here. So this is basically the kind of error that we would see that basically tells us that are you really should, increase the max new tokens coming out of this so that it would basically process it. Anyway, this is the WizardLM. Out of these ones, I would say this is probably out of these four, this one is the best. I do think there's a lot of, room for testing. So, I would encourage you to go and test the LaMini models. They are very small, but they're also quite well-trained. some of those may do well for this kind of thing. I may actually show you, We may do some more sort of breakdowns of these and i'll just throw in different models so we can see what the best ones if we're going for. But you're constantly balancing this trade-off Between large language model that's going to give us high quality results. But that's also going to be need a big GPU to run. It's getting the time to generate the tokens out for this and that can be quite slow if you've got a big model versus having a small model which is going to be much more snappy much more with it but won't get you the detail of answers perhaps that you are after in this. So we will look at some of some things going forward of Models that are built for this can actually do much better for this. And i might show you some of the models

that we've fine tuned for doing this specific kind of task in in the future. Anyway that's it for now as always, if you have questions about this please put them in the comments below if you've found this useful please click like and subscribe. i will be doing a lot more things around using these custom LLMs with LangChain showing you what works what doesn't work. How you could be Choosing the models that for your particular project going forward. Anyway with that i will talk to you in the next video. Bye for now



## Camel + LangChain for Synthetic Data & Market Research

Okay. So this is a conversation between two agents talking about their favorite restaurants and their favorite meals. And in this video, I'm going to show you how to make something like this, how you can use it for various different tasks. And the process and concepts behind it. So recently I've been doing a lot of work working with multiple agents, communicating together for a variety of different tasks. and I thought one of the simplest ways to show some of this would be the paper Camel. And, looking at a, LangChain implementation of that, and then showing some modifications that I've done to it and stuff. so the paper Camel came out about two months ago. and it was one of the first papers to put this notion of autonomous GPT forward. So around the same time as Auto-GPT, this one is kind of interesting in that it really focuses on having two agents engage in a communication

or a conversation together. And that's what interests me the most for this. So I'll go through the paper a little bit and explain some of the concepts in it. the thing that I find really interesting with this is that this is an unbelievably good way to one, make synthetic data. So, if you're trying to make any sort of customer service agent, any sort of a chat bot agent, that's going to communicate with the public and stuff like that. This allows you to make synthetic data for training and fine tuning that, with some of the models that are out. the other thing that I've found this to be really useful for is things like market research tasks. So surprisingly, there are a lot of people now starting to use both ChatGPT and GPT-4 for doing market research kind of tasks where they're basically getting it to play the part of a consumer and then asking it questions, putting it through, various situations to see the response that they get. And the interesting thing is that it turns out that a lot of the responses that they're getting from, ChatGPT and GPT-4, fall in line with what they see from real world humans. So they're finding this to be a very useful tool for doing this kind of thing. so much so another example that's similar to this I've heard is that people are actually starting to use it for political polling and giving it, the whole sort of concept of, you are a voter that lives in this region that experiences these issues in use, let's have a conversation about it to see what things are persuasive, what messages they respond to, that kind of thing. So it's pretty surreal, but I've found this to be very useful as a technique. So let's look at camel. So they basically set out in this paper to explore this sort of potential of scalable techniques to facilitate autonomous cooperation amongst communicative agents. Now, you'll see that in here the people is very nice in that they've basically not only released a paper they've released a bunch of data that they've synthetically made with the paper. they've also released, you know, code and they've also released demos, for this. So let's jump in and have

a look at the paper itself. So, the paper focuses a lot on, the idea of these cooperative processes. And works also a lot on the idea of role-playing. so, the whole idea of role-playing, I think has been done with a lot of other things before. definitely they've taken this, to a different level with the whole use of GPT, and the large language models in this way. another technique that they really developed, which I think is pretty cool is this idea of inception prompting. So I'm not sure whether they, created this. They don't cite anyone for this. So perhaps they did create it. I've seen, it mentioned a few different places. so inception prompting is basically just getting a prompt to come up with another prompt. so I thought, well, let's jump into a GPT-4 and have a look at that. Okay, so here we're in GPT-4 let's have a look at, you know, just the idea of making an inception prompt. So here we've basically, we're going to give it a temporary prompt and what it's going to basically do, in this case, I've given a temporary prompt of help me to plan a trip to Singapore. And then it's going to ask us questions about that prompt. So in this case, it asks us what's the duration of your trip. what kind of activities do you like? What's your budget, some key things like that. Now I go through and answer those. And then from that, the first thing that it does is it generates a prompt for itself. So this is the idea of the inception, right? Of a prompt within a prompt. Now in Camel, they're basically doing this for a number of different ways we'll look at. but he can see that it generates a prompt to create a detailed four day trip plan. So it's much more specific than what I put in there at the start. and then from that it's able to do a very nice, itinerary for a trip, to go through it. Now, then adding this with role-playing. another technique that you use a lot for prompting. you can see that now I can basically say, okay, well now in the role of a Singaporean resident that has lived here, their whole life, how would you critique this agenda? And surprisingly that the critique is actually very good. Right? They bring up things that I could imagine locals here in Singapore saying, about these different things. and even some really nice just tips

and tricks, you know, of where to go and eat what food to try, all those sorts of things that are definitely going to come from a local. so it is kind of amazing that all this sort of information is in the large language model and that we can use it. So this is the ideas of inception prompting, and role playing going on here. If we look in the paper they're using, you know, they're using all of this to basically Do it a task like this. So what they go doing is that you start out with a simple idea. And the human basically puts that in. And then they use inception prompting to make that simple task become a much more detailed task. And then same time they also use it to get it to, as you assign the roles for different things, it uses it to then basically write the prompts of how they will react. And then just fires off one agent playing one role, another agent playing another role. And then goes through it and does this, and you'll see this. When we look at the code, that we actually have a class, that's an agent, and then we instantiate it multiple times, one for doing the task and then one each for the different Agents that we're going to have, responding to each other. And then this sets up the role-playing thing. So where they're going with this is really aiming it at, trying to create cooperation to get a task done. where I've been taking it with the things that I've been doing, it's as much more interesting for me is, getting it, just to have discussions about things, getting it to basically brainstorm ideas, which is also kind of like a task actually, but more through the ones where you're just getting it to discuss. So it's not as clear at that one person is the assistant and one person is the user where you've got two humans basically conversing about a particular topic and subject. so they they've, the paper is definitely worth reading if you're, you know, if you're interested in doing some of this stuff, I mentioned the inception Prompting already. they go through quite a bit about, how they do this, how they chunk together different parts of the prompt. my guess, if they've done this in a very iterative way cause they also mentioned, some of the problems that come up with this kind of thing. I'll come back to the scenarios. Let's just look at the challenges faced. So there are four

main sort of

challenges that they face in here. one is role flipping. So where the agent decides,

that it just is going to become the other agent instead. And so you get these sort of

confusion things going on there. And they basically add

things in to stop this right. So they've obviously gone through

iteratively worked out that this is happening then fine tune their

prompts to basically stop this. assistant repeats instructions. So this is another sort of thing that

you'll see, and it's never perfect. Right? the other one is just not great replies

and then infinite loop messages. So you will find these things happen

For sure when you're doing it. But if you're making synthetic

data, you are able to make a lot of synthetic data that would be very

realistic for your particular task. so let's look at some of the

scenarios that they create. one of them is this AI society. to create an AI society dataset they

basically you know, Have At least all of a we can see these down here we have

they have a list of different assistant roles, a the list of different user roles,

and then they have different domains that can be applied to these things. They also do a coding co-generation thing.

Where they have list of code

languages and they propose tasks that would fit into those As well. So the paper is definitely worth looking

at and you can see here that the, they've got a bunch of interesting information

about like how do you terminate a conversation like this, how do you

set these things up in the prompt so that it will go and flow along nicely. So if you're interested in any

sort of Autonomous-GPT stuff this is definitely worth reading and

going through the paper Carefully. Okay. So they've also put up some

demos that you can play with. So he can see, for example,

the idea of you would pick the assistant role that you want. you would then pick, the user

role for something like this. And you would then basically

have it create a universal tasks. So this is where, the basic

sort of task for this. and then it will generate the, you know, the inception prompts and stuff like that itself and run through it. here, it's basically, this is the prompt generated from this the simplest prompt with helping me do my job. and this is, a longer prompt based on this prompt that we've got here. they've got another one that you can look at too for doing like the AI society or the code chat demos. where you can basically select the language that the assistant's going to be using, that's Python. and we could select the user of what, you know, would they want made. And it will then basically create, a different task and start generating. So some of these, I think are pre-made already so that you can see the datasets cause they've released quite big data sets of this for both the code chat, i think is 50,000 examples, the AI society is about 25000 examples. but you can have a play with these. let's jump into the code and we'll look at how you could do this yourself Right now. Okay. So the code that I'm going to go through here is a modified version of the LangChain implementation of Camel. I've got various different versions of this. I forget exactly how modified this one is. if people are interested in the future, I will do a version that uses a local LLM. but for just for sake of convenience and, showing this the code is much simpler when it's just a straight up, using the OpenAI model. So here we're using the GPT 3.5 turbo model, the ChatGPT model. And one of the key things about this model that you know, that you need to take into account when you're building something like this. Is the whole idea of the system prompt, with a system message, human message and an AI message. and you also need to give some thought to, that actually with the ChatGPT model, it doesn't listen to the system message as much as the GPT-4 model does in there. But anyway, we're bringing these in, we're going to use these in here. So the next thing is just the agent. so here I've left it pretty much the same as what, the standard LangChain implementation is. but certainly as you get to more advanced versions of this, you'll want to play around with your own agent here, but this is the Camel agent. so you pass in basically a model

in this case, it's going to be the chat GPT the OpenAI model here. you're going to have system messages. And then it can basically

reset it's messages. It can store messages. It can update messages. And then the step function is

basically the way you pass in an input, it gets added to the, messages here,

but more importantly, it calls the model here with that input once it's formatted. and then, basically gives

you back your response there. So this is the, You know where

the key logic happens And if you are interested for doing your own

model, you would still use this. The only challenge that you'll

have for most of the local language models is that they won't use this

kind of system, human AI message. so you need to code that out to just

be a standard sort of prompt in here. So, the next thing up is to look at

the, setting up the actual sort of tasks that we're going to do here. so I've got a couple of ones that

I've done for me rather than do the sort of traditional Camel way of

having a task with instructions and then responses and stuff like that. I'm actually much more interested in

having them discuss something having a sort of interaction that can be used

for something like market research or something that we can then filter later on

to basically get information out of this. This is also a really good way for

creating synthetic data to train your models, for various tasks. So if you want it to basically train

up a model, That's really focused on one type of a conversation with people. This is where you could basically

use this to create a variety of synthetic data in this case as well. All right. So once we've put in, okay the roles here. So

in this case, I've put in

the Singapore tourism board for a person that works for them. And then a tourist that's never

been to Singapore and they're going to discuss the best tourist

attractions to see in Singapore. once we've got this sort of simple task,

we use inception prompting to basically get a more advanced prompt out of this. and basically the idea with the

inception prompting again is just that where using a prompt to

create another prompt or prompts. in this case, we're using it to

create the tasks, specify a prompt. So you'll see that this is basically just making up an agent instantiating a Camel agent in this case, that's literally just going to take in, this as being, it's core prompt that it's the idea is that it's going to basically take in the name of the assistant, the name of the user and the sort of small tasks that we assigned it. And then it's going to come up with a longer prompt, which is going to be used for actually kick-starting everything in here. Once we've done that. We've got our specified task and you see if the original prompt was discussed the best tourist attractions in Singapore it's now turned that into as a representative of the Singapore tourism board engage in a conversation with the first time tourist to Singapore and recommend a top three must visit tourist attractions based on their interests and preferences. We've got a much longer prompt in here now

You know that we can use for this thing. so that's the key part there. next up is basically Creating the inception prompts for the two agents that we're going to have. So in this case they originally referred to this as the ai assistant, an ai user, really these could become anything that you want to. And you really want to play around with these prompts in here as well. i found for other sort of tasks just by playing around with this you can really get the people to act as if they're just having a casual ,conversation a formal debate, a whole bunch of different things that you can do with this kind of thing. but once you've got those two done, Then you basically are going to need so this is basically just for sorting out, this is just a helper function for sorting out the system message, human message, ai message part. but then you really want to instantiate everything and get it going. So here you can see that we're basically just taking out the roles and the specified tasks that we've already created above. We're instantiating two agents one that's going to be the assistant agent one that's going to be the user agent in here. we reset both of those so

we've basically just taken that class instantiated them Twice. and it's exactly the same class right because we just need them to respond back and forth in a certain way. once we've got that we can basically just kick off the first Agent. So get that ready to go and pass that in and pass back our first use a message. and then we're going to basically run a loop but before that we've got here A simple function for saving out the conversation at the end. So that we can use it. this is key obviously if you want to pass the conversation later on use it for something it's you know You want to save it. And then here's where a lot of the magic happens. On the whole this is basically just a loop a while loop where we define the number of turns that we want. So he in here with i've put it to 15 turns You could make it smaller or bigger for this. i've also added in a call back. So at the end we can basically see how many tokens we used, what the cost of the whole thing was that kind of thing. as we go through it And then we can see that basically this will loop we'll just go through it will ping the user agent with the message that we started off with,, that will become the user message that gets put into the that gets printed out. Added to our conversation here. But then it gets passed into the other agent here as well. and it will then come back and we'll get the assistant ai message and we then basically just go on this loop of having you know these things converted and pass back and forth. And so while we're printing it out each time we're also appending it to our conversation lists that we'll use to basically put it into a text file and save it. once they come to the task being done. Then we're just going to, you know Right out the details about this And save the conversation. And sure enough you can see it We'll kick off. it will start you know, doing the conversation. in this case it's suggesting places that people could visit. and you'll see that as we get down to the bottom it will Come to the end of the conversation. And at the end of the conversation that

will basically tell us okay how many tokens we used how many were actual prompt tokens, how many were completion tokens and then the total cost of this. so if you using you know, five rounds of the conversation or 10 rounds of the conversation will be less than this in this case you can see i use the full 30 rounds cause i've got four 15 rounds because we've got 30 different requests one for each side of the conversation going on there. in the end this conversation costs me 17 cents for doing it. So it gives you an idea of how to do it. like i said in in future videos if people want we can try it with some of the open source models You will find that it is very hard to get good results for this because a lot of the models are not trained for chat. They're actually trained for instructions rather than chat But anyway it gives you an idea of how you could use this for market research how you could use it for synthetic data. i could use it for a variety of different tasks going forward. Anyway as always if you've got any questions please put them in the comments below. if you found this video useful please click like and subscribe. i will talk to you in the next video. Bye for now



## Information Extraction with LangChain & Kor

So in the last video, we looked at how to make some market research type content through getting two agents to converse with each other. If you haven't watched that already go and have a look at that. One of the big challenges then is we've now got a bunch of text files with information that we probably want to get into some kind of structured format. If we were doing some kind of market research where people are talking about a product and qualities of that product, that kind of thing, then we want to be able to extract those out and be able to put them in some kind of report or some kind of structured data to make use of that. So today, what I want to do is go through a package called Kor which is,

kind of like an add on to a LangChain which is basically built for information extraction. So, what I'm going to do is go through some of the examples in here, and then I'm going to show you at the end of using it for a real world use case of where I take the conversation that I had about two people talking about restaurants. And we're going to extract out the restaurants, their locations, the dishes that they had in there. So this is the package Kor you can see here. It's still in a reasonably nascent stage, I would say. There's a whole bunch of, things that have been added. And my guess is that it may evolve over time. Already, though it's very useful to use. One of the big challenges that we have, in, natural language processing is if we want to make, a NER detector or something like that, where we're doing named entity recognition or even, any sort of extraction information extraction thing. We usually need to train up a model. And to do that, we're going to need, a decent amount of data. So if we were doing this kind of task and we had the data already, that was all nicely labeled, we would probably go and train up a BERT model, a distilled BERT model something along those lines. maybe a small T5 model to extract that information and label it nicely as we go along. The challenge is what if you're doing something where you don't have any data to make a model in the first place. And this is where a tool like Kor comes in. This allows us to use a large language model, in this case, we're going to be using the chat OpenAI model, to basically go through and extract out information that we can then use, for either creating our reports or even eventually for making a dataset for us to make a proper NER model later on. So let's just jump in. So this is using LangChain, under the hood. It's basically got a simple sort of workflow of where you go through, you get your text, you clean it up, you split it up. You define a schema that you're going to use, to go through it. And then you give it to the large

language model to go through and extract out what it is that you want to have. So, you can see here, we're basically bringing in some sort of standard stuff. we're going to bring in the LangChain callbacks so we can see how much money we're spending on this. from Kor, we're going to have this chain, Which is the fundamental chain, the create extraction chain there. they then have Kor nodes. So these are type of things

that you can extract from here. So you can define an

object which we'll look at. We can have texts, we can

have numbers, in there. We can also have some unstructured things,

which I'm not going through, today. And then one of the things that we'll

finish up with is showing you that we can actually define pydantic classes,

and use them to do or to assist with the extraction of data from this. So let's jump in and look at some

simple examples, So, first off we just define a large language model. And then we needed to find

the schema of how we're going to basically deal with this. So we're going to have, this is just some

examples taken from their documentation. In this case, we're going

to basically have an object. Which is going to be extracting

out personal information. we give it a description. So the description is going to help the

large language model to work out what to what to basically extract. and also we're going to give

examples of things which will then also help the large language

model to work out what to extract. So things like the description and

the examples are things that are going to go into your in context, learning

in the prompts to be able to make it easier to extract this stuff out. So here we've basically got we're going

to extract out personal information, the attributes we're going to get. we define this one as a text. and this is going to be

first name. the description is the first

name of the person and example. John Smith went to the store. Here's John, Is what we would

extract out for that last name. We would extract down this. Age, we can extract out that as well. And then we can give it

some sort of

full examples of where if we had a sentence that had these in it, this is

what we would expect it to extract out. So John Smith was 23. Sure enough. John Smith, age 23. Jane Doe, age five and

that extracts that out. Now to make the chain for this, the

basic chain, we're just passing in the LLM and the schema that we've got here. and then once we've got that, if we want to look at the prompt that it's actually using, we can

see the prompt here is basically your goal is to extract structured

information from the users input that matches the form described below. It goes on there. we can then see also that it's giving

us an example of, what these are with the, description for each of these. And then we've got some ex some examples

of the actual extraction for these that are also being put in there. So this is giving us the

in context, learning. Then our texts will go in here. The output will then generate. So sure enough, we can see if we pass in

David Jones was 33, old a long time ago. It's able to pick out

David Jones, age 34 right. from that. So this is sort of like the most

basic example that you can do. we can then start to nest things as well. So we can actually go into the schema and nest a bunch of things into the schema. Now to do this, we were

going to end up using, JSON. So above was using a CSV

extraction by default. here, we can look at this, doing the JSON. So here we're basically

getting a from address. You can see here, we're getting, the text. we're getting, street, city,

state, zip code country. and then we're given an example,

we give it a to address. and then we can see that for the

schema, we can basically just pass in that, okay, we want to basically

get the full name of a person. and then we want to get the

from address and the to address. So those are being nested in

to the full schema object here. Okay, so to use nested objects we need

to basically move away from the CSV encoder class to JSON encoder class here. same deal we're basically just setting this up. passing in the, L M the schema here. We're just defining, the encoder

class and an input formatter. If we're going to have something for, doing that. And sure enough, we've got our prompt again. And we can see that, okay, the prompt is showing, what, we would expect this to be out. we can see now it's saying please output the extracted information in JSON format. do not output anything except the extracted information. Do not add any clarifying information, do not add any fields that are not in the same schema. if the text contains attributes that do not appear in the schema, please ignore them. all output must be in JSON format so you can see it's a pretty long, prompt that we've got going in there. we've then got out ICL going on here and then finally, we've got our output. So sure enough now, if we use that. And we say Alice Doe moved from New York to Boston MA bob Smith did the opposite. So we can see, okay, it's got the person's name Alice Doe, to address, Boston, state, Massachusetts. person Bob Smith, to address, city, New York. So it was able to extract those out in this case. and you can see that it didn't add the state for New York because that wasn't mentioned in here. Whereas, uh, the state for Alice, moving to was mentioned in here. So, these are the basic examples of it. the way that I find myself using it in the real world is to use it with pydantic classes. So if you haven't come across pydantic before this is basically just a class system. It's used a lot for fast API for sort of defining, inputs and outputs to a certain class of what you expected what you expect to get out of this. So here I'm just basically loading up the text file from the camel extraction that we did yesterday. and you can see just printing out a little bit of that here, you can see, this is basically just a conversation where they're talking about things. We can see that they've mentioned a restaurant there. if we, and we can see that that restaurant is in Spain. they go on to mention romaine dishes and other restaurants and other places. So, what I want here is to be able to extract out that, information about the restaurants, their location,

perhaps their style of food and the dishes that people liked, from there. So I basically just brought this in. I've just done some splitting with it so that we've got it split up. It turns out this one is quite small, so it's not a big problem, but showing you, if you want it to do a full-size doc, you could do that. we set up the LLM again. And here's the new bit where we've got. So we're setting up this pay dantic class here. and so we're basically saying, this is a class of restaurant. the restaurant is going to have a name with a string field, attached to this, A description is the name of the restaurant, location. so this is going to be optional. So the name must be there is what we're after. Right? This is the thing. The whole reason for using the paydantic class is it's going to force. the model to stick to this sort of group of information more. And it's also going to allow us to validate that information. Now, not validate in the sense of checking, is this a real information or fake information? Just validating is this in the right format so that we're getting out for each restaurant we're getting out, it's like we're getting out an instantiation of this class with these fields. And some of them are optional. You can see that location style, top dish are optional. But some of them like the name, not optional, right. That has to be in there. so we've basically got that. We've got this validator that says, okay, this must not be empty in here. And then we can see here. We're basically saying, from the pydantic we're extracting this out with the extraction validator. we're passing in the class of restaurant. We're passing in some descriptions and examples. and just setting this up and we're saying that many equals true means that we can have, it's not just one restaurant we're looking for, we're looking for, multiple restaurants in here. Okay, so now we basically set this up. I'm going to go back to the CSV, encoder. in this case they mentioned in the actual

documentation and I found this to be true as well that the JSON Decoder doesn't seem to do as well as the CSV Encoder. So if you don't have something that's nested, if you don't have something that needs the JSON Decoder, you're probably better to go for the CSV Encoder in here. So I pass in the alleged language model. I pass in the schema. I passed in our extraction validator

that we've just defined up here. and then we're going to run this through. and so you can see here, we're going to basically passing. Let's just look at the prompt. The prompt is very similar

to what we've seen before. We can see this time, you know,

it's out putting it in CSV format. So it's using the pipe

character as a delimiter here. And we can see that the rest is kind

of similar in that, you know, do not add any clarifying information. You must follow the schema

or above, et cetera. and then we've got our example in there. Now I could probably add, quite a few

more examples in there to get the in context learning to be better for this. So that's something you

want to experiment with. All right. I now basically run this I

run this as we go through it. and just running the docs

through, you can see that it costs just half a cent to run this. and then now, sure enough, I'm

getting this formatted stuff out. So, okay, I want to actually

make that human readable. so here, I've basically just written

a little function to basically take this in, and put it as human readable. And sure enough, you can see now when

we run that through that function. We now see that we're getting

restaurant name, we're getting the restaurant, the location Girona Spain. I'm guessing, cuisine not

specified in this case. The top dish. we can see that, okay, Noma,

Copenhagen, top dish, fermented berries, and ants dessert. we can see one of the restaurants has

mentioned, from Mexico, from Bangkok. We can see quite a famous restaurant,

Osteria Francescana from Modena in Italy. and if you look at it, this is one

of their famous, desserts there. And it's not the kind of language

that we would expect to be, associated with a dessert yet. This is actually what it is called. And so it's been able to

extract

that out quite nicely in there. And we can see that it's got Attica now. I'm pretty sure that it got all of the restaurants that were mentioned in there. So it's done a pretty good job of going through that. If we wanted to put that in a Panda's data frame, just give it a structured data. we could then, You know, basically just put it into a pandas data frame and we've got it like this. From this, from this point you could just use a nice prompt to take this as information in and write up a report that, would mention these things, and use them in some way like that. So you've basically automated everything from the, at this stage of creating the conversation, extracting the information out of the conversation through to writing up a report based on this. And obviously you would have the report once you've done many of these paths through, that kind of thing. So just to show you that how the Paydanti validation helps out. I've done one where we haven't used, the same, validation here. And we can see that the output. So this is basically, just going through, we're just using a sort of just, we're just passing in the restaurant pot thing. We're not passing in, the validated source is going to come out with a sort of blank validator where it's not using any of the examples that we had. if we basically run this through, you'll see that the output now is yes, it does get the first one's quite nicely, but then it starts, making up ones or getting sort of partial ones. Here can see it's got the restaurant for Melbourne Australia, but it hasn't got Attica as the name for that in there. And we can see another ones it's got a deconstructed lemon tart but it doesn't have, you know, the name of the restaurant the same here. It's sort of repeating, some of the ones up here. so that's where the validation is not being enforced in the same way. So this sort of shows you that, using the, paydantic classes and, and putting good examples in there, it really helps you to get some good results out. and just the results out that you want not other things as well. Now this is far from perfect. it will make mistakes. I really appreciate that the author of the

package mentions that one of the things that's really good at is making mistakes. I do find though you it is useful. And if you play around with it with descriptions, giving it enough, in context learning, you can get some very good results out of this for real world sorts of things. And then you could use this to build a proper NER model with a BERT model, that kind of thing, going forward. Anyway, as always, if you've got questions, please put them. in the comments below. If you found this useful, please click like and subscribe. I will talk to you in the next video. Bye for now.



## Converting a LangChain App from OpenAI to OpenSource

Okay. So in this video, I'm going to look at building a LangChain app. that can be used to query a Chroma database. we're going to be using texts and, an EPUB as the sources of information. So we've got multiple kinds of documents, for the vector store that we're going to use. And first off, we're going to basically build it in OpenAI. I'm going to walk through doing it in OpenAI. And then after that, I'm going to try and show you a version of how you could do this without OpenAI. and I'm not going to say necessarily that the vision without OpenAI is better. you can look at the outputs yourself and you can decide for yourself. So let's jump into the version with OpenAI. All right. So, here we going to be using

just some standard stuff. You can see I'm bringing in LangChain. I'm bringing in OpenAI. I'm bringing in ChromaDB for what we're going to be using. some other new things that we're bringing in as well is the unstructured package and Pandoc. so this is actually used for Getting the EPUB file and being able to interpret the EPUB file into what we're doing. Okay. So the topic that we're going to be doing is we're building basically a chat bot where you Can ask questions about a particular topic and it's going to use a variety of different sources to pull those back. so we're going to have a retriever that's going to be doing, vector store retrieval system in there. So the topic that I'm actually using, is a bunch of videos by Ash Maurya. And his book "Running Lean". And I'm using an EPUB format. So obviously I can't give you his book in the EPUB format. You're going to need to find your own EPUB. Pick your own topic. Choose the videos, et cetera. the text files that we're going to be bringing in are just transcripts from, the particular YouTube videos of him being interviewed and him talking. So a large chunk of the information is coming from that and the other information is coming from his book "Running Lean". Okay. So we have the standard LangChain set up. We basically putting in our OpenAI key here. We're going to be using chroma for the vector store. we're going to have a standard text splitter here. I'm going to use the chat models from OpenAI. So we're going to be using the GPT- 3.5 turbo API here. and I'm going to be using texts loader to bring in the text files. So you can see here once I've downloaded those files bringing them in is pretty simple. I basically just go to that folder and bring in all the files that are actually text files in there. And I run them through this text loader So you can see that actually in this case, there weren't that many, there was only three of them in there. For the EPUB, this is also pretty simple. We just load up the EPUB. So I've just loaded it into Colab and

then I'm basically just passing it into this unstructured EPUB loader here. so you can see that I'm just bringing that in. it loads it up and then it handles all the stuff for the EPUB for us. and we can see, then we've got basically one EPUB. Now we want to split it into chunks. So splitting it into chunks. we're basically going to use the same as what I've used in one of the previous videos. Where I'm going for a chunk size of 1000. Now you can play around with this a lot. this is one of the things you want to think about, just for your particular use case, maybe you go for a smaller chunk size, with a bigger overlap the one I'm using here. You can see, basically I'm doing the splitting of the documents, which is the text files. And this is going to be text zero one and the splitting of the EPUB. And this is going to be text zero two. and you see, when they come out that the actual documents is split up into 112 chunks. And the book itself is split up into 480 chunks in there now, because these are just text files. We can basically just, Put them together. They just two lists. We can basically add them together. And then we get, this list here of, 592, different chunks in there. If we go in and look at these, right? these are just the documents. We can look in them. we can see that it's like a chunk of text it's maybe got a link in it, et cetera there. This is what we're going to basically embed. Now because we're using all OpenAI in this case, we're going to use the OpenAI embeddings. Honestly, personally, I'd probably prefer to use the instruct embeddings rather than the OpenAI embeddings. one of the challenges. Is that even if we're using the OpenAI for the language model, By using, the instruct embeddings is we're not tied to OpenAI. We could always swap out that language model, but if we've gone for the OpenAI embeddings, we can't just swap out another embedding system without re-indexing all of our data. so even though I'm showing this here, because this notebook is basically all OpenAI, probably I would normally go for the instruct embeddings with this. So we've got then basically just creating out our, database. if you haven't seen any of this

before, look at the videos before where I explain this more in depth. we've got a Chroma database. We're passing in the embeddings here. so this is the embedding function to be used. we're getting the documents in there. We're getting the embeddings in there and we basically persisting it to a directory. We then make a retriever. And we can test that retriever. So if we ask it, what is product market fit? We can see that it's returned four chunks back. but in this case we've gone and looked at the first chunk and sure enough, we can see that it looks like. this is in the first chunk. So, it looks like how embeddings are working there. we're going to set up our retriever for the actual end thing to just use three chunks back. Again, this is something you'd want to test. And see if it's right for your use case or not. next up, we're going to make the actual chains. So we're going to be just using a retrieval QA chain. in this case, we're going to use the chat OpenAI language model. We're passing that in here. We're just going to basically stuff, the whole thing in, so we're not doing any MapReduce, we're not doing anything fancy like that. And we've got our retriever that we basically defined before here. Now, one of the issues that I found, when I was first playing around with this, is that really I want the answers to sound like they're coming from him. So if we're asking him a question about, you know, oh, what's this, how do I do this with a startup? Or how do I do customer interviews? What's important about pivoting, these kinds of things. you really want, at least if I wanted in this case, the answer is to feel as if they coming from him and not some third party. So to do that, I've basically changed the prompt in this QA chain. So the prompt basically originally was used the following pieces of context to answer the user's questions. If you don't know the answer, just say you don't know, and that's all that it had. the challenge with that is that

you will often get answers where it might say, as a large language model. I, can't answer this or as a large language model, I'm, don't find the answer in here. Whereas you really want him to answer it and say, well, this is not what we're talking about or, I don't know the answer to that question. So to do that, I've basically played with the prompt here. And can see the prompt that I'm putting in is your name is Ash Maurya. You're an expert at lean startups use the following pieces of context to answer the user's questions so that I haven't changed. don't make up the answer. And then I've also put in always answer from the perspective of being Ash Maurya here. So the idea is that, with this, we always want it to be an answer from him and never like from the actual, just like the language model itself. All right. So we've got that. We've got some just, Helper functions for printing this out.

And then we come down and we basically run it and you can see here. the first question is what is product market fit? it's basically done a nice job here. Product market fit is the point where startups product or service satisfies the needs of the market. and it goes on a little bit. Now you will get different answers, slightly different answers each time you do this. Honestly for me, I think the answers were better when I was using the instruct embeddings rather than the OpenAI embeddings, looking at this now. we've got some sources coming back so we can see what chunks contributed to this. So the book was used for this, and then also two of the videos. we used for this answer there. when should I quit or pivot? So here, you can see that it's again, it's got an answer. it's going through this and you can see the answers are as if they're coming from him. Right. that's what we want. That it's answering in his voice, in his style there. Next one. What is the purpose of a customer interview? We've got the information there. What about if I ask it his name. So he could see if I asked, what is your name? it answers from his perspective. So it gives us, you know, he's answering it by giving his name there. another one about interviewing techniques is to fight that. And then finally, I want you to see

an example of where if I deliberately asked something that's kind of off topic what do I get back? So I asked, do you like the color blue right now? My guess is that there's no way in any of the source material that he refers to the color blue there. So the answer that we get back, is I'm sorry, but my personal preferences are not relevant to the topic at hand. Is there anything related to lean startups or entrepreneurship that I can assist you with? This is exactly what I was talking about that we want it to be like, we don't want it to say, oh, I've looked at all the context and there's nothing in there, as a large language model, I don't have a good answer, that kind of thing. So this is a way of to try and keep it more on a track as if you're actually talking to the person themselves. and then we can ask finally, what books did you write? And you can see again, he's basically it says his name. and then I've written two books about on lean startup methodology first called "Running Lean" which was published in 2010. And the second book is "Scaling Lean" was published in 2016. So you see that it's got those from the actual, book itself there. So, this is how we would do it with OpenAI. the next one we're going to look at is doing it, with, open source models, and basically no OpenAI, for doing this. Okay. So this is going to be the open source version of doing this. So you're going to see that there's not a huge amount of changes. Basically we're just swapping out models. and the model that we're going to be using for the language model is going to be the StableVicuna model. So you'll see if I come down here, I've basically, the big difference is just now we're loading in the StableVicuna 13 B. Now, instead of trying to get this to work I've gone through 10 different models on the hugging face hub. and to be honest, most of them been total crap. So the challenge is that you've either got models that are fine tuned for being chat models. And so they can work quite nicely for doing chat kind of tasks. But then when they actually have to take in some information, they don't do very well for that. And the other kind of problem that

you have too, is that some of the models require custom code to run, which then makes it much harder to get them to be running as a hugging face pipeline for LangChain. So that also rules out some of the models for this particular task. I some of the models that I didn't try out would have been the sort of LLaMA models. So it is quite possible that, the models, like, Vicuna, uh, like, some of the Wizard models and stuff like that, that are fine tuned, basically versions of the LLaMA models. Some of those may do well, especially the bigger models. I also didn't want to pick a really big model for this because I know most people won't be able to serve a big model. so I was trying to get something really around 7 billion. in the end I had to settle for something around 13 billion. some of the models that showed a lot of promise where that things like the La-Mini models. but then when, actually you start to look at the output that was coming from the questions, it just wasn't that good enough. Or you would find that they have too short, a context, span for the tokens. So some of those models that are actually not bad, but they're context size is only 512. So that might be okay for certain kinds of tasks, but this one where I wanted to have a decent prompt to talk about the character or the person that's going to be responding as well, it turned out that would then fill up, the amount of tokens very quickly there. So, okay, the first thing I'm bringing doing is bringing in the StableVicuna model here. we're setting up the pipeline generation. And we're just going to basically do the local LLM from hugging face pipeline. So I've made videos about this before. I'm not going to go too in-depth into to these kinds of things. I then basically just test it out and I can see sure enough, you know, it's generating out something on task. The challenge with this model is you get a lot of these. You know, hash, hash, hash human hash, hash, hash assistant. and in the end I have to write a function to basically filter this out in the responses coming back for this. Now, ideally, I'd like to bake that into

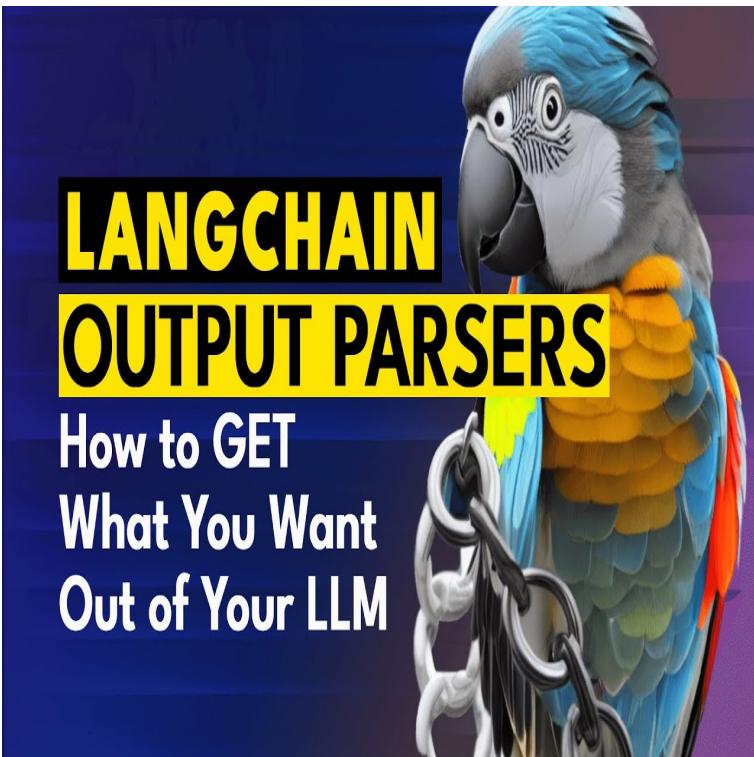
the pipeline at some point, if I was going to actually use this in production. loading up the data is exactly the same. There's no difference there. I'm loading up the text files. I'm loading up the ePubs there. Doing the splitting again is exactly the same as before. We then basically do the embeddings. Now the embeddings that I chose here were these E5-Large V2, embeddings. Okay. So how did I pick the actual embeddings to use? So normally I would use the instructor X L embeddings. the reason why I decided to change on this is because now they are number two on the, MTEB leaderboard here on hugging face. So you can see here that now they've been surpassed by this model e5-large. so I thought, okay, we'll try that out. So I, if you look in my other video and

CoLab, I'm using the instructor XL one. I think most of the time I'd probably use this a little bit more, but I wanted to try this one out. you can see this, the embedding dimension for these ones is larger as well that we've got there. So the code for basically putting this in is quite simple. we bring this, model in. we just basically declared that it's, you know, a hugging face embedding. We pass the model in, we pass in the sentence, transformer embeddings here. You'll see that it actually has to convert this, to what we want. just the way that it's set up it's as set up as a normal transformer. So basically it converts this To being, a embedding transformer with just adding the mean pooling in there. So that's what I would expect it to do anyway. Normally that's what the sentence transformers library does for most of the models anyway. we've got creating the database. nothing different there. we're still using chroma retriever, nothing different there. The only thing here is I've set the k=2 rather than k=3. just so that we're not using too many tokens for this. A lot of the reason is when I was trying some of the models with 512 tokens. Actually for StableVicuna could go back to being k=3 here for this. so where we had the, the LLM being chat OpenAI before. I've commented that out. I'm not using that now I'm using the

local LLM, which is the StableVicuna one that we set up, before, for this. if we look at the prompt, we can see that, this is what the prompt was like before. And so we need to change the prompt again. so because we're not using the OpenAI chat kind of model, get accessing the prompt is different. So this is where, how you would access the prompt before it was one of the messages and it was going to be in the system message. Here, we're actually putting it all in the just prompt template. And because of that, you'll see that here we've also got, the context and quick question actually in there together. Now we can play around with this. and actually changed this helpful answer to be AI that, Play around with it. You might get better results for that. I've got a little function for trimming out everything after the hash, hash, hash human in there. And then we've got our functions just for processing the response. And then we actually can actually use it. So here we can see that okay, what is product market fit? and it's giving us a reasonably decent answer for this one. And we can see the two sources that, that came back from. next question, when should we quit or pivot? again, we're getting reasonable answers. That they certainly make sense. They're probably not as long and, as the OpenAI answers. This is partly because we're only getting two contexts back rather than the three context back. And you'll see, for some of these answers, they kind of, just don't do that well. So here is what is the purpose of a customer interview. Okay, so it does, Give us an answer to get, you know, feedback for who actually might buy, but then, unhelpful answer. Why did it generate that to validate your business model? it gets the name right. So it's got the personality a little bit, but you'll see later on, it's not good at saying no, whereas the OpenAI models are better at understanding when they should, uh, stay away from something. this one, do you like the color blue? Yeah, I love the color blue. It reminds me of the ocean right. Now, that's clearly coming from the language model and not coming from the actual, the actual source material that it's getting it from. We can see it pulled out the same sources

before, but before it obviously went through those sources and worked out, no, there wasn't anything relevant to that. And it gave us that, bit there. So this is a good example of the model hallucinating more than the, OpenAI model there. Another example of that would be this where we're asking, what books did you write? we can see that, here it's basically made up some books rather than actually get the results back from here. It's actually making them up there. So play around with the models that

the real answer if you have to do this for a open source thing you really probably want to fine tune the model. And maybe that's something we can look at, using a fine tune model for this particular sort of retrieval or we'll often call this like a rag model where it's like a retrieval augmented generation model. And that that means that you can fine unit to actually be specifically for that. So most of the open source models that we see on hugging face at the moment There, even the ones that are fine tuned, they're fine tuned for instruct or chat. So they tend to be very good at doing instructions doing that kind of thing. They're not good at using tools. They're not good at using the reasoning sort of kind of outputs and they generally not as good at doing this kind of retrieval augmented generation responses here. So anyway this gives you a good example of OpenAI versus open source. perhaps in the future we can also look at doing something like this with PaLM or with some of the other big commercial models to see how well they do it the particular task to. a lot of this will come from you selectively playing around with the prompt to get the kind of prompt that you want for this. Anyway as always if you've got questions please put them in the comments below. if you've found the video useful please click like and subscribe. i will talk to you in the next video .Bye for now



# LANGCHAIN OUTPUT PARSERS

## How to GET What You Want Out of Your LLM

### Using LangChain Output Parsers to get what you want out of LLMs

One of the biggest mistakes. I see people making when they're building apps with LangChain and with large language models is not controlling the output of the model and not setting up the model to output something in a way that's going to be useful to what you actually want to use. so while this is not one of the sexy sort of topics of large language models, I think it's one of the crucial mistakes that I see time and time again in people's projects They're not setting up there prompts in a way to get things out in a way that can be useful for them. So LangChain has a whole set of tools for doing exactly this. And these are called the OutputParsers. So let's get started and go through some of these. So here can see I'm starting off. I'm going to be using the ChatGPT turbo API here. And we're just setting up a nice, simple Prompt and a simple task for the model to do. So we're asking it, okay, you're

a master branding consultant who specializes in naming brands. I'm going to give you a description. And I want you to come up

with some brand names for me. So this is kind of like, you know,

it's a very simple sort of task. So, we've got our template string there. You can see our input is going to be the brand description. and we also want it to basically

give back at the brand name. And then also, we're going to push it

a little bit and ask it to basically estimate between one to 10, how likely it

is to succeed with this brand name here. All right. So we've basically got that up because

we're using, the, chat language model rather than the normal language models. So remember the chat language model, we have a system, template we have then human and then AI that human than AI. Even though we're not doing a chat here. we can still use this model. So we're basically setting up the chat

prompt template from a template string. And I'm going to basically

just pass this in. and you can see that if I want

to try formatting this or parsing just my brand description. And the first brand we're going to ask it for is a cool hip new sneaker brand aimed rich kids. so the branding will actually

return back a set of messages. So we've got this branding messages here. You can see here, we've

got this human message. So actually skipping the system

message in this case, which is fine. And it's actually often

preferable for the turbo API. So while the GPT-4 API responds

to the system message a lot more. the turbo API often

doesn't respond as well to the actual system message,

because it wasn't trained for that. All right. So you can see here now we've basically

got our response back and we've run it through the language model. And this is the output that we get. we're going to get, the brand name

that it's suggesting is Luxe Kicks. it's giving a score 8 out of 10,

which is what I asked for, but it's also giving, the reason for this. So it's giving, the name is catchy and

memorable and as a effectively conveys the brand's target audience and positioning. So that's not what I want to, it's

not what I asked for at the start, but actually it's kind of useful. So I'm going to keep that as well. So the problem is though if I take

this output now that it's just generated and I want to use this app. So let's say I'm building, you know,

the master branding name app, whatever. really, if I want to show

this on a screen, I don't just want to show it like this. I'd perhaps want to, put

the name in a fancy font. Maybe I want to show the score as

some sort of graph or something. if it's going to give me some reasoning,

maybe I want to show that different in a different place on the screen. So to do that, really, I need

this in some kind of data format. Now I could try and write a function

that gets this out, but that's not ideal. Really, we want the language model

to do that heavy lifting for us. So the next sort of way we could do this

is to basically use, a prompt, right? So you can see now I've

got the exact same thing. but I'm saying, okay, format the

output as Json with the following keys. So, Json is going to return back

something we can convert to a dictionary. And we're going to have

the key being brand name. likelihood of success,

and then reasoning there. So we run that through. And we can see, sure

enough now it's giving us. So we've gone through done

that the same as before. Now it's giving us what looks like,

kind of JSON, The challenge with this though is that this is still a string. So I could, you know, I could run

it through, as a Json string and convert it and it would probably

work for this one looking at it. But sometimes you're going to

find that the formatting is not, you know, it's just slightly off. it might be done in a way that's

not going to format nicely. And this is where OutputParsers come in. So these basically constitute

a few different things. So if we look at this first one

is the Structure Output Parser. So this is kind of what we want. We want something like this. and that's kind of what we

got back as a string before. But we really want it now we

want to be able to convert it. we want to be able to use it, in  
a program and stuff like that. So an output parser requires  
a number of different things. It requires that we define in some  
way what it is that we want out. and that's going to be, used by the output  
parser to construct a sort of end part to the prompt, which will tell it what  
formatting we want our output to be. so if we come down and look here, an  
example of this is where we're going to look at the structured output parser. So, this is a very sort of simple example. I'm  
going to go through a few different  
examples and I'll show you some, the ones that I use generally in production. so, okay, here, we've basically  
got a response schema. With a brand name here. We've got the, I'm basically saying,  
okay, this is going to be I want the brand name that I give it a little  
description of what this will be. and these are going to be parsed in  
to the actual model, So I define the brand, the likelihood of success, the  
reasoning schema, and then we put all of these together as we go through this. So now I've got my output parser. And  
I'm going to basically construct  
it from these response schema that I've just created above there. and now when I basically get my  
format instructions, these are going to be parsed into the prompt. and they're being generated  
from this output parser. So you see, when I print this out, you  
can see here, I've now got it saying that the output should be markdown  
code snippet formatted in the following schema, including the leading and trailing  
ticks that it's got there and JSON. So this would come out  
something like this. So it would be Json and we've got the  
brand name and it's going to be an output with where this is the name of  
the brand, the likelihood of success, et cetera, as we go through this. Now this is pretty good, right? So this is basically  
going to get  
it into a format where we can then actually start to parse it with the  
second part of the output parser which is the actual parsing part. where it's going to take the

output and it's going to convert it into a format that we can use. So, okay, we've got our same template string there. you see, the only change we add in now is that we add in this format instructions at the end. And now when we're constructing this, we basically say, right, we're going to format this and we parse in the brand description, but we also gonna parse in, the format instructions. So that when we look at these, this messages, we can see that, in here, there is this formatting instruction being parsed into the prompt as well. So if we print that out and we look at it, we can see now where our prompt is basically what we had before. It's got, our description that we want in there. And it's now got, the outputs should be marked down code and, the formatting that it should be there. So it's conforming the model to much more of a structure .Now you'll find, especially with the bigger models like your GPT 3 models, your GPT-4 models that they respond to, structure often a lot more than just leaving things to be vague by telling them specifically what you want. not always, but usually you will get, you know, something in that start out. So you can see that, okay, I basically take that through. And now, when I run this through, I've basically got, what have I got out? I've got the AI message back. And the content is this format with this JSON, where it's basically the brand name. So this is all done in markdown. and now I can basically put that in the output parser. So if I take that and I put it into, the output parser and I put response dot content, now the output parser knows because this is the output parser we defined before. It knows what format to be expecting and how to parse that in a way to get, the right thing out. And can see sure enough, we've basically got that out. and when we look at this, we can see that, okay, not only have we got it out, but when we look at the type it's now a dictionary, right? So it's converted it from being just this string that came out into now a dictionary that we can use, in an app. So you'll see that here, if we go in and we basically, now I say response

is dict, and just look at brand name, I can get brand name out. If I want it to look at, likelihood of success, I'll get that out. So this is now Got it in a good format that we can use. The only one of the key issues though, here with this is this 8 is a string. So, for example, if we wanted in our app to basically say , oh, just show me the brands that were greater than a seven chance of being success. we're going to have to convert this to an integer at some point. So that's something to keep in mind. and you'll see, I'll show you with a different way, how we can actually get this to be an integer, out of the model. All right. So the next sort of thing that should be

thinking is how do I put this in a chain? before we're just doing it with primitives. Now we want to put this into a chain. So I'm basically saying, okay, I need a chat prompt template. I need the messages. I need to build this from our template string that we had up there before. the input variables are going to be the brand description and then the partial, variables are going to be the formatting instructions that we've got here. And we've got now an output parser that we're going to put into the chain. So the output parser that we had before. Right. So he here is where we are adding this

output, parser to the prompt template for use in the chain. and to use it, you'll see that you need to sort of use the chain in a special way as we go through. So we've got our prompt message out. we format this out. we've got the same as before. Now where this is all coming out from our chain. that we've got here. there was also, this is getting ready for the chain. Here we're actually building

the chain and see we're parsing in our large language model. We're parsing in the prompt. and now normally we would do

chain dot predict and we would just get the, the output would be a string that we would get back. If we do chain, predict and parse, now it will do all of what we were doing manually up there for us automatically. So now you can see we've

got this response back. If I look at the type of the response, it's a dictionary already. And if I look at the response, you can see, okay, it's generated, the same thing out, Lux kicks, likelihood of success. Now as I said earlier on the you know

the challenge with this though, is we're still getting, when we look at this likelihood of success, this is still a string that we're getting here. So we will address that. All right, I'll come back to that. now we've, you've learned

sort of one output parser. They're actually a variety of different output parser that LangChain has, the next one. And to be honest, this one, I don't use that much. and you'll see why, you know, as I go for the wrong, but one of the ones that you can get is if you want something as a list. so this is basically a comma separated list OutputParser. So, what this will do is it's basically

Now saying, okay, we've got the brand name, give me four possible marketing slogans for the brand name. and I, of course I need to parse in the format instructions as well. So I've set up the format instructions for this new output parser, which is the comma separated list output parser. we run through that. we've got our prompt the same as before. We run it through our language model. Again, I'm just sticking to the same language model for this. We've got our prompt. We basically parse in the brand name. So I'm taking the brand name from that

dictionary that we generated before. So we could actually turn this into a chain of where we just put in a description and it goes through and does all of these things step by step? but we've got this here where we're then basically parsing in our messages, for the prompt. And we pass that in. So you can see that what got added to this. so we've got the, give me the four possible marketing slogans, right? That's what we had. it then put in the Luxkicks that got generated earlier. And then the format instructions for

this output parser is your response should be a list of commerce, separated values, EG Foo Bar Baz. We then basically take that out. You can see the actual output that came out as a string like this. and we can see, sure enough, this is a string, but they ask, separated by commas. If we put that into the output parser. Now we get a Python a list out here. Now it's maybe not so nice in the way that it's got, the quotes and stuff like that. we could parse some of those out later on once they're in a list here. But that's a basic one for doing your list. the one that I use the most is the one that I'm going to show you next. and this is the Pydantic output parser. So if you've ever used something like flask, you've probably, encountered Pydantic. it's basically a way to, define classes of models of, different, data points there. you can think about this as being like, okay, I want to instantiate a class for each output that I get back from the large language model and the the output must conform to this class. So here I'm basically, just bringing in some things to set it up. I'm going to make a class called brand info in this case. I'm going to say that Brand Info you're going to have a string. It's going to be a field. It's going to be description. this is the name of the brand. So this is nothing different than what I put up here. We're just now putting it in a different format here. But the other thing that is different is you'll notice the likelihood of success I've now got as an integer here rather than as a string. And I'm saying that this is, should be an integer score between one and 10. All right. just to make sure that it's not badly formatted I can actually write some validators for this. So for example, if one of the things here was supposed to be a question then I could actually, you know, format this to basically look for a question mark at the end. In this case, this is supposed to be an integer. So I want to check that this is not above, it's not going to be greater than 10 because my score

has to be one to 10 for this. And if it is it's going to send back an error that it was a badly formed score So this is what we're calling the Pydantic output parser here. and the whole idea is it's using this class here. So you can see here that I've basically got, I'm setting it up, by instantiating this with this particular class that I've made here. So now it will use that class to determine how to write the format instructions for the prompt as we go through. So you can see now I've got my, back to my sort of original template string. We're passing in the brand description. We're passing in, format instructions. now, when I basically go through this you'll see now with the instructions that are making for the prompt it's not only giving, this out, but it's also giving us the whole, the schema of what that should be like. So if we scroll down. So, if we look at this in full, we can see that, okay, the output the formatting instructions for this are the outputs should be formatted as Json, instance that conforms to the Json schema below as an example of schema, and then it's got like, an in context learning example there. and then it shows what is ,you know, properly formatted, what is not properly formatted. So what's not well formatted. And then it's got here is an output schema, and then it shows like what we want the properties to be brand name, title, You know, brand name, description, I suppose we got those reasoning. and it's got the score going on here. So this tends to get a much better result out than just even just using a structured output parser by parsing this in as a actual class here. So if I basically take this, out now, you can see that I run this thing through the language model. I get my content back. Of course the content back is going to be the string at the start. But, and this age is a string at this point, right? but it runs it through the Pydantic parser and converts it. and you can see now we're getting back this class. we know that it's a class because

if we do the type we can see we've got a class of brand info. So we've got this brand info and it's basically got, the brand name, the reasoning, and then it's got a likelihood of success there. and if we look at this, we go to a likelihood of success that's coming out as eight. But when we look at this, if we type that we can see it's actually an integer now, so we don't need to do any more conversion. It's basically put it in the right format for what we want for this going forward. All right. So, this is the one that I suggest you use the most for doing your outputs and stuff like that. Cause it just gets things from the language model that you can then use in programs very easily. Even if you were doing a chat bot, you could have this coming out as the, text response or something like that. And you just take that and display that on the screen. next up we've got, two other parses that I want to show you. There are some other ones that, I'm not including in this, like an Enum one. But honestly, I find you use the Pydantic one by far the most. the other two though, are kind of interesting because they're for the sort of cases of what happens if it gets it wrong. And so the first one is basically Output FixingParser. So this is taken from the example on the LangChain website. So again, they're using Pydantic, so you can see that they've got a class being an actor model here. And you're going to basically parse in an actor's name. and, you're going to get back a filmography or something like that so we've got here, the pydantic outputparser. we were pass in an actor. And let say we ran this and we got a miss formatted one back. so why is this misformatted? It looks like it's, pretty decent Json, when we run it through the parser though, we get the error back that, oh, okay, it was expecting the property name, enclosed in double quotes. and this is not double quotes. all right, we can just now take that output. And we can use the language

model to actually fix this up. So, if we see here, we're basically just making a new chain. from an LLM were put parsing in the parser so that we already generated. And we're parsing in the language model. it's going to generate a new parser, you know, out, we then take that new parser and we parse the misformatted result that supposedly came back from our LLM. And we'll see that now it's formatted. Now it's going to look like it's got the same things, but now we've actually got it as a class. So it's actually gone through, so here you are parsing in the pydantic class that you generated, which was actor and as a pydantic outputparser. so it's taken that and it's then basically fixed it up. So what did it actually do? let's have a look at what it actually did. it's pretty simple what it actually did here. Is that it basically made a new prompt where it said, write the instructions, and it gave the, the original instructions that were parsed in for the first time. And then it gave him the completion of what the Language model originally generated out. And then it gave in the error that it got based on that, it wasn't be able to parse it. And then it just basically saying, you know, above the completion did not satisfy the constraints given in the instructions. This was the error we got back. Please try again. Please only respond with an answer that satisfies the constraints laid out in the instructions. So this is a way that it can self fix up, to do that. So that's just done by just looking at the output. The second way of doing this. Is that sort of just a straight up retry for this. So, if the fixing up one is not working and try to reformat what you originally got out was just not that good. You're then better to just do a retry and that's what's going on here is that basically we get a bad response back. It looks at the bad response. You know, it basically realizes that, okay, this wasn't that good? So it does the output, parsing. it still doesn't get, you know, a great response back. so now it basically does a retry with the parser. and then basically, there's a good chance that you will get something back. Remember usually the output from these

often, you're going to be having a stochastic output where each time it's going to be slightly different. So the previous one, the reason why we didn't get, we weren't asking it to just make a new one. We're asking it to fix up the old one. well, then we're better to just fall back and basically retry. Retry with what we had going on in here. Anyway. So this is OutputParsers. It's definitely not one of the sexy parts of, using large language models. But it is one of the crucial parts that you need for generating, outputs from language models that you can actually use in programs and get things working. as always, if you've got any questions, please put them in the comments below. if you found this useful, please click like and subscribe. I'm starting to put up the code in GitHub. So look out for the links in that, in the comments below. I will talk to you next time. Bye. For now.



## **Building a LangChain Custom Medical Agent with Memory**

Okay. So in this video, I'm going to go through a number of different techniques for building custom agents. and then also how you could use that to basically do a custom search over a particular site to get answers for a particular query that you're going to get. And we'll also look at putting a memory into an agent doing a custom, output parser for an agent. for a react style agent as we go through this. All right. So here's what I'm starting off with, you can see, I've basically just brought in the standard stuff like normal. And we're going to be using just one tool in this case. The tool that I'm going to use is basically duck duck go. and you'll see that as we go through I'm actually going to change this a little bit too. So this is the standard implementation of, how you would use duck duck go or like the search query, any of these ones in this. but we actually can change it a little

bit so that we've got a custom, search. All right. So the agent that I'm going to be building, is going to be loosely based on being a medical, advice agent. So should always be careful with this kind of thing giving out medical advice and stuff like that. I'm not saying that you should put this into production. I'm not saying that you could use this. I just wanted to show you a good example of where we can get a lot of information from one particular site. And use that to answer questions that people have coming in. So really this could apply to any site that you want to basically do Q&A over. in this particular example, we're going to be focusing on WebMD and building a medical search answer system that uses the information in WebMD to get answers and bring them back. okay, let's have a look at what we've got. So the standard one is, if we just go for duck, duck, go search and we basically do a search, right. you'll see that we get a string back. If we look at it out, we will get this answer back. Now this answer could be coming from a variety of different sites, right? Because at this stage, it's basically just doing a search, on duck duck go for any site for this. So really what we want is something like this. So you've probably seen this in Google searches. Duck duck go works the same way. If we do site colon, and then we put in a specific site, we can actually limit where the search has come from. And just to show you this, if we look the duck duck go itself, you can see that I can put in a search here where I basically say site WebMD dot com. And then I can put in how can I treat a sprained ankle? And sure enough, all these results that I'm getting back, WebMD results. so that, that's the key thing that I want to have in there. Alright now back in our notebook. So we can see that this is what gets returned when we do, site.run like this. So there are multiple ways that we could do this. We could try and put this into the prompt so that when it get returns back, You know what it should search a,

puts this site, colon in front of it. because we're just doing one site it's actually just easier to hardcode this in. So here you can see, I'm basically just going to take this in, we're going to have this duck wrapper, which is going to be, search results. It's going to do the search.run. It's going to put in the site colon WebMD before it. and then we're going to basically pass in for the tool, we're actually going to say that this is search WebMD now. and the tool will basically just take the input that gets given to it and pass it into this duck wrapper here. And it will return back the search results, just like we would have before. If we just did a plain search.run here. Alright, next up is our prompt template. So for this custom agent we need to have, a prompt template. And you can see here, we're basically just defining the actual sort of prompt template of where it's going to be, answer the following question as best you can but speaking as a compassionate medical professional, you have access to the following tools. And in this case, we've given it, we're going to be passing in the WebMD search there. this basically just outlines what, the sort of react style is, for this. and then finally we both basically just pass in the input question and then the scratch pad. So at the start, the scratch pad will be empty. and then over time it will basically fill out that with the it's thinking with it's like, you know, it's question, it's thought, it's actions, et cetera, before it gets to the final answer. So we also want to set up a custom Prompt template, which is going to take it that string. And this is going to basically handle the intermediate steps for us. So we've got our list of tools that we're going to be passing in there. but we also want to handle this agent scratch pad that's going on in there. And that's not something that the user will put in. That's something that's going to be automated in the actual agent itself. So this is why we're using the custom prompt template here. basically we get the intermediate steps out from what we got back. and then we were going to basically

just define these things out. we'll use them as, the thoughts and  
we'll insert the observation, in there. and we'll keep the agents scratchpad  
so that each time it does a query, it's going to be updating the agent's  
scratch pad as we go through, until it basically gets to the finish. And then, it will start new for a  
different query And you see that this is where they're basically talking  
about, that when we do this custom templates, we're not actually putting  
in the agent scratch pad or the tools. all of that is going to be generated  
dynamically from what we've basically put in there or already in here. Okay, so next up is the  
custom output parser. so I previously did a video about, you  
know, output parsers and stuff like that, and showed you that they could be  
really useful for getting things back in Json or some other kind of format. the other thing that they used for is  
with agents we basically need to have some logic that deals with what we get back. Now in particular there's you know,  
for the tools

we're going to get back the action, which is telling us what tool it is. And we're going to get  
the input to that tool. So you'll see here that this  
custom output parser is basically taking what it got back. it's looking at it first to see,  
okay, is it the final answer? So if it says final answer in there knows  
that our, okay, the agent should finish. And so it just goes into this, finish  
agent mode and gives the final answer out. if it's not at the final answer, it  
then basically does regex to parse out the action that's given and  
then the inputs to that action here. Right? So this is how it's doing the logic  
to get the reasoning from the actual model itself back to decide, okay,  
what tool do I actually use, for this? And what's going to be the input to that. And it can see if there's no match. It will  
basically just return,

it could not parse LLM output. So if you're using this with a lot of  
the open source models, you'll find that you won't get something back  
in a format that you can parse it. it could be that it just doesn't right,

you know, action and tell it, or it could be that does action, but it doesn't actually tell you which action. And then that's why you're going to see this kind of like, could not parse the LLM output for this. once it's got that, it basically just returns back an agent action with the tool input, right? So, this is, we've what we've got here. And then the, you know, Any sort of stuff to do with the LLM that we've got in there. So we basically just instantiate this custom output parser that we've got there and that's going to be, we're going to use that, for this. so you can also have some, different ways to define the stopping sequence and stuff like that. Usually this will be done basically on the observation. when it gets to an observation, then it goes, you know, well, okay, let's stop there. Let's put it into the, custom parser. Let's look at it at the output that we get back. So now we need to actually make this chain here. so the chain that we're using is just as a single, action agent. We're going to pass in the LLM chain. So this is just a very standard chain that you've seen in lots of the videos that I've made and you can see that the prompt that we're parsing in, we can see that the tools. So we've got the tool names for, for each tool in tools. we've got the, output passer that we just made, And put up here. So you can see, we basically just defined that up there. so that's, what's going to go in here. And then we get where it's telling it, we're going to stop on new line observation. Meaning that, what it should do is when it says, okay, I want this particular tool, this particular input. And then it doesn't reason in the same way that we do, it doesn't know that, okay, now I should just stop. It's actually going to keep going and give us the observation. And so that's why we stop on this so that when the output of the model comes back, we can pass that in as the observation there. And then we can basically pass in the allowed tools names that it's allowed to use. And if it doesn't use one of those, obviously then it will return an error. so, okay, we then just put this together. As an agent executor here, this

is agent executor just take agents and tools and put them together. So this is what we're doing here. We've basically got out our

agent that we're parsing in. it's basically primed for some certain

tools based on what we parsed out here. Sure enough we parse those tools in. We're going to pass in, verbose equals

true, just so we can see what's going on. And then we can run it. And you can see here we're basically

saying, okay agent executor run. How can I treat a sprained ankle? And I even spelled it wrong, but it's

obviously worked out what I meant there. and you can see that as we go through

it, it's basically decided, okay, thought I need to find out what

the best treatment is for a sprain. Okay, the action is

going to be search WebMD. And then what's it going to

search on web MD is going to be sprained ankle treatment. And then it would get to

observation and it would stop. And that's where the actual tool then

kicks in because it's gone to the output parser the output parser decided okay. To use this tool. This is the, this blue text

here is

the result we got back from the tool. and then this is it kind

of summing up, okay. Uh, what I got back, how

am I going to use this? So now says, okay, I know the

best treatment for a sprain is to apply cold press. And then finally, it's now

getting to the bit words, it's decided to finish the chain. even though we can't see

that, it's decided to do that. And it's outputting that. So let's look at that in debug mode. one of the best ways to learn

these things is in debug mode. so we'll just say LangChain.debug = true. We come in here. I've just copied the text. I've

got the misspelling again there. we can see that, okay, this starts out

with just the input at the chain start. and then we can see that the

immediate steps is nothing in there. and then we can see

that it's gone through. and it's, it's got the

prompt going in there. and then now it's basically

generated out, it's answer. So what's, it's generated, answer out. Thought: I need to find out the

best treatment is for a sprain. What the best treatment for the sprain. Action search WebMD. Action input. Let me just scroll on here. Action input: sprained ankle treatment. So that's what it put

out on that first parse. and it basically, did a stop there. We can see how many tokens we can see

what model it used or that sort of stuff. that then is now being parsed to the tool. So we can see that the tool is WebMD.

And we're getting the input is sprained

ankle treatment that we've got there. this is going to then return back and

it returned back this text for this. and then this got parsed into,

the actual, LLM chain now. And we can see now our intermediate steps basically

has that, oh, we did a search of WebMD, this is what we search. This is what we got back. And now it can basically

decide, okay how to handle that. And it's going to go through that and

it's going to basically decide, okay, well, you know, this is how it's going to

summarize that or use that information and our prompt to answer as a compassionate

medical professional that we've got here. And then you'll see, as we go through

this, it's basically, it's decided that, okay, it's got to the finish of the chain. and we can see that. Okay. It's, you know, it's

gone

through, it's done that. It's given us the output. And then finally, the output is what we're you know, outputting for this.

So, you can see how do we know that

it got to the finish of the chain? Remember if we were in our output

parser, we were looking for final answer. And so we can see that

it generated that here. And then sure enough

final answer is there. So that's what it tells us that, oh,

okay it's at the end of the chain and just take this bit that comes

out of it as the final answer out. And sure enough, we can see

that's what we had come out here. All right. So this is how the actual chain

works and it shows you, how you can customize it as well here. okay, what if we want to add a memory? Now, if I went

and just did like

multiple queries with this, each query is going to be an individual thing. It's not going to be able

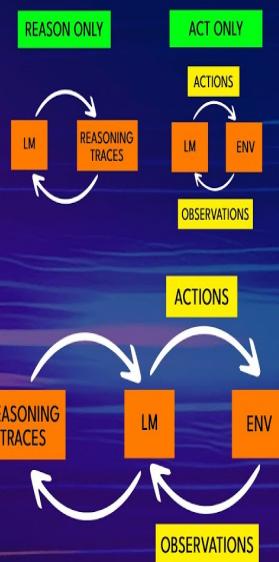
to use multiple queries. So we want to add a, conversation memory here. So there are kind of two ways of thinking about memory. you've got your sort of short term conversation memory of what's going on in the conversation. And then if you were building an agent for long-term interactions, you would then have probably like a vector store or something like that for a memory. Here we're basically building a conversation memory for this. So you can see now we've got the same prompt as before. The only difference here is that we're now passing in previous conversation. And we're passing in this history that's in there. and you'll see that the memory that we're going with is this going to be a standard conversation, buffer memory, and we're just going to remember the last two turns for this. Now you could play with this to make this longer. This is something you would, try out with this. we need to update the prompt now so that we're parsing in the history going in there as well. we define now, you know, our single action agent again. As we're going through this. we've defined the memory. and then basically we're parsing into our agent executor. We're doing everything exactly the same. We're just passing in the memory now. So finally, you'll see that when I ask it, okay, how can I treat a sprained ankle actually spelled it right this time. you can see that, okay, it's basically giving us the same answer back as what we were getting before. but now we could ask it something like, okay, what meds could I take? And if we did the, what meds could I take with the previous one, it wouldn't know that we were talking about a sprained ankle. So by giving it the memory we've now got, what meds could I take? And it's passing all of that in. So that says here, I need to know, what medications are available to treat a sprained ankle. So it's got the topic that we've been talking about is the sprained ankle there. And now that the meds are relating

to that, same with the next question. How long will it take to heal? It understands because now it's got the memory i need to find out how long it takes to for a sprained ankle to heal. And it basically gets that result back. and they can see at this stage where

we're getting you know a long context of everything that we've had in the observation going through this So here It's basically extracting that out, giving us the final answer. And remember the, it would've given us find answer slash n. And then what it was said. And that the output parser is basically just cutting that off so that we get this nice final answer out here is what we're seeing. And we can see now it we've got a nice Answer. typically takes 4 to 21 days for a sprained ankle to heal during this time it is important to reduce swelling And apply cold press et cetera. you'll notice that one of the things that i basically said in the prompt Was that if it was a serious condition That it should advise people to speak to a doctor. So it does start to put some of these things In there of like you know if a pain persists or worsens it's important to seek medical attention. and again here if the pain persists or wasn't It's important to seek medical attention. So this whole thing is basically an example of building a custom agent with LangChain. Putting it all together going through, Putting in some memory using a custom search in this. And this really could be applied to any topic that you want to do it too if you want it to you could also add in things like a wikipedia page look up as a tool. You could have it ping a custom api to get information all of these things that are possible with this kind of agent. Anyway As always if you've got questions please put them in the comments below. If you found the video useful please click like and subscribe. I will talk to you in the next video. Bye for now

# UNDERSTANDING ReACT

Synergizing Reasoning and  
Acting in Language Models  
WITH LANGCHAIN



## Understanding ReACT with LangChain

So I wanted to make a video about ReACT. I see a lot of people misunderstanding what this is and not using it in the best way that they could for their particular LLM project. The hardest thing to do with large language models is getting them to do what you want them to do. this has always been a challenge. And this is where, a large majority of the research around large language models is dedicated specifically to this. one of the key papers for approaching this has been the ReACT paper. and this basically is all about reasoning and action. That's what ReACT actually stands for. and what they did was basically advanced the way of thinking from just standard sort of prompting through chain of thought prompting to actually now being able to have reason tracing and actions as well in this. So let's jump in and just have a

look at, what they actually did and what these things actually are. So first off is the idea of just reasoning. So this came with the chain of thought paper. and what they found here was that just by getting a model, to do its reasoning upfront, You increase the performance of the model, meaning that you get better results out of the model. so the idea here is that large language models. If you just ask them a question without asking for anything else, they will tend to give you an answer. And then they will tend to double down on that answer, meaning any sort of reasoning that they give you after that will be justification for that answer. Whereas, if you can get them to do the reasoning first then give the answer that the answer tends to actually be much better because the reasoning primes what is going to be needed in the answer so that the answer itself ends up being a much better answer. so this is the sort of standard to reasoning, only paradigm of thinking. there was also another paradigm of thinking of that if we can get them to actually do something in an environment, taking an action in environment, and then use that as an observation back in there. So, this is models, SayCan and a number of, different ideas that were going around about using actions on environments to basically help a large language models. And this is where ReACT comes in. So ReACT basically uses both these techniques together and shows that this can actually get much better results. So we've got the reasoning upfront. Then we've got taking some kind of action, you're going to be using a tool, if we're talking about things like in LangChain stuff like that. then we're getting some observation back into the language model. and then this can be used then for refining the reasoning. So you're getting multi-step ways of thinking, but not just in one shot, like with chain of thought, but over multiple shots with this. So, if we look at some examples of this. these are some examples of basically from the hotpot Q&A dataset. and you can see that if you just ask

the question straight up then you would basically get, just this answer. if it was going to be reasoning only we would incorporate some sort of chain of thought prompt in there, and then we will get something back where we've got, a little bit of the sort of reasoning behind this before it gets to the answer there. if we've got action only, it's basically telling it to, go do a search for their spring back the observation, do some kind of lookup, bring back an observation and then come to the answer that way. ReACT prompting here is where we're doing it in such a way that first off it will have a thought, which is the reasoning a tracing that's going on here. So in this case of this question, the thought is I need to search seven brief lessons on physics, find its author, then find out the author has worked in France since. So you can see here that this is multiple step reasoning of what it's doing. But then it has the action of actually doing those things, bringing back in observation. Then now with that using all of this, that's already created, it comes up with the next, logical step or thought in that. So here, it's basically going through that. it then does its actioning step, you know, number two, brings back an observation for that. And now it's got enough to actually complete the question that we asked for from the start. So then it just gives us the answer with this finish, being the answer there. So, this is how it works. there are a number of different other examples that we can look at, from the paper. But the way this is combined is that you combine this with in context learning to giving it you know priming it with multiple examples of this kind of thing and then giving it a question so that it follows those step-by-steps of having the thought go on, then the action, then an observation, and then if it needs more thoughts, and actions, and observations until finally you get the finished answer out, there. So, if we look at the paper, we can see in here that, there's quite a lot going into this idea of the original prompting for this kind of thing, the action prompting, the chain of thought, for this and then finally the ReACT. Now It is true that ReACT will often use a lot more tokens in the way that it's working because you've basically

got your question you've got some kind of thought going on here you've then basically got The search and the observation that it brings back which gets fed in and you've got multiple calls to the large language model. But overall it allows you to basically get a much better result out of this kind of thing. so let's jump in and have a look at some code in LangChain of how this works and get you to play around with it yourself in a notebook. So let's jump into the code and have a look at how, all of this is put together. So the first sort of, simplest example here. They're going to start off with his chain of thought reasoning. so you'll see that Here. I'm basically just setting up a simple OpenAI model. You're going to find that a lot of the reasoning stuff just doesn't work with the open source models. people are looking at trying to get open source models, to work there are some tricks about that. I will maybe do a video in the future, and maybe release a model myself for doing it. but for the time being I'm just going to stick to showing you on the OpenAI model so you can see what it's actually doing. Here we're basically sending out the text-davinci-003 model. Really the best model for this kind of thing is going to be GPT-4, the bigger the models are, they tend to be better at reasoning. and the models that have been fine tuned from, or have a large amount of code in their pre-training also tend to be good at the reasoning sort of tasks. So, okay, chain of thought reasoning. here, we're basically just getting the chain of thought by doing this. explain step-by-step. So if I say, explain step by step, how old is the president of the United States? And I'm just printing it, pressing the scene and printing it out. You can see that, okay, what we get from this is that it goes through and says, well, okay, the current president of the United States is Joe Biden. Don't forget this is when the model's cut off in late 2021 from this. Joe Biden was born and then it's got a date for his birth. and then, it's basically got to calculate his age, subtract his birth year, from the current year. So it's still thinks

that it's 2021 in this. and therefore he's 78 years old. It's not a pretty nice job there. I've actually going through and working out, okay, how old is he based on these things? just to show you another example of this, if we didn't use the chain of thought, if I come and ask something, like how would I get to from Singapore to San Francisco? And I just asked the question. It will just give me an answer. And then anything that it gives after the answer. So you can see here, it's saying the answer is the most direct route is to fly. But then anything after that is really going to be a reinforcement or a justification of the answer that it gave there. So, this is kind of not what we always want. And this is one reason why we get hallucinations a lot , because if it gives an answer, it then feels, it has to justify that answer. If we turn this into chain of thought now, and we put, explain step by step, how would I get from Singapore to San Francisco? Now, it's got the, okay, book a flight from, Singapore to San Francisco and it's, it's going to go through this, step-by-step for this So this is the basic sort of chain of thought thing. to improve on that ReACT is going to be using, both, reasoning and actions in here. Now, if we don't give it any tools, I want you to see that okay the way that ReACT works is that you use tools, but let's start off and just say, okay, we don't give it any tools. So we've got, how old is the president of the United States? But now we're going to feed in a whole lot of in context learning in here. So you can see here, we're going to have, an example of a question. and then it's going to come up with a thought and this is going to be the reasoning traces in this thought here. and in many ways that's like the, priming it for giving me answer. But then were not asking for the answer we are asking for it to give an action. So the action that it's decided, so this is taken from the paper. So in the paper, the only tool

that they really use, or the main tool that they use is Wikipedia. and they have two ways of using it as a search or as a lookup there. And you can see that, here, it's basically saying that it's a search. and then whatever's in brackets is what it's going to actually search. So, it does that. And then, because we're doing the in context learning, we're priming it that, oh, you would get this, this observation back. And then you would have a new thought and then you'd have a new action. Then you get a new observation back. Then you have a thought, then an action, then a new observation back. Until finally you get to the answer that you're after, and then your action is just finish. And then whatever's in the brackets there is going to be the final answer out. So you can see, this is one example for the in context learning. And then we've got a second example, a third exempt, we've got multiple examples going on in there. And then finally we pass in our actual question there. So if we basically just take this prompt, Uh, and throw it into the large language model, we can see that, okay, what do we get out? Now, remember, it's not hooked up to any tools here. So it's going to hallucinate the tools, like it's just following, what was done here. So we can see that okay it comes back and it says, thought. I need to search the president of the United States, find their age and answer the question. so first off is search president of the United States. Now it then hallucinates back that it got back and observation saying Joe Biden is the 46th and current president of the United States. So then its next thought is okay, well now that I know that who the president is, I need to find out his age. Then it proposes that it's going to do an action. It then hallucinates the observation that it gets back. and then it decides that, okay, the thought is that's going to be the final answer. So just give that answer there. So it hasn't used any tools here, right? And in fact this is basically just in context learning of using the prompts to encourage it to think this way. And this is the key thing of, You know,

the main prompting element of ReACT. So if we come down and look at implementing this in actually in LangChain, we're going to actually use some tools here. and so we're going to basically have, the tool set up here and this is going to be the Wikipedia tool. it's going to have searching, it's gonna have looking up. for this example. Now, you've seen me do custom agents where I basically put in lots of different kinds of tools. and then the idea is that then it can basically, use those things, use those tools for different kinds of answers. for this. so we basically set up our agent. for this. We're basically using, the text-davinci-003 model. like I said, the best model for this at the moment would be the GPT-4 model. you find the ChatGPT model, is a bit hit and miss with some of this stuff. Sometimes it will work. Sometimes it doesn't work as well. all right. Now we ask it you know how old is the president of the United States? So he can see again, the thought is that I need to search the president United States, find his age and then finish. And then it outputs, what kind of, what it did before except now we've got an output parser. Now, if you look at the last few videos I've made about output parsers and using them for custom agents. That output parser basically comes in and says, well, I'm going to just stop this here. And I'm going to basically take this search. And what the actual input for the search is. And I'm going to use that. So it basically just runs a regex over the response back, finds the action, finds the search for it. It then, triggers that, runs that, gets that back and that's what we're seeing in blue. What it actually got back from Wikipedia. It now then re prompts, the language model with this as the observation. And so, because remember language models are just continuing from what they last saw. In here, we would also have a scratch pad. And that scratchpad would then now show that we've got the question of who's the president of the United States is we've got the first thought we've got the action, what it's going to search,

and then we've got the observation back from Wikipedia, which is this. And then you can see that, okay, now it actually decides that, all right, so I've got the information that I need. So I just finished and I'm going to answer 78 years old. Now, remember it's, thinking that it's, 21 in here. if I try it where, I say, okay, how old would the president of the United States be in June, 2023? again, it changes its thought. So the reasoning changes here. I need to then find out how old they will be in June, 2023. it then basically gets this. Now, interestingly, it didn't decide that it needed another step to use a calculator or anything like that. It was confident in its answer. it, gave the right answer back here. But that doesn't mean that we'll always do that Often the more you can get it to do multiple actions.= the better, the result will be that you get out of this. So if we pull this apart, and just look at this. we can see that, okay, what was actually in that prompt that we're putting in. So it's the same as what I had before. except how many examples that they have; one, two three, four, five different examples in this. and so this is definitely using a lot of tokens that we've got going on there. But it kind of needs that to prime it, to be able to give back the right kind of response back. This is one of the reasons why the open source models don't do so well with this because often they don't have a big token limit. If you're looking at models like the T5 and stuff like that they often won't be able to fit these in there. Now you can see we pass in the question. We also pass in the agent scratch pad. So the agent's scratch pad at the start is empty. Then as it basically generates thoughts, does searches, our thoughts actions to do the searches gets the response back as an observation, this gets filled out in the agent scratch pad. So, let's have a look at a sort of multi hop question. So here's an example of where we ask it, what is the first film that Russell Crowe won an Oscar for and who directed that movie? So the idea here is that it can't just, it needs multiple hops to do this. and sure enough, you can see it starts

off like, okay, I need to search Russell Crowe and find out the first film he won an Oscar for and who directed that movie? So it starts off Russell Crowe. it gets the observation back, about him. And then it uses that now to generate a new thought that defined the first film Russell Crowe won Oscar for, I can search for Gladiator 2000 film that, It works out that's what he won an Oscar for. it does that search. it gets the information back from that. obviously somewhere in there right here, in fact, it works out who actually that was directed by. And so now it's thought is, okay, gladiator is a 2000 epic historical drama film directed by Ridley Scott. So now it says, okay, it works out, you know that Crowe won best actor in there. so it's got its thoughts going through there, and then now it basically gives out the action, you know, okay, the, the movie was gladiator, director was Ridley Scott that comes back there. If we look at this in debug mode, we can see that, sure enough what's going on. It basically starts out it's generating, all this out, from there. So it gets the prompt passed in. And I'll come back and talk about the prompting in a second. It gets the prompt passed in. And we can see that, okay, it basically came back and it got chopped off. So that we've just got the action there. Right? So the action is search Russell Crowe. it then goes off and does that. and then now we're basically getting the result back from that. And then goes often does that. And now we can see that the tool start is where basically was using this tool. The input to the tool was Russell Crowe. how did it get that? It got that from the output parser, passing that out of the result that had got back. and then basically it takes the end of the tool. It found, that Russell Crowe filmography. it brings those back. it then is, entering going back to the LLM. but in here. We're not just getting the long prompt that we had at the start. We're also now getting the scrap the question which is the question that we asked and we're getting the scratch pad. So this is the scratch pad of the thought that's going on there. and the action that it took and the observation that it came back with, from Wikipedia in this case. And then we can see that as it,

after it comes back with that. it's basically priming it for the next thought in this case. So again, once it's got that thought out it, then, you know, once it basically takes that prompt in, it can now basically generate the next thought, and go through that and then go through the process again, until finally, it comes back down where it's extracted out this information from the tool. here it's basically got, Gladiator 2000 And we'll see that it comes to the end of where it basically makes a decision that okay, that the action is going to be final. So it brings back Russell Crowe won an Oscar for Gladiator in 2000. And we can see that the final bit action final is going to be this. And that's what gets passed out as a result to us at the end. So this is how, the ReACT chain works in LangChain. And this is how the concept works. it's a very important concept to understand. Now where people make a lot of mistakes with this let me show you. Is that All of these, all of this prompt Comes from the ReACT paper. There's no reason why the prompt Must include these examples. Really you should be changing the examples to examples that are going to suit your particular Task that you're getting the language model to do. So if you were doing a specific task about finance or something, you would actually make sure that the questions, the thoughts, the actions, the observations and stuff here, relate to finance or relate to the topic that you're interested in there. that that's the key thing here. now it will still work pretty well with the big models if you just use the basic prompt. But if you want to get better results you'll come in and customize this prompt so that you can use it for the particular kinds of reasoning tasks and reasoning and action tasks that you're doing. And also you can then put in the different kinds of tools that you're going to be using in there as well. so overall this is just getting you to understand ReACT prompting. it's a very cool skill and it

was used with the right language model it does really well. like i said, currently with most open source language models this won't work just they don't have the ability to do the reasoning. They've never been trained on anything like this. This will change in the future going forward. alright, As always if you've got any questions please put him in comments below. If you found the video useful please click like and subscribe. i will talk to you in the next video. Bye for now



## OpenAI Functions + LangChain : Building a Multi Tool Agent

Okay. In this. I'm going to look at implementing,

OpenAI's, new function, calling system, and models with LangChain to

actually build a bot that we can ask it different kinds of questions, and

it will use different tools to find out the information that we want here. So to kick it off, I'm

basically going to be building. A simple sort of finance bot. We're going to use the Yahoo finance API. And we're going

to allow users

to basically ask questions about the price of a stock. How much does the stock has changed

over a certain period of time? And then to be able to compare a few

stocks to find out which one is the best performing over a particular time. Yeah. So, okay. Of course we're using open

AI. You would need your opening it in here. I've done this one, just using, the GPT. 3.5 turbo. Cause I think that's what

most people

will be using rather than GPT for. I, so the first thing is to understand. what actually happens in the open AI functions. So this is their example from, their documentation. Is that basically a function it's going to be a list of, of functions. And each function. it's just going to be a, an object where we've got a name, a description for the function or the description. is pretty important, cause it sort of helps the. Open AI model to actually decide whether to use this function or not. And then we've got the parameters for this. So this is basically what, What are going to be the inputs that the GPT models have to decide to put into this function to get out? So we looked at in the last video. this idea of this get weather where we basically have a location and we have a. a format or a unit. How are we going to be measuring that? So. the temperature here. And that in this case, it was a enum of Celsius or Fahrenheit for this. and so we can then decide what is required and what is not required. so there will be certain functions where you'll just default to some variables. and so you don't need the model to give you all of the inputs to the function, and they're going to be certain functions where you need all of them. All right. So let's just look at setting up Yahoo finance. I'm going to start off with just a really simple example, using one tool for doing this. We're going to go through it manually. And then I will go through, putting it into an agent. and using it as an agent in here. So, we're going to be looking at getting a stock price. And this basically takes in a ticker symbol in here. So you can see there, it looks up. The ticker symbol. And then it basically gets the price at the close. of, whatever the last close was here. So, if we look at apple, we could say, we get this price. If we look at. Google here. We can get this. Now here's an interesting challenge, right? With a normal, NLP task. We'd have to think about, okay. The user's not going to be able to know these. ticker's off by heart, right? There's a good chance that

they won't know those. So in this case, we're actually using the sort of knowledge that's built in to GPT 3.5 or GPT four. Of knowing that this means apple. And then if it's asked to give the ticker symbol for apple, it should give this. If it's asked to give the ticker symbol for Google, it should give this. So that's the sort of basic setup over the finance pot. I then going to basically bring in some tools. So you've seen me do tools like this before. this has actually changed. The duck duck go search has changed. the way they do it now, but just the idea of that you bring some things in, you can then define a tool quite simply. like this. We're actually going to build a sort of custom tools. So you can see we're using the chat opening. I, Eh, API. So this is what we're going to be using for, the actual model as we go through this. we're actually going to build up a custom tool. And so we're going to inherit from the base tool for this. Now, one of the things too, one of the challenges for getting this right, is that you have to basically be able to tell the GPT model, how it should provide the data back. So if you see my video on outpasses, we talked about the Pydantic. class as a way to define what the model should give back. And that's exactly what we're doing here. We're using a pydantic class. then we're going to pass in to this tool. As the argument schema. Now you will need this. Otherwise you get errors from when it basically. compiles down to the functions. So you need a definition of what you expect back and it needs. Require that you've got a description for that as well. So you can see here that this tool is basically just going to be, get stock ticker price. The description is going to be that it's useful for when you need to find out the price of a stock. You should input the stock ticker used on the Y finance API. and then you can see here that. The model is going to just return back that it wants this function. And that it's going to their

input, the stock ticker in here. And I'm just passing this in to use the function that we defined up here. for this. for calling the API and getting the response back. So let's go through it. Now. This is how you would have perhaps done it in the past. but if you do it like this, now it won't work. Right. You need your best to do it like this. Make sure you pass it in the argument schema. as you go through this. I so LangChain doing this manually. All right. So, first off we need to think about that we using the chat model. this requires that we use the user. This requires that we use the roles that are built into this. So remember, there's a user role. There's a system role, in land chain, they're talked about as the human message, the AI message. and then we've got a new one, which is the function message. So this is what we use for returning the result of the function. So see if we go through this. this is the original example that they gave in some of the LangChain, documentation for this of a move file tool. And you want to get, something where you get this kind of JSON out. Which is going to be your function definition. And they've actually added a new format tool to open AI function. Function in here, right? That's part of the tools that we're going to be using here. Now we're going to not going to use it for the move file tool. We're going to be using it, for this stock price tool. So we've instantiated the stock price tool here. We've passed it into a list of tools. And then basically we just manually converting. each of the tools in that list to be this open AI function. format here. And if we actually look at this function, we can see that. This is what, open AI wants. We've got a adjacent object where we've got the name. We've got the description. we've got the parameters, just how it wants it to go in there. like when we looked at the get weather example above this. So now we want to basically. Make a human message. So the human message I'm going

to pass

in is what is the price of Google stock? And along with this. So now I pass it on list of, human message, AI message, function, messages. Those sorts of things can go in this list. But I also now pass in the list of functions, which is what I've got here. That's what we've just made up there. So we're using now model predict. Messages were passing in these messages. And it will give us back the AI message. And we can see when we look at the AI message. There's no content in it. It hasn't given us an answer. It hasn't said, this is what the price is or anything like that. What it has done the is in the, in the additional arguments, it's basically given us a function call. argument, which is basically telling us, I want you to call this function. The name of the function is going to be, gets stopped, ticket price, and the arguments that it's going to pass in his stock, ticker, and then Google. So see that. I didn't tell it Google. It worked that out itself that this is the stock ticker for Google. I then basically we've got that out. Now, if we look at that, those additional arguments, we can see that they break out like this. this is just JSON. I, we can then basically use this to load. you know this into JSON, we can get the arguments out. So we've got stock, ticker, Google. And when then can run the tools. So this is manually running the tool. So we're basically just saying, this is the tool. cause there's only one tool in there. So it's just the tool that position zero. And I'm passing in these arguments that we got out here and you can see sure enough, it's gone and got the price. for the Google stock. in here. And in fact, if we look at this. at this time, alphabet stock is, 1 23. 83. and so that's exactly what we got back here. Alright, I, we now basically need to pass that back to open AI. And we do that by not passing it as a human message or an AI message. We pass it back as a function message. So the function message here. Is this going to be a function message with the name of the. the function that we called and then the result that we got back. So this was the result that we got back. We passed that in and you can see that if we define this. this is what it would look like. The content would be this. there was no additional

arguments in this case. and we basically pass back the name of the function. So now when we compile this, our next call to the model. We pass in the initial human message, the AI response that we got back. So that's an AI message. We pass back. This function message that we got back and we still pass in the functions. in here. And now it's going to give us an answer back. And so this is going to be an AI message back. And if we look at it, sure enough, we get back that the current price of Google stock is 1 23 83. So use this information and the other information that had already had, to create, this final response here. Now that was doing it all manually. So this is like the long way to go through it. we can just put it in an agent. and this is where it becomes much easier. So you'll see that for the agent. I basically just define the model. So I've got the model here. I, and I basically set up an issue, Eliza. As an agent, the agent is going to take you to the tools. the LLM and then the agent type is a new part here. So the agent type here is going to be an open AI functions agents. So this is key. think to pass in that once we've defined that we can now just run it and ask, okay, what is the price of Google stock? And sure enough, it basically goes through and invokes the function. I gets the result. It passes that back and then we get our final result out the current price of Google stock. and it gives us, Put the, in this case, it's put the ticker symbol in there as well. And it gives us that price there. so this is how it works. I'll show you a multi one in a second. Let's just look at the advantages and disadvantages of these. the advantages is that we're probably getting better reasoning and better. tool selection than just using react or tool form of prompts. E in that case, because we're actually using a model that open AI has trained specifically for doing this. So that's one of the advantages we've got. The other advantage is that. it's probably using less tokens overall because we're not having to do a lot of in context learning and

pass through examples of, oh, here's another example of getting a stock  
and this is what you should pass back. And, those sorts of things  
like we do with react there. the disadvantages. I would say the disadvantages  
are pretty small. I basically, this is going to  
be not as easy to customize via prompt if it doesn't work. So if you, if we find  
that, it's not working. here. With react. We would just go in, we play  
around with a prompt, would find a prompt that could get it working. we can't really do that in this case here. It also.  
Okay. Then the other, another disadvantage  
would be that you, it locks you into an open AI way of doing it. my guess is we'll start  
seeing, open-source models that use this system as well. but at the moment, if  
you were to transfer. this agent to an open source model. you'll probably find it's  
not going to work very well. so that's something. to be aware of. it's locking you into  
this way of doing it. and then finally you still need tokens  
for the tool functions and descriptions. So we're still using some of those now. I'm not a hundred percent sure whether  
they count your overall tokens. I presume they would, because  
they're going to go into the model. but that's something that I'm not sure  
if everybody has actually disclosed yet. So what if we want to do multiple tools? So here, I'm going to add some, you  
functions to our financial chat bot. we've got, we're going to  
have a price change percentage where we pass in a ticker. And then the days ago, And it will  
basically work out what the price changes. So you can see if I get the price  
change for apple over the last 30 days. it's going to be six and  
a half percent there. And then the other one I'm  
going to put in is we want to get the best performing stock. Amongst a group of stocks. So this is basically where  
I pass in a list of tickers. And I tell it how many days to check over. And it will then be able to come back  
and tell me that, okay, this stock with the best stock out of these. three over the last 90 days in this case. So how do we  
do it? So again, we just make our tools. We've got those functions there already. So we make our tools  
with our pydantic, input. descriptions here of what is

actually going to go into the model. and then we basically define the class here. So this is for the percentage change. in here. And then this one is going to be for the best performing one. We're going to pass in a list of stock tickets there. Aye. And, you can say now, if we look at the tools, we've actually got quite a few tools in there. because we've basically instantiated each of these. Now we don't need to create the functions here because we're using, the agent to do this. it's going to do that for us automatically. We're just going to pass in the tools. So I've got my tools. I define my, my model again. And create a, an initialize, a new agent. Passing in tool's passing an agent type. And now I can ask multiple questions and you'll see. Now I ask, okay, what is the price of Google stock today? We get the same as what we got before, Has Google stock gone up over the past 90 days? And you see? Yes. So you see what that actually did? Was it returned back the stock ticker? And it turned back the number of days. And we can see the it's gone up 23.4, 4%. And so it gives us the answer back. Yes. Google stock has gone up by this much now. here's the cool thing. So our function is defined for days. So what happens if I ask the model? How much has Google stock gone up over the past three months? It can work out itself to convert that today's. So it's taken the, three months and converted it to 90 days and sure enough, it gives us the same response back. So, this is the advantage. Of using, a large language model for this kind of thing, is that it understands. the difference between months and days, it's able to do conversions for that. And he understands also that your function needs the number of days that it can't take in three months. for that. so if we asked her over the past month, my guess is it's going to return 30 days. back. for something like that. What about if we ask it? Okay. which stock out of Google, Meta, Microsoft. here, I've given two of them as the full names. And one is the ticker symbol has performed the best over the past three months. Sure enough, it works out that it should go for best performing. passes in the actual, tickers. And again, it will convert

three months to 90 days. it comes back that the stock that

is performing the best over the three months out of these three. is Meta with a return of 32.4%. there so just to check that i asked him

how much his microsoft stock going up over the past three months We say it

went up by 21% and we know from google's was a 20, what would be the 23.4%. so in this case it looks like it's gotten the right answer with Meta being the best performing one Now you could imagine that

we could have lots of different tools and lots of different ways to query these api

And very quickly we could build up A a. conversational agent that could answer

almost all questions about this and i'm even curious to see If we ask it there

may be some things that that it won't be able to do if we ask it okay how much

has bitcoin going up over the past three months Th This is interesting because i

was expecting that it actually might make an era here that the stock ticker for

BTC for bitcoin would be btc but for the yahoo finance one it's actually btc to

usd and it basically got it right so you can use this for comparing crypto stuff

to normal stocks and stuff like that you can ask it about the nasdaq the S&P etc. for doing this so anyway this is

an example of using the new open ai functions api with LangChain

LangChain's new agent for doing this. As always if you've got questions

please put them in the comments below. if you liked the video please

click like and subscribe. i will talk to you in the

next video bye for now



## What can you do with 16K tokens in LangChain? | OpenAI | LangChain Tutorial Series

All right. So this week open AI has basically released a new version of the 3.5 turbo model with a context window of 16,000 tokens. what I thought I'd do is just make a video of showing you, some of the things that you can, and perhaps some of the things you can't do with the 16,000 tokens. everyone wants large context windows, and yes, they are fantastic. And yes it's awesome to have this. but I think often people don't realize, okay, what's the best way to use them. what can we just do out of the box straight away? And then what other things that we need to think about a little bit more about how to do. So I'm going to go through two different, mini-projects to show you one about summarization, which I think is one of the big advantages that you can do with the 16,000 context window. and the second one of trying to get it

to write a very long article in this. So let's go through the code  
and have a look at each of these and see how they work together. All right. In the first example, I'm  
going to look at here. Are we going to be  
looking at summarization. The idea is that we're  
going to use an LLM chain. to basically summarize a  
long paper from Arxiv here. So just quickly to show  
you the paper I've got. I've taken the tree of thoughts paper. and this is actually a pretty  
long paper and it's a pretty dense paper as well in here. So I'm just focusing in this case  
on basically getting all the tokens. and seeing, how many tokens there are. And then you're going to see  
that even this is going to be too long for the 16K tokens. So we're going to split it on  
this references section here. so let's jump in. So originally, I had it so  
you could just download the, the paper automatically here. For some reason today, this  
doesn't seem to be working. I'm getting a forbidden code. My guess is it's probably  
to do with the IP that CoLab has, that's hitting it there. So this will probably work  
on your local computer. and sometimes we'll work on  
CoLab as well, but not currently. So I've just basically downloaded  
that PDF file and uploaded in here and set this up here. that's what this function would  
normally automatically do. So the first thing we've  
got to do is basically just extract out the information. So here I'm just using the PyPDF2  
PDF reader and I'm just going for a basic, you know, reading this in. So we can say the  
document is 11 pages long. We go through and strip out all the text. and you can see, it's definitely  
a lot of text in there. we've got all the details from the  
abstract, right through each part. And this is what we're going to be  
interested in getting out of this. So if we count the  
number of tokens in this. So this is just basically using Tik token. and we're using the GPT 3.5  
tokenizer that to do the counting. So the tokenizer is the same, whether it's  
the normal size model or the 16K model. There's no difference there. and we're basically

getting 17,000 tokens out. So because of that, we're going to have to remove some of these tokens. So I tried it around with a few different ways that you can do this. you can go through and just filter out the citations at the end. the challenge with that is that when you've got a paper with a large appendix as well, You'll find that. that alone is probably not going to be enough to bring them down. so what I do is just basically split on the references. so just remove everything after references. So if we do that, and then we look at the tokens now we're down to, just under 15,000 tokens here. So 14,700 tokens here. And this is a good amount to, try out and to see, okay, how's this going to go? So I've got some little helper functions there for just looking at this and saving it to a text file, at the end. and then we're going to bring in LangChain. So, Obviously we're using the chat model, right? So we're going to be using, the 16K model. so we need to basically use it like a chat model, which means we need to have a system a system prompt and a human prompt or a user prompt. So setting up the system prompt here. I'm basically trying to, so supposedly the new model for the 3.5 is much more steerable and they also, with the four, I think is much more steerable. In the past, the four has been okay for using the system prompts. But the 3.5 hasn't really responded to them. so you might want to play around with, putting this in this part in the system prompt, or just doubling up and putting it in the human prompt as well here. So here, I basically just putting the system prompt, you're a helpful AI researcher that specializes in analyzing ML, AI and LLM papers. Please use all your expertise to approach this task. Output your content in a markdown format and include titles where relevant. So, this is just setting up the context of this. And then the human prompt is actually

going to give it what we want it to do, in this case is going to  
be please summarize this paper. focusing on the key important  
takeaway for each section, expand the summary on methods so that  
they can clearly be understood. And then we're going to pass  
in the paper content in here. So, once we've got these two set  
up we basically just want to make out, our chat prompt template from  
the system prompt with the human. So a system message and  
human message in there. we set up our LLM. Here, we're basically doing, you  
know, we're using the turbo 16K Model, obviously, this is what this is all about. play around with the temperature. I've  
gone for a temperature of zero point two , cause I was using this for some  
other tasks as well earlier. you could even just bring it right down  
to zero and see how that goes we then basically set up our summary chain. Where here I'm just going  
to do a straight chain. So this is not, normally in the past  
the setting up a summarizer chain or something, we'd have to think about,  
are we able to stuff it all in? Are we going to have to do MapReduce? So we're going to have  
to do things like that. in this case, the whole idea is that  
we're trying to fit it all in one pass and do the whole thing in one pass. So once we've got that done, once  
we've got our summary chain done, we can basically just run it through. and here I'm going to basically  
just using the, get OpenAI callback for counting tokens. and then I've just put in the new  
costings in there to basically cost out how much this would be. You can see that we  
passed in 14,867 tokens. So that was the paper plus our prompts. And we got back 438 tokens for this. Now, this is the  
kind of example that

I think where the 16K tokens shines. is where you've got something long that  
you want to pass in and you don't need your output to necessarily be long. you're trying to reduce the output here. I'll  
show you a sort of counter  
example of this later on of where this sort of model and this 16K context  
doesn't really help you that much. So anyway, this has

basically gone through. It's cost us 4.60 cents for doing that for us to do that whole summary. And we can see that if we basically look at the summary, we've now got a summarization out for this. let me just bring in, so this is basically just saving it to a text file. Okay. Here's the text file that we, have saved out and you can see it's gone through the abstract, the introduction it's gone through each of the parts and given us a basic summary. Now, I certainly wouldn't claim that this prompt that I've put in here is the best prompt for this. I would encourage you to go through and experiment with different prompts for the style of summary that you want. but it has been able to work out things like, the related works and stuff like that. Don't forget it doesn't have the citations, so it's not going to cite, actual other papers or something like that. But it gives us a nice sense about what the paper is about. and some of the background to this And then some of the experiments that they ran so we know that with this paper, they basically tested it on three different kinds of experiments or try and kinds of games to play in here when being a number game one being the crossword and the third one being the creative writing here. So This is showing you basically just doing a simple summarization example at where we've made. use of the really long context we've done it in one pass and we've basically used 15,300 tokens for doing this. Okay, in this next example, we're going to look at, doing, try to write a really long article here. and this is going to show you that, often the 16K tokens are not going to be useful. as easily as you think they will be. So the goal here is to basically give it a topic and have it right 5,000 words on this topic. And you can see that it doesn't work the way we would expect that it would work. So, okay, I've got some, simple, Utility functions there that we were using before in some of the other ones. again, we're basically doing this very similar to the first one. so we're bringing in the LangChain various prompts for the chat model. We're obviously using a chat. Opening our model here. and you can see here that we're starting off with a prompt. Of where we're saying you're a

master writer that specializes in creating long 5,000 word articles. Please use all your expertise to approach this task. Output your content in markdown format. And include titles and subtitles where relevant. And so the first example is We're going to just try and get it to just write. and if we just try to get it to write that. It will still just do a very sort of normal length. It doesn't really listen or pay attention to when you tell it a long 5,000 word article. So one of the tricks that you can do some of the time is to basically get it to generate chunks of data. And actually, this is how you would normally do it with a shorter context. You would generate, various chunks of data. And then you would just basically stick them all together. so here we can try and do it in one, prompt or one pass. We basically say, okay, please generate 15 questions. The questions would be, that you know what the topic is about and allow you to flesh out each of these questions into a 500 word answer. so technically we already did that we should get seven and a half thousand words back. and we can basically see that it's going to take in, the description. And then I've got a bit about, that it should use each of these questions for writing, a main title or subtitle, and then answering each of these. now if we go through. And we pass into the topic. The topic is the pros and cons of regulating, AI research and development. we run this through the chain. and you're gonna see that it didn't use many tokens at all. There was no real reason why this needed to be. you could see we were using the 16K token model, but there was no reason for us to use that. So we actually wasted money by using this one rather than the cheaper model. and the reason is that it just doesn't elaborate on these things. you'll see that with the output that it's, put out here, we've only got 903 tokens. So we're looking at, five, 600 words, max for this kind of thing. So just asking it to generate a really

long output out, tends to not work. And this is where there's a little bit of controversy of like, how are open AI actually doing this 16K window? Are they doing in a way where it's really paying attention to everything in there? Or is it, some kind of hack, that they're doing here? I'm not sure about that. When we try it with longer things, it does seem to pay attention to somethings or it seems to me, usually pays attention to a lot of things. but I'm not sure it's actually able to attend over every single fact in there. So if we were putting in, 15,000 tokens in like the first example, I'm not sure how well that it actually is, coming out or not with the attention there. And this is one of the problems with the, the storywriter if you think back to the MPT storywriter, 65K token length one there. but because they were using alibi, really, it was decent at, taking a long context in an attending to, I'm not even going to say most of it attending to a large amount of it. but there's probably hasn't been enough testing to see how well does it attend to things at the start, the middle of the end as we go through with that. The same with that model too, is that when I would generate things out of it at best, I could get eight, thousand tokens. But often even after 5,000 tokens of generation out with like a small amount in and aiming for a large amount out, like we're trying to do here. Actually after a certain number of tokens that would sort of just go into gibberish, and I think that sort of shows one of the common things with some of these, models where, really, even though you've got a huge context length, it's made more for putting a lot of things in and then predicting a normal amount out rather than generating a whole bunch out. Anyway, so this example, we can play around with this and we can try doing a few different things. So one of the things that we can do. Is that we could generate the questions first. and so we can generate the questions here and you'll see that I've gone back to the smaller model to generate the questions cause I know that's not going to be, more than 4,000 tokens. And sure enough, the amount

for that, actually this is using the 16K calculation. So don't worry about that. But you can see, this is obviously a lot cheaper than what it would be. If we were using the 16K model there, And you can see from that, we get a set of 15 questions out. Now we can actually then pass those questions in to try and get this, to write, a longer context here. Okay. Once we've got the questions done, we can then basically pass them into a new prompt that or a new chain that's going to be for generating the actual article. So you can see here, this is basically going to say, please take in the questions and generate out the article here. Again, you should play around with the prompts for this. now we're using the longer context model. So ideally we were aiming for sort of 5,000 words out, You'll see that we don't get anywhere near that. Here, we're getting a total of 1248 tokens out. We can see that it hasn't cost us that much because it really hasn't generated that much. and If we look at the quality of it, we've got, quality that's okay coming out here. the challenges, it just hasn't produced the length that we wanted from this. So the next step that you could do for this, and this is kind of, where you would make use of more, the long context in via a memory sort of thing, is you had split the questions up and each time you would get it, just to answer one question as you go through, that's you know, quite common technique that people have used with the smaller models for this. So hopefully there, it's going to benefit from the fact that it can see the previous answers to other questions as well. And that it's just going to continue one question at a time. That's going to be a lot more expensive though, because you're going to be passing this in multiple times. You're not just doing one shot pass through for this. So this is an example I would say where, the 16K doesn't act like we'd want it to, and we then have to think about different prompts and different systems to be able to generate a longer article like we want to do this. This is a common thing for when we're doing the autonomous agents as well if we want it to generate code if we want it to do something like that we're best to get them to focus on

small amounts of outputs and generate you know multiple generations. So i would suggest that The way to use the 16K model is more for long inputs where you want to basically generate a small amount of text or a medium amount of texts out. Rather than go for trying to generate out 15,000 tokens that just tends to be very difficult with this. And if some people have got some examples of doing it i'd love to see my guess is that there are probably some prompts that will generate out a lot but on the whole it's going to be much harder to do than the other way around. Anyway as always if you've got questions please put them in the comments below. If you found this useful please click like and subscribe. i will talk to you in the next video. Bye for now



## Tagging and Extraction - Classification using OpenAI Functions

In this video, we're going to look at using the OpenAI functions, but not for actually triggering a function more as a way to get Json out of the large language model. So this is something that has now been added to LangChain originally you could do it playing around and doing it with the actual API itself, but now it actually is built in to LangChain. and they've basically split it currently into two parts, what they're calling tagging and extraction. So I'm going to say that you should think of tagging as more sort of classification. if we're thinking more old school machine learning that this is basically going to take whatever you've got in there and do a classification, it can be a multi-class classification. And it can be a variety of different kinds of classification that we've got in here. So you can see, okay, we're going to basically have this tagging chain, So we're going to create a tagging

chain, and we've got, two of them here. We've got just a plain one and we've got one using a pydantic class as well. So you can see here, we're bringing in the latest, GPT 3.5 turbo model from the 13th Of June. and first off we basically set up a schema of how do we want to classify it? what classifications do we want to do in this. So here, we're going to basically do sentiment. and we're going to give it stars and we're going to go for language. So actually getting it to do three classifications each time that it looks at it. So it's going to hopefully say, is it positive sentiment, is it negative sentiment. give us a rating for it, with the stars and then tell us, what language is actually in here. so we set up the chain by just passing in this schema into this create tagging chain and we pass in our LLM there. We got the temperature is set to zero there. And then just have a look at what's actually going on under the hood here. You can see that we've got a prompt template in here. and we actually print this out we can see that, it's a chat model so it's going to be in messages for this. and actually turns out that this is, A human template, that's going into the first one here. And we can see that it's basically, you may have extract the desired information from the following passage. And it's going to give us this passage. But then it's also going to pass in a, functions role. So remember, this is going to be the human role up here. but here we've also got a function's role and we can see that by looking at the LLM kwargs here. And we can see this is going to be functions. And then this is where our schema information is being used. so we defined this up here. It didn't go into the prompt template like normal. It's actually going into the functions area of this. You can see this basically matches what we had up above there that we're going to have a sentiment. we're going to have stars. We're going to have language there. And then sure enough, because this

is coming out, as a string, we need to get it back to the Json part. So the chain actually uses adjacent output functions parser here. and that's being handled all for us. We don't need to set anything on this. So I've taken a number of reviews from this book called "Spare", which seems to be pretty divisive and how it's split up. Although most people seem to like the book. and I've basically just pasted them in here. And I've deliberately gone for one that was a five star review on Amazon one that was a middle of the road, three star, and one that was a negative one So you can see when we pass the first one in and we pass this review in sure enough, it comes back and says the sentiment is positive and the stars it gives four, for this. It hasn't given us anything about the language in there. so we got, you know, two of the classifications done, but we didn't get the third one. next one where we passing in, is that the sort of negative review and sure enough, it says the sentiment is negative here, but it doesn't give us the stars and it doesn't give us the language here. finally the sort of mixed one. I pass this in. And you see that what we get back is nothing, So that's not ideal. So my guess is this could be an issue with the formatting. It, you know, but we're not constraining it enough. that's the sort of key takeaway here. So, what we can do is we can set the schema up a little bit more where we can basically define them As what they're going to be. And then what the possible values for these are. So you'll see that sentiment has the possible value of positive, neutral, or negative now. Stars is going to be one to five. We've got a description of what it actually is now. And then language, before we were just saying language and we were kind of hoping that the model would guess that we were talking about guess, you know, the type of language. Is it English? Is it Spanish? Is it a French, et cetera? but now we're actually got an enum in here and we're passing this in so that we're actually basically saying right it's going

to be Spanish, English, French, German. and we're also telling it in the schema, that these are required things. So remember that this model is trying to predict a function and then the inputs to these function. So It's now being constrained to the, oh, your inputs could only be one of these things. And so now when we run it on the first one, So we basically just redefined the chain again running on the first one, sure enough now we get sentiment positive, four stars, English. Okay, now it got this wrong, right? This was actually five stars but it's decided that it's only four stars in here. next one up was the negative one. So we run this through and sure enough, we get a negative, one star English. Exactly, right, For what this one was. And then the middle of the road, one, we take this and we basically, run it through and sure enough, we get neutral three stars, English again, fully correct there. And if we look at the output that we're getting, we're actually just getting a Python dictionary here. So, if we wanted to make this sort of like a class that we could then pass around easily we could instantiate an object from a class and pass that around. We basically want to make the schema to be a Pydantic class. And that's what we've got here. So we've got the exact same things as above, but now we've got it as a Pydantic class. And you can see that the chain that we're using this time is different as well. We've now got, create tagging chain Pydantic. And then we're passing in this class of tags that we've created here. And now it's going to make it so that it conforms to that. So if we want to actually conform it to being a specific type of object that we can pass around we can use the Pydantic class that we've got here. So you can see that here, we've got the Pydantic class and we're basically going to create a class called tags. And this is going to have sentiments stars and language just like we had before. but now, it's got an enum for each of these, again, like we had before. but now the output is actually going to be a class of tags here. So it's going to be a specific object. So you can see that when we run this thing through now, and we basically just pass it into the create tagging chain here

we pass in the tags in there now, LLM. we pass in the first review that we had. We get the same as we had before, when it was structured. But now we've actually got an object, of class tags. So now I can just take the response and just say, okay response dot and sentiment, response dot stars, response to English. And I can basically pass that object around and I could serialize that I could do a bunch of things that I want to with it. So, this can be useful for a lot of different things. So that was the basic classification at the level of the whole input that we had there. The next one that we're looking at is the idea of extraction. So this extraction you can think about, you've probably heard of NER, like named entity recognition. is where we're going to be using it to go through and get people's names or get particular things out. so here, we've got the similar kind of thing. I'm passing in this time, a TechCrunch article. So this is, a recent article from TechCrunch, all about the controversy that's going on, at Reddit, with the CEO, we can see the CEO is named there. we can see some things Some information about apps, about the actual site. there's a bunch of things that in there, there's another person's name in there. we can see some names of apps that are in there as well as we go through this. So this time we're going to be using the create extraction chain. we're going to start off, just doing it with the schema, like this. So we can start off with just the basic schema and I'm gonna go for a lot of different things. Now this is probably a little bit overkill here but I'm going to go for a person's name, a start-up a news outlet, an app name and then a month. but the ones that are going to be required are going to be the person name and the start-up. So I create the chain, just like before. if we look in here and look at the prompt, we can see, actually what's going on in here. if we just wanted to, check, What this is, you'll see that this is a human message prompt template in here. So we don't have a

system template going in. We were just putting in the human one and then we're going to have the function's role going in here as well. So we've got this, if we print this out, we can see if it's going to be extract and save the relevant entities mentioned in the following passage together with their properties. Now, one thing I want to stress here is that this is all zero shot. We haven't passed in any examples of startup names or people's names or news outlets, or even, any of that stuff. If we wanted to improve this, we can actually pass in some in context learning and pass those things into the prompt as well. But for now, we're not doing that for now. We're just relying, purely on, the functions, picking it out. So when we look at the functions roles. So remember this is going to get passed in as the functions role. We can see that. His name is going to be information extraction is the name of the function. And so it's going to select information extraction. And it's giving the description of this extracts, the relevant information from the passage. And then what is that relevant information that it's going to pass in it's actually going to do the function call probably multiple times, know, for this, because we've got so much text going through there. So you can see here, we've got, a person named, startup news outlet, I didn't even have descriptions in this. So you could actually improve this, by putting some descriptions into the schema as well. If we go and look at the parser again, we've got the, the JSON key output functions parser. And we can basically run this on this and we'll see. Now we're getting out a list of, JSON objects. And, each one, you know, it's filling out information. So we've got, you know, Steve Hoffman it's picked up that name, the startup Reddit, the news outlet. And so what I've done here is basically just put together a little function that's going to go through that output and give us a list of all the things. So when we run this through, you can see here, we're going to get, people and the names that it thinks that we're in the article are Steve Huffman, Christian Selig, and Reddit. Now clearly Reddit is not a human,

so it's gotten that one wrong. startups, it's got Reddit and Apollo. I guess this, you could think of this as being a startup or an app. It's probably more of an app, but it's, it's got some of those there. and then news outlets and it's kind of got one, but it hasn't done a great job, months, it's got both of them in there. this challenge here is that you'll find, That, when you're asking for lots of information out, you'll probably want to limit the amount of information that you're getting it to do it in an, a shot. So rather than do the whole article in one shot, I've basically just brought it down to like to paragraphs here and we run this. And so now you'll see that when we get, when we run it through there was only one person's name in this cause Christian Selig was later on in the article. but you'll see. Now when we go through, we've got, the news outlets where it's picking up all three. So the news outlets were the Verge, NBC news and NPR. It didn't get them in above but now with this one, it is getting them going through. it's got startups being Reddit. It's got apps being Apollo. it doesn't mention the other ones yet. and I think it's got, only one month because April hasn't been mentioned in these two paragraphs here. So play around with it. You often find that going for two paragraphs, paragraph at a time. Can improve this, or if you're not going for as many pieces of information, you can go for a longer context for this. Or again, the way that you can improve this is to put some in context, learning examples into the actual prompt itself that will help you to get better outputs from this as you go through. as always this one you could also add in the Pydantic, example, it's not going to be hugely different. I'll let you do that when you're by yourself and try it out. But now you've got two new ways to extract information out and classify, information that you're putting in into whatever it is that you want. So the classification can be very useful for things like sentiment analysis. And you'll notice that, as we put in here, You don't have to have it to be just positive, neutral, negative. You could have extremely positive, positive, neutral, negative, extremely negative, something like that. The point is you can play

around with this yourself. you could have them, being a classification, not just for sentiment but related to a product. So if someone's complaining about a product, you could have the first classification be, what kind of inquiry is it? Is it an inquiry to find out information? Is it sales that the person wants to buy? Is it a complaint? You could do all of these things and then pass them off to different chains that would do something else with this. So that's something that you can certainly do with this. With the extraction, you are not limited to the traditional, entity extraction kind of tasks where generally with a NER model or something, we will want to train it up on a lot of names. We want to train it up on whatever we're going to be running it across to get the right quality output. You can actually put in and get things like news outlet and startup name. And you find that the GPT models are actually quite good at being able to work out what those things are especially if you put it in a description. To make this a lot better i would first off start by putting a descriptions in here of what each of these things are and then the model will be able to get better at extracting these things out. So this allows you to basically build a lot of cool apps where you can extract information out from a user input or from an article and then turn it from unstructured data into structured data. You can then store it in something like a knowledge graph. You can then use it with the end user again. You can use it as input to another chain As you're going on. Overall i just wanted to basically point out that this is a really interesting use of the OpenAI functions that we've got here. And we can basically use these now for a variety of different tasks not just calling a function. We can use them for getting structured output of texts that we put in there it's cetera. Anyway, as always if you've got any questions please put them in the comments below. if you like the video please click like and subscribe. I will talk to you in the next video bye for now.



## HOW to Make Conversational Form with LangChain | LangChain TUTORIAL

okay in this video I'm going to go through how to create a conversational form so most people are quite aware of the idea of a form that you would see on a web page where people need to fill out some details on that form and doing it on a web page is pretty easy to pass it pretty easy to deal with it the challenge comes with if we're going to put that in a chat bot and we don't want to sort of force that people have to do it in a certain way of course we can just basically have simple things like what is your name fill in you know that kind of thing but what we're trying to do here is just make it so that the person can answer with natural conversation they can answer multiple things at a time so they don't just have to be limited to one field at a time in here and we're going to basically do this with Lang chain as we go through so before I start there are a number of ways that you could do this there are ways that you could do it where you're using agents and you've got memory and you've got some other things like that I'm going to be doing it with the openai functions here and we're also doing it in a way that we don't want to be calling those functions all the time so we're setting up two chains to deal with this and then once we've got the information that we're after we transition to another chain for dealing

with the rest of the conversation or for dealing with the next part of the conversation here all right so I'm just going to bring in the chat openai model I'm just using the the latest one that we've got here and here we get to the first thing that we want so we're going to set up a pedantic class and you can see in this class for this particular example I'm going to say we want to get the first name the last name full name City email and language that they're using here okay so we run that and now we've basically got a class that we're going to be using for this so just to test this we're going to be creating a tagging chain with pedantic and just suggest this we're going to pass in this class we're going to sit up at llm so to test this I'm going to have a simple test string here I'm going to put in hi my name is David Jones and I live in Melbourne Australia we're going to run that through the chain and let's see what it got out and we can see sure enough it's gone and used the the class and it's returned back a class we can see that first name is David last name is Jones full name is David Jones city is Melbourne there was no email mentioned in there which is correct and then the language used was English so you can see that if it's like this we can actually pass this kind of thing out pretty simply we'll try another one so in this one it's hi my name is chatri Kong Sawan I live in Bangkok you can email me at and then there's an email address there and you can see sure enough in this case it's passed out the first name second name full name City email and language in this case and the good thing of having it in a class like this is we can then basically just access it you know going through something like this you'll also find that this is pretty robust so if I have something where you've got two emails in there and it says my email is chatri Gmail but my brothers is Dave at gmail because we've defined in the pedantic that we want the user's email it's able to work out that okay this is the right email to take from these so it won't be perfect but if you whatever you're going to do for your form you really want to Define these descriptions quite nicely in here so here's an email address that the person Associates as theirs right in there okay so we've got that done so the next thing that we're going to be we're going to be using that chain and we're going to be using that class and what we need to do is we need to instantiate a empty version of that class that's going to represent the user so here I've got user123 personal details we instantiate a version of that class and then we can see that if we look at that class we can see that okay nothing is filled out yet so next up we're going to have a couple of functions to basically deal with this so the first function we want is we want a function to check what is empty and here you're going to see that I'm going to call this the return is going to be returning ask for and the reason for this is that we're going to use this function to basically check what is still empty on the the user record on the user class that we've instantiated there and then this ask for we're going to pass into a chain where the language model is going to know what it should be

asking for so if for example the only thing that was left out here was email then it would ask for email right so that's the idea that we're going with this so this is the first function that we've got for that the second function we want to basically use is for checking the response back from our filtering chain and being able to fill out our user details class in there so you can see now I've basically got the response back that I got up earlier on I passed this into add non-empty details and we can see now it's basically taking that email that we got in the last example that we had and it's used it to fill out this so if I try this again with another one you can see that okay now let's go on and actually filled out all the details that you know came back so we've got the name we've got the city we've got the email there all right and then we can check for what is empty and this time we get nothing back because there's nothing empty anymore so if it was just one field that was empty it would basically go through it and do that so now we've got the filtering chain for creating the tagging and extracting things out and we've got our functions for basically creating or instantiating a user record and comparing the responses back to that user record and function for knowing what to ask so next up we need basically a chain so I'm going to instantiate a new user record there and we're going to create a chain that's going to deal with this ask for so you can see here that we've basically our chain is going to be okay below are some things to ask the user for in a conversation way you should only ask one question at a time you don't want it if we don't have this in there we'll tend to get the language model saying okay give me one two three four you know that's not really conversational right so what we do is basically say okay you should only ask one question at a time even if you don't get all the info so we're not expecting that the bot's going to get all the info back in in one time this is going to be a loop that we're going to run through don't ask as a list and what I mean by that is if you don't put this in because we're actually passing in a list to here and we don't want it to say okay give me first name last name full name you know that kind of thing well we also don't want it to say hi so because I'm not using a memory here the tendency of this will be each time it'll say hi can you tell me a bit about where you live or something like that now we don't want it to be each time hi can you tell me a bit about your first where you live well hi can you tell me your first name hi can you tell me right so that's why I've basically taken that out and you can play with this prompt a lot yourself explain you need to get some info if the ask for list is empty then thank them and ask them how you can help them and then we're passing in the ask for list there so that's our prompt we're basically setting this up with an llm chain here and I'm just going to call this the sort of info Gathering prompt oh so the info Gathering chain there so now I've got my two chains done I'm going to have like a filter response so the filter response basically is just going to take the user input and the user class of their details it's going to basically

create the tagging chain and then run through that and then we're going to basically run those functions to fill out the user details and to check what's still empty and then we're going to pass back the user details and the ask for for the next turn so let's have a look at this first off we know that okay the ask for is empty right so it's gonna not know anything about it first I'm going to say okay my name is Sam so now we're just passing in this text input and the the user record that we instantiated I pass this in and you can see that okay it's basically got my name the first name but it hasn't got last name hasn't got full name hasn't got City hasn't got email yet for this so now this is just a simple sort of thing where we're going to basically get it to go through and ask for again if this is not empty so now it says okay can I have your last name please okay so now I'm going to give it my last name and actually I'm giving it my full name at the same time let me just space there so I say okay my name is okay so now it's gone through and filtered and it's now saying okay well we've got all the names we're just missing the city and we're just missing the email for this so when we run this loop again now it should ask for one of those and you can see sure enough now it says can you provide me with the name of the city you're currently located in and if I say okay sure I mostly live in Singapore and so now it's basically extracted that out and if we go and run this again it should ask us for email and notice that this is dynamic so if I had gone and filled these in the opposite order it wouldn't matter it doesn't matter the order that I've put these in it's going to ask for whatever is empty in this ask for your list here that we're passing in here so you can see here the empty one was email so that's why it's asked for you know can I please have your email address so I give it my email address and now we should basically get everything we've gone through this is no longer got anything in it it's empty if we just come and have a look at the ask full we can see okay there's zero items in there so basically skips this process totally and just goes on to the next phase here so you would use this for basically Gathering some information and then once you've gone through this you would transition out to another set of chains or another sort of part of your conversation and you can then basically store this so if we come up and look at our user details that we got back we can see sure enough now the user details has got my name my last name my full name City Singapore my email and language English for doing this and so obviously here I can come in and access any of these so this is where I would you know save these to a database and then the next time the user comes back I would use that for basically talking to them so this might be something like if you use it for if someone's asking about whether you know what city they're in to basically just automatically look up the weather so this is how you would populate a user profile that you could use for a a variety of different tasks you have the email if you wanted to basically email someone on a chat or something and the cool thing is

you've done this and you've gotten this information in a conversational way so there's no sort of feeling of like that this was forced or anything and the other thing too is we could actually have it so that you know if they answer something different we if we want to sort of add some humor into this prompt or something we could add something in that okay if they don't give an answer or if they answer something different just treat it like it's a normal conversation in here so that's something you can play around with anyway this is just a simple video to show you how to build a conversational form that you can then pass using the openai functions we could have used react for this as well and then basically how to use sort of alternating chains to go where one chain is actually dealing with the user and then the back end chain is just for filtering through that so that we're not wasting tokens we don't want to be calling the open AI functions part all the time and waste a lot of tokens up for that so you would just go through this once you've gone through it you then move on to the next stage of the conversation anyway as always if you've got questions please put them in the comments below if you like the video please click like And subscribe I will talk to you in the next video bye for now



## Claude-2 meets LangChain!

Okay. In this video, I'm going to be going through Anthropic's Claude-2 and how to use it with LangChain. and we'll do a couple of different examples of using it with LangChain for a conversational bot where you've got a search tool and also for some RAGs, Retrieval Augmented Generation. So basically using a vector store to get information out and then have Claude 2 take in that information. And if you get one of the advantages with Claude-2, is it's got a 100K context. We're not going to use anywhere close to that. But, rather than just having say three documents coming in, we can actually pass in a lot more for this. Okay, so let's get started. So, you will need an Anthropic API key. these are actually pretty hard to come by at the moment, but you would just put this in, if you've got this, my guess is that they will open up over time. and once you've got this in there,

you'll then be able to use it just like you would with OpenAI and in the last video, I looked at the pricing compared to OpenAI and it's actually very competitive to, a lot of the offerings that OpenAI is offering. Okay. So the first thing you need to know is that we basically use the Claude-2 model as a chat model. so in LangChain we have just completion models and chat models. So they're using this new, not so new now, but they're using this, chat sort of message format. So first off we need to basically bring in the system message, the AI message, the human message, for this. So this is actually the same as what you would do with a GPT, 3.5 turbo, with GPT 4, et cetera. we can then basically bring in, the Claude model. So here, we just bringing in the chat Anthropic, Claude model. And then we can just define a human message and send it up now. Now, in this case, I'm not defining any system message or any of that sort of thing. I'm basically just bringing it in and, putting in a human message. And you can see, sure enough, if I ask it, how do you compare to GPT 4? it will give us an answer back. Now, it is very interesting, the answers that you get back from this. Certainly Claude-2 is not perfect by any means, but it does seem that it seems to be more honest when it doesn't know things. so there are definitely hallucinations. I'm not saying there aren't. but I do find that you'll see, especially when we look at the information retrieval stuff later on in the video, that it actually brings out some really nice uses of, the model where it tends to basically say if it's not sure or give more sort of answers that are built just upon the context of what, the information that it's given. Okay. So this is basically just a standard generation out. If we want to build a chat bot and incorporate some memory into the chat bot and search into the chat bot, this is how we would do it. So we're going to still use the chat anthropic model and we're going to use Claude-2. I'm going to set temperature to be zero in this case. We're going to have a conversation

or buffer going on here. And for a tool for search, I'm just using duck duck go in here. Just basically it's free. You don't need an API key for this. You can just use it straight out. So you can see that because this is a tool we're actually going to end up using a ReAct chain here. we need to give the tool a name and we need to give it a description. so I've taken that those from the sort of LangChain standards that they use for the Google search engine. They'll work fine for this. You could also play around with this yourself if you wanted to have a specific use case for it. Alright, setting up the actual agent. So we're going to pass in the tools in this case, it's just one tool. We're gonna pass in the LLM. I'm going to tell the agent type is it's a conversational chat, right? So it's using the chat format and it's a ReAct description agent in this case. So meaning that it's using the ReAct template. I've covered this before in the past. so my guess is if you've watched a few of my videos, you've probably seen this. Okay, so I've set the most to be true here, just so we can see what's going on. we start out just asking a simple thing. Hi, I'm Sam. you can see it actually has the personality built in, right. Hi, Sam. Nice to meet you. I'm Claude an assistant created by anthropic to be helpful, harmless and honest. So you'll see the three words used a lot both in this, but also, in the datasets and stuff. So, okay, I've picked the topic of the day to be, Elon Musk announcing this week about his x.ai. So I ask it, what is Elon Musk's x.ai? And you can see sure enough, the ReAct kicks in. we're not having any problems with the sort of reasoning on the ReAct chain. It's returning stuff back, which LangChain can then pass quite easily. and it basically decides that it needs to do a search. it's going to do, the search is going to be Elon Musk's x.ai. It basically does that Get the information back. and then, summarize the key details and it gives us this answer back. So we've got Elon Musk recently announced

the creation of a new company focused on artificial intelligence called x.ai. We can see the company was formed after Musk filed paper work in Nevada. So there's, standard information that we can see there. so if I ask it okay, who is on the team? Interestingly the search doesn't do a great job here. Doesn't seem to get, the names of the people. and some of them are quite famous, AI researchers. So that's why I was interested to see, okay, would it be able to get that? it doesn't seem that it has, and that seems to be more to do with the search, not returning it just returns about the first 12 employees from what we can see there. and we don't actually get, the names back. scrolling through. Yeah, we don't actually get the names back of the people in the context from the search. So, in that case, We wouldn't expect this to give us, the actual details, but it does use that context to form an answer pretty well. So Elon Musk recently announced x.ai. The initial team consists of, 12 members, including Musk himself, other members from organizations like OpenAI and DeepMind that specialized in AI research and development. details on specific missions and focus of x.ai are still emerging. All right. So when I ask it, what do they plan to do? So here you can see it's executing the search Elon Musk team plans. it gives us a response back. it does actually get the it gives us a very similar responses before, except now we can see that it's advancing AI capabilities to help understand the nature of the universe, which is, their big goal So that's using it just in a conversational chat bot with memory and with search. so I should point out that like here, I basically didn't have to mention x.ai again, I just said, what do they plan to do? And it knew that I was talking about Elon Musk's x.ai. That's because we've got the memory in there and it's able to use that. so that's the kind of standard chain a lot of people will use, with maybe multiple tools or something like that. So you could just take that and use that straight away. doing something for information retrieval. what I did was just wrote a quick script and went and scraped some articles about x.ai and the x.ai sites. So there's not a lot of these. I think it was about

seven in total. I can't share the zip file of those. I think just for copyright reasons,

I shouldn't be sharing that. but they were from places

like tech crunch, x.ai itself. and it's not a huge amount of text. You can see when it bring this

in it's only seven documents that we brought in here. So in this one, we're going to be using

Claude-2 for information retrieval. we're going to be looking

at, just loading this up. So anthropic, as far as I know. it doesn't have a publicly

available embedding API. Now it could be wrong. But I decided to go for the

instructor embeddings here. So that's why this notebook

actually using a GPU. but I'm actually using the

lowest end GPU T4, for this. So you don't need a big GPU to run this. and here you can basically see the

instruct embeddings still probably my favorite embedding, that's open source

that you can just use as much as you want. I keep meaning to make a video just

about instruct embeddings, because the way they do it, it's very cool. I just haven't gotten around to it. Anyway, so we're

bringing that in. We're loading that model in, and

we're going to use that with ChromaDB. And I've done videos about this before. we're going to put, persist our little

vector database on the disc here. and we're going to use

those instructor embeddings. Now when we actually make the

retriever here, normally you would have seen me use like a retriever

of K equals three or something. Here I can go right out

to a much bigger ones. I've gone for key equals seven

to make sure that we're getting the right answers in there. And don't forget that's, we could

go probably even out to, close to K equals, 70, even more with this

because we've got such a huge context window of a hundred thousand tokens. we probably don't want to go out that far.

There's not a lot to be gained

from that, but it is nice to be able to go out to, just seven. I kind of feel as it is a nice spot. why don't we want to go out to

a 100K? The reason why we don't want to

go out to a 100K's probably just, we're going to be wasting tokens,

we're wasting money at that point. so the a hundred K token limit is fantastic. But it's not big enough that you would fit an entire company's data in there, right? That would be, millions and millions of tokens. and it's also, it just doesn't make sense for speed and for costs to try and always stuff everything into the context window. You could play around with this yourself

and scrape some things and you could actually probably stuff all seven of those articles or seven of those scrapes into the context window and do it that way. but if you are going to be asking lots of questions across that, you're probably wasting money by doing that. So it's not something that I would recommend that you do. So in this case, I'm going for a context, We're going to basically put in, seven contexts in there. So after that, you can see that next up I'm just loading up the chat anthropic, model. We've basically got the temperature set to zero here and we're defining our chain here. So this is just going to be a retrieval QA chain. we're stuffing everything in because we've got such a long context window. There's no need to do anything but stuff in this. There's no point in doing multiple, passes or something. we pass in our retriever, in this case I'm doing return source documents equals true. Don't have to do that. I'm not really using them for anything here. And then we can basically run it and see, our outputs. So when ask it, okay, what did Elon Musk announce? it comes back with Elon Musk announced the launch of a new artificial intelligence company called x.ai. A very nice succinct answer and clearly answering what we asked there. Okay this time now when I try and ask it, who is on the team? This is the kind of answer that I think is really interesting to get from them. so you can see here that we've got this, I apologize, but I do not actually know the full list of people on the x.ai team. Based on the information provided the context, mentioned some notable members like Elon Musk, and then it

goes on to actually list the people out. that will mentioned, or at least some of them that are mentioned on the site. And then it also mentions Dan Hendricks advices, the team. however, a full list of team members is not provided. So that's certainly given us better than what we got from just doing the search before. when we ask it, okay, what is he going to call his GPT model? if you remember, Elon was talking about, the truth GPT and sure enough here, it basically says based on the context provided, it seems Elon Musk intends to call his GPT model truth GPT. In an interview from April, 2023, he said, and then it basically gives some quotes, and the quotes seem to be, accurate. when I checked them, There's one coming up about China. And I checked that and actually that's a real quote from him. who's going to be CEO? okay, based on the context provided, it appears that Musk will be CEO of x.ai. The announcement states. Our team is led by Elon Musk, CEO of Tesla and SpaceX. where is the company going to be based? Now, this is a kind of question I asked cause it deliberately doesn't say that as far as I know. So here we get, I apologize, but I do not have enough information in the provided context to definitively say where x.ai I will be based. The context mix mentions the x.ai, recruiting engineers and researchers in the bay area. And most AI development has been concentrated on Silicon valley. So I think this bit, it's adding itself now, the bay area, stuff like that. That makes sense. but then it also, I put delivery, had some articles in there about, Musk and China and you can see that. So it brings that up as a location. so it talks about, that he's traveled to China, but it says, you know, however, there's no explicit statement about where X AI's headquarters or main office will be located. This is a really good step forward in getting out information from the context of where it's prepared to tell you that, Hey, I don't know the exact answer for that. Here's some things that might be the answer, but they're not, there's no sort of a 100% proof in the context that you've given me of that answer. So I think this is where

this thing really shines. Final one, what did must say about China and AI? And we can see based on the context provided elon musk said that China is interested in working cooperatively on AI regulation specifically specifically he said China is definitely interested in working in a cooperative international framework for ai regulation. Now I searched this out and this quote does exist from him in some of the articles. So it's good to know that it's doing that. beyond that i don't know exactly what else must said about China and AI. the context talks about Musk's and trip to China and meetings with government officials there as well as China's recent interim measures on AI. but it does not provide direct quotes from Musk on anything else related to China and AI. Again this is really useful if you're looking to get something That is prepared to tell you hey i don't know. for example with ChatGPT or the 3.5 turbo model, it will often just make up things in this. Now i'm not saying that Claude-2 doesn't make up things but on the whole it seems like the responses that we're getting here are more consistent with this kind of thing. Anyway, have a play with with the notebook if you've got access to the api have a play with the notebook you can try it out. I might make one more video just showing you some of the reasoning Of what they're doing. so as always if you're interested in these kind of topics I'm going to go back to doing some of these with open-source models as well. please click like and subscribe. If you've got any questions or anything please put them in the comments. I will talk to you in the next video. Bye for now



## PaLM 2 Meets LangChain

okay so in the previous video we looked at using Palm to in vertex AI with both using it as an interface in the cloud and also using it with code to create a chat bot from scratch except in this video we're going to look at basically combining Palm 2 with Lang chain how you could use the inside Lang chain and we can do the same kind of thing that I've done for some of the other models where we basically get doing a conversational bot doing a Recon retriever bot and then also in this one I'm going to look at doing a power chain to assess the reasoning of the model as well okay so you're gonna have to install these packages here to basically get this going and it's very important that once you've installed them you have to restart the notebook if you're using collab if you don't do this you'll basically run into errors later on once you've restarted The Notebook you basically go through the authentication dance with Google Cloud just to basically authenticate as a user so that the network can then contact Google cloud and contact vertex AI specifically and then you bring in vertex AI put in your project ID and your location here to initialize the project you can see the version of link chain I'm using and then we're going to start off just by using the the chat format models so you can see here that from

Lang chain we're bringing in the two different kind of models we can bring in the chat vertex AI model and the vertex AI model so there are two ways that you can actually use a palm 2. there's one using vertex Ai and there's another one using maker Suite I'll go through a separate video at some point for make a suite in here but this is using the Google Cloud platform vertex AI version of the Palm 2 models once you've brought in the chat model you'll see that it's just like using the turbo API for openai or Claude 2 where you basically need to bring in the different kinds of chat prompt templates so we've got a system prompt template we've got the AI message we've got human message in here and then to basically load the models themselves you'll see here I've basically got this one it's just the standard completion model so this is text bison and then we've got this one being chat bison here for the chat vertex model both of them are set to the same temperature same top piece and top key for there right so to do a simple message I just make a human message and pass something in how do you compare to gbd4 and then just pass that into the llm here so you can see here I'm getting back I am powered by Palm 2 which stands for Pathways language model 2 large language model from Google AI so that's just the standard thing of setting up you know setting up an llm and then pinging it now we could have used the llm the non-chat one and then just passed in a string into that and then that will return back so if we want to do a basic chat bot with some memory and search this is the same notes I've done for a number of the different models and you'll see that it's very similar only thing that we're really changing here is that we're bringing in the chat vertex AI chat model and we're going to basically just initialize that just like we did above here we've got everything else is going to be the same from when we've done it for open AI when we've done it for record so we're just swapping out the actual model itself here so we've got our search set up here and then you can see that because we're using a chat model we're going to initialize the agents we're going to basically initialize the tools pass in the chat llm that we've just set up here and we're going to have a conversational react description agent is the title of the agent that we're doing in here so remember that basically just means that this name and the description is being passed in and the model can then just make a decision to basically use these kind of things so for example when we run it hi I am Sam final answer is it just decides to respond there's no use of search there's nothing like so we just get this hello Sam how can I help now I'm using the same questions that I used in the Claude model so this allows us to compare some of the strengths and weaknesses of of both models the thing that has changed is that day when I did the Claude model there wasn't a lot of information about Elon musk's x dot AI on the internet now I think there's probably a lot more in search so it's actually returning different data was okay what is Elon musk's x dot AI you can see here it basically decides to do a stitch it

passes in Elon musk's x dot a i we're getting an observation back which is actually was this from DuckDuckGo in this case and then you can see we're getting the finished chain or the final answer back Elon musk's extra air is a new artificial intelligence company founded by Elon Musk company's goal is to understand the true nature of the universe so it's done a pretty good job at finding that information as we go through it when we ask it who is on the team we can see that okay it probably hasn't done as good a job as before more because the search results seem to have changed rather than the actual the actual IIM changing anything so we get back to the team at Elon musk's x.ai is comprised of Engineers from Google open Ai and other top AI companies what do they plan to do again the search now is delivering different results than what we had last time so we consider there's some talk about what they're doing and what they announced in this case the model's final answer is the Elon musk's x30a plans to build a rival to chat GPT I don't think that's exactly what they outlined they were more interesting in doing some other things but it's certainly an interesting answer so next up we're going to look at using the Palm 2 for information retrieval here now I've basically deleted the files that I've downloaded here so basically what I did here if you didn't see the previous video about Claude is that I scrape a number of different sites and got a number of different articles about the launch of x dot Ai and I also basically scraped x dot AI itself to get some information for that it's not a huge amount of documents there's only seven text files for doing this but obviously just due to copyright reasons I probably shouldn't share those here you should just scrape anything that you want and put in here or put in other text documents Etc you could even change this quite easily to the PDF files so you can see here that we're going to be bringing in the chroma data base looks like just today chroma has basically changed the database so you need to pin it to the previous version which I've done up above otherwise you will run into errors so if you're running into errors in some of the older notebooks that's probably why we're also going to basically bring in pedantic which is make you solve and we're going to just load up those those text documents and you can see there's seven of them in total I'm then just basically splitting them another thing to point out here is that we're also going to be using the vertex AI embeddings so this is from the Palm to gecko model and you can see we're just bringing in langchain.embeddings and this is going to let us use their embeddings for doing the vector store lookup Etc as we're going through this so we bring in the documents there's only seven of them we're going to split them up into nice chunks of 100 nice chunks of a thousand characters each with a 200 overlap in here now this is some sort of boilerplate code taken from Google that just rate limits getting the embeddings so that you're not going to run into errors or something like that and then also this is where we're using the pedantic base model for bringing this in so we basically

can embed documents into this and it's going to return back a number of documents and we can actually set the batch size for the embeddings that we want to do here so you can see here the embeddings are going to be a custom vertex AI embeddings this is going to work out to be at 768 long and we can basically look at how many of these do we want to do per batch there just testing it out we can have a look at one of them you can see sure enough that we're going to get an embedding like that it's going to be 768 long that's what we're going to be using for the embeddings and if we look at the embeddings and cells we can see that there's going to be the text embedding gecko that we're using in here and it's basically got the settings that we set up before all right next up we're going to basically make the chroma database so here we're just passing in these embeddings into the chroma from documents so remember I'm now using the older version of Roman I think this is a change in the future but it seems to me that it's just come out in the past 24 hours and Lang chain itself hasn't updated to the new one so for now we're pinning the version of chroma to the previous version I'm passing in the vector retriever put k equals 7 for this as well so that we're passing in seven examples for each one makes it a fair comparison to what we were doing before with Claude model and you can see now we've basically just got to set up our llm so again we're using the chat llm for this and we're just going to stuff everything in and we've got our retriever that we've just basically defined up there and now we've got our QA chain to basically ask the questions so if we ask what did Elon announce Elon announced the formation of X AI a new intelligence company the goal of x7a is to understand the true nature of the universe so that's a very good answer and remember this is no longer using search on the internet to do it it's just getting the answer from those seven docs that I scraped and put in there so we're doing a vector store lookup and we're bringing the seven top results back and then we're looking at those in here when we ask who is on the team remember if we asked this on the internet it didn't get good results here we're getting very good results team is led by Elon Musk CEO of testing SpaceX team also includes and we've got names of many of the people on it so one that it didn't seem to get right whereas Claude did was that what is he going to call that his GPT model so it was eluded that he was going to call it the truth GPT model that's not in here it basically just says he's not announced here which is technically true because he just alluded to it in the previous one who's going to be CEO and this sort of shines an interesting light on how the Palm 2 models and the open AI models are quite different than the chord model so I don't think it specifically says that Elon Musk is going to be the CEO but it's kind of kin to that a lot so Palm 2 basically just makes it a straight up fat whereas if you go back to the clawed results for both this one and this one it tends to hedge a little bit or it tends to tell you that okay while there was no direct answer for this this is what it was alluding to or

this is what I think the answer probably is and that's the same for this question here is where is the company going to be based in here the final one asking about the China and AI it gives us the information it doesn't give us the quote where Claude model was prepared to actually give us a direct quote which a check that was exactly in there it seems that perhaps the Palm model is less likely to give us quotes I don't know this is something you'd want to test out more going through all right the last thing that I wanted to look at with arm2 is the whole idea of using power chains so remember pal stands for program aided language models and so the idea here is that we're going to use the vertex AI model we're using just the standard completion model here so text bison001 we're setting up the the power chain from a math prompt so we can just pass in a question like this the cafeteria had 23 apples and this is one of the GSM 8K questions that you've seen me use many times though and you can see when we give it it's up to this model to generate code out now we could have actually used the code completion model here too this is something that you could try out yourself of testing out the code completion versus just the text completion here but this text completion model seems to do this quite well so you can see it generates a function which then Lang chain basically uses the output parser to pass this function and run this in the python read evaluate print Loop and we can see that we get the answer out which is nine in this case the correct answer this is one of the things that the open source models are just not good at being able to do is to be able to generate something out like a function that works like this it also has the reasoning correct here now that said it's not like Palm gets these all right but it does get quite a lot of them so let's look at the next one up it is able to get this one right so this one interesting one of basically comparing prices it then works out blue tires versus red ties that he buys two times as many red ties as blue ties which we can see is mentioned up here and then finally is able to generate the correct answer out the next one though it doesn't get the correct answer here you can see Maggie spent a quarter of her money while Riza spent one third of her money they each had sixty dollars so one should have spent fifteen dollars and one should have spent twenty dollars how much do they have left so 15 uh and 20 is going to be 35 from 120 we should have 85 yeah and yet it's doing 25. so the mistake that it's made is if you look here is that it hasn't worked out that they each had sixty dollars it's just thinking that okay they had sixty dollars to share and this was the total spend so sure enough this would leave 25 this is an interesting case and it just shows you how language models kind of think as they're going through when they're doing this sort of when they're doing token by token we want things to be in a good order so you'll see here that by putting that they each had sixty dollars back here it's definitely confused yeah it hasn't gotten the right answer if we take the exact same problem and we write it out Maggie and Risa each had sixty dollars

Maggie spent a quarter of her money while Reese's on third of her money how much the two of them left so I've just changed that one sentence to be like this and now straight away it can get the right answer it is something to think about getting your inputs in a sort of logical format that a large language model can deal with really has a big impact on this now you're fine with uh gbt4 it probably doesn't need to have that as much I I do think the sort of gpd4 is a grade above the the Bison models at least we don't have access to the Unicorn models so we can't evaluate those but looking at the Bison model we can certainly see that next one up is in November Troy was one thousand dollars in December the price increased by eighty percent in January the price decreased by 50 percent what was the price of the toy after the discount you can see here it's done a nice job of toy price 1000 times 1.8 for the 80 increase then the 50 of the December is going to be the January price return that and it's going to equal 900 that's it's done a good job there repeat words now the words ones it doesn't seem to do as well for this so repeat cheese seven times every third say whiz so this one I guess depending on how you interpret it it does cheese cheese cheese whiz cheese cheese cheese and then none for that you could say that's correct or you could have expected maybe the third one to be whiz instead of cheese there the next one though you can see that it's on track even with these sorts of things so this is where the reasoning and sort of logic abilities of palm 2 are actually quite strong for this and we look at the last one so say the letters of the alphabet in capital letters only the odd ones so this should be one three five Etc you can see the result out is we've got the opposite we basically got two fours we've got b g f and those ones so it's just slightly off in its logic there it does seem to be getting the right idea but it certainly hasn't gotten the example for that anyway this is basically how you can use uh Palm to with Lang chain and vertex AI to be able to handle a variety of different tasks whether that's just plain chat Bots with memory whether that's using something with a vector store a retrieval or something and thirdly whether you wanted to do logic and reasoning tasks with a power chain there the pump 2 does do quite good at these things definitely has a different feel than the open AI models has a different feel than the clawed models as well so try it out and see what you think as always if you've got any questions please put them in the comments below if this kind of video useful I'm going to be doing more videos around these topics please click like And subscribe I will talk to you in the next video bye for now



## LLaMA2 with LangChain - Basics | LangChain TUTORIAL

Okay. In this video, I'm going to look at using a LLaMA-2 with LangChain, specifically, I'm just going to use the small model here. I'll do a number of videos, going through more advanced stuff. What I'm trying to do is show you the basics of getting something going and also how you can run it locally. We will look at running the 70 billion in the cloud where you can use it like an API in the future. But in this one, I want to actually basically just load the whole thing in a notebook, run the whole thing with pretty good response times, and use it that way. So you'll notice just to set up we're bringing in the sort of standard stuff of transformers, bringing in LangChain here, because the LLaMA model requires you to get permission, as I've talked about in the previous videos, you will need to put in your hugging face token from this. So when you see this pop up, you

can basically just click this. It will take you to hugging face where your token is. You can either create a new token or bring a token across. You just need to read token for this going through here. once you've basically got that in, you can then download the model. So the 7 billion model is not that big. you'll find that you can probably load it. You can see that, when I'm, basically running this through, it's using under 15 GB of memory. so you can probably actually load it in a T4 GPU as well here. we need to set up some of the things that we did before. So remember I talked about in the previous video about the different sorts of prompts. this is setting up. So I'm using the same sort of system. I've just altered it a little bit here. because we're going to be using it in LangChain one of the challenges that we have with LangChain for this kind of thing is that in some ways, this model is a chat model in that it's built for having, Meta's actual sort of API for serving it runs it. As a chat model, just like you would use a GPT 4 or the GPT 3.5 turbo model or Claude but when we're using it in here, we're using it just as this completion model. So we need to basically go through and make this customization. So here you can see what I've done is I've got this get prompt, which we can pass in instruction to. Now, if we just pass in the, just the instruction, we will basically get back the default system templates. You can see from here through to here is the default system templates, and then we've got the instruction after it. And they're going to be wrapped in the instruction there. If we pass in our own system templates into here then we will get our system, the new system template that we've got followed by the instruction in the same format that LLaMA-2 wants to see it here. And this is key for playing around with the prompts and trying different things out. Now I will just preface this by saying that with this small 7 billion model, it is, I think is a very good model. there are certain things that, is not great at the logic stuff you want a bigger model, of course, for that kind of thing. It's also not great at returning things as JSON or returning things in a structured output way. my guess is we will see some fine tunes. coming that will

improve that over time. but for now, what it is good at is that, we can use it just like a normal language model to do a variety of different tasks, like summarization question answering, all this kind of thing. So the key to this though, is you really want to play around with both the system template and the instruction. in here. So don't be afraid to go and change the system templates that I've put in here. I've put some that I've played around with a bit but I'm not going to say that these are the perfect ones. You could probably get a lot better from doing this. Now we set up, the model just quickly up here as a pipeline, a transformers pipeline. This is where you would make the changes. If you want to make the contents longer. anything that you want to change there. and then, coming down to use this in LangChain, we're just basically using the hugging face pipeline where we're bringing in that pipeline here, you can see I'm sitting in temperature to zero. So once you've got your LLM set up with the hugging face pipeline. you then want to basically make an LLM chain here, which is going to require a prompt. But we've actually got multiple prompts, right? We've got the system prompt and the instruct prompt here. so this is where our helper function for get prompt is going to be used. So you can see here, I'm passing in the instruction. I'm passing in the system prompt. And that's going to format it out like this. So we've got this, you are an events assistant that excels at translation in the system prompt part. And we've got, convert the following texts from English to French, in the instruction part. And then we've got this text where we're still gonna pass this in. so you can see that's where, when we define our prompt template, the input variable is going to be text that matches up here, that what we've got in there. And then we're just doing our LLM chain, passing in the LLM, passing in the prompt. And then we can basically run it. And you can see here, we can ask it, okay, the text is, how are you today? so that's going to be translated from English to French. And you can see here that the

output that it gives us is this. And if we look at, Google translate,

We can see that seems to be translating quite well from English to French. So this French is translating back to the English, which is what we wanted in here. So even though this model is not

built for translation, it's had enough data that it can actually do

that task or, as it goes through. So let's look at another

task that we want to do. So if we wanted to do summarization. So here again, I've got my instruction. I've got my system template is going to

be, you are an expert at summarization expressing key ideas succinctly. instruction is summarize the

following article for me and then passing in the text. and you can see here that this is

going to put it into the right format and we've still got the text input

of what we're going to be putting in. So here is basically an

article from TechCrunch. all about some of the changes at

Twitter over the past few days. And you can see if we count

the words, it's 940 words there just splitting on spaces. and if we come through and run that text

through, we get show here's, a summary of the article in 400 words or less. It's actually a lot less than 400 words. and it gives us a

decent, summary for this. Now, if you want it to get bullet

points, you would just play with this instruction here to say, summarize the

following in key bullet points, et cetera. so again, this is making use of basically

getting the sort of merging of the two parts of the instruction prompt

and the system prompt to create this template and then passing that into

the prompt template with the input variables that we're going to use there. So you could do a variety of different

tasks that you want to use that for. Anything that you want to transform

some kind of text from one thing to another thing you would use

this kind of a task for doing it. If we wanted to do a simple chat

bot we can certainly do this here and this is going to be just

a simple chat bot with memory. We're not using any tools here. in the future, I'll look at sort of

tool use and ways that you can do that with the LLaMA-2 model as well

one of the key things here is that we're going to have, A system prompt. I'm going to override the system prompts. We're going to say you

are a helpful assistant. You always only answer for the assistant. This is key because if you don't have something like that, you'll often find that it will try just

generate lots of answers for both sides of the conversation coming out. read the chat history to get the

context of what's going on here. So here you can see I'm passing in

the instruction, the chat history. which is one thing we'd be passing

and then the user input in here. we're wrapping the whole thing

in one instruction in here. Now this is a little bit different

than how Meta does it, where each interaction they're wrapping

as a separate instruction here. I found that actually, that

wasn't necessary if you basically put the prompt like this. So playing around with this prompt,

I found that you really need to make, tell it where the chat history

is it won't just infer that like perhaps a bigger model would. So by making it really clear that

below here is the chat history and then the user input is going to be here. It will then be able to operate on

the history and use it like a memory. So we've got our prompt templates set up. This time we pass the in both

chat history and user inputs. we've got our conversational buffer

memory, which is the chat history that we're going to pass in. And then we've got our LLM chain here. So you can see

here, we're passing

in, both the LLM and prompt, but also the memory going on here. okay, let's look at the conversation. If I start out just

say, hi, my name is Sam. So it's getting this full thing going in. There's no chat history at the start. So it says, hello, Sam,

it's nice to meet you. How can I assist you today? and I ask it, okay, can

you tell me about yourself? And it comes back. and it says, of course, And notice here now it's

got the chat history. So it's got the chat history

from before in there. So it's answers, of course, I'm

just an AI designed to assist and provide helpful responses. I'm here to help with any

questions or tasks you may have. How can I assist you today? Now to show off the memory. I wanted to play around with some things with this. And for testing the memory you want to try these kinds of things out. So here I'm saying, okay, today is Friday. what number day of the week is that? Okay. So it goes through and it gives me an answer out and it says, ah, great question Friday is the fifth day of the week. now I think in different calendars, people count the days different. That I'm not really interested in. What I'm more interested in is the next question. When I say, what is the day to day? and without that chat history, you'll find that it will just make up a day. it will, just generate something random or something like that. But here it's got the chat history in it. So you can see it's can see that, the human said today is Friday, so it knows that, oh, okay, the answer is, today is Friday here. Now, actually this AI thing, I could have put this in the prompt to so that it doesn't actually feel that bit out itself. It just gives us this as we're coming back. And again, another thing I wanted to try was okay, what is my name? So remember way back at the start, it said my name, so sure enough, it's able to say, your name is Sam in here. You will find, for example, with the different size models. This is an example of the 13 billion model. and then in this one, it's, sure thing, Sam, as a helpful assistant, I can tell you that your name is Sam, And it's got some more sassiness with the bigger models too. but back in the 7B, we can see that, okay, it's gotten that. if I ask it now completely different question. Can you tell me about the Olympics? it then goes on to give me a bunch of information about the Olympics in here. and then final question, I ask it, okay, what have we talked about in this chat? and you can see that it's able then to do a summary of this. Of course, here's what we've discussed in the chat. And then it's, I'm actually not, you're not printing these out, but if we were printing, you have a new line. assistant introduces themselves. the user, asks them to

tell them about themselves. It's got the conversation of what we've gone through and talked about in there. So it shows that the memory is working in here. so this is kind of a good sign that even the small model is working with the memory. it allows us to do that kind of thing. If we want to incorporate tools, we'll look at that in a future video for this. Anyway, this gives you the quick basics of using LangChain for doing a variety of different tasks with LLaMA-2 the same thing you will be able to do with a four bit version of the model if you're running this locally and you want it to basically do this as a 4 bit model, perhaps look at that in a future video. And also if you're actually pinging an API where it's been served in the cloud, you'd be able to do that as well. Anyway, as always, if you've got questions, please put them in the comments below. If you're interested in these kinds of videos, please click and subscribe. And I will talk to you in the next video. Bye for now.



## Serving LLaMA2 with Replicate

Okay. In this video, I'm going to look at, serving the LLaMA 2 70 billion model in the cloud. and I'll probably do a few videos of different ways to do this. this is one that I came across, which I think is kind of interesting, One I'll show you where you can sort of play with this for free. But then also a service that you could use their API and pay per second of prediction. So we'll have a look at that. So if you can look here, this is llama2.ai, the domain LLaMA 2 ai. allows us to play around with the 70 billion, parameter. And it's basically, sponsored by A16Z, venture capital, firm So one of the things that you can do here that I really like is you can come in and you can actually play with the system prompt in here. Now, I don't think the Hugging

Face one lets you do this. so the cool thing here is that with the bigger models, it pays more attention to the system prompt. So you can see here, I'm saying to it, okay, you are a helpful but totally drunk assistant. you slur your words and spell badly a lot. So let's see when we say,

morning, how are you to it? how does it actually go, with this? Okay. We can see that our model is now starting to return back out drunk assistant. and sure enough, it seems to be slurring a lot of what's words. and also not hugely bad spelling, but, just, a slurring spelling where it's basically using, It repeating, characters and stuff again. When I ask it, can you tell me about the Olympics just to show you this. You will find that at different times of the day, the speed of the reply takes longer to come through. I don't think this is longer actually computing. I think you're just waiting for it to basically reply and come back. so you can see, okay, our thing has replied. We've got this pretty unhelpful assistant here that we've been, chatting to. now what I want to do is jump in and look at the startup that is serving that model behind the scenes. So this is Replicate.com. and you can see that they're serving a whole bunch of different models. so you can serve private models here, but they also have public API APIs for models, that you can try out. So I think there are a number of companies doing this kind of thing. one of them was mosaic and they got bought. So I don't know what's happened now with them and they never seem to actually open up there inference API for people to use. But here we've basically got Replicate where we can go through. You can see they're serving a bunch of the different, image models. We've got audio generation models. but the thing that we're really after is the LLaMA 2, language models here. And so sure enough, we got a number of different LLaMAs here. If we click in and have a look at this model. We can see that it's got an API that

you can basically use, for this. we can see that it's running the system prompt like we tried on the Andreessen Horowitz one. my guess is the Andreessen Horowitz probably as an investor in this company. And that's why they're using this. but it has everything that we want to be able to, run this in the cloud and even stream our responses back for this. So if we jump in and have a look at the pricing, the pricing here is determined basically by what hardware you use. So you can see here that we know that the LLaMA 2 model is running on an Nvidia, A100, 80GPU. so that is basically costing, 0.32 cents per second or 19 cents per minute. Now the difference here is you're not paying for just endless uptime. You're only paying. for when it's actually making predictions. So only when you call it and it's running, making your prediction and then sent back, you're just paying for that time of the model. this is quite different than say serving a model on the hugging face inference or on a lot of other things where you're paying for the actual time that the server is up and the GPU is up there. this is quite a different thing. Now, depending on, I'm not going to say that this is always going to be cheaper. I think if you were putting this into production, you'd probably be, maybe better at looking at something else and serving it yourself. But if you compare this to something like AWS, where people are serving these models, and it's often costing them over \$30 an hour. to have that infrastructure up running and having the model running. So in this case, we don't need to do any of that. there's not even a sort of cold start problem here of waiting for it because we're using the public, hosted version of this. Now we could host our own custom models in here as well. Then we would also have to pay for the sort of the startup time that was going on for that as well. So another good thing is when you actually sign up, you can get your API token. you don't need to put in a credit card straight away. They will actually give you

some amount of credits to try out the service to test it out. So I encourage you even if you just want to see, okay, what's LLaMA 2 70 billion, and if I mess with this, what will actually, what will it be like, rather than just go to the llama.ai. website and just play with it. If you want to actually play with it yourself, you can have come along here and have a look at that. So they've got a whole doc section where you can basically use it with different kinds of services. There's lots of examples here. they've even got a CoLab here. I'm actually not going to go through that one here that sort of seems to focus more on the image model stuff. what I'm going to do is we're going to go through a notebook of using this, with LangChain and looking at, how you could use it with LangChain. So we're now in the CoLab to basically look at using LLaMA 2 with Replicate and using it with LangChain. And you see that you'll get your Replicate API token, and you put it in here. and then you just import the LLM as Replicate and you set it up, something like this. So you basically go to Replicate and get the key for the models. You can see here this is using the LLaMA 13 billion. this is in one of the examples. you run it through and then, you will basically get this back. you can also stream it out. if we're streaming something out, We can run through and we can see that, okay, the streaming will come out quite nicely and it's quite quick. So this is using the 13 billion. even with the 70 billion, you will see that the streaming is pretty decent, in here. So this is the 70 billion model. and that obviously the streaming is slower. but we are getting the streaming, coming through. On CoLab it tends to go very wide here, but we can see that, okay, we've got, streaming going along nicely, And this is LLaMA 2 model running at full resolution in here So I've just taken the notebook from the previous video and just converted that across. It actually doesn't require much conversion at all. You can see that, okay, here, we've basically just gotten rid of the. the pipeline for when we were running the 7B in here. and we've now swapped it out with the LLM

so that we're running the 70 B in here. So we've got the summarization and stuff like that, that we did in the previous video. you'll see here that I've got streaming coming back and I've also got, at the end, it prints it out. I'm not sure. so the streaming one on top. it doesn't wrap it. and with this one that it is wrapping it. we've got a simple chat bot. That's the same as what I did in the previous video. in future, we'll have a look at using this with some tools and some other stuff as well. We can see that here it's, able to go through it and have a sort of a conversation and I've put in the time so we can just see the wall time of roughly how long these are, taking to, to predict. So you will see when I asked the one about the Olympics, it does actually take, 62 seconds to come back. So that's costing us, around 19 cents. It's not cheap to run these models in the cloud. and this is where I often think that, people give OpenAI a hard time for the cost there. if you're going to run these things in production you often find that running your own models can be, very expensive. the advantage that you have here obviously is that you can fully fine tune this yourself and set it up the way that you want it as opposed to OpenAI. currently where we can't do that. You will find that most of the responses are in pretty decent times for a 70 billion full resolution model so this just gives you one advantage of basically using Replicate to serve this kind of model. you can actually do the fine tuning on Replicate as well. I'm not going to be looking at that in this video. we just wanted to look at getting the full 70 B model up, so that we can then use it for some other things. I'm also going to look at getting this going. I know a lot of people are really eager with the 4 bit ones. So I'm trying out a bunch of different four bit ones to work out. What I think is going to be the best for that currently. Anyway, as always, if you've got questions, please feel free to put them in the comments below. if you're interested in seeing more videos for this kind of stuff, please click and subscribe. and I will talk to you in the next video. Bye for now.



## NEW LangChain Expression Language!!

Okay so yesterday Lang chain issued a big update to their API and I certainly made some interesting things so in this video I'm gonna go through Lang chain expression language which is their new update of how you can basically do things with Lang chain how they've sort of gone back I think more to the concept of chains and you can actually see what's going on a lot better than that you have been able to in the past so over the past months or so Lang chain has faced a backlash of criticism over two main things over the docs being very confusing and then over the API itself being overly complicated when it doesn't need to be like that I think that the Lang chain expression language is the key way that they're reacting to that and basically updating the API to make it much easier for people to see what's going on and be able to get a sense of what is actually going on under the hood as well as you're doing this and making it much more declarative so we will look at some different kinds of chains with this new Lang chain expression language so first off they've put out a blog post for this and they're calling it a new syntax and it really is a nice declarative way to be able to define chains so that you can actually see what's going on in a chain much easier than before so I know from reading

people's comments and talking to people that a lot of people have had issues just understanding what is actually Lang Ching doing underneath the hood for a lot of these things so certainly the new Lang chain expression language this new syntax makes it a lot easier for you to be able to see what's actually going on here so they've put the language out they've also put out a app that you can actually use to learn the language through a chat form and I think they're going to open source that which would be really cool to see so in the blog post here you can see that they're basically talking about different types of chains and different types of calls so at its fundamental language chain is really about prompts and llms and how do you organize those then how do you incorporate other tools like passing the response back from the llm doing some Vector search to find some data to put into your prompt for sending to the llm all of these are part of chains so the cool thing with this new syntax is it makes it quite easy for you to see what actually is going on in here so just if we look at this sort of really simple example you can see that you've got your model you've got your prompt and then you chain it this whole idea here of basically having the components of a chain and just piping them together so that you can see from one element goes into another element is what this is all about and it makes it much easier to do things another thing that they've updated as well is that in the past it was very confusing which chains could do batch processing which chains could do streaming could do async that kind of thing so all of that they've also updated here so that it's now quite easy for you to basically run dot invoke for just running a normal chain you can run dot batch for a batch chain it can run dot stream for a stream chain this is something that definitely makes it easier going forward as well let's jump in the code and then have a look at what they've actually added and look at some of the things that are in there all right so let's jump in and have a look at the code so I see I'm just installing some really basic stuff we've got Lang chain chromodyb that go we're using an AI in this case and you'll see that actually at the start I wanted to just quickly show that this can work with both the chat API and also the older one as well it will actually work with the open source models and stuff as well you can just use bring in so the hugging face model that that should be fine for this so okay I'm setting up a model and model 2 here we set up a prompt and this is where the magic happens so these two are quite basic sort of things that you would do normally here is where the sort of change comes and this is where we've got this sort of declarative way of defining a chain so here we're basically just saying that the chain is going to be prompt and then we've got the pipe command and we're just piping The Prompt into the model and that will give us our output and return that so you can see that my prompt was tell me an interesting fact about and then put subject in and here I can just put in the subject of zelvis and and then basically run it now here I'm doing chain dot in row I'm passing in just a

dictionary of what I want to pass in here if I wanted to do batch I could just change this to batch and then I put a list of these going in there and it would be able to work just as well for this I wouldn't need to change anything about the actual chain itself to do this so you can see here because we used a chat model it passed back an AI message with the result back so if I want to strip that out and just get a string back I can redefine the chain and basically say chain equals prompt pipe model pipe string output parser and then run that when I say run I mean invoke that and then you see that now we're just giving the string out back here if we do the exact same thing with the other models so this is the text DaVinci 3 Model you can see that we Define it exactly the same way we've got a prompt we're going into the model and then we've got the Apple put password just getting the string out and we can see we invoke it now and we get a different result back and the result back here obviously is quite different because the model is quite different itself next up the thing that we can do is we can actually add bindings to this so the bindings can be a number of different things here the sort of one simple one would be just to basically add a binding of a stop so this is kind of like where we're telling the stop token or the stop series of tokens for the model that's going in here so we can see that we've got our prompt being piped into the model then we've got this bind stop equals new line and then we've got alpaca so you see here I've changed it now to tell me three interesting facts about the subject so really what the model would do is return a list of three things and we could see that we're only getting one of them because we're basically got the stop after the new line so where it finishes the first one it then basically has a new line character and that's because we've done the bind to stop there it will actually stop there next main used of the bind is for adding in open AI functions so here's a function schema defined in the way that you can pass it into openai you can see that this is basically this is one of the examples from the language in site they're asking for a joke and the joke comes in two parts it has a setup and then it has a punchline and both of those parts are required here so the functions chain is going to be the prompt with the model.bind function call passing this in and passing in the functions there and you can see that basically once we invoke that passing in that we want something about bears we get back this result and the content in the result is actually nothing but the function call is basically our Json that we get back so we get here the arguments are going to be the setup is why don't Bears wear shoes punchline is because they have bare feet this is a way that you can basically use the open AI functions in this new format I'll probably make some a few videos about doing some fun things with this I've been playing around with the functions a lot more recently for some other tasks so this certainly makes it quite easy to basically put in the functions and invoke them on what it is that you're doing next up is output passes so if we just wanted to basically get something

back with these functions that we passed in and we won't actually get the Json out now we can basically add to this that we've got our prompt we've got our model.bind with the functions stuff and then now we're bringing out the Json output functions pass now when we run the same thing that we did before response we've basically got a dictionary or Json coming back there and if we actually go into that we can access it you know as a dictionary to get out either the setup or the punch line another one that they have is Json key output so this is similar but where you're specifying exactly what key name you want back in this case the key name is only set up for the setup not for punchline so we just get that back as a string here next up we look at retrievers and one of the key things here is that sometimes you want to pass something in and use it multiple times in the chain for example if we're doing a simple Vector store and we ask the question if we're going to ask a question in this case you'll see the question I'm asking is who is James Bond we want to basically ask that question first of the vector store so we get back the relevant information from that then we want to put that into the in context learning and pass that in with the prompt to the language model so that we can get an answer back and then in that one we also want the question as well so if you think about it the first one is we want to basically use the question to get our context then and we're going to pass that in but then we also want to pass in the question again here so this is where this runnable pass through comes in an item getter comes in so these are basically used for passing something into the chain which you're going to use multiple steps of the chain without the transformation going on there okay we're bringing in a vector store chroma we're going to use open Ai embeddings and you can see that here what we're going to do is just make a little set of fake docs about James Bond right so these are just a set of strings and we're going to embed each of these with our open AI embeddings into chroma from text there and then we're just basically setting this up as a retriever so now we've got our prompt which we know that takes in the context which takes in question and then now we're going to set up the chains the chain is first off the context so we know that the prompt needs a context and a question so here we're basically defining this context is going to be from the retriever and then the question is going to basically be something that we pass through to use later on in the prompt as well so these are both being set up here so you can see we've got this set up we can run it through with who is James Bond it's now going through it and basically using that question first for the Retriever and then once that's filled out that context gets used here and then because we've used it once we're now using it again we've got this runnable pass-through of the question being used again so the question was first used for the retriever now it's being used for the prompt in there and you can see sure enough when we run this we get James Bond as a spy who works for MI6 what does Jameson like to do based

on the given context it can be inferred that James one likes cats right that's what we had in there so you can add more to this too and here's an example that they have of adding in that we want it to answer in a specific language so now we've got three parts to the prompt so we've got our context which is going to be getting the question passing the question into the retriever that will give us back the context for this we've got our question which is going to be the question that we just passed in as a string the start or as a question in a dictionary and then we've got our language which is also going to be something we're going to pass in Via a dictionary so these item getter things are used to basically get it out of the dictionary that you pass in so you can see here we're passing in a dictionary and in that dictionary we've got the question and we've got the language so if I invoke this chain I pass in where does James Bond work language equals English I get james1 works for MI6 if I do the exact same thing with Italian I basically get it in Italian now the key thing here is that as we've gone through this first off this part has been run right where it's getting the context via using the question on the retriever so passing the question into the retriever so we could actually do this with multiple retrievers so if we had one retriever for a certain kind of documents another retriever for another kind of document we could actually you know put these two and then combine them in here and then we've got a question passing in and then our language and this item getter is just basically getting it out of the dictionary that you pass in here next one up is some tools some tools here is just a simple use of DuckDuckGo again this is not an agent kind of thing it's not a react thing what this is basically doing is that it's going to have the template you know where you're going to run it through a prompt first go in and the prompt is just asking it to rewrite it for a search engine then we basically run it through our output password to get a string back and then we pass that into the DuckDuckGo search so when we pass this first one in you can see this this is what we get back here if we didn't use the search and we just relied on the model we can get a response back as well but it's probably not as reliable or different than perhaps just doing a straight up search now I think going forward it will be interesting to take this and then take the search and pass it back into another prompt and model as well another one just quickly just to show you arbitrary functions so this is another example from the langchain docs this one they're basically taking some functions and just calculating the length of what you pass in here and turning that into a math question yeah so you can see here we're passing in a dictionary of two that values and we can see that first off this value is basically something's going to be passed to a prompt and the prompt is just going to be what is a plus b right so a is being using the item getter to get this and then it's running it through this runnable Lambda length function which is just going to calculate the length of that and then we basically here we've got a text one and a text two going on again

now we've got a runnable function of multiple length function that's going on here you can see that this one is actually using another function in there as well as you go through it and then finally it gets piped into the prompt piped into the model and then we get our output back so when we run this thing through you can see we get okay four plus 16 equals 20 in this case so that's been calculated because this is for uh long in here we've basically got 4 times 4 which gives us the 16 and then the model itself is doing the four plus 16 in there okay so this has been a few of the examples in there definitely in the next few videos I'll look at using some more real world examples of putting these into action to actually do some tasks and stuff I think definitely things like summarization some of the chat stuff all of this can now be done in in quite different ways so far it looks like this new expression language replaces a lot of the chain stuff that was built in but perhaps not agents so you might see this you know in future it gets added to basically two custom agents with this syntax but it certainly makes it much easier for us to just see what's going on in the syntax that's here anyway as always if you've got any questions please put them in the comments below if you like this video I'm certainly going to be doing a bunch more of the Lang chain expression language going forward so click like And subscribe and I will talk to you in the next video bye for now



## Building a RCI Chain for Agents with LangChain Expression Language

Okay. So one of the key things with a lot of autonomous agents that are built around using large language models is the way that they evaluate themselves. So in this video, I want to look at the concept of what an RCI chain is, and also how we can actually build one using the new LangChain expression language. and so you can get a sense of how this all comes together. So where, autonomous agents go off the rails a lot is when they're not checked. So you need to find a way to check them. And it turns out kind of amazingly that one of the best ways to check them is just to get the language model, to look at its own output. Now you can either use another language model, a bigger language model to check a smaller language model, but even, with a decent size language model, it can, it has the ability to check its

own output and we'll often see errors. Now this is to do a lot with the auto regressive nature of the language models. But it is really kind of magical in many ways. So the thing we're going to look at is this idea of an RCI chain. So this comes from this paper called language models, console, computer tasks here. and the idea also comes from another paper, sort of reflexion these two papers have a lot of sort of commonalities in them. but the one we're going to be looking at specifically is the RCI paper language models, console, computer tasks. And in that people they talk about, okay, what is an RCI chain? So, RCI stands for recursive criticism and improvement. And the idea is actually pretty simple. They show that, what you can actually do is you can start off with a zero shot prompt where you basically just ask the language model a question or something. You then take the output of that. And you then prompt the model to basically criticize the output to check the output and to see, okay, is this a good output? Is it not a good output? what needs to be changed and stuff. And then finally, you actually have an improvement prompt, which takes both the original question the, the output, the criticism, all in one prompt and then writes a new version of the prompts. So it's supposed to be an improved version of the prompt there. this is what an RCI chain is. It's basically recursive criticism and improvement. and it can be done the whole recursive thing just means it can be done multiple times. So you can actually, take the output and then check it again if you wanted to, or that kind of thing. We're going to focus on three prompts and three chains for this as we approach it. So the first one obviously is going to be our initial question. Now, it doesn't even have to be an initial question. It could be, that you get it to write an email, with your first Prompt. Then the second prompt is going to be an evaluation of the output of that, prompt based on the question or the prompt that went in as well. So we have to pass both the output of the initial question and the actual output that the language

model generated into the critique prompt with its own prompt as well. we then take the output of that. So we're where it's basically being

critiqued and said, all, this part is wrong or this part could be improved. And then finally we pass it into the improvement chain where it's going to basically taking all of these things. and, improve upon them. as it goes along, what we actually

have is something like this. So if you look at these three steps,

you've got like our initial question. And the top arrow here would be the output

of the model going into the next prompt, but we also need to pass that initial

question into the next prompt as well. That's going to be used as well. Then we get the critique out. We're going to pass that into the

improvement prompt, but we also need to pass in, the initial question

back into the improvement as well. And then finally here, we're

taking the output and that's going to be our final output. Now we could actually be recursive

and come back from the improvement and feed it back into the critique

and do a second critique for this. So you could have multiple steps. You'll find that for simple

questions that's not needed. but sometimes for things like writing

longer piece of a document or something like that, it can be useful to have

it do multiple critiques or multiple improvements as it goes through that. And that can be either done, by breaking

down the document or it can be done just feeding the whole document in if you've

got a long context window, et cetera. All right. So we need to basically

build three chains here. the initial sort of prompt and initial

question chain, The critique chain which is going to take the outputs

of the initial one in And then we're going to have the improvement chain. And then finally we're going

to have our output from this. So let's jump into the code and

have a look at how we would do this. Okay. So in this notebook, we're

going to look at building an RCI chain with a chat model. I'm using OpenAI here, but you

could use Claude-2, you could even use LLaMA-2 70 billion with

the chat set up for doing that. So just quickly showing you what I'm bringing in. Not bringing in a lot at all. We are using obviously a recent version of LangChain to make sure that we've got the LangChain expression language in there. And the first thing you're going to start off is just go through how we do multi chains with this. So you can see here, I've got two prompts. I basically set up my model. I've got two prompts. And this one is basically tell me an interesting fact about, and then I put it in a subject. And then the second prompt is like a reverse of that, based on this interesting fact, which is a chunk down from a Meta subject recover what the Meta subject is and then tries to predict this. Now you could probably improve this prompt quite a bit, if you actually wanted to do something like this. But here I'm more interested in showing, using the two prompts. So with our first prompt, we can make a chain pretty easily, we just have prompt pipe model pipe, and then our output parser. And so if I pass it in, okay, tell me an interesting fact about Elvis. It goes off and gives us a nice, interesting fact, Elvis Presley was a black belt in karate. And it tells us a bit about that. Now, what we would really love is if we wanted to do the multi-pronged thing is to just chain this, like where we could go prompt model, Output Parser, feed that into reverse prompt model and then output parser. Unfortunately, this doesn't work. Because it turns out that for the second sort of chain part or second part of this full chain, the second chain itself. We actually need to pass in as a dictionary, not just as a string. So we could actually invoke the first chain with just passing in a string. But after that, we need to pass in a dictionary. So we'll see that if we run this and I've turned on LangChain dot debug equals true here. That when we go through it, you can see it, it works fine for the first chain, but then when it comes to the second chain, we run into an error. And it tells us that it must be a mapping, not a string going in there. So that brings a question of how do we do this with a multi chain. So the way to do it is actually pretty simple. We define out two chains. And the input for this chain 2 is actually

just going to be interesting fact, right? Because we use that in the prompt, the F string prompt that we're substituting out is interesting fact. And then we just pass it in chain one. The output of chain one is going to be that interesting fact. So we've got our dictionary here, with the output of chain one being the interesting fact that gets passed into our reverse prompt, a model. And then our output parser. If you see now, if we invoke chain 2. So we actually have to run chain 2 and pass into what we would pass into chain one. It will first basically make this. And to do that it has to run chain one. So we can see that it's going through running chain one. It then gets the output of chain one the stuff about Elvis and the black belt in karate. it then passes that into chain 2. And it can run chain 2, and then it can reverse that. And you can see that it comes back saying that the subject is probably Elvis Presley's interest and proficiency in karate. So it doesn't get back to just Elvis. Now you could possibly change that by playing around with the prompt there. But I wanted to show you. That's how you do a, like a multi chain, right? When we were linking chains to chains here. so now we want to move on to the RCI the actual RCI prompt here. This is going to be a little bit more complicated because now we want to, were using a chat model. So chat models have system templates and human templates. So here, I'm going to basically start off where we're going to create this chat prompt from having a system template which is going to be, you are a helpful assistant that imparts wisdom and guides people with accurate answers. And then the human template is just going to be the question that goes in there. So you can see that if we start off with this. we basically have to combine these, the system prompt and human prompt to make our chat prompt template from messengers. But we can then feed that into our chain. No problem. So you can see here, we can

basically pass that into, chat prompt model string output parser. And so the question I'm going to use here is one of the ones from the GSM 8K logic questions. Okay. Roger has five tennis balls. He buys two more cans of tennis balls. Each can has three tennis balls. How many tennis balls does he have? So it's basically should be five plus two times three. And so it's going to be 11. Now the initial answer out here actually gets it. So it basically does that calculation and it gets it right here. A lot of the open source models, won't get this right. This is one of the simple tests that I give a lot of the open source models. So let's pretend that it got it wrong. So I'm actually making a fake initial answer here. Where we were going to say, okay, Roger initially has five tennis balls. Each can has three tennis balls. He bought two cans. but when I put it into the math, I'm just going to make it five plus four equals nine. So it's clearly the wrong answer. so now we've got our critique prompts. So this is the first of our three chains here. We've got our first of three chains. So now we come to the second chain where we've got the critique chain. So this also has a system message and has A human message in here. And you can see in the human prompt here, we're passing in the question. We're passing in the initial answer. Now I'm actually going to pass in the fake initial answer for this. So we can see if it actually critiques it and works out that it's wrong. Okay. So I've got my RCI prompts. So this is the critique prompt in here. And this is, the one that we could run recursively. If we wanted to. here you can see I'm basically invoking this. and so I've got my initial question. And I've got this fake answer. And sure enough, you can see that the criticism that comes back is the problem with a given answer is incorrectly calculates the number of tennis balls roger has now. The answer has five tennis balls and then adds 4. And so it basically tells us that, okay, it's got it wrong. And really what it should have done is it should have had two times three, five plus two times three equals 11. Therefore the correct answer is 11. So now

that's our

constructive criticism there. So that gets passed into our third prompt. So you can see here again, we've got a system prompt, we've got the human prompt now in the human prompt, we're passing in the question, the initial answer and the constructive cause criticism here. So you can see when I basically make this again, this is just a simple chain. When I make this, I've basically got my question, my initial answer, where I was practicing in the fake initial answer. And then I've got the constructive criticism. And this chain is able to then take

all those things and rewrite it. Okay rewrites it, roger

initially has five tennis balls. And it goes through and eventually gives

us the right answer of 11 tennis balls. Actually gives us what we

had at the start there. Now remember you don't have

to use this for a question. You could use this for writing

an email for doing a whole bunch of different things. The idea is just that you're getting the

critique on what the language model did And then revision of improvement for that. Now we've got three separate chains. So this is okay if we were going

to write like some kind of function where we did this, but really

we wanted in a combined chain. So to put it into a combined chain

we've got our three chains here and I've actually remade the chains. And we're actually going to need

to pass some variables around. So before I was doing it in an open

scope, it was easy to pass them around. Now to basically pass them around

I'm going to use this itemgetter. So when we pass in the question in

here, Anytime we're going to use the question later on we need to basically

have this itemgetter question. So that when that's going to be used

for the first chain, but in the second chain and in the third chain it's

going to be, the item getter question. Now let's look at how

we actually, do this. So you can see here, obviously the chain

one is going to produce a result out. That's going to become the initial answer

for the chain 2 that's going to produce result out, which is going to become the constructive criticism for chain three. If you can think about this, you think about it backwards that okay, chain three is going to need both chain one done but it's also gonna need chain 2 done. So actually when it does chain 2 it already gets chain one done there. And then, because we're passing this around, we're using the itemgetter for the question here that we've got going on. Actually the other thing I could probably do is actually change this chain one to be the itemgetter initial answer in here. something I'll experiment around with. But by putting it together like this,

we've basically got the three chains going on and we can now just invoke the last chain, give it something to do in this case, I'm saying the question right in an SMS message to say I'm tired. It will basically go and write that. I've turned on the LangChain debugging and just try and get again. Now you don't have to stick to the prompts that I've used here. So here for example, it's writing an SMS message. This is a pretty long SMS message, right? What I might want to do is change the critique prompt to say that the answer shouldn't be too long. So then it could be used to actually shorten an answer. And just include the key points kind of thing. In this case though We're basically running it, it goes through, it writes how I so we can see here, we're basically going to get our response out. So we're here, right? Hey, [ recipient name] just wanted to let you know. So that's what we've got there. We then basically want to feed that into the critique chain. So we've got the second chain going on. And we'll see the output of that chain come back and say, there are no apparent problems with the given answer. It accurately conveys the message of being tired And the need for rest, that we've got going on there. And so then the final one basically can just rewrite it. You can play around with these prompts. And this is what lot of good agents do is they have these sort of recursive prompts that look at outputs and decide, okay, are they meeting a set of criteria? So rather than just looking for one criteria in that critique prompt, you could list out that,

okay, is it as concise as possible? Is it as precise as possible? All the different sorts of things that you might want to use. If you are building an agent to do a specific task, to write something to human or, write an email to a human or some thing you might want to check that is it in a friendly manner. Is it all these things that the critique could do And improve on the original generation in there. So that's something that where these things really tend to shine for this kind of thing. Anyway you've seen it finally comes out with the output which i think is pretty similar to the original one if not exactly the same To what went in this case. But this is definitely a skill that you should be using if you're building any sort of agents for this. Anyway, if you've got any questions please put them in the comments below. As always if you found this useful i'm actually going to do a whole series about agents. It's something i've been working on coming up. So click and subscribe if you would like to see more videos about that kind of topic. And i will talk to you in the next video. Bye for now



## How to Run LLaMA-2-70B on the Together AI

all right in this video I want to look at using the Llama 2 70 billion model again with an API so the service provider that I've chosen this time is together AI this is not sponsored by them I'm not receiving any funds or anything from them but looking at their offering I think it's quite interesting certainly a lot cheaper than some of the other things that we've looked at and I think for many people this is going to be the only way that you can run one of these Llama 270 billion models in Full Resolution yes you can probably run it with a 4-bit model or something like that but for most people they're just not going to be able to spin up for a100s or multiple a100s to run this model and try it out with a decent token speed so together AI is a new startup it's actually done a bunch of cool things already in relation to research so if we look at the team behind this a lot of them are academics from Stanford from Mila from eth and you can see some quite famous names like personally who's been the supervisor on things like the generative agents paper and a number of other key papers recently so the team certainly knows what they're doing both with research and I think also with you know putting these things into production for people as well and they've basically released an API that you can use to access not only

llama 2 but a number of the different models so let's take a look at that so here is what's called their playground so I'm looking at the the different models that they've got here and you'll see that they've got you know the sort of standard llama models they've also got some of their own fine tunes including number of different interesting models they've built a lot of the original llama one models the fine tunes like alpaca vacuna koala those sorts of things that you could access but if we look at it here the one that we're interested in is the Llama to in this case the chat model here now I only started using this over the past few days and it's very interesting that the price seems to have gone down so when I was actually testing this out you were paying a very small fee maybe like 15 cents an hour for the hosting of the model and then something like you know 1.5 cents per thousand tokens on this one and obviously a lot cheaper for the 7B model the 13 model B model Etc I'm not sure if this is a UI error that's showing zero zero at the moment it certainly wasn't like that yesterday when I checked this out but maybe they just you're getting enough volume now that they can just make their money off serving of charging people for tokens so we're going to use this just to show you quickly once you're making an account in here you'll have access to your profile to your API keys and enter billing now one of the cool things that reasons why I chose this is that they're giving quite a number of credits for you to try out their service so they're giving 5 000 credits I've done a bunch of testing with the Llama the 70b model and I've still got lots of credits left so this is cool for checking out here they've also got a bunch of documentation about the apis and about the different models but we can see that there's a whole bunch of different models that you can pick from for doing inference they also support fine tuning here they're fine-tuning offering is interesting in that it's done basically by your data set they calculate how much to charge by how many epochs of that data set you want to train on a specific model in here so anyway let's jump in and have a look at using the llama2 chat model here we can just start the model up like this by clicking that and you'll find that if we want to go in we can actually just use it like a playground as well in here where I can basically just ask some simple questions and you will see that the speed of tokens coming back is actually quite fast okay so I'm going to ask it what's the difference between llamas alpacas bikunas and you'll see that the speed of this is actually very quick that we're getting back here so this is using the full 70b model on four h-100s for this and we're paying per token here the cool thing is that not only can we use it like this we can actually jump in and use it with the API and with code so let's do that now okay so in this notebook we're going to be using a llama 270b on the together API so you'll see that I'm bringing in Lang chain I actually don't even need hugging Place Hub or any of the others but I do need to bring in together that's one of the key things there and you will need to get your API key and stick it in here setting this up is

pretty simple we're just going to set up our API key from this we can start to look at the actual models in here so you'll see that they have a lot of models available but we can see that the original llama models are available in there but we can see also their red pajama fine tuning are also available in there and the one that we want obviously is the Llama to 70 billion chat model to basically start this up we can either start it up manually from the actual UI we can also use the together model start and pass in the string that we actually want to start up for the model so you can see I'm doing that then here now unfortunately link chain doesn't support the together API out of the box so I'm not sure why this is the case my guess is it will come pretty soon but for now what I've done is just write a little class that will basically use an API as an llm for Lang Chang in here and we can see that with the model that we're going to use and we could pass in a different model if we were using a different model but in this case I'm just going to set it up to be default to be the Llama 270 billion chat model you can see that our API key is going to go in there temperature we can pass these in and change them Max tokens Etc so once we've got this set up we'll be able to call it and get the responses back so here you can see I'm just doing an instantiation of the model I'm passing in actually what the model is I'm passing in a temperature I'm passing in the max tokens now in my testing I found that it seems like it does error out sometimes if you set the temperature to be only zero so I've gone for 0.1 here in this but we can basically just check and see how that's set up and then we can just use this like we would normally so here I'm just asking a simple question and we can see sure enough we're getting an answer back okay so now with Lang chain we know that llama needs a very special prompt in the way that it's set up I've basically got that in here this full prompt that they use now you could come and play with this and you'll certainly see later on I also play with it but that gives us the default it allows us to set this up you know quite easily okay just showing you normally if we were using the 7B or even if we were using the 70b chat model with save four a100s something like that we would probably be using it with the hugging face pipeline in here but rather than that we're actually just going to use the together llm class that we made before pass in the details and now we've got our llm to pass into our chain as we go through so the first off I'm just going to be doing is a translation and you can see just the prompt sort of just shows you okay we've got the instruction our system prompt being you're an advanced assistant that excels at translation and then we've got the instruction convert the following text from English to French and then we're going to pass in the text they're setting up our nlm Gene it's just simple like this and then we can basically just go in and pass in text of how are you today and it's going to give us a translation back here okay same thing for summarization we're going to want a system prompt that you're an expert at summarization and expressing key

ideas succinctly and we're going to pass all of that in and in this case I'm going to pass in the Twitter article basically about new CEO at Twitter or x.com as it is now and you can see that the word count for that was 940 words with the 70b chat model it's down to 98 words and we've got our summarization going on yeah I'm not focused too much about the quality of these I would suggest that you yourself go and have a play with them see what they can do and see the different results for this but you can see that it's certainly doing summarization in here using that same 70b chat model we would now want to set up a simple chat bot up for this and so to do this we're just going to have our LM chain but the big difference now is we're going to have a conversational buffer memory in here so I've got my instruction set up my system prompt setup in here so you can see what is going on into this and then you can play around with using human using Ai and these things I actually found it changed from when I did it a few days ago to now so I've changed the system Pro to actually be assistant here rather than AI but I suggest you experiment yourself see how it goes our print template is going to have a user input which is what we're saying to the chatbot it's also going to have the chatbot history which is our memory you can see we're defining the memory here as this conversational buffer memory and then the memory key is going to be chat history that we've got going in okay once I set all this up I've got verbose equals true just so we can see what's going on we're passing our llm passing our memory here and you can see I can start off by saying hi my name is Sam you see we're going to get a response hello Sam it's nice to meet you is there something I can help you with now we could change that the system bot to basically have more personality or to respond in certain ways but really what I wanted to test here was using the memory can it respond to the memory how does it react with that so you can see okay first off I give it my name and I'm asking it can you tell me about yourself sure I'm just an AI I don't have a personal life or experiences like humans do and then it gives us you know a bit of information and then what I wanted to do was basically Let's test out its memory so I'm going to tell it okay today is Friday what number day of the week is that and see we get back sure I can help with that today is indeed Friday which is the fifth day of the week now I'm so interested in how we whether we count from Sunday whether we count from Monday what I want to test more is it gonna remember that today is Friday so you can see I've asked it that and then the next one I ask it okay what day is today and sure enough it does remember so it's able to say sure I can help with that today is Friday then again also ask it okay what is my name and it's sure enough it's remembering that your name is Sam so this is a good sign for a model that it responds to the memory it gives us the correct answer many open source models won't do this they will get confused they will give you a default answer something like that next up I ask it about can you tell me about the

Olympics we get a quite detailed answer about the Olympics including is there anything specific you would like to know about the Olympics I'm not really interested in that so much I now wanted to do some summarization so I asked you okay what have we talked about in this chat and you can see that as we're looking at this we're getting the chat history being passed in here so it is this is what's giving it the memory and going into the context that we've got sure enough when we ask it to summarize the chat we get back sure I can help with that in this chat you've talked about one your name which is Sam two the day of the week which is Friday three the number of the day which is five and the Olympics and it's giving a brief overview is there anything else you'd like to know or discuss so it's done a good job of being able to do this now I would suggest you if you want play around with say the summary memory or some of the other kinds of memory in here because you'll probably find that the 70b model can actually do some of these things where with the other open source models and the much smaller models are going to struggle at some of this finally just turn this off we just go together dot models dot stop and pass in the model and then we're no longer being built for this hopefully this gives you a simple example of spinning up a 70b model the token speed of this is actually very good that you're going to get back because they're using four h100s to serve it as you use it and while it's not totally local on your computer you actually now do have access to a very big model that you can use rather than just having to spin it up with your own gpus or use it like as a 4-bit model which still would probably be reasonably big for this so anyway as always if you've got any questions please put them in the comments below if you like the video please click and subscribe I'm going to do a few more showing perhaps some other service providers and then we're also going to look at serving these models locally and what you can do with some of the smaller llama 2 models as well bye for now



## RetrievalQA with LLaMA 2 70b & Chroma DB

Okay. So in this video, we're going to

follow up using the LLaMA-2 70 billion model for some retrieval QA. So this is just the basic,

chat model that I'm using here. So it is fine tuned for instruct use,

you couldn't see here that I'm using the Together Compute one that we looked

at in the last video of how to set this up really the way you're using the

model, whether you're using API, whether you're loading it yourself with

four GPUs et cetera, really shouldn't make a difference to this, right? The way I've set it up is here

I'm using the together API. So I'm going to have to make

a LangChain LLM here, which I basically just set up there. And then I'm just going to

bring in a zip file with a bunch of different papers in it. So the papers, I think are flash

attention paper, LLaMA-2 paper, tool former and the react paper, maybe a

couple of others on and augmenting LLMs in there .

the idea here is that we're going

to basically just bring these in. We're going to , you know, have multiple PDF files. We're going to stick them into chroma DB. So we're not having to

put this in the cloud. We're not having to do anything like that. this is, Fully open-source here that we're using the model that we're allowed to use. We're using chroma locally. I guess the one thing that you know here is that this is probably not technically a local model in that it's

where you pinging it from the cloud. but you could be running this locally if you've got enough GPU's to do this here. So, then we're basically just

going to bring it into a LangChain. We're bringing in chroma I'm I'm bringing the directory loader for this. the embeddings that I'm using here

are the instructor embeddings. Now there are some new embeddings

that have come out recently, which are supposedly getting better scores in these. I'll probably do a video about looking at some of those. I still think the instructor

embeddings are a really good way to just go for a really solid embedding. And you'll see that in this, they really seem to work well too, in relation to the questions and

finding the right context back. So then I'm just going to basically

ingest in these PDF files. That's what we're doing here. We're using the PyPDF loader to basically

bring them in, that's split them up into pages, et cetera and chunks and then we've

got the, we've got these documents now. Now the documents, I'm going to

split with a character splitter. I'm not a fan of the character splitter. you know, really, I think there are a

lot more intelligent ways of splitting texts than doing it like this. the crazy thing though, is it

still pretty much works I do like having a chunk overlap. so that if you've got one idea between two

chunks of text, you want it to basically be overlapped so that you can actually

get that in one full chunk by itself. And then we basically just set up these, hugging face instructor embedding. these are the, embeddings from

the Hong Kong University's NLP team. I'm using instructor X L here. So it actually is a little bit slower on

the T4 GPU for running this as you'll see if you go through and run this yourself. so Once we've got our embeddings our embedding maker, you know, embedding model loaded. we can basically just go through and make the chroma DB here using, these instructor embeddings. And so this will take a bit of time because it's going to go through the 200 documents and they've now been split into chunks of a thousand characters. And it's going to go through and embed each of these, into this. once this is run, you will see on your drive here, you will see that you've got your database. They're all locally for this. So next up is to build our retriever. So we're going to just be using a vector store r here. we're going to have the search arguments to be k = 5, meaning we're returning back five contexts from this. And what I'm going to do is actually have a little citation that basically tells us which PDF file it came from, in there as well. So that's something that I know a lot of people want to know, okay, where did these, this answer come from in, the chunks of information there. and so while we're not being specific of showing quotes and stuff like that here we are showing that, oh, it came from this PDF or from a combination of these PDFs. Alright, I basically then just instantiate my LLaMA-2 70 billion chat model here. I've got the temperature set to 0.1. I've got max tokens set to 10 24 in here. I then basically just assemble the retrieval chain here . So here I'm just basically passing in the LLM. we're just going to stuff, everything in. we shouldn't have any problems fitting into the 4,096 context window of LLaMA-2 here. So we're just going to use stuff. the retriever is our Chroma DB, which is going to basically return five contexts from the actual, answers that we've got them. And then I've got a few little sort of helper functions. just so that we can see. We're returning source documents equals true here. And we're basically just using that to work out, okay, where did the PDF come from in here So once we've got that done we're ready to actually go through this quite simply. So here you can see, I start off with just

asking it, okay, what is flash attention? and, this is actually the flash attention one paper not the the newer flash attention 2 paper that we've got in here. and you can see that, okay, we've got this, what is flash attention? Flash attention is new attention algorithm that computes exact attention with far fewer memory accesses. So it's giving us a bunch of information about it it's also giving us the sources. So remember we preset key equals five in this example, and all five contexts have come back from the flash attention PDF. So that shows that our embeddings are looking up quite nicely from taking this question, embedding it, and looking for the most similar examples in our chroma database there. We can see that doing this, there's a follow-up question about IO aware mean from flash attention. here, it's basically able to get that as well. What if we ask it something about LLaMA-2. So here you can see ask it, okay, what is the context window of LLaMA-2? And i Comes back the context when you have LLaMA-2 is 4096 tokens, which is correct. Now it's interesting that a number of times, it gets some of these sort of weird things where polite answer, I don't know. But it's actually given us the answer already in here. Again, we can see that our embeddings have worked well for this one and this one. In that we've asked about LLaMA-2 we've got LLaMA-2 back how many tokens was LLaMA-2 trained on? LLaMA-2 was trained on 2 trillion tokens of data. what about, you know, asking it something that it doesn't know. When is LLaMA-3 coming? And this is where I think, that this model is pretty good, is that we see, I don't know, the paper only discusses LLaMA-2 and its variants. There's no mention of LLaMA-3. This will have returned things that have been closest to this kind of query . But none of these would have actually mentioned LLaMA-3. So the language models had to basically go through it itself. And look for, okay, what, you know, is there a mention of LLaMA-3 in here? Is the relevant information in here . Now some of the danger with smaller or less well-trained, open-source models is that you will see with this kind of thing, that it will just make up an answer because it uses something from in here. So it's quite good that

it hasn't done that here. What about if we try and confuse it even more? And I say to it, okay, what is the new model from Meta called? Now here, this is a trick question because, it could be LLaMA-2, which is what the answer gives us, but it also could be tool former because tool former also came out of Meta And remember, there's no dates on these papers. I don't think, where it's basically using that to make the decision. So we can see that the sources that's getting it's get 2, from LLaMA-2 to from augmenting LLMs. and another one from the LLaMA-2 one as well. so, the augmenting LLM survey is basically mentioning a lot of different models. So my guess is that's why it gets some of the things out of there. but in this case, it does come back with that, okay, it thinks that, the new model is called LLaMA-2. but it also then talks about the safety reward model which was used for the RLHF in LLaMA-2 chat model and, or the helpfulness world ward model again, these are kind of interesting that, they get returned in there. It's a little bit of an unfair question, but it's good to test out your system and see like how it will respond to things like this. next up, what is tool former straightaway it's able to get what to form it is. it basically, it gives us okay Definition of I wouldn't say It's a great definition of what tool former is. , what tools can be used with tool former okay search engines, calculators, translation systems, via api calls. how many examples do we need to provide for each tool? So this is interesting because this certainly is something that you know Would it be mentioned both in tool former and the survey paper that covers tool former and other papers like that . So we see some things about that And what about react We've got the react paper in there. Sure enough react is a novel prompt based paradigm that combines reasoning and acting i language models for general tasks solving That's a pretty nice definition that it's given us there for looking at this. So on the whole i think this

model has done pretty good. Now if you were going to use this in the real world, You probably wouldn't just go with this model straight out of the box. You would go for a fine tuned version that was going to be better on your particular types of retrieval questions and you know what you were going for like that .

So From testing this i can see

that okay this has got potential. and Then i would look at okay are there some fine tunes out there tha focus more on the rag or retrieval augmented generation tasks. Or i would look at making one myself and perhaps that's something we can look at in a future video here. finally because i'm using the api in this case i just basically stop it so that i'm not going to be charged anything else for this If you're using your four gpu's you probably also want to stop them to not get a big bill for that unless you're running them locally I guess. Anyway so this sort of just shows you the strength of of LLaMA-2, for doing a retrieval QA And for doing RAG or retrieval augmented generation. It's definitely something that can be done with this for the best results that you will want to fine tune for your specific task for this .

All right. as always if you've got any questions please Put them in the comments Below. If you found the video useful please click like and subscribe i will talk to you in the nex video. Bye for now.



## How to use BGE Embeddings for LangChain and RAG

so this is the massive text embedding Benchmark leaderboard that we've got here so this is hosted by hugging face it basically benchmarks all the different embedding models from here so we can see that one of my favorites for a long time has been the instructor XL embeddings in here but we also have some of the open AI embeddings in here so just quickly I try to do a timeout and tell people why I think you probably shouldn't be using open AI embeddings so there's a lot of confusion up there about embeddings and embedding models you certainly don't need to use the same embedding model provider so if you're using open AI gbg4 or gpg 3.5 Etc that doesn't mean you have to use open AI embeddings in the same vein there are some projects out there that are taking some of the Llama models and trying to turn them into embedding models that probably doesn't make sense for most projects the embedding models are trained to be embedding models meaning that they've while they've had some you know they've had a little pre-training they're also usually tuned in a way so that they can take text and find the similarities Etc so you don't want to just go out there and look for like I saw recently you know a bunch of people talking about a llama 2 embedding model that's probably not

going to be the key thing so why do I recommend that you don't use opener so first off if you look at the open AI model down here it's way behind the open source models now things have moved on but that's not the main reason that's certainly one good reason is that you know you can get other models now including some of these models that are very small like these GTE base the the BGE small Etc these are small enough that you can probably run them on a CPU as well so you don't even need to worry too much about going hardcore for a GPU Etc one of the big reasons though that you don't want to use this model is that you don't want lock in to a certain provider so all the other models here are open source you can download them you can use them on any platform you can use them on any hardware Etc the moment that you're using the open Ai embeddings and you basically embed a large Corpus of data that you're going to use in the future you're going to find that's going to lock you in to having to use those going in the future otherwise you're going to have to start from scratch and reimburse did everything again now of course that's something that you can do it's going to be costly if you're having to pay for all these embeddings as well another issue is that at some point probably not that too far off in the the future opening AI is going to deprecate this model because they're for sure coming up with better models as well as everyone else is coming up with better models and when they deprecate this model there'll be a certain time overlap where you can probably use both or something but again you're going to have to re-embed everything from scratch so I do think that the open AI embeddings can be useful if you want to just quickly test something out you just want to use an API you want to Ping something okay sure in that case use it for testing use it for trying out an idea Etc but for long term if you're building a major project I think you really want to go with open source embeddings all right let's get back to the BGE embeddings so just as of the past few days we have a new leader on this leaderboard and these are the BGE embeddings so in this video I'm going to go through looking at the BGE embeddings and using them with Lang chain to do some retrieval QA with a Chromo Vector store and we're going to look at putting all this together so let's jump into it what are the BGE embeddings I hear you say well these come out of the Beijing Academy of AI and they're basically a new set of embedding models that they have released and you can see here looking at their GitHub we connect we can actually see that these embeddings they've got both an English embedding and a Chinese embedding but the other interesting thing is they're also working on a multilingual embedding which is not out quite just yet so these have skyrocketed to the mteb leaderboard just in the past few days one of the things that I love about these is the size so the instructor XL embeddings that I've been using up until now just under five gigabytes for the size and you can see that these are now no longer near the top of the leaderboard the BGE ones that we've got

not only are scoring much better but they're tiny in comparison so the one I'm going to use today here is going to be the BGE base English model which is about a tenth the size of the actual one that we were using with the instructor Excel now you could go for the large model which is just over a giga byte and the embedding dimensions of that are actually bigger as well you're probably going to find that for a lot of things that the 768 Dimension embedding is quite standard and this is what I'm going to use with chroma and with Lang chain in this to look at how does it perform compared to those instructor XL embeddings that we were using before which to refresh what I covered in the last video we're basically using the Llama 270 billion model hosted on the together API I'm not going to change any of that bit that we were using the embeddings there before was we were using these hanging face instructor embeddings here so this is just the hugging face implementation of these embeddings and like I said these embeddings have been my favorite for a long time and the only challenge with it is that this is a five gigabyte model and when we come to actually make the embeddings with the database this actually takes quite a bit of time on a T4 right you're sitting there waiting for a while because we've got about a thousand pieces of text that we're actually making embeddings for and so sitting there waiting for this takes quite a long time if we compare this to the new one that I've done today here the code is basically the same for all of this the only thing that I've changed in here is actually the embeddings right so rather than but using the HF instructor Excel embeddings we're using these BGE embeddings now like I mentioned earlier on the model that I'm going to use is this BGE base en right for English we're going to basically use cosine similarities so we're just going to normalize the embeddings as true and then we can just set it up like this and then when we come to using it with chroma DB we can actually just drop it in to the embedding like this similar to what we did before with not a huge amount of difference in the code here now where it was taking quite a few minutes to process with the instructor XL you can see this whole thing was done in sort of 35 seconds for a thousand embeddings and we then basically make our retriever just like before rest of the code is just like before we've still got the sources we're going to get basically return Source documents equals true and we can look at the outputs here we see that sure enough really the main thing is looking down here sure enough it's finding the right outputs for each of these now it is interesting that the language model itself is actually writing some other stuff as well so I might look at making a video one of the things you can do with this is do a second pass into the language model to clean up the output so really probably we'd want just this maybe with some of the explanation we don't want the incorrect answer in there for many of the others it's okay for this one again the language model is providing a bit of a strange output in that we're getting the correct answer first up where

it's basically telling us it's been expanded from 2048 to 4096 tokens but then we're getting this kind of weird stuff that's going on here so this is what we'd want to clean up if we're doing this with the other ones you find most of them are pretty much fine how many tokens was llama 2 trained on strained on two trillion tokens all of that it is fine the sort of trick questions around when is llama three coming it's interesting here that basically says there is no official announcement from Meta regarding the release of llama 3 current version of llama 2 was released July 2023. it's interesting that it's giving us a few other answers now I'm guessing this is due to the actual system prompt and stuff that we're using in here that we're just using the basic QA retrieval so we could optimize that for llama 2 and use their system prop more and you know putting that together over going through this going through the others you're going to find answers similar outputs similar to what we saw before so the big win here is not necessarily that it's you know hugely better quality we could go to the bigger model and perhaps get better quality retrieval but actually we didn't have bad quality before and this is something that you will see for a lot of these things the huge win here is that rather than be five gigabytes that this is only 438 Meg for this actual model so it basically means our influence time is going to be quicker the amount of ram that we need is less for vram ETC on the GPU you could probably even do the inference on this with a CPU and maybe the time is not going to be totally uh awful for that anyway so these are the new general embeddings the BGE embeddings from the Beijing Academy of AI you should certainly check them out and if you're interested in biddings for multilingual things I would keep a very strong watch on when they released their multilingual models because my guess is that you might see a nice bump in multilingual models you might see a nice bump on the results for outputs for the multilingual models going forward anyway check these out as always if you have any questions put them in the comments below if you find this kind of video useful please click and subscribe and I will talk to you in the next video bye for now



## How to use Custom Prompts for RetrievalQA on LLaMA-2 7B

Okay. In the last video we looked at adding the

BGE embeddings to our retrieval QA, the rag model that we've been working with. Now, up until now, we've been looking at

using the LLaMA-2 70 billion model here. You can see that adding the BGE

embeddings still meant we got good quality output, but we're definitely

getting, some artifacts on some of these things where we're getting the

right answer, but then we're also getting incorrect answers after this. and so in this video, I

want to do two things. I want to look at one, can

we go for a smaller model? And then after that, can we actually,

you know, Improve this so that we don't get things like this. So here we can see that we've got

the context window for LLaMA-2 has been expanded, and this is

the correct answer, but then we've got all this weird stuff after it. So it's giving us the correct dancer and then going on with this. so first off, let's see, okay, how well does this sort of notebook work if we just changed the model to say the 13 billion. All right. So here's LLaMA-2 13 billion model. and you can see that all, I'm still using Together API for now just cause it's nice and quick for us to test. I'm still using the same class for instantiating the model. And you'll see, this is set to the 70 billion. If we had it by default. But we're going to instantiate it with the 13 billion in here. So everything is the same in here. We've got the BGE embeddings going on. We're bringing in, you know, the different stuff we're making our chain. and we're getting out pretty decent answers with the 13 billion. we can see that not all of them are totally correct though. So the context window for LLaMA-2, it's just saying that it's 40% larger than the pre-training Corpus. So it's missed out on getting that one, right. we can see that it gets that it was trained on 2 trillion tokens. we can see it gives us a nice answer for, you know, LLaMA 3 is not out although it says it's not coming, which is funny. and then some of the other ones it's doing pretty good job. And actually, you know, it doesn't have as bad sort of add on outputs to this. though, there are some of them there when you play with this going through it. What about if we try with the 7 billion model? Okay. So this is now using the 7 billion model. and you'll see that, You know, again, everything exactly the same. The only thing we've changed in here is that we're moving to the LLaMA-2 7B-Chat, in here. And we can see that, okay, back to getting like the correct answer, but then we've got like unhelpful answer and some other stuff in there. I'm not sure why it wants to continually give us a lot of unhelpful answer things in there. And then we can see that, okay, this answer is clearly wrong. What is the context window for LLaMA-2. and it basically says it's 50% of the safety day to use. So at this stage, we know that the context

that getting back from the BG embeddings are the same as what we were getting for the 70 billion for the 30 billion in here. So the problem is probably not in the context that we're getting back. It's probably going to be in the language model taking and using those here. Now we can see it still gets a lot of the answers right. although we do have a lot of this note, this answer is based on information. We've got lots of extra stuff in there that would probably, don't want in there. So the next way to fix this is to look at our QA chain and work on the prompt in there. So in the next part, we're going to stick to the 7 billion parameter LLaMA-2 model, but we're going to play with the prompt, and change the prompt. So let's jump into that. Okay. So here we are, again, we've got everything just set to the LLaMA-2 7 billion Chat model in there. We've got our BGE embeddings in there like before. We're setting up the chroma DB, like before where we're making our retriever return, five contexts back. And this is where we start to change it. So here is the example of the default LLaMA-2 prompt style. Remember, if you go back to some of the videos where I talked a lot about LLaMA-2, we have both a system prompt and an instruct prompt in here. So this is their default system prompt. You are helpful, respectful and honest assistant. Always answer as helpfully as possible while being safe, your answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. please ensure that your answers are socially unbiased and positive in nature. So this is heavily slanting it, you can see even here and then we've got some nice stuff about, if a question does not make sense or is factually is not factually coherent explain why instead of answering something not correct. If you don't know the answer, please don't share false information. So I like the last part of this. But we don't want it to be perhaps as politically correct or in a way, getting confused by trying to remove all the toxic and dangerous stuff. For some things, you're going to

want some of these things in here. So I'm not saying that you should always take them out. but they themselves are going to have an impact on the answers that you get back. So you really want to craft this system prompt. So the one that I've gone for, and this is just after playing with it a few times of trying out different things, seeing what worked. the one that I've gone for here is something that I'm trying to stick as close to the default as possible. But I want it to use only information from our context, and I want it to basically stick to that context and not give us any of the extra stuff out that we were getting a lot with the seven B model and the 70 B model and a little bit with the 13 billion model as well. So the prompt them go for the system prompt is you are helpful, respectful and honest assistant. Always answer as helpfully as possible using the context text provided right. we're going to, we're going to flag the context text we put in there. Your answers should only answer the question once. So don't answer it twice, right? And not have any text after the answer is done. So that's the key, you know, this is my way of trying to stop it from just going on with things later on if a question does not make any sense or is not factually coherent, explain why instead of answering something not correct. If you don't know the answer to the question, please don't share false information. So that one I've kept exactly the same as what Meta had. Now in the instruction I'm going to pass in context two new lines and then pass in the context in there. And then I'm going to have question and then the question. So you can see that our final prompt comes out. Something like this, where it's formatted for the instruction prompt with the system prompt in there. We've got our context in there. We've got our question in there. All right. I just instantiate the LLaMA-2 model just like we did before. At this case, I'm not trying to change temperature. I'm not trying to do anything else

fancy at first, we're just trying to get this prompt to be pretty solid. and now I basically need to instantiate, you know, the prompt template. So this is a prompt template that we've had up there. It's just regenerating it, passing into the prompt template here. The two things that we're going to swap out as input variables will be context and question. Now for generating the actual prompt template, I'm going to basically use our function, get prompt, to pass in our instruction template our system template in there. and then from these, we're going to pass in 2 input variables to these. So this is going to be the context and it's going to be the question. We're then going to basically, so I'm calling this the LLaMA prompt, right? I could call it the LLaMA rag, prompt, or something in here. and then I'm going to basically say you chain type. and because this is going to, it's going to be extra arguments to get passed into our chain. And in this case, we're going to pass in the prompt in here. So you can see here now when we generate our chain, we're passing in our LLM, which we know is going to be the LLaMA-2, Model. So it's our TogetherLLM in there. we've got our retriever, which is going to pass back five, you know, examples each time. Let's just, you know, go over that. and then we're passing in these other arguments in here, which is really where we passing in our prompt in there. I'm still going to do the return source documents equals true so that we can cite our sources for this. And then the rest is basically the same, right? So I've just got, you know, the things for helping cite the sources. And then now when I run it, you can see now, okay, what is flash attention? So it gives us the answer for flesh attention. we can see that we're getting all this is, you know, going to be the same cause we still using the same BGE embeddings, et cetera. But you'll notice now we're not getting anything extra coming out of this. So we don't have answers after this. Now, remember the context

window is one of the ones that it kept getting wrong before. Even though we were confident that the answer was in the context that we were passing in because we know it worked for the 70 billion, but it didn't work for the 13. It didn't work for the 7 billion But here we can see it's working. The context window for LLaMA-2 is 4,096. Now it's interesting that it's not giving us any of the other junk after it, because our prompt is saying, only give us one answer. Once you finished with the answer. Basically don't say anything else, right? how many tokens was LLaMA-2 trained on? 2 trillion tokens. Again, we don't have any of the unhelpful answer or any of the other junk that we had coming out after this. Okay. So when LLaMA-3 coming out? The answer to your question is not available as the texts you provided does, and you see when it says this, it shows us that it's using those contexts, Does not mention any information about the release date of LLaMA-3. The text only provides information about the current version of LLaMA, which is LLaMA-2, and its performance on various benchmarks. this is definitely a much, More sort of succinct, truthful answer rather than trying to make up something. when we try to confuse it about, okay, what is the new model from Meta called? In this case, it's pulled all LLaMA-2 stuff. So we would expect it to say LLaMA-2. Which is what it does. it is interesting that it's, I need to probably check the paper again. It's not doing the two capital L's and the capital MA here. So I'm not sure even the paper do they actually just go default back to referring to it like this? Possibly. All right then again Tool Former, we've got a good answer. Again nothing, else coming out in here. And then the interesting thing that I think that this prompt helps it do too, is that if it wants to pull sort of parts of the answer from multiple contexts, it can do that as well. So you can see here, how many examples do we need to provide for each tool? So it starts off the number of examples required for each tool depends on various factors, Including complexity, of the tool. So this is a good answer. And then were sort of getting, You know, because we got some interesting stuff back from Tool Former, from LLaMA-2, from

augmenting LLMs, it's incorporating all of this into, you know, this answer in here. no, and that's, you know, maybe it's not the perfect answer. And maybe the 70b would do certainly a lot better. so, you know, you could go back and try the 70 billion with this new prompt and see, how it does as well. I do think that with this answer, we've definitely gone to a better quality answer with the retrieval, augmentation for LLMs. Here, we're getting, you know, much more stuff that the augmented LLMs is a survey paper, meaning that's covers lots of different types of things. it's pulled back a few different types in here, and we're seeing that we're getting that sort of covered. We've got the dense and sparse retrievers. We've got external knowledge sources. There's a bunch of different things in here which are kind of useful for this. finally, our, what is react thing? Again, just notice that we don't have any of the, you know, and unhemmed helpful answer any of that stuff out. So just by, you know, By playing around and fine tuning this prompt or tuning this prompt. And when I say tuning, Here I'm not talking about doing anything with code. I'm just literally talking about sitting there and playing with it until you get a prompt that works nicely. we're able to improve the result of the overall thing in a big way. So if you're building, you know, so you should definitely customize your prompts. The big thing I wanted to sort of say here, and this is something often, I will show you some new skill or something and I will use the default prompts. But if I was going to build something from production, I will go through and spend quite a bit of time playing with the prompt, getting the prompt, working on all the things related to that prompt. Anyway, I will give you these notebooks. You can have a play with yourself. Don't forget the notebooks, I always put the notebooks in the description, So that you can actually load up the notebooks. I know some of you asked me in the questions, hey, can you, try out this prompt on it or something? You can do that, right? Just open it up. do it, especially these ones

I've been doing recently. you can just go and get your free API key from together. you can, use the, I'm using the T4, which is the free GPU that you can use on CoLab here. so all of these things you

can play around with yourself. Anyway, as always, if you've got any questions or comments, please put them in the comments below. if you found this useful, please click like subscribe, share the videos on social media, et cetera, to help other people. and I will talk to you in the next video. Bye for now.



## Advanced RAG 01 - Self Querying Retrieval

Okay. So in this video, I'm going to address one of the biggest issues that I see people having problems with in relation to building RAG systems or retrieval augmented generation systems. and that is that they try to use semantic search for everything. you only want to use semantic search where it makes sense to use semantic search. If you're doing search on things that would be the kind of things that are in a normal database that you would just look up an integer, look up a string, that kind of thing. You actually don't want to use those for doing semantic search. You want to do semantic search where you've got text that you're trying to extract semantic, meaning out of that. So this brings us to the whole concept of self querying retrieval. if we look at the diagram for this, we can see that the idea here is that. We have a sort of step between

the retrieval and the input. So the person types in their query. and then we use a large language model to reformat that query to get both the semantic elements of that, but also to be able to convert it so that we can actually do searches on metadata as we go through this. So this is a fundamental fact that if you're looking for a movie and you want to basically specify the year, you don't want to look for the year using a vector store in semantic search. You want to basically just do a lookup that looks at the year and filters the results back based on that year that you put in. Just the same if you were doing something for doing searches on Spotify or doing such as with music, if the person gives you the name of the artist, you don't want to use semantic search to look up the name of the artist. You want to do a query that looks for that artists and then uses the semantic search for doing the parts where semantic search is actually strong in this. So let's jump in, have a look at this in LangChain. So we're going to be using the self querying retriever here. now I'm using OpenAI embeddings and the OpenAI models. you can swap out, these for, you know, the other models like I've done in many videos before. And maybe at some point I'll do an end-to-end example of building an app with this using LLaMA-2 and say BGE embeddings. I've just gone for these cause so that these parts are not the important part in here. And the code doesn't take up too much room. So I'm using chroma as my vector store in here. like I said, I'm using the OpenAI embeddings. and then I'm going to pass in data. So this is where you would put a lot of effort into preparing your data. In this case, I'm going to be doing search over wines. And wine's going to have a number of different things in here. So you can see that we've got a description of the wine that's in here. We've got the name of the wine. We've got the year. We've got the rating that it's

got. We've got the type of, grape, in there. We've also got the color of the wine. And finally, we've got the country of where the wine comes from. So all of these things are metadata that we're putting in, And he, you can see that I've got these for a number of different wines in here. Now, Which part where we use the semantic search on? We would use it on this description that we've got here. So the description is what's going to be actually used for doing the search. So you'll see, as we go through it, that if we talk about things like, fruity notes or that kind of thing, It's able to work out that, okay, apricot and peaches are fruit. black fruit, stone fruits. These are sorts of things that relate to fruit, citrus flavors. but if we're talking about a year, we don't want it to just do semantic search over that. We want it to use a specific year and filter by that year. So we've got our data in here and you could spend quite a bit of time going through and, working out the best metadata for your particular use case one of the things that I see a lot of people, you know, if they've got a CSV file and a lot of the data would make good metadata. And then part of it would be used for the semantic search. This is a perfect example that you could do this. And you could basically write a little function that would go through your CSV file, take it out and convert it to something like this for all your different examples. All right. once we've got these, we basically need to embed them and put them into our vector store. So this is what the last line here is doing. It's just making a chroma DB from documents. We're passing in these documents. We're passing in what to use to do the embeddings. And it's going to embed these part and it's going to keep the metadata separate for these. All right. So next step, what we need to do is create our self querying retriever here. So this is something that's built into LangChain. the key thing that we need to do though, is we need to tell it the metadata info. So you see that each one of these

relates to one of the types of metadata that we've got in there. So we've got grape, which is the great use to make the wine. we've got the name of the wine and you'll see that these are strings or list of string in here. we've got the year is going to be an integer. we've got the country here, we've got the color of the wine. We've got the rating. Now the rating in this case is an integer, but if you were using something like, you know, score out of five and you could have 3.7 out of five, then you would change that to be a float for this kind of thing. And so the model that we're using actually knows what a Robert Parker rating is. So this is a famous wine reviewer and the ratings that we've got there, based on that kind of rating in there. I'm not sure if they're totally accurate, but it gives you a sort of sense of this. country, obviously that's going to be a string. and then for the semantic bit, this is going to be a brief description of the wine that we've got there. So once we've got that, we then basically set up our large language model. Again, in this case, we're just using the OpenAI model. But you could use a LLaMA-2, you could use, you know, a variety of different ones that you want here. Right then we basically set up our retriever. are we going to pass in the LLM? The vector store? All of this is very similar to just normal retrieval, augmented generation with LangChain. the next things we want to pass in is okay. What do we actually do the query on, and also the Meta data fields info in here. So these are the sort of two differences that we would do from a normal rag system. Now you'll see that we can go through and we can query. and what actually happens is that the model will take our input, which in this case, is the, what, you know, what are some red wines? And it will write a query. Now, in this case, there's nothing semantic going on in the query, right. It's just purely going to be filtering, but the filter is going to be a comparison. and it's going to be a quality comparison. We see that we can see that attribute is going to be color. The value is going to be red. So sure enough, if we look at

what gets returned back, we can see that we've got back. these nice results here of where we've got, an Italian red wine. We can see the, you know, for this color, red color, red color, red color red. So we've got a variety of different ones there. If we want to do some sort of semantic search now where I'm saying, okay, I want a wine that has fruity nodes. and, it's going to basically now do the search and I can see, okay, this wine has a crisp white, tropical fruit and citrus flavors, right. Which fits out our description. You can see that we get back all the metadata as well. we've got another one where this has got dark fruit flavors. Again, that fits the fruity nodes in here. so we can do a variety of different searches. What if I say, okay, I want a wine that has fruity nodes and a rating above 97 in this case. so here it's got our comparitor where I've got my greater than at rating, 97. and it's bringing things back that again, relate to fruit. So apricot and peach red fruit, earthy notes there. What if I say, I want wines from Italy. So now this is no query on this. You can see here, the query up there for that last one was fruity. Right? we can see that this one has no semantic query, but it has an equality country, Italy. We get the Italian red ones back. what if we want to do a composite of a few things? So here I could say, okay, I want a wine that's going to be that's all earthy. and it's going to be between 2015 and 2020. and sure enough, you can see that it's worked out that the, The semantic element to look up is earthy in here. And then here, we've got an operator of and, and we've got this comparison of, greater than year 2015 and greater than a year, less than 2020 in here. And it's able then to basically bring back the data that we want. So this shows us that we can do a Whole variety of different searches that comprise both filtering the metadata, but also using the semantic. look up data of the vector store on those descriptions that we've got in there. Another thing that we can do is we

can actually limit these things. So if we do want to just limit it to, if we've got, you know, 10,000 wines in there probably don't want to return 5,000 wines. So we can do a limit in here. So here I'm saying, okay, what. and that limit is basically the model working it out in this case. So you can see here, we've got where I've put what are two that have a rating above 97. So there's no Semantic query in here. we've got the greater than rating above 97 and we've got limit equals two. so it's just probably giving us the first two, in there. I want you to see that because we're using the large language model to rewrite our query. We can do things, you know, wrong in here where I can say, okay, what are two wines that come from Australia or New Zealand? you'll notice that I've deliberately, not put the capitalization's at times. And you'll find that even if we put some, misspellings and stuff like that. The language model. If you've got a good language model will work out the difference there and be able to then fix this up. So in this case, it's returned back to wines, which is what we asked for. We can see that we've got the quality equals Australia or quality equals New Zealand. That limit equals two. and then finally, Sure enough, we get back our cloudy bay wine with a rating of 92 from new zealand. And we also get back a Penfolds grange from australia here. So hopefully this gives you a taste of how you can build a more advanced retrieval augmented generation system That uses. the concept of metadata to filter things as well as purely semantic search. So this concept will work with other language models It doesn't have to be with OpenAI. i've just done that for convenience in here like i said you could do this with LLaMA-2. You will need to make sure that you're getting a language model that can take these kinds of queries And process them for this. So try this out. i find that this is a way that people can actually do a lot more with retrieval augmented generation than just the sort of minimal standard looking things up by semantic search alone. Anyway as always if you've got questions please put them in the comments below. if you found the video useful and

want to see more videos like this please check out my channel click

like and subscribe et cetera. i will talk to you in the

next video bye for now



## Advanced RAG 02 - Parent Document Retriever

okay. So in this video, I want to look at, the idea of what's called a parent document retriever. so this is something that becomes really useful in RAG. And it's good to understand the difference between LLMs and embeddings here. So, While there are embeddings used a lot in LLMs, here we're looking at embeddings for retrieval. And they act quite differently. So whatever we can actually put in context, learning for an LLM, as long as we've got a decent model, it's pretty good at being able to work out what is relevant and what is not relevant. So when you've got a decent quality model for RAG, if you put a lot of things in there, as long as you don't put too many in there to confuse it, you'll find that it's actually pretty good at being able to extract out pieces of information. So you might have something

like this where you've got a bunch of different documents. This is taken from an earnings call for Google. And here can see we've got, Sundar Pichai giving an update about various products, about earnings, about things like that. Now you could have mentioned that in this particular call, they talk about a whole bunch of different detailed stuff. So we don't want to put the whole thing in there. Right. Really to get a better result, we want to help the language model as much as possible and sort of trim out the bits that are not relevant. And that's what RAG is all about. And with a conventional sort of RAG, we have something like this where we have our original documents. We then split those documents. Let's say this is one of them. We split it up into multiple parts. And then we take an embedding for each of these parts. Now, remember that this embedding is a representation of all the semantic details that are in this particular part. So if this particular split doc here is going to be quite big. Then the embedding can get very sort of washy, right? It can be nonspecific. Whereas that part of the document is quite small, the embedded can be actually quite specific. So you could imagine that if you've got a document where they're talking about, say earnings and they're comparing different products or different things like that. And if multiple figures are being used and multiple descriptions are going to be used, the embedding for that particular thing is going to be more sort of general, about the earnings. Whereas if we split them up so that, you know, they're one part is about, okay, this is how much money ads made. And then another part is this is how much money that, Bard made. And this is how much money that PaLM-2 made or something like that. you find that the embeddings and they're going to be much more specific to the particular question that we want. That said though often we want to make use of the fact that the large language model can actually handle multiple things in there because it might actually want to do a comparison to be able to say that, ah, okay., this particular product. this much, and we can see that more in comparison to product A and product B. And to do that, we'd want a specific embedding for the chunk that we want, but then we want to sort

of pass in a bigger chunk of the overall sort of context in here. And this is where parent document retrievers come in. So the idea here is that, where, you've got your original documents. And you split it up into sort of chunks that are a decent size. We're going to call those the parent chunks. And then rather than do a straight embedding on those, we're going to take those parent chunks and split them up into child documents. Right. So we've got the parent documents and then we've got the child documents. And you might find that one parent document actually, has three child documents. And remember all of the content from these just comes from the parent document in here. But now we make an embedding for each of these child documents. so you can see now we've got three different embeddings describing what, before we only had one embedding for. So the idea here is that we're getting a much more specific representation with our embeddings that as we go through this. Finally, when we want to actually use

this, we have our question embedding that lines up to the child doc, but what we can then pass back is rather than just pass back this child, doc, we actually pass back the parent doc. So that the parent doc has got a lot more context in it. So the question is able then to, you know, the large language model gets a bigger context but the embedding representation is on a much smaller context. So it's going to be much more specific and then the large language model can take advantage of having the extra context that's in there. That's what parent document retrievers do. let's jump into the code and have a look at how you would do this, in LangChain. All right. Let's have a look what's going on here. so one of the things I'm doing different in this notebook is I'm actually using the BGE embeddings rather than the OpenAI embeddings. I'm bringing in a zip file of some of the LangChain blog posts here that I've basically just scraped and save to text files in there. and we're going to use that

for doing this searching. All right. So the parent document retriever, has two ways to use it. one of the ways I explained in the section before. but let me just go through these two ways and we'll look at both ways in code. So the first one is rather than sort of return bigger chunks we return the full documents. So if our documents are not that long, one of the things that we can do is just use the documents as like the parent documents. So that when the smaller chunks do the lookup, they just returned back the original document. This can be really good if your original documents, let's say you've got a lot of them, but each one in itself is not very big. So this could be like product descriptions, things like that, where maybe it's like half page of text and it's like one article for each of those docs and maybe you've got a lot of them in there. you can then split those up into smaller chunks. Do the look on those, but then return back the actual full document for that. The second way is more like, what I explained before with the diagram is that you're going to return the bigger chunks from a smaller chunks look up. So let's have a look at these. So we were just doing some standard imports, were importing the parent document retriever here. we're going to basically have some text splitting and stuff like that. I've commented out the stuff on the, OpenAI embeddings, but I've left it in there so that if you want to use that, you can use that. The embeddings that I'm using in here are the BGE embeddings. I'm using the small and these have just been recently updated, I think as well. The small at 1.5 embeddings and these are tiny like the whole, embedding model is only 134 Meg. now I'm running it with a T4 in the CoLab here. If you don't have access to a GPU, just drop back to the OpenAI embeddings as you go through this. All right. So I'm bringing in a couple of these, blog posts here. I'm bringing in one that's

announcing LangSmith. Another one that's about, benchmarking QA. and we can see that sure enough, when I bring these in, I've got two documents. if we look at one, you can see the way I've scraped them is I've saved the, title and the URL to this And we could actually sort of filter those out if you wanted to have them as metadata, but here, you know, basically, I've got the URL first and then I've got the title here for the blog post, and then I've just got the text for the blog. So this is using our custom scraper that I built, or maybe I'll release this as open source. It's actually quite simple to do for this. So in this case, each doc is one full blog post, here. Now, what we're going to do then is we're going to split these up into smaller chunks. and we're going to be using, you know, a child splitter here. We're going to use the recursive character splitter. And I'm just taking these settings basically from the LangChain docs in here. Now, of course I've swapped out the embeddings to basically use the BGE embeddings. If you are using the OpenAI ones, just uncomment this and delete this out. And then we've got our full-size docs are going to be in a store. that's going to be in memory here. so he can see our full doc retriever is going to be a parent document retriever with our vector store, which is the small chunks that we've split up. the full-size docs in the doc store, and then we've just passed in the child's splitter. So when we do this add documents, it will run through. it will basically make the embeddings for us. It'll put it all together. Now, if we look at the store in here, we can see that there's only two documents in there. So in this case, we've only got two blog posts in there. But these are the idea is that they're the big chunks in this. And if I do a vector store similarity search for what is a LangSmith and I set it to just K equals two coming back. You see, sure enough, I get, two docs coming back. And if I go in and look at one of these docs, we can see that this first doc is basically today, we're

introducing LangSmith a platform, right? And we can see, that's not very long. It's just a few sentences long in there. So this is the small split in there. Now we've got, I've got two of those coming back. If I pass the what is LangSmith into the full doc retriever? What it does is goes and works out what are those small chunks. But then it works out what the parent chunks are. in this case, those parent chunks are the full docs and then it returns those. So you can see here if I say, okay, what is LangSmith? Sure enough, I'm getting the sort of matching full doc for this first one here. And we can see while this one was, just probably a few hundred characters long, what we get back here is 11,600 characters long. and that's because it's the full post in here and sure enough, what post is going to be announcing LangSmith in there. So this is one way that you can do it with the whole thing in one shot. The other way that you can do it is just retrieving larger chunks. And this one you would use where if you're parsing in the full document, it's just going to be too big. So let's say now your full documents are multi page documents. You don't want to pass the full document into the language model at the end. So now we have to have sort of two layers. We have to have the original documents. The parent chunks. And then the child chunks or the big chunks, and then the small chunks in here. So again, we've got our, we're going to use the, child splitter that we had before, but now we've got the parents splitter, which you'll notice is a much bigger size, right. So we've got 2000 for the parents splitter and we've got a 400 for the child's splitter in here. So again, these, just take them from LangChains docs. You would experiment around with what's the right size for your particular thing. We've then got our Chroma, setting up again, using our BGE embeddings. We've got the store the same as before. There's nothing different. Now the only thing now. Is that when we create the parent document retriever, we've got both a child splitter now and a parent's splitter. So we're going to have

multiple layers of, docs. And in this doc store now is going to be the big chunks right in there. We've got those two blog posts. When we look at the store, now we can see they've been split into 18 big chunks. That's because each of those is roughly 2000 characters long. The sub docs, when we do our search, you know, these are going to be much smaller. So the sub docs now, when I do a search, I get four of them back. All right. when I look at the first one, Sure enough, we can see today we're introducing a LangSmith and we can see, okay, what's going on there. and I could change the K for the number ones I'm getting back there, right? The standard default is four, earlier on I changed it to two. okay, now we can see if we do the sort of, retrieved big docs. So this is big chunks retriever. Get relevant documents. What is Lang Smith? You can see now, sure enough, we've got only two docs coming back. And when we look at them, they're big, but they're not the full document. They're not the full blog posts like we had before. But if we look at the two that we get back, sure enough, we've got this one, which is the start of the blog post about that. And if we look at the second one, it's probably also, looking at it, it looks like it's also from that blog post, but it's a different part of the blog post. So now we've got two chunks of the blog post as two separate things that we're going to pass into the in context, learning in here. And then finally, just to sort of finish it up and try it out we just put together a retrieval QA chain. here, I'm just using OpenAI for the language model. we basically, you could put in any. you want a decent language model to do this, right? But you could certainly put in, the LLaMA-2 models should work pretty well for this. We're just going to pass everything into in context learning. and you can see the retriever, I'm just passing in the big chunks retriever now. So now when I say, what is, LangSmith? It's going through, it's running

the queries to get the small chunks. And then from those getting  
the big chunks back. and you notice that early on remember  
we had, the four sub chunks, but they only gave us back two big chunks. So that obviously overlaps in  
those, which is a good sign. and sure enough, we can  
see here that, okay. What is LangSmith? Now we're getting a proper thing. What's going through a  
language model at the end. LangSmith is a platform designed  
to help developers close the gap between prototype and production. We've got a full answer now for this. So this shows  
you how you can use  
the parent document retriever. It's very useful for a lot of different  
things where you've got a lot of, sort of fine grain information. So you want embeddings that are very  
specific, but you also want to return back a bigger context for the language  
model at the end, to be able to give her a good coherent answer for this. Anyway, as always, if you've  
got questions, please put them in the comments below. if you found the video useful,  
please click like and subscribe. I've got a few more of these videos  
coming out about different things about retrievers and tips and tricks about,  
using this foot RAG going forward. All right. I will see you in the next video. Bye for now.



## Advanced RAG 03 - Hybrid Search BM25 & Ensembles

Okay. So one of the things that people want

to do often is have hybrid search in their particular retrievers. So what is hybrid search? Hybrid search is basically the combination

of having both a keyword style search along with a vector style search. So we've got the advantages of doing sort of keyword lookup, but also the advantage of doing

the semantic lookup that we get from embeddings and a vector search. So this makes use of a tool called BM 25. now

BM 25. Is not a new algorithm at all. This has been around from, through the

seventies, eighties for a long time. And to be really honest as someone

who's battled against this, this is a really good algorithm. meaning that for a long time, as

people have worked on trying to come up with better embeddings. and using deep learning, often

many techniques were actually just beaten by this BM 25 algorithm. So I'm not going to go

into depth about this. You can certainly look it up on the web and read a little bit about it. But basically this is all about creating sparse vectors. So where you're counting words or N grams in here, and you're doing a whole bunch of things with TFIDF So if you've come across Term frequency or inverse document frequency stuff before so you'll know that this is the driving force behind BM 25. one of the things that's really good with BM 25 is it's very quick to compute. So often where using vectors and using dense methods, which are embeddings they will be, slower than just doing this kind of, counting of words and working things out that way. So one of the cool things that you can do in LangChain is you can actually make a BM 25 sparse retriever And what it does is actually sort of, hides a lot of what's going on in the background. it uses the BM 25 package that you can put in here To do your lookups. But you'll see in LangChain, it's actually very easy to implement. So we basically just have, you know, from LangChain retrievers, we import the BM 25 retriever in here. Now, what we're going to do is I'm going to go through the BM 25 stuff and then we're going to combine it with just a simple sort of embedding retriever and combine them together in an ensemble retriever so that you've got both keyword look up and you've got, you know, the sort of normal embedding lookup for semantic lookup as well. so here I'm just bringing in some vector stores. I've got, I'm using open embeddings here. We're only using open embeddings for the vector store part with the ensemble retriever, not for the BM 25. BM 25 will calculate its sparse vectors by itself here. All right. So you can see that I've made a simple sort of list of documents. And you can see here one of the things I'm trying to do with this, and this is taken from their example, on the LangChain side, I've just played around with it a little bit more. And what I want to sort of show you here is that we've got you know, I like apples, I like oranges, apples and oranges are fruits. I like computers by Apple. I like, I love fruit juice. Right. So we've got sort of the word apple here but it's got some different meanings in some ways. So, the key word retrieval will

tend to just retrieve the keyword. So if a word matches, it will be brought back. And we can see this if we sort of instantiate our BM 25 retriever. We pass in this doc list. We set, our K returns to be two. And I give it the word apple. Sure enough. It comes back with, I like computers by apple. and it's actually surprising that it's not getting the other ones, cause they're not direct matches in here. If I ask you it a green fruit, it gives us fruit juice cause it's got fruit, I guess. and I'm not sure exactly why it's giving back, I like computers by Apple in here. but you can see that when we look at this, in the dictionary, this is just passing in our documents are in here. So, this is sort of like a key word search. So BM 25 is used in things like elastic search. It's used in a lot of very quick search algorithms that are out there. The embedding or the dense, retriever here, we're just using a normal sort of embedding lookup. Now, in this case, I'm using OpenAI embeddings. but really any embeddings do the same sort of thing. And I'm just going to bring this into a faiss or a faiss vector store. and we can see that here again we're just setting up to K equals two. And you'll see that this time, when I say a green fruit. straightaway, it gets, you know, apples and, yeah as fruits and also, I like apples. We don't get the computer one in here. So this is doing a more of a semantic lookup. So the thing that we can do is we can combine these. So what happens if we combine them is we basically make an ensemble retriever. An ensemble retriever we just pass in the retrievers that we want to have, normally you probably going to use this for a sparse retriever and a dense retriever, but there's no reason why you couldn't use it for different, Retriever's as you go through. And then we set up a weighting system. so this is going to help with determining how it re ranks the results from the two of these. You can see now when I put this

in and I say a green fruit. the first thing I get back here  
is, you know, I love fruit juice, apples and oranges or fruits. I like apples. And the final one is the you  
know, Apple computer one. Whereas if I put in Apple phones,  
so this is not you know, computers, but semantically, it can work  
out straight away that, ah, okay. I like computers by Apple is the  
one that goes at the top here. So here we're getting the advantage of,  
this hybrid search system where we're getting both the keyword search look up  
and we're getting the semantic look up. This is something that you can try  
out for different projects and see when does having hybrid search for  
your use case actually help as opposed to just purely semantic search. It certainly becomes useful for things  
where people know exact words that are going to appear in a text or something. And they're looking for someone's  
name or something like that. then the hybrid search can actually be  
really beneficial for that kind of thing. So, anyway, This is a very  
simplified example of this. have a play with it yourself. Try it out for some of your own  
use cases and see how you get on. As always. if you've got questions, please  
put them in the comments below. if you found this video, check out  
more in the series of different sorts of retrievers and different  
techniques that you can use for retrieval augmented generation. I will talk to you in the next video. Bye for now.



## Ollama meets LangChain

Okay. So in this video, I want to go through how having Ollama models running on your computer locally allows you to run LangChain locally and use those models to do different tasks. So I'm just going to look at some simple tasks, like how we set it up and stuff like that. and then finally, I'll finish up with a task where we're going to get it to go and do some scraping for us and extract some information all using, the LLaMA-2 model. Okay. So in here we can see if I come down, I'm just using vs code here. and I've just got a couple of Python files open. I've made a condor environment just to install LangChain. there's nothing special about that there. You can certainly set that up or just use, your local Python, if you really wanted to. So if I come into the terminal, I can just come and have a look at. what we've got going here. If I can see the models that

I've got going in there. So I've got my Hogwarts model from before. I've got the LLaMA-2 models. and, I've got some other models

which I'll show you in the next video about setting those up. so first off let's look at just the most simplest thing, how do we actually load the Ollama model. So in this case, we're going to use the pre-made LLM in LangChain for Ollama. and we can just set that up with

a streaming callback for this. and then we just instantiate our LLM,

passing the model that we want in this case, I'm using a LLaMA-2, and

then passing a streaming callback. Now I can just call the model

locally from my Python code here. Okay. So now I run the code. And you'll see. Sure enough, it's basically triggering the Ollama model. So it's doing this via an API. It's actually, one of the things that I

didn't cover in the first video is that Ollama is running an API that we can

actually trigger both with a LangChain, but we could also do it with just a

standard user interface, et cetera. So that's something that perhaps

we'll look at in the future is building a next JS app With

LangChain and Ollama are there. All right. So the next step up is I'm just

going to make a basic chain here. So I've got the same things from before. don't forget, this is an LLM,

so I can add in, temperature. I can add in max tokens. I can add in things like

that if I wanted to here. what I am going to do is just

make a simple prompt template. So just give me five interesting

facts about, and then I'm going to insert that in here. So I could say Rather than

Roman empire go for the moon. So, okay, you can see that

we can just set up our chain. we're just going to pass in the run

chain and print it out as we go in. So in here, we can also pass in, you

know, verbose equals true or not. If we want to see, the

streaming out of this. Okay. So in this case, I'm going to

turn off the callback manager. So we're not going to run that and

I'm going to turn verbose to be false. and now if I run it. You'll see that it takes a bit

of time because it's actually running and getting the data. but sure enough, once it's finished, it will actually just print the data out. So this is probably more useful if you wanted to write something to a file, et cetera. We can see there that it's gone and done it. And we've got the five interesting facts about the moon. And it's gone rounding and gotten those quite nicely. All right. Let's look at the last example. So this is definitely a bit of a jump from the previous two examples. So now we're going to be doing some RAG and we're going to do it using a web based loader. So we're going to load a web page in. we're going to run that and split that up. we're going to put that in a chroma DB. So you will need to have chroma installed for this. and, we're going to also use an arg pass to actually pass in a URL that we want it to get back some data. So let's have a look at the imports here. We can see that. Okay. We're bringing in recursive text splitter. we're bringing in the webpage loader with web based loader. We're bringing in chroma. And then we've got some embeddings here. So I haven't looked into it too much, but apparently Ollama seems to, according to LangChain, has its own embeddings GPT-4 All has also made, some quantized embeddings that you could use in here for this. we can see we're bringing in Ollama like normal. so we're gonna just have our main function, which is gonna just pass through, looking for the argument URL. we're then going to load that URL. We're then going to split that URL. we're then going to put it into chroma. We're then going to split that data up. and we're going to then put it into a chroma DB here. Then after that, we're going to set up our LangChain prompt. So here is the prompt that we're using, We're pulling it in from LangChain hub here. Now, you can certainly print this out and have a look at it and play with it yourself and change it. And then we're setting up the retrieval QA chain where we're just going to pass in the LLM. We're going to pass in our vector store, which is chroma. We're going to pass in our prompt here. And then I'm just going to ask it, set

up a question, the question, what are the latest headlines on, and then I'm going to pass this in and get the results out here. watch them stream out. So to set this up when I actually run the file this time, I need to pass in an argument called URL. And I'm going to pass in TechCrunch in here. And that's going to kick start everything off so that we can see that, okay, it's filtered out the TechCrunch. it's loaded some documents. It's loaded the LLaMA-2 model. And now we're going to see that it's just going to basically go through and give us the headlines from there. So, if we look at TechCrunch, we can see that these are the headlines that it's gotten out. So I haven't have to write any parsing code to do this. The language model has done this itself. And you could imagine that we could have lots of little tasks like this, where we could just send it off with a Cron job or something during the day to bring back information. and save it for us on our local drive. And you can see here that it's managed to get those top stories and list them out. Now interestingly sometimes it will list out five sometimes it will list out 10. I didn't put anything in there about that. but you could certainly play with that yourself. So this gives you an example of just using LangChain to do simple tasks with a local llm that you could do a variety of different tasks for this. If people are interested we could look at setting up a full local RAG system for documents and stuff like that using this model Certainly the LLaMA models are pretty decent at being able to get through this and you can run a chroma locally et cetera for this. Anyway as always If you found the video useful please click like. And if you've got any questions please put them in the comments below. I will talk to you in the next video. Bye for now



## Advanced RAG 04 - Contextual Compressors & Filters

Okay. So one of the biggest problems that you can face in RAG issues is what are you exactly bringing back from the retriever? And how useful is that to the language model overall for finding the particular answer that you want. So the challenge here can be that you bring back a few things. And some of them are useful and some of them aren't. Or you can often bring back large chunks and only very small amounts in that chunk are actually useful to the overall answer. So this can be where a particular question is going to require facts from multiple chunks and it needs to synthesize them together. It can also just be for clarity on questions where you don't want other facts getting into your in context learning window there with the actual query. So, this is where we introduced contextual compression and filters. So the idea of contextual compression

is that we have some kind of base retriever that goes and gets a bunch of different, pieces of information. And then we have these document compresses and filters, which go through and process that information to extract out only what is useful to answer the question. So this could be, for example, where if you've got, you know, a bunch of the information that's needed, is in the middle of a particular document that's being returned. The compressor can wipe out, you know what actually, is at the start and at the end of that, and just extract out the information for that. And the idea here is that your compressor or often a compressor pipeline of doing multiple things is going to sort of clean up those documents and give just the most useful stuff to the model at the end, to be able to answer the particular query as it goes through. So with these, you always start with some kind of base retriever. and then you add on some sort of filter as part of the compressor in here. So one of the first ones that can be is just simply a call to a large language model that can be an LLM chain extractor. and the idea there is that it's going to basically use a language model to extract out and clean up the bits before you pass it along. Now, this doesn't have to be the same language model as you use for the final answer, you could actually fine tune model, to do this particular task. I've seen examples where people have fine tune models to extract information out of emails. So were they, you know, disregard the simple things like, you know, hello, how are you? That kind of stuff. And it just gets to the meat of the email. and puts that in and then tags that. this is one of the key sort of features that you can do with fine tuning is make it a fine tune language model for a very specific task. And it turns out, you know, these kinds of things work very well when you fine tune for a particular task like that. Another example of a compressor filter is just to have the language model look at each of the responses and see, should this go in or not? So often this will be at the start of your pipeline for these. Another thing is that once you've started

to manipulate the outputs, you then may want to take an embedding again. So you've got your original query which you didn't embedding to get the information out. now that you've run it through a compressor pipeline, you then can often take another embedding filter at the end to see okay Out of these final ones how many are actually really still close to the query in this case? So one of the key things here is that this is allows you to get a lot often larger amounts of documents bring them, back filter them through, and then sort of compress the information in a way that you can send to your final language model call with the query to get the answer back here. All right let's jump into the code and have a look at how these actually work Okay, so here, I'm just setting up some standard LangChain stuff. We were sitting up the BGE embeddings, et cetera, that we're going to go through. So first off, we're going to have some simple imports. Where we're going to basically just set up FAISS for this. We're going to set up a text splitter. In this case, I'm actually going to use the BGE embeddings locally to do this. So that makes it quite easy. I'm going to bring in my documents et cetera. I'm going to split them up. we can then basically just set up a simple little helper function for looking at what we get when we do a search on these documents. So you can see start off, I'm going to use the same question and ask it what is Lang Smith? And you can see here, I'm getting, four different contexts back. And there's definitely stuff that's relevant in here, but there's also a lot of stuff that's not relevant in here or, it could be certainly be less verbose in the way that it's got different stuff back. So the first one we're going to do is we're going to look at adding a contextual compression with LLM chain extractor. So this one is basically just setting up an LLM chain, and what the chain is going to do is, in fact, we can just look at the prompt, right? We can see that the prompt is going to say, given the following question and context extract any part of

the context as is that is relevant to the answer to the question. If none of the context is relevant returned, no output Do not edit the extracted parts of the content. So we're basically just asking it to rewrite or actually not even rewrite, we're asking it to trim off sort of the irrelevant parts and just extract the parts that are useful to us. So you can see after we run this, we asked what is LangSmith? We can see, you know, we're getting stuff that is much shorter than what we had earlier on. now we've got, announcing LangSmith a unified platform for debugging testing, evaluating. you can also see that we've gone from four documents down to three here. So this is the first one. Now this requires, obviously it requires, another call to your large world language model. So sometimes it, you know, it can slow things down, but it's probably getting to give you much better quality of getting the actual outputs out. The next one I want to look at is a filter. So this one the LLM chain filter. I think the best way to think of this is just like a yes, no kind of thing. so the idea here with this one is if we look at the prompt again, this is again going through a large language model. If we look at the prompt, we can see that it's saying, given the following question and context return yes if the context is relevant to the question and no if it's not. So what it's going to do is basically go through and, and decide, okay, this one is a relevant one. This one is not irrelevant one as it goes through this. So this allows us then to basically work out that oh yeah, okay, this bit is relevant. Now notice that, this URL and title got trimmed off in this one. because it wasn't kind of the most, you know, the best part of it for answering the question. But certainly the announcing LangSmith unified platform for debugging testing was relevant, right? and we can see that here too that this time we've got the full part of that context. but we can see that, you know, that certainly is a relevant thing to answer in here. So this would have been returned, yes. and these other ones would have been returned, yes. Another trick that you can do is an embedding filter. So here, looking at this is where you basically do your retrieval first and ideally you want to do, I'll probably

talk about this in another video, but you want to do retrievals of sort of five contexts, or more sometimes, maybe you're going to retrieve five from different sources to bring back together. but here, what we can do is we can use an embedding filter to actually, grade these or to sort of like, pick the top ones and rank them One of the things that you may ask is okay, well, wasn't I already using embeddings to actually get the context in the first place? Yes. you may use different embeddings, for this. So, I could be using BGE for one thing and then OpenAI for another thing or I could still use the same embeddings, but you'll see that I could put this in a pipeline. So, when we do the actual context retrieval, we're doing the embeddings and we're retrieving the most similar, uh, context to bring back here. but if we would then to say, run this you'll see in a second that when we run things in pipelines, this is where things start to change. And so if we were to run the extractor first, And then create these new ones where we've got, you know, this being chopped off, like we saw before. And then we can now take a new embedding on what comes back and see, okay, what are the really good ones in here based on this new embedding that we've got going on in here? So this is something where we can set a similarity threshold we can go through it. we can basically do a new sort of embedding lookup to return stuff. Okay, next step is where we can start to join all these together. So this is a document compressor pipeline. so the idea here is that we can do a whole bunch of different things and stick them in a pipeline. so, We can get our context back. And then if those contexts were say 1000 or 2000, et cetera, And we want to split them we can then come into here. do a splitter on these again, split them up from being 1000 now to multiple chunks of 300. we could put in an overlap in there would be good as well. and then once we've done that, we couldn't have this pipeline that's going to basically do the splitting. Then we're going to do a new

embedding filter to see, you know, which ones of these are redundant, that we should just Chuck away. And then we're going to then basically finally bring back just the relevant ones within a similarity threshold. So this would allow us to say, take a five 1000, character, chunks back that we get. Split each of those up into 300. So let's say we've got some overlap in there. We're going to get maybe four chunks out of each of those. so we've now gone from five chunks to 20 chunks. And then those chunks are going to be smaller and then we will take the embeddings of those chunks to bring back the most relevant information for that. And you'll see that when we run that through. now it's going to basically split these things up and bring them back from what we've got there. We could also though do something like this, where we basically take a splitter, we've got our compressor for actually compressing, with using the large language model. And then we've got our embedding filters. So that when we bring it back, we've got something much smaller now. Now, in this case, you can see that, okay, probably this one is not that useful. but we can see that some of the other things we're getting back, are going to be, you know, quite smaller context. Now, this is probably a little bit overkill in this particular example, but you can have pipelines that do a whole bunch of stuff. You can have, pipelines that sort of filter things, then do the rewrite. then you check with embeddings. You can retrieve from multiple sources with an ensemble, do a filter, do a rewrite. You could even then check embeddings, et cetera return it, retrieve, split them up, check the splits with embeddings, filter it, do a rewrite. So there's a whole bunch of different ideas that you can try with this pipeline's idea. and so this is basically just setting up a pipeline compressor, where you take some of the things that we've talked about before and start to use them in your actual document retriever pipeline in there. All right. So the contextual compression

tasks and filters along with the pipelines themselves. one of the key things for building a really good and useful RAG Now, you always have to balance the speed if you need it to be real time versus the number of different things you're doing. But if you're doing something like summarization where you perhaps don't need it to be as real time, you can do quite a number of these things. and you can summarize, you know, custom summarizations, based on getting these chunks out. and in that case, you could go for quite a lot of chunks that get used for answering a particular summarization question, et cetera. If you aren't doing sort of real time question and answering you want to basically check out your timings of how long has each of these things taking? and you'll probably build multiple, compressor pipelines where you test, you know, doing, Different kinds of rewrites, different kinds of filtering, different kinds of splitting et cetera. The other big thing to remember is don't forget that you can actually rewrite all these particular prompts. So things like this prompt that you've got here if you prefer to rewrite it for a very specific use case, Far too often I see people are building a RAG for a very specific use case but they're using very generic prompts There's nothing wrong with honing in your prompts on your specific use case And allowing the language model to know that okay you're only interested in things that relate to medical things, you're only interested in your particular types of things depending on the domain that you're working in Alright I encourage you to play around with this and try out different versions of this for your particular tasks. As always If you've got any questions please put them in the comments below. If you found the video useful please click like and subscribe. I will see you in the next video Bye for now



## Advanced RAG 05 - HyDE - Hypothetical Document Embeddings

Okay. In this video, we're going to look at the concept of Hyde or hypothetical document embeddings and, this comes from the paper called precise zero shot dense retrieval with without relevance labels. so remember dense retrieval is just, looking up things with a similarity search or semantic search. usually with a vector store nowadays involved here. so this paper came out last year. at the end of last year, it's one that I kind of feel like not enough people have paid attention to. It's a technique that while it's very simple, is very powerful in the way that you can improve your rag system overall. so if we look at this diagram from the paper, this shows you the fundamentals of how this thing works. And this is that we're going to input a query into our system. And then rather than straightaway just look up the embedding for that and compare that embedding to all the possible chunks to find the best answer. we're actually going to use a

large language model in the middle and what the large language model in the middle is going to do is write a hypothetical answer. So why would you use this? Let me explain this. sometimes if someone asks a question that's very bland or, often that perhaps doesn't have any nouns in it or anything that we can easily look up to find an answer that can be quite challenging. So if we look at the example of McDonald's right. we all know the concept of what does McDonald's sell. It sells food. but if someone's asked the question, okay, what a McDonald's best items? Now, if I ask you that question, there's a high probability that you will know that I'm talking about food and that perhaps I'm talking about burgers or shakes or fries. The type of fast food that is. the challenge is none of that is mentioned in the query. So if I'm taking an embedding of the query and then looking up, an embedding of something else. And let's say, I didn't mention the word, McDonald's maybe we've been talking about McDonald's and I just say, oh, okay, what are their best selling items? Now bestselling items, plus, you know, looking for big Macs in our semantic vector store. The chances of finding it, it's probably going to be reasonably small. So, in this case, the one of the best ways to do this is to have the language model write an answer first. Now we're writing the answer not to let the end user ever see the answer That we're writing the answer just to get something which we can then take an embedding from. So for example, if the Large Language Model writes the answer and it says, oh, you know, their best selling item is big Mac or their best selling item is the Tuesday burger or something like that. even if it's the wrong answer, you'll notice that it's talked about a burger, it's talked about McDonald's is talking about stuff now, which when we take an embedding of that will be a good representation to look up and find the right answer. So this allows you to do something where you're not doing query to answer embedding similarity. We're doing answer-to-answer embedding similarity. And it turns out that this

often works really well. Now it doesn't always work well. if you're talking about a topic that the language model has no knowledge of at all remotely. You probably want to stay away from this because it will tend to hallucinate, things that, are totally different. And I'll show you an example of that in the notebook. but on the whole, If you're dealing with something where you need to look up something that's going to be quite specific it can often be quite good to have the language model do this. Now one of the other things it can do is rather than just generate one answer you can generate multiple answers and then you can feed those into you know you can average those out in an embedding model. So you can take an embedding of each of those and average them out which will allow you then to basically look up things as well. So there are a number of different ways you can sort of use this But it is a very powerful technique for this

Let's jump into the code and have a look at how this actually works in the code So remember that the HyDE is hypothetical document embeddings. and the idea is that we're going to create a hypothetical answer with a large language model. And then we're going to embed that and use that for the search against our documents in our RAG here. So, okay, we can see that, that the code part to do this as we need some base embeddings or base embedding model, and we need an LLM chain. and you'll see that we'll change the prompts. We've got some basic prompts in there, but we'll also then go for some custom prompts as we go through this. All right. So, you can see what I'm bringing in here. I'm using OpenAI, just for simplicity for the large language model For embeddings, I'm going to be using the BGE embeddings. And you'll see that as we go through that this is going to be generating out the embeddings. You could swap this out for any embedding system that you wanted to use, which would be fine. even if you were using like a local system, you could use the quantized embeddings for this in here. All right. So this is where the magic sort of starts. so you can see that we're going to get the embeddings out of this, but actually what we're doing is making a chain in here. So this is, from large language model chain. we're parsing in the actual

embedder that we've set up here. So we've set up our BGE embeddings,

And what this is going to do, is this gonna take in our query. And you can see that when we look at

the prompt for this, we can see that, okay, what's it going to do, please

write a passage to answer the question. So this is a very simple sort of example. You'll see later on that it's often good for us to write this and customize it to our particular use case in this. But okay we can see what we've got there.

Let me just sort of go through and

explain again what I talked about earlier on was that, something like,

what items does McDonald's make. Now we know that McDonald's makes food. And it makes burgers and it makes fries and things like that. But you'll notice that none of those

things are mentioned in the query. So, this is where if we're taking

an embedding of the query versus the answers back it might not be a great

thing because you know, the answers we would want back would be, French

fries, big Mac, that kind of thing. but none of those are

mentioned in the actual query. So here the large language model, and

I've just turned on LangChain you know, debugging here so that we can see the

output and every step We can see that it goes into large language model. It basically says,

write an answer to this. He answered that, OpenAI writes is

McDonald's is a fast food chain, which is known for making a variety of things. These include signature burgers, such

as big Mac, quarter pounder, as well as chicken sandwiches, wraps, desserts. We've got a lot more stuff now that's

content that the embedding would fit this. So something like, big

Mac is mentioned in there. now when it's going through

looking for pieces of text, it's probably going to match things

that are big Mac related in there. So that's the first way and the

simplest way to do it with just their built-in prompt, that kind of thing. The next way we can do this as we

can also have, multiple generations. So rather than just generate one answer,

we can generate multiple answers. So here you can see, you know,

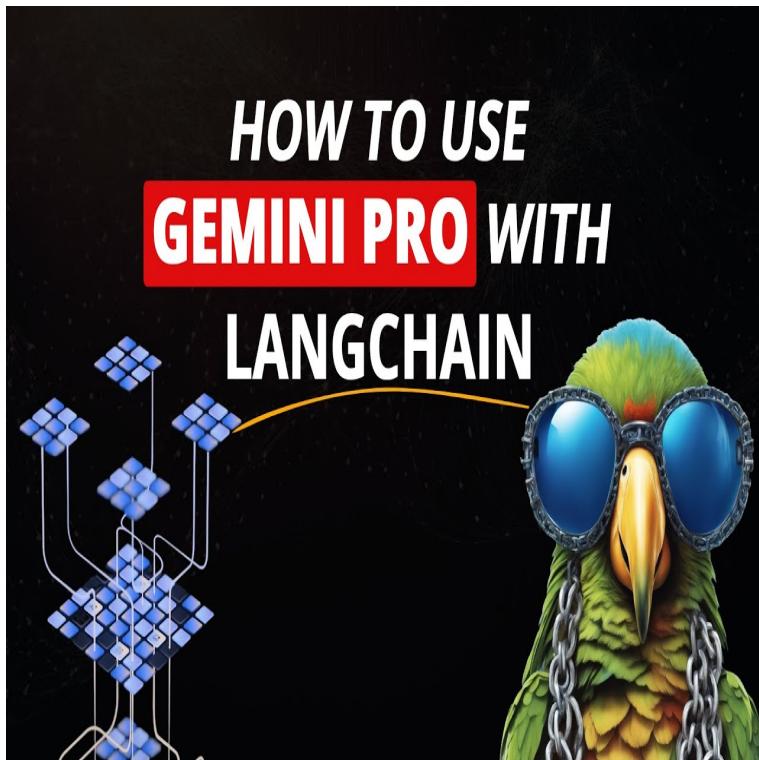
When we ask it, okay, what is McDonald's bestselling item? Again, we're not mentioning, actually

we're not even mentioning food here. Right. So if we were just doing the query, sort of look up, it's possible that it could get, confused of not knowing, that, okay, we're actually talking about food. Whereas the large language model kind of gets it. Even if it's not giving us the exact right answer. It's giving us oh, McDonalds is one of and the most popular fast food restaurants in the world. its menu items include a variety of items, but one stands out is the bestseller, the big Mac, right. And we can see that we've got different answers in here. bestselling item is undoubtedly, the big Mac. and we've got to, you know, a number of different that we get big Mac mentioned again here. So when we take the embeddings of these and we combine them, we definitely going to get this representation of big Mac coming through for when we do our search with that embedding there. So this is one of the key things that makes HyDE so powerful, right? Is that you could take a number of these things. Combine these embeddings. Another thing that you could also do is use your own prompt. So if I don't want a long answer like that back, and I just want it to mention, the food I can say, okay, please answer the user's question as a single food item. so now I kinda know, I would use this in a case where I kind of know what the question's going to be about or the topic. And I would guide the LLM to that topic. So if we were doing this for the McDonald's website, You know, we could put in there, you are a helpful assistant that works for McDonald's. you answer questions about different things on the menu. please answer this, by recommending, a single food item kind of thing in there. So now when somebody just says, well, what is McDonald's best selling item or even worse in this case, it's where people would just say, oh, what's your best selling item right now? What's your bestseller? And it doesn't even have McDonald's in it for the embedding that wouldn't go down well. Whereas we can see that, if we've set up this user prompt quite well, we can get something back, which then in this

case we just wanted a single item. What did it return? Big Mac. We can take our embedding on big Mac. There's our embedding for being Mac. And then we could do a search on it, for this kind of thing. So just quickly to show you how to put it all together. so in the original example, they were just using a single file. I've put in some text loaders in here. and what I'm going to do is ask it. Now, I got two examples here. one that's not here. One that I am showing here. So let me share the one that works here first. and this is, you know, please answer the user's questions related to large language models. Again, we're guiding it with a custom prompt here. When I ask, okay, what a chat loaders? it kind of makes something up that's wrong. Right? Chat loaders are software tools used to load large language models into chat bot applications. And that's not quite right. But it doesn't matter because it's mentioned all the key things so that the representation is able to then find these chat loaders So that we can, you know, we can come in here and we can look at this, this article chat loaders, fine tune our chat model in your own voice in here you know, for doing this. So this is an example of how it would, you know, it would work. You'd have to be a little bit careful. So originally I was going to do the example of LangSmith like I've done in some of the other notebooks. the challenge there was that the large language model knows nothing about what LangSmith was. So unless I was to overly guide it in my question prompt it's it will just come back saying that, oh well, it's language learning software So you want to be careful for if it's something that's totally new that the language models perhaps never heard of, You know then you want to be a little bit careful about using HyDE. But for other situations using HyDE is often going to get you better results Then just embedding the query and doing the query against the answer sort of similarity match there. So this is a very powerful technique encourage you to check it out and try it out on your own projects and see how you get along. As always if you've got questions

please put them in the comments below. If you found the video useful

please click like and subscribe. And i will talk to you in the next video. Bye for now



## Gemini Pro + LangChain - Chains, Mini RAG, PAL + Multimodal

Okay. So in this video, I'm going to look

at using Gemini pro with LangChain using a Google AI studio, API key. So just quickly, the things I'm going to

cover in here, I'm going to basically show how you set up the LLM with LangChain. How you can actually

do like a simple chain. With LangChain. Then also we'll do a through

little mini RAG system. Using both the Gemini pro model and also The latest Google embeddings model. That's in there

as well. And then we'll have a look at

using a Gemini pro for something like PAL for a PAL chain. So that's a program. Aided language model chain. In there.

And then finally, we'll wrap it

up with looking at some multimodal things of how you would actually

pass in an image using LangChain. So some of the things that I've done

already in the previous video, but now we're going to look at actually how you

could do it in LangChain and then use it for building LangChain apps, et cetera. All right. So I'm bringing in a few different things here. Some of you probably don't really need all the ones that I've brought in here. I've been playing around with a few different things. But you will need to have a few sort of key ones. One of the key packages to install is this LangChain, Google gen AI package. So, I think this is one of the first sort of third party. packages that LangChain is doing now where the language model stuff for this is in this particular package, not in the core LangChain library anymore. All right, I've got that set up. I've got my secrets here in CoLab. remember just to, put your Google AI studio. API key into the secrets, then you can access it like this. Obviously I've run this notebook already. So it's not asking me for permission. It's already got permission to do this. and then I can have a look at the list of models that we've got here. So you'll see that this is basically what we had in the previous videos. We've got the old legacy. texts, bison models. we've got the old, embedding gecko model. and then we've got the Gemini pro model. and then Gemini Pro Vision, and then the embeddings zeros. Zero one model in here. and in the attributed question, answering model. I'll do another video for that. all right. So if we just using the straight up normal API, this is how we would do it. So we looked at doing this in the previous video. We basically just get, get the model to be Gemini pro. I'm passing in a prompt. and you can see that then I'm just taking that prompt, printing out the markdown and we can see, okay. I am a large language model. What did we ask it? Who are you? And what can you do? I'm a Large Language Model trained by Google. I'm developed by Google brain team. It's actually the deep mind, Google DeepMind team nowadays they've changed, they've changed their name. but anyway, then it shows you go through some of the things. that's, that it can do, et cetera. And obviously this will change each time that you run it as well. So, okay. We, if we want to do the same kind of

thing in LangChain, what do we need? So the main thing that we need is we need to bring in this chat, Google generative, AI. model. So from here, we can basically tell it that we want to use the Gemini pro API or the various Gemini pro API APIs. and you could have mentioned it's going to be the same for other Gemini models going forward here. All right. So I'm basically just defining the LLM. and this is sort of just like a key replacement. of what you would do if you were using open AI or something like that. And then just in this one, I'm just basically saying to it. Okay. just take the model and let's pass in a prompt. So we're going to invoke the model there with the prompt, and you can see that we're getting back. something, kind of similar to what we got before, where, we've asked it, what is a LLM? LLM stands for large language model. We've got the whole result back. You notice that it's returning back marked down so that we're getting some things that we're getting bullet points. We're getting a bolded text. We're getting a number of those things in there. just like with the other one you can stream. in here. This is just basically passing it through it. Won't always pass back one chunk at a time. it will tend to, do a number of chunks and pass them back depending on the speed of it, et cetera. All right. So the next thing we want to do is look at building sort of like, some different chains. going on here. So here we're going to make our chain where we're going to have just the joke chain, a very standard sort of LangChain thing. We're bringing in chat, Google generative, AI, so that we can set up the Gemini pro model. We're going to pass it in a temperature of 0.7 in here. and then we're going to just have a chat prompt template where we're going to pass this in. tell me a joke about, and then we can change the topic. We're also going to have an output parser. And we're also going to be using the LangChain expression language to put this all together. So you can see here, our

chain is going to be a prompt. Piping or passing into the model. Taking the output of the model, passing that into the output parser, which is going to convert it back to a string. and, then it will just, print it out in this case. So. If I go through here

and I run each of these. We should see now tell me a joke about machine learning. Okay. Why did the machine learning algorithm go to the doctor? It had a kernel panic. okay. Not the best jokes, but it's certainly doing what we're asking it to do in here. All right. So let's move on and have a look at it a little bit more. complicated sort of chain, and this is kind of like a mini rag in here. so now we're going to be using the, Gemini pro chat models. So we're going to be needing the chat, Google generative, AI, but we're also going to be using their new embeddings. So this is Google generative, AI embeddings in here. we're going to basically just, we're not going to use a vector store. to save to disk or anything fancy there, we're just using this sort of array in memory search. vector store in here. And we're going to pass in, a number of different things. So we can see here, I've got Gemini pro is a large language model. was made by a Google deep mind you have, and I can either be at a star sign or a name of a series of language models. a language model that should be a language model is trained by predicting the next token. LLMs can easily do a variety of NLP tasks as well as text generation. So this is what I'm going to embed, right? Each of these is like a little mini document that we're going to embed in here. So in passing in the embedding, that we've got here, so we're using this Google. generative AI embeddings. And the model that we're using is embedding, 0 0 1 here. so we're passing that in. So that's going to be our embedder model. That's going to do the embeddings for us. And then we're going to just set this up. as a retriever. so that we can basically ask it questions and it will retrieve back. these documents, but in an order, that's going to be based on what our search is. So you'll see that if we just ask it just, what is Gemini. It's going to probably go more for, the star sign. Right. So Gemini can either be a star sign

or a name of a series of language models, and then it comes Gemini pro. there if ask it. What is Gemini pro? You'll see that now it comes with, back

with the Gemini pro at the top there. So this is the most relevant,

query that it's finding based on the embedding lookup in there. All right, so we need a prompt. So the prompt here is very simple. One, answer the question. As based only on the following

context we passing in the context you pass in the question. That gives us a prompt. We're going to chain all of this together

using the LangChain expression language. so I'm going to bring

in runnable map here. I'm not going to go too much into detail

of what actually each part is doing here, but we've basically got a, this runnable

map so that we can pass things through. We're going to have. the context be the retriever taking in the

question, right from our prompt up there. and then we're also going to pass in

the actual question in here as well. then we're going to pipe

that into the prompt. Pipe that into the model, pipe

that into the output parser. So that's going to be our chain here. And you'll see that if we

ask it, who made Gemini pro. It should come back. Google deep mind. And sure enough, it came back. Google deep

mind. Now if we want it to get the answers

back, say as a, as a full sentence, right? So we'd come up here. We changed the prompt. answer the question. as a.

Full sentence. coma based only on the following context. So you'll see that. Now, if I run through

with that new prompt, I should get the same sort of answer

back, but now as a full sentence. So I've got Gemini pro was

made by GoogleDeepMind. in here. What about if I want it to get back? Let's say I want to do

something like return. Your answer. As JSON. All right. So we want the answer back in JSON. All right. I run through the

prompt. So I've, I've just heard the prompt here. I'm asking the same question. And you can see now I'm getting JSON

and I'm getting back, like a, a JSON dictionary here where answer equals do

you want me to pro is a large language model, made by Google deep mind. in there. and we could, play around with

this. We could also say return your answer. Let's say in. Three. Back ticks. All right. And you see now we're getting the

answer

back with three back ticks new line. Answer new line three back ticks in there. So you can actually put things like your  
formatting and stuff like that in here. as well as and obviously, if we  
were using, we could use the Jason output parser to actually get back  
a dictionary and stuff like that. So some of those things I've looked at. in the past in some videos. Maybe I'll look at doing  
some new videos about that. If people are interested in  
more stuff on the LangChain, expression, language, All right. So we've done that one. one of the ones that  
really sort of shows off. The strength of a model is, the PAL chain. So the PAL chain, I used to work  
with a very specific model at open AI, which they deprecated. And with the 3.5 model, it was tend  
to be sort of hit and miss of whether it works, whether it doesn't work. so if we try this out with. So if we try this out with  
Gemini

pro, we can actually see, okay, can. can Gemini pro do this. So in passing in, basically  
the cafeteria had, 23 apples. I remember the PAL chain is  
program aided language model chain. it, what it will actually do is turn  
whatever the question is into a Python function and then run that Python  
function and return back the answer. for that. So you can see here. we had 23 apples. This is a standard one. I've used  
a lot 23  
apples, minus 20 plus six. should we nine? Right? We can just turn it into  
a Python function here. The cafeteria had 23 apples. if they use 20 for lunch and bought  
six more, how many apples do they have? Apples initial 23 apples  
used 20 apples bought six. It turns it into a math equation. and then returns the result and  
sure enough, it basically gives us the roll Zola out there being nine. So one of the ones that a  
lot of the models used to get wrong was this time one. And it's interesting that. this one kind of understands  
it's interesting to sort of see what it actually gets here. So let's look at the question. If you wake up at 7:00 AM and it  
takes you an hour and 30 minutes to get ready and walk to school at  
what time will you get to school? So it should be 8:30, so we can see  
that, it's got minutes, it's got this. the arrival time, it kind of gets it

right, because it's saying like 8.5, which probably means eight and a half. So, you could translate this as being, wrong or right. Depending on, On what you're doing here. but It's certainly better than a lot of models, which will either just throw an arrow with this one or we'll return back, you know, some random number in here. So this is definitely a good sign to see that some of these PAL chains are actually working quite nicely with the Gemini pro model. in here. All right. Finally, let's have a look at how you would do multimodal stuff. So here, I'm just bringing in a picture. bringing the same picture from the video before of the earth. And what I'm going to do is I'm going to ask it. So I'm going to basically ask it a question about this picture. I need to make sure that I'm bringing in this. Google provision model here that I'm not just using the text model, obviously now. Doing the. the vision model in here. And one of the key things here is that. I'm passing in. a URL with this. So now I could here I'm passing in this sort of public URL. I actually have a few different choices in here. I could pass in a public URL. I can pass in. local URL. So if I've got it on the drive, I can pass the path to that. I can also pass in a Google cloud storage bucket URL if I've got it set in there. So, that would be quite easy if you're doing an app and you've got people. giving you. Images, you would upload those to a Google cloud storage bucket quite easily. And then you would get that return path and then be able to do any Q and a that you want over that. that's obviously gonna help, be quick with the model as well In this way, the model is actually downloading the image from the internet. as we go through it. the last one thing that you could also do is you could do a base 64 encoded image and pass that up as well. in here. So I'm basically just passing this in. you can see that I've asked it. Okay. What is in this image? And who lives there. The image shows the earth as seen from space. There are clouds, land, visible water through about 8 billion people living on the earth. So it's answered what I asked there. So there are a bunch of different things

that you could do with this kind of thing. Maybe we'll look at in the future. If people are interested in doing a sort of multimodal rag, where we could actually have, you know, images being looked up, at doing things where we were, could ask it questions and get it to find different things like that. in here. But anyway, this gives you a sort of quick sense of how to use LangChain. with Gemini pro. and with the Gemini series of models. to be able to build apps and get various tasks going with this. anyway, as always, if you've got questions, please put them in the comments below. If you found the video useful, please click like and subscribe. And I will talk to you in the next video. Bye for now.



## **LangGraph Crash Course with code examples**

Okay. So in this video, I want to

have a look at LangGraph. so I'm going to talk a little bit

about what it is, and then I'll go through some coding examples of it. So if you are interested in building

LLM Agents you will want to learn this and then maybe over the next few videos,

we can look at going more in depth with building some different agents

and some different, use cases here. so first off, what actually is, LangGraph? you can think of this as sort of the

new way to run agents with, LangChain. So it's fully compatible with the

LangChain ecosystem and especially, can really make good use of the

new sort of custom chains with the LangChain expression language, But

this is built for running agents. So they talk about the idea of

being a graph and what they're talking about, you know, a graph

here, is where you've basically got nodes joining to different edges. and they're not always going to be directed. So this is not a DAG or a fixed directed graph in any way. This is basically where nodes can make decisions about which node they can go to next. So another way of thinking about this is it's like a giant state machine that you're building, where the graph is basically the state machine that decides, okay, what state are you in now? What state will you go to run a particular chain or to run a particular tool, et cetera? and then, how do you get back? And then also things like, how do you know when to complete or end the graph or end the sequence, in here? So LangGraph is built on these ideas of trying to make it easier for you to build, custom agents and to build, things that are more than just simple chains, with LangChain So there are a number of key parts to this. You've got this idea of a state graph. So this is where your state is being persisted in some way throughout the agent's life cycle. And you can think about this as a sort of way of passing the dictionary around from chain to chain or from chain to tool and stuff like that, and then being able to update certain things. And you can update things, where you can just overwrite them, or you can add to them. So if you've got a list of things like immediate steps, you can basically add, to that as the agent is actually going through running the various parts of the graph, et cetera. The next part, which is key to this, is the whole idea of nodes. As you build the graph, you want to add nodes to the graph. And you can think of these nodes as being like, chains, or actually they're also runnables, so it could be a tool, it could be a chain, and you can have a variety of these different nodes. And you think of those as being like the components of your agent that you need to wire together somehow. So while the nodes are the actual components, The edges are what wires everything together. And the edges can come in different forms as well. So you can set an edge where, it's just going to always go from this node to this other node. So if you have a return from a tool going back to the main node, you're just going to let that, you're going to want to probably hardwire that in there. But you can also then set edges

which are conditional edges. And these conditional edges allow a function, often going to be, the LLM, to actually decide which node that you go to next. So you can imagine that, this can be useful for deciding if you're going to go to a tool, what tool you're going to go to. If you're going to go to a different sort of persona in the agent. Let's say your agent has got multiple personas and you want to go from one to the other, or you want to have a supervisor that's basically delegating to different, personas. all those things are going to be on sort of conditional edges, that we go through. Now, once we've set up these nodes and we've set up these conditional edges, you then basically want to compile the graph, and now the graph acts, just like a sort of standard LangChain runnable. So you can, run invoke on it, you can run stream, etc. it will now basically run the state of the agent for you, and, give you the entry point, you will define an entry point as an entry node, and give you the sort of end point, and it will be able to get through the whole sort of thing of wiring these things together. Now I can see that there's going to be a lot of use for sort of making, reusable, sort of agents that you would then wire together on a graph. So you might have lots of, little pre made things for using tools, that kind of thing. And you could imagine also that you've got agents that use certain kinds of prompts based on, the inputs that come before them here. what I want to do now is go through some of the code. we'll look at some of the examples that they've given. I've gone through and changed them a bit just to highlight the, what's going on, and then we'll also look at it in LangSmith's. So we can actually sort of see what actually happens at each step and what gets, sent out to the large language model, etc. You'll find that I'm using, the OpenAI models in here. There's no reason why we

can't use other models. The only, I guess, challenge is that a lot of those models need to support function calling. If you're going to be using function calling on these. Now, if you're just running sort of standard chains or something where you're not using the function calling, you could use any sort of model. but if you want to have the parts where you're using function calling to make the decisions and stuff. then you're probably looking at models like the OpenAI models, like the Gemini models. And now we're starting to see some open source models that can do this function calling stuff as well. All right. Let's jump in and have a look at the code. All right, let's start off with the simplest, sort of example they give, which is probably not that simple in some ways. the agent executor. So this has been around in LangChain for quite a while. you can think of it as a way of, building an agent where you can then use function calling to get, a bunch of responses in here. So, what I've done is I've taken the notebook, I've sort of changed it a bit. I'm going to make some things a little bit, simpler and I'm going to add some more things to it so we can sort of get a sense of, okay, what actually is going on in here. so first off is basically setting the state. So I've left a lot of their comments in here. There are a number of key things that you want to persist across the actual agent while it's running so in this case, they're persisting the input, they're persisting a chat history. so this is more a sort of traditional way of, adding the memory and doing that kind of thing. you'll see in the second notebook, that we move to more just a list of messages going back. But this is using more of sort of a traditional way of having a chat history, and then having things like intermediate steps here. And you'll see that some of these, things, can be basically overwritten. So this agent outcome, gives us the outcome, from something that the agent did, or gives us this agent finish of

when the actual, agent should finish, So in this case, this can be overwritten as a value, in here. Whereas things like the, intermediate steps here, this is basically a list of, steps of agent outcomes, or agent actions, rather, and then show the results of those actions. And you can see in this case, this is being, operator.add. So this is just adding to the list as we go through it. So this state that you start out with, we're going to pass that in to make the graph later on. Alright, now what I wanted to do is set up some custom tools in here. many of you have seen, custom tools before. I did some videos about it a long time back. I probably should have done some more videos updating, as things in LangChain changed, for it. But, if you think, custom tools, You can basically, pick a bunch of pre made tools from LangChain, and there are a lot of those already. but you can also do, custom tools. So here I've made two sort of, silly little custom tools. and one is basically just going to give us a random number. between zero and a hundred. And the other one's just going to take the input of whatever we've got and turn it into lowercase, right? So these are very simple functions. You can see here we're using the tool decorator to basically convert these into tools. And then when we do that, we're getting the name of the tool. We're getting the description of the tool, in this way. so it's a nice way of just quickly making tools, in here. and you can see that when I want to run these tools, I basically just say, whatever the tool is or the function and then basically just dot run. in the case of random, I'm having to pass something in, so I'm just passing in a string. Really, the string can be anything in here. It doesn't really matter. you'll see that the agent likes to pass in random. So I've given this as an example here, but really it could be an empty string, it could be, a string with whatever in it. so in this case, the input

is not that important. In this case, the input is important. in this case, we're basically, if I pass something in uppercase, it will be converted to lowercase, right? So whatever the string gets passed in, that will be converted to lowercase and then passed back out. Now, they're simple tools you could change with this with, a bunch of different things like search DuckDuckGo , Tavily is what they originally used in here. but I kind of feel these are nice simple tools where you can go in and then see very clearly what is it That's going on is, you know, what I think going on rather than getting this long JSON back, of a search or something like that. All right, next up, we've basically got, the way of making, an agent. Now, remember a graph can have, multiple agents. It can have multiple parts of agents, can have multiple chains in there. in this case, this is the the sort of agent, the standard sort of agent, which, Basically uses, OpenAI functions, right? You can think of it as an OpenAI functions, agent here. So here we're basically pulling in a prompt, this is they had originally where they're pulling in the prompt from the hub. if we go and have a look at that prompt, we can see that there's really nothing special in there, right? It's basically just got a system message saying you are a helpful assistant. It's going to have a placeholder for chat history. It's going to have a human message, which is going to be the input. It's going to have a placeholder for the agent scratchpad in there. So that's what we're getting back, from that, when we just pull that down from the hub. we set up our LLM. And then we've got this, create open functions agent, which is going to be, an agent runnable in here. So I've passed in, the LLM, the tools, the prompt that we're getting back here. And then, you can see that, that if we actually look at the prompt, it's, it looks quite complicated because it's got a bunch of different, parts going on in there. And we can look at the prompt two ways. We can just look at it like this. We can actually, get prompts and stuff like that. And you'll see that now, if I've got that agent, I

can just pass an input into it, and I need to pass in a dictionary, right? So I've got a dictionary with my input text, I've got a chat history, I've got intermediate steps. Neither of those have got anything in here. All right, so we've got these inputs, we pass this in, and you can see that the outcome that we're getting from this is that we're getting an agent action message log response back. So this is basically telling us, give me a random number and then write in words to make it lowercase So you can see that, all right. What's it doing? It's basically deciding what tool to select via a function call. So, if we come in here and look at Langsmith, we can see that when we actually passed this into the LLM, we were passing in these functions and tools in here as well, right? So we can see that this has got, the, details for, that tool, we've got the details for the random number tool, and they've been converted to the OpenAI functions format for us in there. we then basically have got our input, our system input, and then we've got the human input there. And we can see the output that came back was this function call saying that we need to call random number. And if we look back here, we can see that, okay, it's actually passing back that we're going to call random number with the input being random, and we've got a message log back there as well. Alright, so this was a basically one step in the agent. so this hasn't called the tool for us, it's just told us what tool to actually call in here. So that's showing you what that initial part does. Now we're going to use that as a node on our graph. And we're going to be able to go back and forth to that between that particular node and the tools node as we go through this. first off we want to set up, the ability to execute the tools. So we've got this tools executor here. We pass in the list of tools that we had. So remember we've got two tools, one being a random, number, generator and one being, convert things to lowercase. If we come up here and we look at, okay, the first thing we're going to do, what are we going to put on this graph? We're going to put in the agent, but we're actually going to run the agent. So we've got that agent runnable invoke, and then the data that we're going to pass in. and you can see that we return

back The agent outcome from that. So in this first case, that, agent outcome is going to be telling it what tool to use. If we put the same inputs that we had before there. we've then got a second function for actually running the tools. so you can see here that this is going to basically, get this, agent outcome. that's going to be our agent action, which is going to be what tool to run, et cetera. And then we can run this tool, executor function and just invoke this with the, telling it what tool and what input to pass in there. Now I've added some print functions in here just so that we can look at, okay, the agent action is what actually it is, and also then the output that we get back from that. Finally, when we get that output back, we add that to the intermediate steps there. The next function we've got is for dealing with, okay, do we now, so remember, each of these can be called at different stages, even though I'm going through, the agent tools and stuff like that. these are separate things, at the moment. the next thing that we've got, this function, is basically determining, okay, based on the last, agent outcome, do we end or do we continue? if it's going to be like an agent finish, then we're going to end, right? We're not going to be doing something. So if it's coming back where it's saying, giving us the final answer out, we don't need to go and call tools again. We don't need to call another language model call again, we just finish, there. So these functions you're going to see, are what we're going to add in here. So first off, we've got our, workflow, which is going to be the state graph. And we're passing in that agent state that we defined earlier on. We're then going to add a node. for agent and that's going to be running the agent there. We're going to add a node for action and we could have called this actually tools, right? tool action or something like that. This is going to be that the function that we've got here for actually, running the tool and getting the response back and sticking it back on immediate steps like that. Alright, so they're the two

main nodes that we've got there. We set the entry, node in here. So we've got this, entry node. We're going to start with agent,

because we're going to take the inputs, we're going to run that straight

in, just like we did above there. and then you can see, okay, now we need

to basically put in the conditional edges. the conditional edges is where we're

using this function, should continue. and we're basically saying, that, after,

agent, The conditional edges are going to be, okay, should we continue or not? you'll see down here, I'll come back to

this in a second, but you'll see down here we've got a sort of a fixed, edge where

we always go from action back to agent. So meaning that we take the output

of the tool and we use that as the input for calling the agent again. But then the agent can then decide,

okay, do I need to use another tool? Or can I just finish here? And that's what this conditional edge is. So after the agent, it will decide, if

I ask it something, that's totally not using any of those tools, it's just

going to give me a normal, answer back from a large language model or from

the OpenAI language model in here. But if I give it something where If

it's going to be an action, then it's going to, continue, and it's going to

go on to, to use the tools and stuff, in there, So here, we want to sort of

decide, this is this sort of conditional edge part, and you'll see this in,

one of the other notebooks that this can, get a lot more complicated if

you've got multiple agents, going on the same graph, as we go through this. All right, we then compile, the graph,

I've tried to go with the terminology as much as, as they've got here of

like workflow and stuff like that. But really this is the graph. We're compiling it to

be like an app in here. If we look at it, we can actually

see the branches, that are going on. If we look at it, we

can also see, the nodes. that are on this and the edges that are on

this so we can see, okay, what goes, from what, And we can also see the intermediate

steps, of how they're being persisted on that graph, okay, so now we're

going to basically stream the response out so we can see this going through. I'm basically just going to take this app, remember I can do dot invoke, I can do dot stream, And I'm going to pass in the inputs. The inputs here are going to have an empty chat history. but I'm going to pass in, the input basically saying give me a random number and then write in words, should be write it in words, but anyway, write it in words and make it lowercase. So you'll see that, all right, what happens here? So we start off and it decides, ah, okay, I need a tool, right? So its tool is going to be random number. and in this case, it's putting the input being random number. it then gets that response back. Now I've printed this. So that it then basically sends that to the tool, right? so you see each of these is where we're going from one node to the next node that we're printing out here. so we go from this agent run node, the outcome being that, okay, I need to run a tool. coming back and then in this one we're going to, basically now have it where, we've run the tool. It's given us back a random number. The random number is 4 in this case. And so now, it's going to stick that on the intermediate steps in there. So now, the, that's going to be passed back to our original agent node. And now it basically says, okay, this was my initial sort of thing. I've got this number back. Oh, I need to write it in words. And then I need to make it lowercase. So to make it lowercase, I need to use the tool. And the tool is lowercase in this case, right? So the input is gonna be four with all capitals, and you'll see that the lowercase that we're getting out here, is gonna return back. Somewhere here we'll see this. It's going to return back, yes, here, we're going to see the tool result is 4, in lowercase there. So again, this is a tool, this is the straight up agent, this is the tool, this is the initial agent again, this is the tool, and then finally we go back to the agent again. And now it says, okay, now I can do agent finish. Because I've done everything that, I was asked to do in there, I've got the random number, I've got it in

words, I've got it in lowercase, here. So we can see that the output here is, the random number is 4, and when written in words and converted to lowercase, it is 4. All right, so it's, a bit of a silly sort of task to do it, but it shows you how it's breaking it down. And we can see, if we look at the intermediate steps that we're getting out there, we've got the steps, for each of the different, things going along. We've got the message log and stuff as we're going through this. All right, if we wanted to do it without streaming it, we could do it like this. if I just say invoke. I'm not going to see, each agent broken out, I'm just going to see, okay, the first off, it's going to pick the random number tool, get 60, Takes that as a word in uppercase, puts it into the lowercase tool, and we get the result out in here. Now, I've saved the output to output in this case. So remember, these are print statements that I put in there. That's why we're seeing this, come along. and then we've got this agent, get agent outcome, if we return values and get the output, we can see the random number is 60, and in words it is 60 all in lowercase. If see the intermediate steps, we can see the intermediate steps there. Just to show you, sort of finish off, if we didn't put something in that needed a tool, if we just put in the, okay, does it get cold in San Francisco in January? it comes back. Yes, San Francisco can experience cold weather in January. So now, notice it didn't use any tools. It just came straight back with a finish. and there's no intermediate steps, right? We've just got that one, call going on in here. So if we come in here and have a look in, LangSmith, we can see, this going on. So we can see that, okay, we started out with that call. It basically gave us a return to a call random number. We got random number. We got four out of that, from that, we then basically went back, so that, remember the action is like our tools, we went back to the agent, and we can see that, if we look at the OpenAI thing here, we can actually see what was getting passed in here, and we can see that, okay, it's going to come back that the input is four in capital

letters, We'll go into the tool lowercase here, this is going to transform it to just deliver back 4 in lowercase. And then finally, we're going to pass that in, if we look at, here, we're passing all of that in. With this string of okay, what we've actually done in here as well. And now it can say, okay, the output is going to be the random number is four. And when written in words, it's converted to lowercase four, right? And you can see the type that we got back was agent finish. so that's what tells it not to continue, as we go through this. All right, let's jump in and have a look at the second example. So the second example in here is very similar. The big difference here is it's using a chat model and it's using a list of messages rather than this sort of chat history that we had before. So we've got the tools here. Now, one of the things with doing it this way is that we're not using, the, createOpenAI functions agent here. So we are using an OpenAI model, but we need to basically bind the functions to the model. So we've got the model up here, and we can basically just bind these. so we just go through for each tool that we've got in here. We run it through this format tool to OpenAI functions format, and then we basically bind that to the model. So meaning that the, model can then, use that and call back a function just like it did before in there. Alright, we've got an AgentState again, like we had before, this is the sort of state graph, that we had here. In this case, the only thing that we're going to have though, is just messages. We don't need to have the intermediate steps we're not doing, any of that stuff, and because the input is already in the messages, we can actually get at that here, so we don't need to, persist that either. All right, our nodes, so we've got the should continue node again, so this, again is going to, decide whether we, go back to the sort of original, agent node or whether we go to the tools node, and in this case, you can see that what it's actually doing is it's getting off that, the last message that we got back. And it's using that to basically

see is there a function call in that or not, If it doesn't have a function call, we know then that's not using a tool, so we can just end. If it does, we can then continue. We've then got basically calling the model, so this is, taking our messages, passing this in and, invoking the model. We're going to get back a response for that. so we've got, this response, and we're just putting that response back in there. we've got a function for calling the tools. okay, here again we're going to get the last message. because this is what's going to have the actual, function, that we need to call, or the tool that we need to call. So you can see that we get that by just getting last message, looking at function call, getting the name, and then that basically pass that back of what the tool is. And then same kind of thing for getting the tool input in here. and then here I'm just basically printing out that agent action, is that action again, so we can actually see, what's going on. And the response back, The same as what we did in the previous one. and then, we're gonna basically, use that response to create a function message, which we can then basically, assign to the list of messages, so it can be the last message that gets taken off and passed back in to the agent again. alright, we've got the graph. here, same kind of thing, we add two nodes, we've got one node being the initial sort of agent and one node being the tool or the action that gets called, here. We set the entry point to agent again. We've got our conditional, the same as we had before as we go through this, so we've got a conditional edge and we've also got a hardwired edge, being that always from action, we always go back to agent, in this case. Compile it, and then now we can just run it. So you can see here that we can, I'm just going to invoke these as we go through it. But you can see that, I've got, give me a random number and then write the words and make it lowercase. We can see we're getting the

same thing as what we had before. So this is replicating the same kind of functionality, but, you're going to find that in some ways this can be, this allows you to do a lot more things in here, in that if we were to come in here, you see how we're popping off the last message when we come in here. We're also, able to basically summarize messages, we're able to play with, the messages, we're able to limit it so that we've only got the last 10 messages in, memory so that we're not making our calls, 35 messages long or something like that. Even with, the GPT 4 Turbo, we can go, really long, but we don't probably want to waste so much money by doing, really long, calls and using up really large amounts of tokens in the context window there. so we can run that through. you can see here, we've

asked for the random number. Sure enough, it's done the same thing. It's got the tool. and remember these are coming from the

print statements that I put in there. and, we're then basically getting this output in here. so that's the output of the whole thing that's coming back with the various messages that we've got going through and those messages, everything from, the system message to human message to the AI message to a function message to an AI message again, to

a function message back to an AI message for the final one out there. if we just want to try it

where it's just using one tool. . So if you and I have put in, please

write Merlion in lowercase, you can see now it just uses one tool, just uses the lowercase one, goes through and does that. And then again, if we want to just try it with, no tools if I ask it, okay, what is a Merlion? A Merlion is a mythical creature with a head of a lion and a body of a fish. Alright, so this sort of shows you. that it can handle, both using the tools and not using the tools. And it also shows you that each time though, we're getting these list of messages back, which is, the way of us being able to see what's going on and persist the

conversation as we go through this. Okay. In this third notebook, we're going to look at the idea of building a sort of agent supervise us. So where you've got it so that the user is going to pass something in the supervisors, then going to decide, okay, which agent do I actually route this to? and then it's going to get the responses back. and some of these agents can be, tools. some of them can be, just other, agents that actually, are not using a tool, but a using a large language model, et cetera. So let's jump in. so we've got the same inputs that we had before. I'm setting up LangSmith in here. I'm bringing in the model. So the model I'm going to use for this one is GPT-4. And then we've got a number of tools in here. I've got the, my custom tools that we used in the first two notebooks. So the lower case and the random number there. but we've also got the PythonREPL tool, right? Remember, this is a read, evaluate print loop, a tool. So basically you can run Python code. So you always want to be a bit careful, of, what prompts you're letting go into that because, it can be used maliciously. Obviously, if it can run anything that Python can run, it can do a lot of. damage in there. All right. So then in the example notebook, They've got these helper, utilities. this is basically for making a general agent and you can see that we're going to pass in the LLM. We're going to pass in the tools that the agent can use. We're going to pass in a system prompt For this. And it will then basically assemble it with the, messages with the scratch pad et cetera. and it's going to make that, create\_openai\_tools agent Just like we had in the first notebook, that we went through and it's going to then return that, executor back in here. So this is just a way to sort of instantiate multiple agents, based on, their prompts and stuff like that. So the second helper function here is this basically this agent node which is for converting, what we got here that, creating the agent into an actual agent node, so that it can be run in here it's also got a thing where it's going to take the message and convert it to being a human message. because we've got multiple, agents which

are going to be LLM responses and stuff, we're often going to want to convert those to be human responses to get the sequence of responses as we go through this. Alright, next up is creating the agent supervisors. So, this case, is where you're going to determine your multiple agent personalities and stuff like that. So the personas I've got here. I've changed their example. So I've got the lotto manager, which is obviously going to use the tools that we had before of the random number, et cetera. And we've got a coder. So I've stuck to the original example that they had of having a coder that will make a plot out of this. but what we're gonna do is plot out the lotto numbers for this. So we can see here that this supervisor has got a very sort of unique prompt, right? It's basically that, you're a supervisor tasked with managing a conversation between the following workers. And then we're passing in the members. So the members is this lotto manager and coder. given the following user request respond with the worker to, act next. So each worker will perform a task and respond with their results and status. When finished respond with finished. So this is what's guiding the supervisor to decide the delegation and to basically decide when it should finish. for these. So for doing that, delegation, it's going to use an OpenAI function, and this is basically setting this up. So this is setting up like the router for deciding the next roll of who should, do it. and then, passing these things through. and passing in like this enum of, the members and finish, so it can decide, do I finish? Do I go to this member? Do I go to this other member as we go through this? And you can see that because that is its own call in here, we've got a system prompt there that says, given the conversation above who should act next? And then we basically give it the options of finish lotto manager or coder in this. And then basically just putting these things together, making this supervisor chain, where we're going to have this prompt, we're going to bind these functions, that this function above that we've just had to the LLM and then we're going to pass that out. getting that back .

So hopefully it's obvious that

will then become like a node on the actual graph as well. So now we're going to look at,

actually creating the graph. So we've got this agent

state, going on here. and so this is our graph state. again, we're going to have the messages

that we're going to be passing it in. So we're sticking to that sort

of a chat executor like we did in the second notebook there. And you can see here that we're going

to basically have the lotto agent. So I'm just going to instantiate

these with that helper function for create agent. And so here, I've got, a lot of agents

going to take in our, GPT-4 turbo model, it's going to take in the tools. And then the prompt for this is you

are a senior lotto manager, you run the lotto and get random numbers, right? it's telling it that, Hey,

this is the agent to do that. It's telling that it's going to have

to basically use the tools to do that. so that's the lot of agent. And then the second agent

is this coder agent. So this coder agent

is just using the tool. So I passed in all the tools

in here for tools, by the way. And this particular agent is, just

going to use the PythonREPL, tool. And this is basically saying you may

generate safe, Python code to analyze data and generate charts using matplotlib. So it's just setting it up

to do the charting in there. So if you look carefully, you'll actually

see that, I think I accidentally passed in the PythonREPL into these tools

as well, So it's not ideal in that we would want to limit, the number of

tools that we pass into something to as few as possible, one, it saves on

tokens and two, it just makes it easier for the model to make the decision. But anyway, we've got those. and then we've

got, this basically

setting up the node here. And so we've got our lotto node. We've got our code node. we can then basically pass

these in as we go through this. We need some edges. So the edges we've actually

got a lot more edges cause we've got a lot more nodes now. and you can see that they're just

using a four loop to make these edges. So, from every agent or persona,

whether it's the lotto manager, whether it's the coder, it always goes back to the supervisor. So even if we had 10 different agents, as you can see we've got to being lotto manager and coder. it will go back to the supervisor at the end of that. and then we've got conditional ones. where it will determine, this is sort of setting up a conditional map for, The conditional edge of being the supervisor going to what? So, this conditional map, in fact, maybe in the future example, I would just hard-code this out so people can sort of see what's going on here. But basically it's just making a dictionary in here. it's adding in the, finish node in there as well that it can basically use as a condition. and we can see that we can go from supervisor to any of those on the conditional map, which is going to be our members and is going to be finished in there. finally we set up the entry point. So the entry point is going to be the supervisor. Compile a graph and then we can use the graph. So you can see now when I've asked it to do is human message in, get 10 random lotto numbers and plot them on a histogram in 10 bins. And tell me what the 10 numbers are at the end. So this runs through. it does the plot for us. So we don't really see that much here. But let's jump over to LangSmith and see what's going on here. if we look at the LangSmith for this, we can see that it starts out. and we've got the router as the actual, function calling thing at the start, right? Not the tools. This is the router that is basically deciding, do I go to lotto manager? Do I go to coder? Do I go to, finish of this. We pass in now prompt there and you can see now it's got the workers being lotto manager, and coder. which got, you know, put it in there. and then we've got, when finished respond with finish. and then we passed in the actual sort of human prompt. And you can see that it's decided that okay, from this select one of these. It's a solid, okay, need

to go to lotto manager. So that's where we get to lotto manager. Now, lotto manager. basically it looks at this and now it's getting tools in there. So remember I said, I accidentally passed it in the PythonREPL in here. I probably shouldn't have done that. But anyway, we've got, you're a senior lotto manager, get 10 random lottery numbers. Were passing in that, in there. And you can see it's going to, it's worked

out that, okay, it needs to do this random thing and it needs to do it 10 times. So it goes through and runs. the random number tool 10 times. So we get 10, separate, random numbers back. from that. it, then, can take those. and decide, okay there's our 10 numbers back that we got. and it can decide, okay, now it needs to go to the coder. now, in this case, actually, because it had the PythonREPL in here, it just did it itself in here. But you'll see on some of them, we'll actually go back to the coder in there. and then finally, we've got the supervisor out, which is giving a lot of numbers out. telling us that we can't see the, the plot, we can't pass the plot back cause it's already plotted it out. Here is our plot out. and if we went along, we can see that. here are the numbers that correspond to the plot out that we've got there. Anyway, this is just running it. two times. If we look at the final response out, we can see that this is what we've got. if we want to actually just sort of give the human response back out, we can get this out. So we've got this, the histogram has been plotted for the following numbers, passing in the numbers with new line characters, et cetera as we go through it. Okay. So this shows you the sort of basics of building a supervisor agent that can direct multiple agents in here. So in some future videos, I think we'll look at, how to actually, go through this, more in depth and actually do some more real world agent things with this. and then from this, you could basically take it, you could deploy it with a LangServe. You could do a variety of different things with it to make a nice UI or something, for this. But hopefully this gives you a sort of crash course in what LangGraph actually does. And what some of the key components are for it. if you just think of it as

being a state machine, this is fundamentally how I think about it. if you've ever done any sort of programming for games and stuff, you often use state machines there. a lot of sort of coding will often have some kind of state machine. And the state machine is basically just directing things around this. so don't be intimidated by it. It's pretty powerful that, you can do a lot of different stuff. I would say You can get confusing at times when you're first getting your head around it. But once you sort of work out like how, you're setting up the different nodes, what the actual nodes are, how you're going to have conditional edges between the nodes and then what it, you know, what should be hardwired edges to basically bring things back is another way of thinking through this. So for me, I'm really curious to see what kind of agents people want to, learn to actually build. Agents is something that I've been interested in with a LangChain for over a year or so. And I'm really curious to see, okay, what kind of agents do you want? And, we can make some different examples of these. in the description, I'm going to put a, Google form of just basically asking you a little bit about what agents you're interested to see and stuff like that. If you are interested to find out more about this. fill out the form and then, That will help work out what things to go with going forward. anyway, as always, if you've got comments, put them in the comments below. I always tried to read the comments for the first 24 hours or 48 hours after the video is published and reply to people. so if you do have any questions, put them in there. and as always, I will see you in the next video.