Korey Stegared Pace (Microsoft)Full Course (Lessons 1-10) AI Agents for Beginners

Summary

This comprehensive video series, "AI Agents for Beginners," provides a foundational and practical guide to understanding, building, and deploying AI agents. Starting from the basic concept of AI agents, the course covers essential components such as large language models (LLMs), memory (both short-term and long-term), and tools integrated via APIs or functions, which collectively empower AI agents to reason, plan, and act on user requests. The lessons progressively delve into more advanced topics, including agentic frameworks, design principles, tool use patterns, retrieval augmented generation (RAG), trustworthy agent design, planning and multi-agent design patterns, metacognition, and finally strategies for deploying AI agents effectively in production environments.

The series emphasizes hands-on learning through coding examples using frameworks like Microsoft's Semantic Kernel, Autogen, and Azure AI Agent Service, illustrating how agents interpret natural language, interact with tools, maintain conversation context, and collaborate or self-improve over time. Key AI agent concepts such as system message frameworks for prompt engineering, handling uncertainty, multi-agent workflows, and reflective metacognitive behavior are explained with practical demonstrations. The course also highlights important considerations for security, privacy, error handling, and evaluation in production, helping developers create reliable, adaptable, and user-trustworthy AI agents.

Highlights

- 🤖 Introduction to AI agents: components include LLMs, memory, and tools.
- 🛠️ Exploration of popular agentic frameworks: Semantic Kernel, Autogen, and Azure AI Agent Service.
- 🔄 Demonstration of tool use design pattern enabling agents to call external APIs and functions dynamically.
- 📚 Explanation of agentic RAG, a multi-step retrieval-augmented generation process enhancing complex queries.
- 🔐 Emphasis on trustworthy AI agents using system message frameworks and human-in-the-loop architectures.
- 📋 Planning and multi-agent design patterns for task decomposition and collaborative workflows.
- 🧠 Introduction to metacognition enabling AI agents to reflect, learn, and improve from past interactions.

Key Insights

- 🤖 AI Agent Architecture Integrates Reasoning, Memory, and Tools: The fundamental AI agent architecture consists of a large language model for reasoning (task understanding and planning), memory for maintaining context and learning over time, and tools for executing specific functions or accessing external data. This integration allows agents to convert natural language input into actionable tasks, making AI agents more capable than static models or simple APIs. This is critical for real-world applications demanding dynamic, context-aware interactions.

- 🛠️ Agentic Frameworks Facilitate Control, Collaboration, and Context Management: Frameworks like Semantic Kernel and Autogen provide developers with structures to manage multiple agents, maintain environment context, and coordinate complex workflows. Azure AI Agent Service simplifies single-agent deployments and integrates well with other Azure services. Selecting the right framework depends on project scale and goals—enterprise production vs. research experimentation—highlighting the importance of tooling choice in AI agent development.

- 🔄 Tool Use Design Pattern Extends AI Agent Capabilities Beyond Language Models: Leveraging the tool use pattern, agents can invoke external APIs, databases, or code functions to fetch real-time data or execute domain-specific operations. This pattern is essential for tasks that require up-to-date or precise information (e.g., flight status, customer data) beyond static knowledge embedded in LLMs. The combination of natural language understanding and tool calling enables sophisticated, automated workflows and enhances reliability.

- 📚 Agentic Retrieval Augmented Generation (RAG) Enables Complex, Adaptive Query Handling: Unlike basic RAG which retrieves relevant documents to provide context, agentic RAG allows agents to break down complex queries into subtasks, use multiple data sources and tools iteratively, and verify if the retrieved information suffices to answer the query. This adaptive approach supports continuous improvement and long-term memory, making agents more effective at multi-step reasoning and dynamic problem solving.

- 🔐 Trustworthy AI Agents Require Clear Instruction, Transparency, and Human Oversight: The system message framework empowers developers to craft precise, repeatable prompts that guide agent behavior, responsibilities, tone, and interaction style. Incorporating human-in-the-loop architectures enables human intervention or approval for critical decisions, improving safety and trust. Transparent controls analogous to video playback options (pause, stop, feedback) help users maintain confidence and control over AI agents.

- 📋 Planning and Multi-Agent Design Patterns Support Scalability and Specialization: The planning design pattern breaks down complex user requests into subtasks assigned to specialized agents or processes, enabling modular and scalable AI systems. Multi-agent patterns such as group chat, handoff, and collaborative filtering facilitate communication and cooperation between agents, improving task completion quality and enabling sophisticated workflows like content creation and review cycles.

- 🧠 Metacognition Enables AI Agents to Reflect, Learn, and Personalize Over Time: By implementing metacognitive capabilities, agents can analyze past decisions, track user preferences, and adapt future behavior accordingly. This reflective process enhances user experience by reducing repetitive queries, personalizing suggestions, and improving accuracy. Designing agents with long-term memory and feedback loops is key to building intelligent systems that evolve with users and environments.

- ⚙️ Effective Production Deployment Involves Robust Evaluation, Error Handling, and Cost Management: Launching AI agents in production requires comprehensive evaluation at each step—from user intent recognition to tool integration and final response quality—to monitor performance and identify issues. Handling errors gracefully (e.g., fallback APIs) ensures continuity despite external service failures. Providing UI feedback mechanisms and leveraging both manual and automated evaluations help refine agents over time while managing operational costs.

- 💡 Hands-On Code Samples Are Crucial for Mastery: Throughout the course, practical coding examples in Jupyter notebooks using Semantic Kernel, Autogen, and Azure AI Agent Service demonstrate real implementations of core concepts. These samples encourage experimentation with different LLMs, plugins, system messages, and multi-agent setups, bridging theory and practice and accelerating developer learning curves.

- 🌐 AI Agents Represent a Paradigm Shift in Human-Computer Interaction: By combining natural language understanding, context awareness, tool integration, and multi-agent collaboration, AI agents enable more intuitive, efficient, and personalized user experiences. They go beyond traditional applications by autonomously reasoning, planning, and adapting, making them powerful tools in customer service, travel planning, research, and many domains.

Conclusion

The “AI Agents for Beginners” course offers a thorough pathway from conceptual understanding to practical coding of AI agents. It highlights the critical components, design patterns, frameworks, and operational considerations needed to build intelligent, trustworthy, and effective AI assistants. By leveraging large language models, memory systems, and external tools within flexible frameworks, developers can create agents capable of complex multi-step reasoning, collaboration, and continuous learning. Emphasizing hands-on examples and best practices for deployment and evaluation, the course prepares learners to harness the transformative potential of AI agents in real-world applications.

What are AI agents? What are the best use cases for them? And what do we need to start building them? We're going to answer these questions in this first lesson of the AI agents for beginners course. In this course, we're going to take you from concept to code, covering the fundamentals of building AI agents. And in this short video follows along with the written lesson including translations and a code sample that you can find the link to above and below this video. But let's get started

00:33

looking at the parts of an AI agent. And the first part is a large language model which pars the reasoning behind our AI agents. And what we mean by reasoning is being able to identify a task requested by users, create a plan to complete that task, and perform the actions of that plan. Next up is memory. This can be short-term, the context of the conversation between the user and the agent, or long-term, which is a collection of data that allows the agent to improve over time in completing the task. After that is tools.

01:18

This can be different services accessed by APIs that perform an action, data to help determine what action to take, or different functions that we will run to send information to the agent. And combining all of these things, an agent uses the LLM to recognize the task of the user would want to complete, identify what available tools are needed to complete that task, and memory to gather the information and data that's needed to complete that task. A simple real world example you do every day hopefully is

01:55

brushing your teeth. You need to plan out when and where to do it. Your tools are your toothbrush and toothpaste. And you have some in context memory of the current status of you brushing your teeth. And long-term memory, maybe your preferences of toothpaste flavor. I'm a Spearmet fan myself. Thanks for asking. But we aren't here to brush our teeth. We're here to build AI agents. So, let's head over to the code to do so. Okay. So now we're here in our code editor to run the first code sample attached to lesson

02:28

one. In this example, we're going to actually work with semantic kernel and a gentic framework from Microsoft and GitHub models, which gives us free access to large language models. If this is the first time you've seen a Jupyter notebook or you if you want to learn how to get set up, we actually have a chapter dedicated to setting this up and running the code in the course. But we won't focus too much on specifics of semantic kernel in this lesson as the next chapter will be a focus on the

02:55

different agentic frameworks, how they work and their differences and benefits. But the first thing I want to point you to is this class destination destinations plugin. This is basically where we start to list the functions or tools that we basically want the agent to being called. In this case, we have an agent that has an availability of different a list basically of different destinations around the world. And we also have a function that makes it that each time you call it, it will choose a

03:25

random city out of that list uh and give it to the user or basically return it. So, we're going to again not really focus too much exactly on how semantic kernel is working, but this is the syntax of adding that plugin or adding that tool like we explained earlier. agents have access to onto that agent as well as assigning the large language model which in this case I'm going to use GPT40 mini. Feel free to experiment with different large language models and see how they react. So we're going to go ahead and

03:54

now uh go to the code. We're going to actually simulate a little bit of difference like of how a user would be interacting with this agent. So the first thing is they're going to ask for a day trip. And I think this kind of really illustrates some of the powers of working with large language models is a user can kind of communicate in natural language and not just say I want a vacation for example but a day trip and a large language model should be able to interpret that this is uh asking or

04:20

maybe requiring calling this function this random destinations because it again the user has not really provided a specific city and the large language model is aware of how uh what uh tools are being assigned to it and And then we're going to also play with a little bit of the memory itself by allowing the user to say they don't like that first destination and they should plan another vacation. So if the agent has context again that in context of the memory of the conversations, it should not repeat

04:48

itself on the vacation destination. So if we go into um this now interaction, we can see the user saying plan me a day trip and then uh the travel agent comes in and says suggest Sydney. So again to confirm Sydney is a part of the list of vacation destinations that we assign into the function. Uh the user says they don't really like this destination and plan me another vacation. So sorry anyone watching from Sydney. Uh but then again though the agent comes back and plans a nice trip to Rio de Janeiro. Very

05:24

similar format. So again you can see that the agent is operating with that function that returns the cities. It's not just pulling it from every city it's aware of. Uh and it also is working with the context memory to know what the conversation is happening and displaying an alternative. So wherever you are, I hope you enjoy your trip to Sydney or Rio de Janeiro. But our trip on this journey in AI agents beginners goes to lesson two where we'll explore the different aentic frameworks out there.

05:53

See you in the next one. What are agentic frameworks? Why do we use them? And how to decide which one to choose? We're going to answer these questions in the second lesson of the AI agents for beginners course. In this course, we take you from concept to code covering the fundamentals of building AI agents. And in this short video actually follows along with the written lesson including translations and a code sample which you can find at the link above and below this video. So let's start by

06:30

defining what agentic frameworks are. And these are tools that enable anyone building AI agents to have a bit more control over task management. Because as mentioned in the first lesson, agents are focused on completing tasks. And we'll talk a bit more about this in our multi- aent chapter. But for some use cases, we need multiple agents working on a set of tasks and agentic frameworks help us to decide what agent will complete that task. Also contextual understanding. See, agents need to have information

07:16

about the context and environment state. If an agent, for example, is to book a hotel room, it needs to know that there are hotel rooms available to book. So, Agentic frameworks allow us to better manage this context. Also, agent collaborations See how these agents complete the task together is up to us to define and agentic frameworks allow us to do that more effectively. They create spaces and use protocols to allow agents to communicate between each other. These frameworks also come with tools or

08:12

connections to allow us to observe and evaluate our agents performance. So which one should you choose? There are many out there and in this course we have focused on three different agentic frameworks and services so you can get a better understanding how they work. First is the Azure AI agent service. This currently is only designed to be a sing used for single agents both through code and the UI and it integrates really well with existing Azure services and capabilities you might be already using.

08:44

We also off use two different agentic frameworks semantic kernel and autogen. Both these frameworks can utilize agents built with the Azure ai agent service as well as other services like GitHub models which we have code samples for throughout this lesson. Semantic kernel on one hand is an enterprise focused framework and the team behind it are really focused on a developer experience for teams building AI agents in production. It offers support for C, Java and Python and a variety of different connectors to other model

09:18

services. We also have Autogen which was born teams in Microsoft research. has a strong focus on taking the latest in aentic research and enabling other researchers and developers to test and experiment with those ideas in code. And as always, the best advice is to start small working with one agent through something like the Azure AI agent service. And then when you have an agents working, you can combine them by using one of the frameworks that has multi- aent support like semantic kernel or autogen depending if you're building

09:50

for production or just exploring the latest in aentic research with autogen. We have more information on how to decide which tool is right for you in the written lesson. And while listening to someone talk about this is helpful. Thanks for watching. By the way, getting hands-on is the best way to get a better understanding of how these tools work. So, let's head over to our code editor and to do so. So, now we're here back at our code editor and we have three different samples. One using semantic

10:19

kernel, autogen, and the Azure AI agent service. As mentioned, you can find this code at the link above and below the video to run it yourself. We'll start with the semantic kernel example. This is something we already kind of ran in the last lesson where you we be be able to define different plugins. In this case, we have this destinations plugin which takes a list of different destinations and allows a random destination to be returned whenever a user is asking for a planning of a trip. And again, we're defining that agent

10:53

using a chat chat completion agent, the kernel. So everything that we've been adding, so all the services and tools, uh we have some instructions and then any other settings, additional settings that we need. And basically we're going down here. We can see the uh that the user is planning a day. We can also see that it's actually making a call to that function. So get random destination uh and then getting the result which is New York. And then the agent is going ahead and then taking

11:22

that information from that function and making a plan for New York. Sounds like a lovely trip. On the autogen side, a similar sort of setup. Uh in this case, we're going to uh create a client, which is a the check completion of the model behind it. We can also do some different settings, whether it's like a JSON output, which is really important when we're talking about doing different working with different functions on different parts of our system. And then we can define the agent in a

11:51

way where you we have the name of the agent, the model client that we just defined. We don't have any tools here, but this is where Autogen allows us to define any custom tools. And then lastly, the system message, which is similar to Samantha Colonel's agent instructions. In this case, we're just going to ask the for a great sunny vacation. Uh so giving the agent a little bit more details here on, you know, the type of vacation that we want to have. Uh and then we see uh you know the interaction the agent comes back or

12:20

the assistant comes back with the plan and it's you know 7-day trip to Maui. Sounds lovely. So and lastly we're going to look at the Azure AI agent service. So again this is your first time seeing this service. We have a setup video uh that will actually show you how to set up these connection strings cuz this is how we're using this agent uh through the project that we've made in the Azure AI Foundry. And with the azure AI agent service, it's a little different setup. In this case, we're going to create an

12:53

agent kind of at the runtime in terms of interactions with the user. We're going to create a thread. And a thread basically is the setup between uh the messages that are being collected between the user and the agent. And in this case, we're going to start with the user's interaction asking for a bar chart of, you know, with the different amount of travelers and destinations. And the way that we're kind of also focusing on this as a uh in terms of the ability for the agent to use that is

13:22

that we have different tools built into the Azure AI agent service. And in this case we have the code interpreter tool. So this allows the agent to generate some code and in this case it's going to generate some Python code uh to actually generate bar chart. So we we see this thread and then we have this run status. So this is actually the uh interactions between once the agent start running the different tools and then we can see that it's actually created an image. Um, and then we can even display that image

13:51

right below here with a bar chart uh with the amount of the destination and the amount of travelers. So, go in, look at the code samples, play around with different settings, and like I said, we're going to be running different samples throughout this course, so you can really see the differences between the different services and frameworks that we're offering in this course. But we'll see you in the next lesson. What makes for a good AI agent? What are some principles for creating them? And

14:22

are there any examples of actually applying these principles? We're going to answer these questions in the third lesson of the AI agents for beginners course. In this course, we take you from concept to code covering the fundamentals of building AI agents. And this short video follows along with the written lesson including translations and a code sample that you can find at the link above and below this video. But let's get started at what makes a good AI agent. And there are three key parts.

14:56

First is space. This is the environment that the agent is working in. And in this space, you should focus on connecting events, people, and knowledge. An agent should be easily discoverable while being able to transition from being in the foreground and background based on the user's needs. To apply this principle, we can build this in the UI or UX of our applications, such as providing basic instructions on how to use the agent, like what the agent should be used for and if it has certain limitations.

15:33

Second is time. This is how the agent operates over time with the user. This is important because agents can improve over time if we design them well enough. And this can be done through connecting the agent to past events and enabling a reflection design pattern which we'll actually talk about later and implement in a later lesson. To apply this principle, you can show users their past prompts and inter interactions in the UI to show how the long-term context the agent is actually operating on. And

16:09

lastly is core. This principle is about brace embracing uncertainty. Since we're allowing LLMs to create plans and action on those plans, uncertainty is a key part of agentic design. Enabling trust and transparency with humans users by empowering them with visible controls and feedback tools is really important here. Much like you have controls over this video like pause, play, captions, and speed control for those like me who watch everything at 2x speed, your users should have a similar level of diverse

16:46

access of controls of an agent like turning it on and off. And so let's see how we can actually apply these principles in code. Okay, so we're at our code editor now. But we're going to continue from the last two lessons of using this sample using semantic kernel and the GitHub models. Again, this code sample you can run it on your own. You can find it at the link above and below this video. It's a GitHub repo. And just to refresh your mind on what this actually does is it takes a destinations

17:15

plugin. So what semantic kernel uses in terms of different tools and functions that we can call uh again we have a list of these tools I mean sorry destinations that are available. um for being able to book a trip and then we also have this function which is basically returning one a random location every time it gets called. So in in case to modify this uh to apply to the design principles we just spoke about we want to make it clear to the user about what is this agent is capable of and also give it

17:49

some suggestions in terms of uh what interactions we would like to see the agent complete. So in this case we have a very much a more specific uh agent instructions here both on you know being able to you start the opening interaction that the you're able to plan vacations also giving it kind of some tips around planning a day trip for a specific location allowing you to rand choose a random location or find destinations based on specific features. So a user could say that they want to you know travel to the mountains or

18:21

beaches and then even plan alternative trips if they don't like the the first suggestion. So again this is a good agentic design principles because a user kind of knows both the capabilities and the limitations of this agent first as they start to first interact with it. Uh so if we go in down here to see like what that interaction actually looks like. You know, the agent comes in, starts out with this hello, uh, you know, your travel assistant, they help plan vacations, gives you a list of

18:49

things that are capable of, and then similar to what we've done before, the user says, you know, plan me a day trip. And then they also the agent now that we've given this more a more specific system prompt is going to ask if they want to specify a specific location or you know again the user again from that first interaction can say I just want a random loc uh location. In this case the the user goes in says they would like to go to Bali which again sounds like a great place to go and then we get this

19:19

nice day trip itinerary from the agent. So that was how to apply some of the agentic design principles around prompting and being very transparent with the limitations and capabilities of an agent in code and we'll see you in the next [Music] lesson. What is the tool use design pattern? How does it help AI agents do more? And what should we plan for when using it to build AI agents? We're going to answer these questions in the fourth lesson of the AI agents for beginners course. In this course, we take you from

19:53

concept to code covering the fundamentals of building AI agents. And in this short video follows along with the written lesson, including translations and a code samples that you can find at the link above and below the video. So, let's get started looking at what the tool use design pattern is. And while while LLMs are very effective on their own in generating content, tasks that are requested by users sometimes require external tools to complete and the tool design pattern allows the LMS to interact with these

20:28

tools such as a calculators APIs to tell us about flight status or a function within our application handling currency exchange. To understand the benefits of this, let's take a look at a few use cases. For example, an AI agent can create database queries using tools for generating code and a be able to retrieve real-time information from a customer database of the customer's travel, loyalty points, and travel preferences. Or an AI agent can be connected to a CRM system and to answer specific questions about a customer's

21:02

travel booking rather than needing a human intervention to do so. And while we've been talking about single-use tools, agents can also combine tools in order to automate workflows. For example, analyzing an email, retrieving the relevant B knowledgebased information, and then forwarding it that email to customer service representative to make that process more efficient. There's a lot to consider when talking about tool calling. For example, security, making sure AI agents have only the type of access required and

21:34

handling errors because sadly we don't live in a perfect world where the services that the agent might be using are always running. Maybe one day, but we will cover these topics more in depth in our trustworthy AI agents lesson and AI agents in production. But let's head over to our code and build a tool calling AI agent. Okay, so now we're at our code editor and we're using the semantic kernel tool example with the notebook. Again, you can find this code at the link above and below this video.

22:08

And we've been covering tool calling in the other lessons as well, but I want to show you just a few other kind of options and abilities that we have within this kind of world of tool calling. And in this case, we have this destinations plugin. uh we're g using semantic kernel and we have one that just has a list of destinations that are available basically a list of destinations and then the other one is the actual availability. So you can kind of think of this maybe as two different databases or maybe one that has uh you

22:38

know a list of things that are destinations available and then the current status uh whether they're you know being able to be booked or not. So we will also show this in in terms of the function behavior. So this is another kind of setting built built into semantic kernel where we can have this either be auto. So it basically allows the agent to choose when to call the function or when not to uh whenever whenever it deems it appropriate or we're required when we want to make sure that the agent is calling a function

23:08

depending on the kind of use case and scenario that we have. Again we're setting up the agent like before. But in this case, we want to have three user inputs. And I want to show kind of the value of working with tool calling here. First, the one is the user input. The question is, you know, what destinations are available? Then it's going to ask about a specific destination. And then it's going to ask, are there any destinations available not in Europe? Uh, and we can look at the interaction

23:34

here to see how this goes. So, first is what destinations are available? And as expected, this is going to call the get destinations plug-in or function. And this result is going to give back our list. And then it can go ahead and show that list to the user. Um, and this again is just saying this is just getting the destinations, right? Not just specifically about the availability of them, but what ones can I actually book? And then it's going to actually the user is going to ask about is Barcelona available. So again kind of

24:05

similarly query but in this case instead of calling the get destinations since it give us a specific destination we can call the get availability and see that the result is that Barcelona is unavailable and the travel agent responds accordingly saying Barcelona is currently unavailable. But I want to show this as another kind of example of why uh you know we're not just making API API recalls here but also using the power of large language models to interpret natural language because in this case you know there are there any

24:35

vacation destinations available not in Europe. Uh we don't have to as developers are building these agents define which ones are in Europe or not because the large language model has all of that within it sort of base training data. So in this case you don't need to have a new function saying is in Europe or not in Europe or anything like that right we can just first that large model is going to call the get availability in this case it already has deemed that Tokyo Japan is not in Europe

25:04

and that New York uh is not also in Europe but it's going to call that specific uh function the get availability to see which ones are not available and then we get the result for both of them and then perfectly it says that New York is available and also gives an kind of extra information that Tokyo, Japan is currently unavailable. So again, this really shows you the power of using agents, large language models to both use their abilities to interpret natural language to have that sort of interaction as well as tool

25:33

calling to give it the specific data based on that interpretation and that plan from the agent itself. So that was tool calling. We're going to continue to use this as a feature in other samples, but we'll see you in the next lesson. What is identic rag? How is it different to basic rag? And how can we use it in our applications? We're going to answer these questions in this fifth lesson of the AI agents for beginners course. In this course, we're going to take you from concept to code covering the

26:07

fundamentals of building AI agents. And in this short video follows along with the written lesson including translations and code temples that you can find along the link above and below this video. So let's get started looking at what a gentigra is and we'll start looking at basic rag first. So rag or retrieval augmented generation enables LLMs to take a query made from a user and retrieve relevant information from a database. That information is then added to the context given to the LLM to provide more

26:44

relevant responses outside of just the trading data of the LLM. And while that framework isn't going away, a Gentigra adds a new level of capability to it. To start on the query side, the agent performs an analysis of the query to create a plan of smaller tasks to use the data sources and tools it will need to answer that query. And because while there is no such thing as a bad question, there are complex questions that require multiple steps and systems to answer. And after that information is retrieved from those

27:24

systems, the agent can verify if the information is enough to answer that query. If not, it can repeat this process of tool calling until it does. And what is really great about this is that the agent can maintain long-term memory of this process. So it can recall the previous attempts it tried to answer the query and improve the next time without taking those steps. And this is why I would have called it adaptive rag. But no one asked me when we were named these things. So aentic rag it is. That's enough talking.

27:59

Let's head over to the code and see how to build this. Now we're here at our code editor to look at the agentic rag sample. I am using the semantic kernel Azure search example uh because both we can now create an index using Azure AI search which we'll show you in the setup lesson uh how to actually set up to get the right credentials and again this code is located at the GitHub repo at the top and the bottom of this video so you can work with yourself on your machine. So the first thing I want to

28:29

point to is this class prompt plugin that we've created. Basically, this allows us to after the prompt or the request from the user is sent, we can actually augment the prompt so that it gives a lot more direction to the agent in terms of being able to retrieve information. So, we basically allowing this augmented prompt to come in uh and it basically essentially like rag where we will take the retrieve context. So whatever we retrieve from our Azure search database or index and then we're

28:58

going to take the user query and we're going to direct the agent to first review the retrieve context and see if it answers that query. If it doesn't, we're also going to direct it to say try any of the available plug-in functions because that's again the power of aentic rag. It's not only to do the retrieval part but to combine those elements to other tools and things to get information to answer the query of the user. So after that we will make a plugin just so that the the agent has another

29:29

option. In this case we have this class uh weather info plugin and this basically just returns a few different locations and an average temperature. So in a in an ideal world that when the user is asking about temperature we don't actually have that information in the Azure search because again maybe that changes quite often but we also have a plugin or another tool that retrieves that information and when a user is making that query we should be able to combine those things. Um so now we will also add these documents so uh

29:58

into our Azure search index. So we have these travel documents. Uh we're going to use our credentials that we we set up in the setup chapter. And then we basically have just some three or sorry five documents here. Um it has just some basic information around insurance, the destinations that are available. Um and even you know kind of the benefits of those destinations. Now we're going to go actually look at the interaction from the user and the agent. And I' I've kind of set this up so that we can kind of

30:27

see how different parts of the application are handling different things in terms of the agentic rag. So in this example, the user is asking around explaining a Kontoso's travel insurance coverage. And the context that it retrieves is a few different documents. So we've given it directions to pull in the top three. uh in this case you can see that you know obviously the first one is really relevant here in terms of really answering the question around travel insurance but the other ones might be able to support it. So you

30:55

can always see the context that the agents operating by using this drop down and then you can see the response that they the agent gives out. In this case this is asking about what the average temperature of the Maldes is. So it does a few different things. First it retrieves that this is a destination available or offered by Kontoso. But as you can see here it since the first review or the retrieve context doesn't answer that question we can actually go back or look at this function calls and

31:27

it's actually going to make this function call which is the get destinations. Uh so again this is the destination it's calling with Maldes and it's going to get back the average temperature of the Maldes. Um I'm not sure exactly this is correct. uh average temperature from all these. But again, this is just to show you uh the ability to first look at the context and see what available tools that it needs to be called. And then the last kind of context or last example is when we need

31:53

to handle both retrieving from rag perspective and the function call because like I said one of the things is that with the gentic rag is that we can handle more complex type of queries than just you know the basic rag. So in this case they're asking for a good cold destination offered by Kontoso and what is the average temperature. So there's two things kind of steps the agent needs to take right first to see what destinations Kontoso offers and then also the average temperature of that. So

32:21

if you look at this example first it's that the rag context it retrieves a document that says the popular popular destinations. So Maldes, Twis apps, Swiss Alps and African safaris. And then it also makes that function call to belt the Swiss Alps to see that this is a cold destination and it gives out the nice response which is a good cold destination offered by Kontoso is the Swiss Alps with the average. So we've done both a rag context as well as a function call. So that was that. Again,

32:52

you can run this code on your your side, play with the questions, maybe add some more documents, really experiment and seeing how the powers of a rag. But I'll see you in the next [Music] lesson. How do we build effective AI agents? What should we think about in terms of security? And how do we maintain users privacy when developing AI agents? We're going to answer these questions in the sixth lesson of the AI agents for beginners course. In this course, we take you from concept to code

33:25

covering the fundamentals of building AI agents. And in this short video follows along with the written lesson, including translations and code samples that you can find at the link above and below this video. So, let's get started about looking at building effective and trustworthy AI agents. And that's by covering this concept of a system message framework. As you may already know, the system message is one of the places that we as creators of AI agents can have the most impact and control

33:56

when working with LLMs. This is even more important when we talk about AI agents because we want to set clear instructions so that the AI agent performs the actions we want it to. One way to build effective prompts is in a scalable and repeatable way is using a system message framework. This starts by creating a system message for generating system messages for other AI agents. What this allows us is to use a basic system message like you're an agent responsible for booking flights and give

34:40

it to an LLM that has a system message template for generating better system me system prompts and get back a prompt that has a clearer and more specific instructions covering responsibilities, tone, and style and interaction instructions as well as any additional notes. When developing agents, it's rare when working with more complex scenarios that you will get the perfect prompt the first, second, or even third time. Building prompts is an iterative process and the system message framework allows

35:21

you to iterate better by making tweaks to the template generating system messages over time to improve across all your agents. The written chapter goes into more details about the various security threats. So do check that out. But another tool that I would like to cover is the human in the loop architecture. When working with AI agents, you might be building AI agents that require a human approval or intervention. And human in the loop enables this by adding a human user in the multi- aent cooperation.

36:03

We will cover more about this in the multi- aent chapter, but as a concept, your AI agents are instructed when the specific user statement like approve enters the conversation to perform a certain action like terminating the agent runtime. But before you terminate this video, let's head over to our code editor and look at this in practice. Okay, so now we're here at our code sample for chapter 6. Again, this code sample is available at the GitHub repo in the link above and below this video.

36:35

What I want to look at in this example is actually the system message framework concept that we just discussed earlier and kind of putting that into practice. So, what we have here is the role and three variables. The role travel agent, company, Ktoso travel, and responsibility booking flights. uh what we actually want to do is then use this as a kind of a a way to build a more sophisticated prompt for actually using an AI AI agent to do these sorts of things. So we actually have this message here which is going to system

37:07

message is describing that you're an expert at creating AI agent assistance. Uh and that's going to actually go in and then take in uh basically allowing it to sort of define the name role responsibilities and other information that might be useful uh for providing a system message to another AI agent. Then we're going to have a user message that basically just going in saying uh you know you are the role. So this case can travel agent company Ktoso and then you have responsibility which is booking

37:36

flights. So from this very basic definition of a a prompt we will should have a very more specific and detailed prompt that we can then use to build out agents. uh and in this case if you look at this example we have uh this system prompt AI assistant Ktoso travel the core responsibilities all outline in terms of being a flight booking assistant some personaliz personalization and optimization flight availability and pricing and travel requirements and policies. So helping and listening listing all these

38:08

responsibilities that the agent should be able to take care of and then even having some more notes in terms of uh you know making sure to focus on customer satisfaction and ensuring accuracy to minimize booking errors. Of course we can edit this if we think that maybe this is uh too many responsibilities because maybe we don't have the ability for an agent to do that. But then again, this is just shows you the power of using a framework like this because we can easily come in and kind of change some of these

38:33

responsibilities uh in terms of the agent and you know kind of duplicate and scale the amount of prompts that we have for our AI agent so we don't have to continue to rewrite this over and over again. So that was applying the system message framework. I encourage you again to look at the written chapter for other examples and we will see you in the next chapter of AI agents for beginners. See you there. What is the planning design pattern? How does it help AI agents complete task? And how can we use it with the agents

39:05

that we build? We're going to answer these questions in the seventh lesson of the AI agents for beginners course. In this course, we take you from concept to code covering the fundamentals of building AI agents. This is a short video that follows along with the written lesson, including translations and code samples that you can find at the link above and below the video. But let's get started talking about the planning design pattern. If this is the first time joining us, welcome. But if

39:34

you've watched other videos, you might have heard me talk about how a feature of agents is is to take a task and make a plan to complete it. The planning design pattern makes that more clear by having an AI agent list the subtasks that make up a more complex task. For example, generating a travel plan for a 3-day vacation somewhere warm can be broken down into subtasks like booking flights, hotels, transportation, and personal activities. And while we have a focus on using one agent to create a plan, where this really has

40:20

impact is working with multiple agents. In this case, not only is the plan divided into multiple subtasks, those subtasks are then completed by a separate agent or another process. To get this to work really well, we can also structure the output of the agent either using features of the LLM itself or through other tools. This is good because then another agent or system can process and parse that information. We can also add validation to the response. So we know we have all the information to continue working with

40:57

the steps of our agentic system. Well, I hope you processed all that information just now, but it might be easier to see it applied. So let's head over to our code editor and see this in action. Okay, so here is our code now code sample and our code editor looking at the chapter 7 example which again can be found in the GitHub repo at the link above and below the video. In this case, I want to first start pointing out that we're going to import this pyantic pattern uh sorry package which is

41:27

basically allowing us to apply some rules or validation uh of that data in terms of the structure that we will get back. And this is going to be super important again when we're using this the responses from the agent uh to either call other agents downstream or other systems. So we've imported that and then what this uh package allows us to do is a couple of things. So first we can define a subtask um basically structure where we have the assigned agent which is a string and it's going

41:58

to have the specific agent assigned to handle the task and then the task detail. So a description of what the actual task is. Then we're going to make a kind of model above that which is the travel plan itself. So there's going to be a main task which is going to be probably you know the overall travel request of the user and then a list of subtask and again the subtask going to have an assigned agent and the task details. So should this should be a very structured response from the agent uh

42:26

again then we can take that structured response apply some validation so we make sure that we kind of have all of the information that we need and then we can pass that on to other agents or other systems. In this case, we're also going to include in the agent instructions the available agents that that it has. So, flight booking, hotel, car rental, activities, destination info, and a default agent for general request. All right. And we'll have some brief descriptions of what those agents

42:52

do. So, we have we'll come in here and we'll see the user's input, which is to create a travel pen for family of two kids from Singapore to Melbourne. So after that u we can see the response that we get back which is uh a validated travel plan again because we're using pant we can do some of that validation and in this nice sort of JSON format the main task the assigned task flight booking so this is going to book roundtrip tickets uh this is going to then look at the hotel booking so assign

43:22

agents to the hotel booking and I think this shows also some of the powers of using AI agents versus a simple booking system or an API I call is I also add these details like finding booking a familyfriendly accommodation or familyfriendly activities and this is again because the power of large language models to understand natural language and you can think of that how powerful this is going to be if we want to then relay this to descriptions of hotels that we have in a database and we make sure that then this is a

43:51

family-friendly hotel or maybe we have a filter within our database or other tools where we can take this description of this task detail and then use that filter to find specific hotels that will match this travel plan. So that is applying the planning design pattern and showing how we can use large language models and AI agents to have a defined plan have defined data structure and then use that downstream for making other calls. So that's the end of chapter 7 or lesson seven and we will see you in the next

44:20

[Music] lesson. What is the multi- aent design pattern? When is a good time to use multiple AI agents instead of one? And when is what kind of control and visibility do we have when working with multiple agents? We're going to answer these questions in the eighth lesson of the AI agents for beginners course. In this course, we take you from concept to code covering some of the fundamentals of building AI agents. And this is a short video that follows along with the written lesson, including translations

44:52

and code samples that you can find at the link above and below this video. To put it simply, the multi- aent design pattern is where we have multiple AI agents working together to complete a common goal or task. But the real question is how exactly do the AI agents do that? And there are different design patterns that apply depending on their goal. There's group chat where just like a group chat with your friends and colleagues, every message in the chat is broadcasted to each agent and depending

45:33

on the contents of that message, a group chat manager, typically another AI agent, will select the appropriate AI agent to handle performing the task. For example, you might build an airline customer service application where there are separate agents that handle booking or customer complaints or questions about flight status. A customer could type their message into the chat and there would be a task request would be routed to the proper agent. But there may be scenarios where an AI agent needs information from

46:11

another AI agent before completing its task. And this is called a handoff pattern where there is an AI agent completing each step in a defined workflow before handing it off to the net. And lastly, there are cases where you want each AI agent in your application to perform some task. And this is called collaborative filtering. In this design pattern, each agent acts as a specialist in some task or area and responds to that task in their own unique way. And this is great when performing some type

46:49

of analysis where you want different inputs or perspectives on some data. And as you can see, multi- aent design patterns cover a wide spread of different use cases. But let's really see how it works by heading over to our code editor. Okay, so now we're in our code editor looking at the chapter 8 or lesson 8 example. Again, this code sample is available at the GitHub, GitHub repo above the link above and below this video. In this scenario, we're actually going to have two agents. one kind of producing content or writing

47:21

content and another agent reviewing that content. So this is kind of the reviewer checker pattern or again utilizing multiple agents to produce a better result. In this case, we're going to have one reviewer agent which is the concierge and we've given specific instructions to the concierge in terms of uh being able to f both focus on uh recommending recommending non-ouristy experiences for travel. So we want kind of this local and authentic experience uh that the a good concier should provide right and then we also got to

47:54

make sure that uh it doesn't actually uh you know provide any insights uh I mean also just only provides insights on recommendations of improving that suggestion but doesn't provide a specific example. So we don't want the you know uh the concier to just make new examples and we because what we would want to happen is we have this other agent which is the front desk agent and we've also given it kind of instructions to provide one single recommendation per response. So again allowing the

48:23

concierge to review that and make any suggestions but not new recommendations but we also then want the uh front desk engine to consider those suggestions and refine uh that idea that they have. So in this case uh we're going to say then uh what's maybe new with multi multiple agents here is that we have this termination function and this is going to be a function that we want to that essentially uh terminate the the interactions between the agents. In this case we have we've defined that this is

48:56

going to be when the concier has given approval of the most recent response. So the other of the most recent suggestion uh that the front desk when it meets their standards in terms of having a local and authentic experience and then we also have the selection process which is where we're going to define basically how this conversation will take place. So we're going to have that you know each participant takes a turn in the next conversation. Uh we're going to choose from these participants again the

49:24

reviewer and the which is concierge and the front desk. And then we've also defined what's going to happen. You know, the user is going to have an input or a question. It's going to go first to the front desk. Then the front desk is going to reply. Then it's reviewer's turn to uh do their instructions, which is going to review the front desk suggestions. After that, uh they're going to provide some feedback. And then it's up to the front desk to then uh suggest something new. So in this case,

49:50

then we want to go look at what this output will look like. And again, we have this input. And this person would like to go to Paris. Uh it was a general question and then the front desk is going to recommend the Lou Museum. The Louv is uh you know great museum and and the concier recognizes that says you know this is undeniably a worldrenowned site but it's also one of the most popular tourist destinations. Again we want to give a very local and authentic experience something kind of off the

50:22

beaten trail in terms of suggestions. So go suggesting a lesserk known museum or local art gallery. So this is kind of again giving feedback to the front desk and then the uh front desk comes back. The front desk agents says visit the museum de launch uh for an intimate experience with Monet's water lily. So great, the concierge is disapproved which then that terminates our um conversations because now it meets the uh kind of criteria for the concierge and then we can see that this is complete is true. This is just one type

50:57

of pattern that we will have can apply to in terms of using multiple agents. Uh you can see here now the kind of power between not just having one agent make suggestions and maybe having you know in hoping the user makes feedback but even using an agent to refine those uh results and giving back a better suggestion. So that was the end of lesson 8 or terms of multiple agents but see you in lesson [Music] 9. What is metacognition? How does it apply to AI agents improving over time? and how can we use it to our advantage when building

51:33

AI agents? We're going to answer these questions in the ninth lesson of the AI agents for beginners course. In this course, we take you from concept to code covering the fundamentals of building AI agents. In this short video that follows along with the written lesson, including translations and code samples that you can find at the link above and below this video. But let's start by defining what metacognition is, which is thinking about thinking. But how does that apply to AI agents? So AI agents can use data

52:07

and analysis to identify errors and make improvements in its plannings and responses. This enables improvements over time, which makes them more unique from a basic application using an LLM or simply making API calls to different services. While this is a growing field of research and but an important one because it allows agentic systems to be more transparent on its reasoning and decision-m more making it more adaptable to changing environments and improving the accuracy of interpreting the data

52:37

for of their environments. So but to put this in real terms let's say we have a booking flights and we ask an agent to book the best flight And best can be defined by our preferences such as being cheap or convenient or just with our favorite airline. Where metacognition comes into play is enabling the agent to reflect on this decision of how it's defining the best flight as well as including preferences of the customer going forward as a pattern for making decisions in the future. This also requires designing the system

53:22

in a way to gather that feedback, store it, and retrieve it later on for future interactions. And as purposes change, an AI agent should be able to adapt to these changes and be able to communicate why it has changed it and its setting based on that. And while your preference might be just to listen to me talk about this topic, I know there are some of people out there who also prefer to see this concept in code. So let's head over to our code editor and see how this works. Okay, so this is now as a chapter

53:52

9 code sample. Again, this code sample is available at the GitHub repo at the top and at the bottom of this video. In this example, we want to kind of sort of mimic what some of the qualities around meta uh metacognition is. And in this case, uh, we're going to kind of focus first on the agent instructions that we defined. So we have some, you know, function calls that will kind of come into play here, but we also want to, uh, have the agent maintain basically a customer preferences object. So this is

54:23

to mimic, uh, it kind of tracking again customer preferences over time in in a production or in a a larger application. Maybe this object is stored somewhere else and the agent is actually reading that. but we're going to just kind of have it kind of maintain this in memory or in context of the conversation that's going on. In this case, we're going to make sure that we track the preferred flight times of the user. And we want to make sure also that before it suggests anything or ask a user for the preferred

54:51

time that it goes back to this object, it reflects essentially on that information or interactions in the past. So again, makes it a bit more of a streamlined experience for uh using this agent uh versus, you know, if you're booking something, you know, on a UI and you have to kind of continue to select these things. So in this case, we're going to have user inputs here basically say, book me a flight to Barcelona. Uh we're going to prefer later flight and this should mimic a conversation with an

55:19

AI agent. So if we run this code, you can see some of the examples. Uh so first is you booking a flight to Barcelona and naturally the travel agent is going to ask you know when is your preferred time to uh for flights uh they're going to the user is going to come back and say prefer later flights and this is going to make a function call to get those flight times uh and again since it has again one of the you know powers of using large language models is being able to interpret natural language. So, it's going to say,

55:47

you know, out of all these flight times I just got returned, uh, which one's the later flight time? And in this case, it's the 10:45 p.m. flight. So, this is going to cut different to where what we might have been experiencing just using a regular flight booking application. Uh, then the user going to say, you know, this is just too late. I want the earliest flight. So, it's going to make the call back again just to get the flights that are available. It doesn't need to do any sort of uh you know

56:14

calculations on the the the time differences or whatever the available times are the differences between what was suggested first because you can also now pick in the earliest flight which is 8:30 a.m. And just some added benefits here. Um we have uh you know a request for more activities and then the user is sort of stressed out about the amount of time uh that they have. So again, it's kind of having um in context here that the user is actually asking that they want to leave uh after the last flight.

56:43

So again, we know the the agent has uh acknowledged that the last flight is 10:45 p.m. So keeping that context and that sort of cognition in terms of okay, well, I know this is the last flight, so I need to build an agenda that's going to fit in within the first flight to Barcelona and the first next flight out. Essentially uh this will then will go in uh and kind of make a defined structure of uh a flight plan because the customer is stressed out. They won't have enough time and again operating within those

57:14

confines. So that's kind of some bits of metacognition here in terms of just operating with the context of the conversation. Let's say we want to sort of mimic making a new conversation, but we're still going to use the chat history from last time uh to make sure that the agent knows some of our preferences as a a customer in terms of booking flight. So, in this scenario, we're just going to say, "Book me a flight to Paris." So, in this case, what we'll go into is the book me to Paris.

57:42

It's going to make the flight call function call again to get the flight times. And then based on your since based on your previous preferences for later flights, uh I'm going to go ahead and suggest the later flight time which is 7:15 p.m. And as as kind of good in instructions, we all make sure you know would you book this flight or would you like to look at alternative times? So this again shows you some hints of uh some of the benefits of metacognition uh in terms of you know using the agent to

58:09

take preferences or long-term memory uh and kind of apply that into an improved customer or user experience over time. Um this is just scratching the surface of metacognition. I do encourage you to look at the written chapter as well as it shows you a little bit more of some of the samples and benefits of metacognition. But that's to close chapter nine and we'll see you in chapter [Music] 10. How can we deploy AI agents to production effectively? What are some uncommon mistakes and how can we fix

58:42

them? And what are some ways we can manage the cost? We're going to cover these questions in the 10th lesson of the AI agents for beginners course. In this course, we take you from concept to code covering the fundamentals of building AI agents. in this short video that follows along with the written lesson, including translations and code samples that you can find at the link above and below this video. But let's talk about getting AI agents into production and into our users and customers hands. And that journey begins

59:13

with evaluations of your AI agents. To evaluate AI agents properly, you need to look at the entire system that the AI agent operates in and set up evaluation points at each step. This includes but not limited to the initial request to the large language model or service. Making sure you have a proper connection response times and model selection and how this might affect affect responses over time. Then the agent's ability to identify the intent of the user, which is helpful to make sure your agent has the ability to

59:55

complete the task that's been requested. Then the agent's ability to identify the right tools to perform that task so that the agent is achieving the goal of the user. also the tools response to the agent's request. Things to consider here are any errors or malformed responses from those tools or perhaps uptime issues with the service if you're using an external service. Then also being able to collect the feedback of the agents response is also a part of evaluations. This includes providing a UI fe or feedback

01:00:46

mechanisms in the UI like a thumbs up, thumbs down or how the users are satisfied with the responses as well as using manual evaluation as well as nlms to judge responses. And having evaluation at every step of the workflow enables us to both see changes over time and it allows us to make changes to our agentic system. Things like changing the model or different services for tools and we can better identify the effects of these changes because we're evaluating at every step. And the next step of this

01:01:27

lesson like all the other lessons before it is to head over to our code editor and see this in action. Okay, so welcome to chapter 10 and this code sample which again can be found at the top of the GitHub repo the link above and below this video. So you've probably seen this code this sample before if you followed along with all of the lessons uh cuz we have this kernel fun this kernel function or function call that we have which is about getting destinations. So these are destinations available and we also have the get

01:01:57

flight times. But the difference of this one and I want to kind of mimic scenario that you might experience when launching AI agents into production is that when we make this call, we're actually going to get this HTTP error for four uh because flight times aren't currently available. So again, if we're using an external service, we might experience this because maybe our credentials are expired or the service is down. So, we need to make sure that we have a way to then allow the agent to continue to

01:02:27

operate and continue to use whatever tools that are available. In this case, we're essentially going to just replace that with a get flight times backup. That should have also the same, you know, the flight times that we expect. Uh when a user asks that, then you can maybe even also has instructions on what, you know, even directing the agent if the service is down, how to handle those errors. In this case, what we we we are going to do is actually define this get flight times function as well

01:02:54

as this get flight times backup and define this as a backup function uh for flight times. And then you know if flight time service is down, we're also going to make sure uh that it knows that it can start using this backup flight times. So again, we're not uh you establishing these grand rules in terms of error handling, but you can do that in terms of especially when you're operating with multiple different services out there. So what this interaction should look like is you know we have this user input

01:03:22

which is book me a flight to Barcelona. So again this is the request from the user and we can actually see I drilled down here about the function calls that it makes. So first it's going to try the get flight times uh you know passing in the destination Barcelona and what we get this http4444 error. So next request is to get flight times back up which luckily that service isn't down or has no errors. So then we're going to get this flight times back up and these flight times and be able to send that to the user. So this

01:03:55

is just one scenario and one type of um you know scenario you should prepare for when you're working with AI agents in production. We have have a list and a written lesson of other things that you might want to look out for when launching a agent production. And this will conclude at least in our initial chapters. So, uh, 10 lessons on AI agents for beginners, but we wish you the best of luck and we will continue to improve this course and add more materials as the world of AI agents continues to expand. But thank you for

01:04:23