Prompt Engineering for Improved Performance

by Kesha Williams

Course Overview

Course Overview

Hi everyone. My name is Kesha Williams, and welcome to my course, Prompt Engineering for Improved Performance. I am an AWS machine learning hero. Prompt engineering focuses on the thoughtful design of prompts to elicit correct answers from large language models or LLMs like ChatGPT. My course focuses on advanced techniques for prompt design and optimization to significantly enhance performance guiding large language models to generate more accurate coherent and relevant outputs. Some of the major topics that we will cover include exploring the latest research and advancements in prompt engineering for generative AI, designing prompts that improve model performance and inference time, analyzing model performance and identifying areas for improvement, optimizing prompts for specific use cases and applications, gaining hands-on experience with advanced prompting techniques. By the end of this course, you will be ready to harness the capabilities of language models effectively ensuring they align with specific use cases, minimize biases, and deliver high quality outputs. Before beginning this course, you should have a basic understanding of prompt engineering and the ability to read basic Python code. I hope you'll join me on this journey to learn about advanced prompt engineering techniques that improve performance in the Prompt Engineering for Improved Performance course, at Pluralsight.

Prompting at a Glance

Introduction

Don't you just love how technology and IT roles evolve? Just a few years ago, prompt engineers and prompt engineering did not exist. Hi. I'm Kesha Williams. I've been in IT for over 25 years, and I've enjoyed watching the IT landscape evolve. Welcome to my course, Prompt Engineering for Improved Performance. Advanced prompt engineering techniques can significantly enhance the performance of large language models by guiding them to generate more accurate, coherent, and relevant outputs. In this course, you'll gain the ability to harness the capabilities of language models effectively, ensuring they align with specific use cases, minimize biases, and deliver high-quality outputs. First, you'll journey beyond the realm of basic prompting techniques and learn how to leverage advanced techniques that elevate a model's performance on reasoning tasks. Next, you'll discover the art of prompt optimization for specific use cases. Finally, you'll gain insights into the key factors that impact the latency and performance and explore mitigation techniques. When you finish this course, you'll be a prompt engineering wiz, so let's get started. The first module is Prompting at a Glance. We'll explore prompt engineering basics and review common techniques like zero-shot learning, few-shot learning, and instruction prompting. We'll wrap up with reviewing what causes prompts to fail.

Prompting Basics

Let's start with prompting basics. What is prompt engineering? Prompt engineering shapes the communication between humans and AI models, enabling better interaction and achieving the desired outcomes. It's simply about getting the model to do what you want it to do at inference time by supplying the proper context. Prompt engineering is like crafting a finely-tuned instrument for a musician. Just as the instrument must be tailored to the musician's preferences and style of play, prompts are carefully designed to guide the language model and elicit desired responses. By adjusting the prompts, we can finetune the model's behavior like adjusting the instrument's strings, keys, or valves to produce the perfect sound. A prompt can be as simple as can you tell me one cheese that pairs best with the Pinot Noir, and the answer from the model would be brie. A more complex prompt called an instruction prompt could be as detailed as proofreading correct this article summary, make it more humorous, ensure it follows the APA style guide and targets a technical reader, return your output in markdown format, and you would also include the text to summarize. Prompts typically contain several elements and not all elements are required. First, you have the context which helps the model understand the task or the desired output. Context stares the model to better responses. You have your input data, which is the actual data you want the model to process. Next, you have instructions that textually describe what you want the model to do, examples which demonstrate the correct behavior to the model, and constraints which restrict the output of the model based on specific requirements. Let's look at an example. Here is a sample prompt. First, we have the context. You are an AI assistant that generates personalized book recommendations based on user preferences. Next, we have the input data, which is a user profile. Next, we have the instruction. Can you recommend five books for this user? You can also include examples, so example input and example output, and you can also include constraints. Ensure the generated recommendations are within the user's preferred genres and align with their favorite authors. Do not recommend the books in the example and also maintain a friendly and engaging tone throughout the generated response. Now that you understand more how prompts are composed. Let's see a prompt in action.

Demo: ChatGPT in Action

In this demo, we are going to review ChatGPT's response to our prompt. The response is also called a completion. Let's navigate to the Jupyter Notebook. In this Jupyter Notebook, we are using OpenAI's API to interact with ChatGPT. In order to do that, there are several libraries that need to be installed like OpenAI, urllib, etc. I've already installed these libraries, so I will not execute those lines of code. In order to programmatically access the OpenAI API, you'll need to set up an API key because it's secured using an API key, and to set that up, you will need to create an OpenAI account and then go into your account and generate that key, the key I've stored locally in an environment file, and I need to pull that file in and retrieve that key. That's what I'm doing here in this chunk of code. First, I'm importing OS, then I'm importing OpenAI, and then here on Lines 5 and 6, I am reading my environment file. Next, you need to authenticate to the API using the API key, and the way you do that is from the OpenAI library, you initialize the API key and I'm pulling it from my environment variable. Next, here's the prompt. So this is what we are going to send to the large language model. This is the prompt we saw before. In this first line, rows 4 and 5, I'm setting the context. You are an AI assistant that generates personalized book recommendations based on user preferences. On the next few lines, I'm sending in example inputs and example outputs. Here on Line 10, the first input is a user profile. On Line 14, I'm expecting the model to output book recommendations. This is a way to guide the model to produce the specific output that you want. Here on line 21, I'm providing another example input, the user profile, and example output. On Line 30, I'm detailing the constraints, so ensure the generated recommendations are within the user's preferred genres. I do not want the model to output books from the examples I provided, and I also want the model to maintain a friendly and engaging tone throughout the generated response. Here, I'm providing the input data, which is the user profile. The person is 12. Their preferred genre is science fiction. I'm also passing in two of their favorite authors. On Line 42, I'm giving the specific instruction to the model. Can you recommend five books for this user? I'm going to go ahead and execute this code because it does take a while to run, and now while it's running, let me explain what we're doing. Here, we are using the OpenAI library to call the ChatCompletion API, and on Line 2, we are detailing which model we want to invoke, in this case, it's gpt-3.5, which is also known as ChatGPT. Passing in the messages, we will get a response. On this Line 1 right here, let's execute that to print out the response from the model, and notice here in the response, we have five books. The model says absolutely based on your preferences, I have selected five books that I think you'll love, A Wrinkle in Time, Superfudge, Ender's Game, The Giver, and The Hitchhiker's Guide to the Galaxy. And the model wraps up with a very friendly tone. I hope you enjoy these book recommendations. Happy reading. You may notice that when you run this code locally on your laptop, the responses may differ and that's to be expected. The responses are generated randomly based on the input, and even if the input is the same, the output could differ because the model is considered to be stochastic. So even if you use the same prompt, you'll see different results each time. You can think of stochastic as the model's variability or randomness because the specific output can never be predicted with certainty. There you have it. You've seen ChatGPT in action.

Prompt Development

Now let's talk about the iterative nature of prompt development. Prompt engineering is an iterative process allowing continuous refinement of the prompt to improve the model's performance. We can achieve better results over time by experimenting with different variations, techniques, and optimizations. In this example prompt, you may start with something general like which cheese pairs best with the Pinot Noir. You may refine that to say, can you tell me one cheese that pairs best with the Pinot Noir, and you continue to iterate and you may land on, can you tell me one French cheese that pairs best with the Pinot Noir? As you're learning about prompt engineering, you'll learn how to tweak and refine your prompts to improve the model's performance. Now, it's time for you to gain hands-on experience with prompt engineering. You have three hands-on challenges to choose from, and your choice is really based on where you are in your technical journey. The first option is just to explore ChatGPT, and you go to chat.openai.com. This website is a simple web interface that sits on top of the OpenAI ChatGPT large language model. You can play around with creating prompts, refining prompts to get the optimal output. The second challenge, you can explore the OpenAI Playground. This is a web interface or playground for developers. You will be able to choose from the different large language models. Remember that GPT 3.5 is ChatGPT, but you'll be able to go in there, explore, set the different parameters, and this option, like I said, is more geared toward developers. And the last option, option three, the

Jupyter Notebook option, if you're familiar with Python, pull down the Jupyter Notebook code that I demonstrated in the prior clip and set up your OpenAI account, generate your API key, update that code with your specific API key, and start interacting with the OpenAI APIs programmatically. Prompt engineering is crucial when working with generative AI systems and achieving optimal results with large language models. Prompts contain several elements, and not all elements are required. We looked at the context which helps the model understand the task or the desired output. The content, rephrase, the context stares the model to better responses. We looked at the input data which is the actual data you want the model to process. We saw instructions which texturally describe what you want the model to do. We looked at examples, you pass an example input, an example output which demonstrates the correct behavior to the model. And we wrapped up with constraints which restrict the output of the model based on your specific requirements. Before we move to advanced prompting techniques, let's make sure you understand the basics. We'll look at zero-shot learning, few-shot learning, and instruction prompting, which is up next.

Demo: Zero-shot Learning

You should know a few basic prompting techniques before diving into advanced techniques that push the boundaries of what we know to be possible with machine/human interaction. Let's level set and review zero-shot learning, few-shot learning, and instruction prompting. Let's start with zero-shot learning. Zero-shot learning is the most simple prompt engineering technique. If you've played around with ChatGPT writing simple prompts, you're familiar with it. With zero-shot learning, you simply describe the task you want the model to solve and let it generate a response. In this case, recommend a book in this genre, mystery. Let's see zero-shot learning in action. We'll look at the Chat Completions API, authenticating via an API key, and using OpenAI's Python library. Let's navigate to the Jupyter Notebook. You've seen this code before. These are common steps that you'll need to use in order to use the Open API library. You'll need to install the necessary libraries, you'll need to gain access to the OpenAI API using an API key, and you'll need to authenticate to the API using that key. Now let's take a look at zero-shot learning. When you're working with large language models, you'll have the message here on Line 1 or the prompt that you pass in. This very simple prompt simply says, recommend three books in the genre, mystery. This is called zero-shot learning, zero-shot prompting, it's the simplest form. Here on Line 1, I've created a function called zero-shot learning. I've enclosed it in a try/catch block printing out any errors that may occur. In this function, we are calling the Chat Completion API, the create function, and we're passing in the model that we want to use, gpt-3.5, which is ChatGPT, we're passing in the messages, and we are returning the response. Let me run this code while I explain it to you. Here on Line 1, I'm calling the zero-shot learning function. On Line 2, I am printing the response. I've asked for three books and notice it returns three random books in the mystery genre, Gone Girl, And Then There Were None, and The Girl with the Dragon Tattoo. Now, if we use the next level or few-shot learning technique, we can improve the model's performance in what it returns. We can also drastically improve the performance of a large language model by providing more specific details. We've talked about the different elements of a prompt like the context, the example input and output, and the constraints. When you include those additional features, you'll see how it improves the model's performance, so let's take a look at few-shot learning.

Demo: Few-shot Learning

Now let's take a look at few-shot learning. You've actually seen few-shot learning before. You just didn't know that it was called few-shot learning. This technique adds input and output examples to the prompt showing the model the correct behavior. In this demo, we'll see few-shot learning in action. We'll use the Chat Completions API, we'll authenticate using an API key, and we'll use OpenAI's Python Library. Let's navigate to the Jupyter Notebook. In this example, I'm going to run this code before I explain it to you because it will take a while to return. With few-shot learning, you add more description to your prompt. If you remember, let's take a look back at our zero-shot learning prompt. It was a very simple prompt, recommend three books in this genre, mystery, and the large language model returned three books in the mystery genre. When I personally look at these books, it appears to be chosen at random, and none of these books really appeal to me. Enter few-shot learning. In this prompt, I include more detail so that the model can perform better to my taste. Here on Line 3, the first line of the prompt is your job is to recommend six books in the mystery genre. My favorite authors are Louise Penny, Ann Cleeves, Harlan Coben, and Lee Child. Output the recommendations in JSON format using these elements, title, author, description, rating, and length. Here on Line 1, I have created this few-shot learning function, calls the Chat Completions API using the ChatGPT model, passing in the messages, and returning a response. Here, I am storing the response in a response variable and printing out the content. Now, look at this output and compare it to the one-shot learning output. Much more detailed. It includes books that actually appeal to me. Here, the first recommendation is a book from Louise Penny, the rating is 4.16, and 320 pages. I can continue to just look through this list. I would definitely choose the book with the highest rating, so this one is 4.15, 405, 412, 388, so it's probably the Still Life, and the story sounds very appealing. We have improved the performance of the large language model. There is a downside to few-shot learning, and it's related to the large number of tokens sent and returned with each request and response. When you're using the OpenAI API, you are charged per token. So the conciseness of the request and the response is crucial. Just remember these pros and cons as you consider tradeoffs between the cost versus the performance of the model.

Demo: Instruction Prompting

The last prompting technique we're going to review is called instruction prompting. Instruction prompting provides specific instructions or guidance to the model to generate specific responses or outputs. Did you know that ChatGPT can assist with story writing, code generation, and even API design. Let's navigate to the Jupyter Notebook and review instruction prompting. I've navigated to the Jupyter Notebook, and here in cell 16, I've created an array of instructions. The first instruction says write a creative story about a magical adventure. Please make it whimsical and lighthearted, emphasizing the importance of friendship and bravery, ensure the story is around 500 words in length. That's the first example of an instruction prompt. The second example is related to generating code. In this prompt, I'm telling the large language model to build a Python function that calculates the factorial of a given integer. Ensure the function takes an input parameter in and returns the factorial value. Please optimize the function for efficiency and handle edge cases, such as negative numbers and 0. I also tell the model to provide clear comments and use appropriate variable names for readability. I think this model is going to produce the perfect code. And the third instruction prompt, design a RESTful API for a blog application. I go on to say that APIs should support CRUD operations, create, read. update, delete, for the blog posts, comments, and user accounts and ensure the API follows REST principles and uses appropriate HTTP methods and status codes. Document the API endpoints, request response formats and authentication, authorization mechanisms. I cannot wait to see what the output of these three instruction prompts will be. In Block 17, Line 1, I have created this function called instruction prompt where I'm simply passing in the specific instruction or the ID, and it's going to pull it from the array above. Notice the model I'm using, it is the text-davinci-003 large language model. I'm passing in the prompt, and I am defining a set response size of 500 tokens. I've already executed the three prompts because it does take a while for it to generate. Now, let's look at the output here in Block 18. The first instruction prompt was related to writing the whimsical story. I'm not going to read this story word for word, but it starts with once upon a time deep in the heart of an enchanted forest, there lived three brave adventures. Definitely a story I'd like to keep reading. And notice the max token length on this output should be 500 and that's important. Remember, I told you that when using the OpenAI API, you are charged based on the amount of tokens passed in and the amount of tokens in the response. The next instruction prompt was to generate the factorial function using Python. This is the code that the large language model produced, and I simply copy and pasted that code into the next cell. I ran that code. There were no compilation errors. And then here in Block 22, I'm testing out the code by passing in the number 10, storing it in the factorial variable, and printing out the value. Very cool. And finally, the third instruction prompt was related to generating the design for RESTful API for a blog application. And let's just quickly look at the output, the large language model define the API endpoints. You'll see an option for registering a new user using a post request, and the response back is a token. You'll see logging in, creating a new blog post, getting a list of all blog posts, getting a specific blog post, and updating and deleting. So when I look at this API design, it seems pretty thorough. And at the very top, it talks about using token-based authentication. This is a great starting point for designing an API. It definitely needs to be further refined and reviewed for completeness, but it's a great starting point. Who knew that ChatGPT could do all of that. Now it's time for your next hands-on challenge. I encourage you to try your hand at zero-shot learning and then refining that prompt using few-shot learning and then just use your imagination and creativity to write an instruction prompt. Let's have a quick review of everything we've learned thus far. We looked at zero-shot learning, which is a simple basic prompt with no context, examples, or additional detail. We saw that we can use few-shot learning to improve the model's performance, which includes examples to guide the model's output and improve that performance. And we looked at instruction prompting that provides detailed instructions, and you can use instruction prompting to aid in story writing, code generation, and even API design. We have one more topic to cover in this prompting at a glance module before we look at advanced prompting techniques, and that topic is understanding why prompts fail, so that's what's up next.

Prompting Failures

ChatGPT and other large language models seem to have all of the answers, but do they really, and can you trust what they say? Not always. There are times that models fail, and you need to recognize the patterns. We'll look at prompting failures and strategies that you can use to mitigate those failures. Why do prompts fail? There are several reasons. First, prompts that are too vague. Prompts that are vague or ambiguous can lead to inconsistent or unpredictable responses from the language model. If the prompt has biased and leading language, that could cause the model to fail or even steer the model towards a particular viewpoint that can result in biased or skewed responses. And lastly, models fail because of evolving language and knowledge. Language models are trained on a fixed dataset, which means they have a cut-off date and may lack awareness of recent events and emerging trends. This sometimes causes the model to hallucinate or simply make up an answer where the model generates responses not grounded in reality or factual basis. There are several options to mitigate a model failure. One, you can improve your prompt, which is what this course is all about, or you can finetune the foundation model, and we'll talk about what a foundation model is in a moment. You can finetune it for your specific task, or you can assume the model can't do what you need it to do and simply solve your problem a different way. Let's talk about a foundation model like GPT versus a finetuned model like ChatGPT. When considering large language models, or LLMs, there are generally these two types, the base and the finetuned. This explains the difference between GPT and ChatGPT. GPT is the foundation model, and Chat GPT is the finetuned model from that foundation model. So for example, GPT-3 has a broader range of natural language capabilities and it's knowledgeable up to 2020, and it takes this text-in and text-out format. Whereas ChatGPT, also called GPT 3.5, is finetuned for conversational dialogue, and it's knowledgeable up to 2022, and it uses more of a conversation-in and text-out approach. A lot of people interact with ChatGPT using the web interface and that web interface sits on top of GPT 3.5.

Conclusion

We've seen that prompts fail for several reasons. First, prompts that are too vague, prompts that have bias and leading language, and evolving language and knowledge can cause a prompt to fail. We looked at several mitigation strategies, we can improve the prompt, we can finetune the foundational model for a specific use case like chat, or solve our problem in a different way. This wraps up our module, Prompting at a Glance. We looked at prompting basics, we looked at common prompting techniques, we learned about zero-shot learning, few-shot learning, and instruction prompting, and we also looked at what causes a prompt to fail. Up next, you'll learn how to master advanced prompting techniques in the upcoming module.

Master Advanced Prompting Techniques

Introduction

In this module, we'll look at mastering advanced prompting techniques to improve the overall model's performance. We'll start with chain-of-thought prompting which improves the model's ability to perform complex reasoning tasks. We'll look at ways to augment prompts using knowledge augmentation to train the model to produce better output. We'll look at examples using relevant keywords, structured data, additional context, and more. And we'll wrap up with improving your prompts through prompt tuning and the differences between prompt tuning versus prompt engineering, which is a question I hear often. So let's get started.

Chain-of-thought Prompting

Let's start by discussing chain-of-thought prompting, which is an advanced prompting technique. Chain-of-thought prompting is a technique that generates a chain of thought or a series of intermediate reasoning steps that improves the model's ability to perform complex reasoning tasks. This prompting technique enables models to decompose multi-step problems. Chain-of-thought prompting can be combined with few-shot prompting to get better results on more complex tasks that require reasoning before responding. Let's take a look at this prompt. You are an AI assistant that helps customers book flights on the Kesha Air Airline. Follow these steps to answer the customer queries. First, determine whether the user is asking a question about a specific airline or flight route. Step two, if the user is asking about specific flight routes, identify whether the flight routes are in the following list. Step three, if the message contains flight routes in the list above, list any assumptions that the user is making in their message. You will see that this prompt is very detailed and it describes the reasoning steps, step one, step two, and step three, that the model should follow before responding. You'll be surprised that the answers from chain-of-thought prompting are more accurate, detailed, and interpretable than answers from other types of prompts. Chain-of-thought discourages the model from generating quick answers.

Demo: Chain-of-thought Prompting

Let's see a few examples. We will instruct the model step-by-step by presenting the steps involved. We'll start with a few simple examples and end with more complex examples. Let's see chain-of-thought prompting in action. If this is your first time using this code, there are several libraries that you'll need to install. Next, I'm importing the libraries, and you'll need to make sure that you have an OpenAI API key. You'll need to generate that. I've saved the file locally in an environment file, and I'm pulling in that environment variable here on Line 3. Next, Line 4, there is a function called get_completion. This function is used to send the prompt to the OpenAI model, in this case, the model I'm using is ChatGPT, also known as GPT 3.5. Now let's take a look at this very first basic zero-shot prompt. Let's look at the prompt plan a vegan weekly meal plan that provides balanced meals. Let's look at the output of this basic zero-shot prompt. It's pretty straightforward. The model spits out this vegan weekly meal plan that offers up breakfast, lunch, dinner, and a snack for each day of the week, so Monday all the way through Sunday. Now let's update that prompt to use chain-of-thought prompting, and here in Block 12, you see the update. Walk me through the process of planning a weekly vegan menu. Step one, describe the healthy ingredient selection. Step two, describe the balanced meal using those ingredients. Step three, describe the specific time of day to eat the meal. Step four, describe exercises that should be included for optimal health. Provide a weekly meal plan using this process. When this prompt is passed to the get_completion function, we'll see the output printed down here. Step one, the model is giving advice around the healthy ingredient selection saying it's important to focus on incorporating a variety of nutrient dense ingredients, and it goes on to list out those ingredients, whole grain, legumes, fruits and vegetables, nuts and seeds, and plant-based protein. Step two, it talks more about the balanced meal and how it needs to include a combination of protein, good fats, complex carbohydrates, etc. Step three, it talks about the specific meal timing. Step four, it talks about exercises for optimal health, and then it wraps up with the weekly meal plan. This time around, it provides breakfast, lunch, snack, dinner, and dessert. Let's look at the next example. Again, we start with the basic zero-shot prompt, develop a new dessert recipe, and the model returns with raspberry chocolate crumble bars, gives us the ingredients for the crust, for the raspberry filling, and for the chocolate crumble topping. It sounds pretty good, and it also gives instructions. Now, let's tweak that prompt to use chain-of-thought prompting. And here in cell 19, let's read the prompt, develop a new dessert recipe following the steps below. Make sure to describe your reasoning steps in detail as you go through the steps. The final output should be the new recipe with cooking instructions. Step one, brainstorm and refine the concept. Step two, consider the flavor combinations. Step three, consider the texture contrast. Step four, consider the presentation that will make this dessert truly remarkable. And let's look at the output. For step one, the model talks through how it came up with the recipe. It wanted to create a unique and delicious treat that will leave a lasting impression. And after brainstorming, it decided to combine the flavors of chocolate and orange, which I agree is always a crowd pleaser. Next, the model considered the flavor combinations. The model decided to use a rich and dark chocolate cake as the base infused with fresh orange zest. It sounds really delicious, doesn't it? Next, step three, consider the texture contrast. So the model wanted to create an enjoyable eating experience, and so it incorporated multiple texture contrasts into the dessert. The outer layer is moist and fluffy while the center should be decadently gooey. This texture contrast impacted the timing for how long the recipe calls for the cake to be cooked. Notice here, it's going to be cooked on the outside, but still slightly undercooked in the middle. Step four is to consider the presentation that will make the dessert truly remarkable, and what the model decided to do this time around is to add a garnish to the dessert to improve the overall presentation. The model will top each dessert with a twist of fresh orange zest and a sprig of mint, adding a pop of color and freshness. I don't know about you, but this second recipe is definitely sounding better than the first, and that was a quick look at chain-of-thought prompting. Now it's time for your next hands-on challenge. For this challenge, you'll improve the weekly meal planning prompt. Take inventory of your pantry, fridge, and freezer and include the ingredients you have available in the prompt to tailor the model's response to what you have on hand. We discussed chain-of-thought prompting, which gives the model specific steps to follow before generating output. This improves the model's overall performance. Next up, we'll look at knowledge augmentation.

Knowledge Augmentation

Let's talk about knowledge augmentation. An advanced technique is to automate prompts using relevant keywords, structured data, additional context, and external data, and this is all to improve the model's performance. Large language models learn a lot of information during training. However, augmenting your prompts with additional context relevant knowledge for your specific use case like structured data is a great alternative to retraining or finetuning the model. This extra information is used as additional context by the model while generating the output leading to better results. Knowledge augmentation is similar to data augmentation in natural language processing. It's a useful technique to improve the accuracy and variety of chatbot responses. With ChatGPT, knowledge augmentation provides additional training data. Let's say we have this simple prompt, and the prompt is using the chain-of-thought prompting technique. We can combine the chain-of-thought prompting with knowledge augmentation. In this case, along with the prompt, we would send in structured data, for example, flight routes, and then the model will take the prompt along with the structured data to produce a better output. The additional data can be added to the prompt using multiple techniques, adding keywords and phrases, incorporating the structured data, providing additional context, pulling data from a vector database, and incorporating semantic search results. Let's look at this example more closely. Imagine an application like a tutoring app that is powered by ChatGPT. Out of the box, ChatGPT can handle a multitude of topics because it was pretrained on the vast knowledge of the internet. In this example, we are going to supplement with knowledge augmentation. Here, we have a prompt. What's the difference between a function and a method? And this is the prompt for our tutoring application. We have the prompt. We can then implement business logic to query for additional information. For example, we can pull data from snippets off of Wikipedia, and we can add that to the prompt which is then sent along with the prompt for the large language model to process and provide a response. This is a great way to give the model factual up-to-date knowledge on any topic. Let's see an example in code.

Demo: Knowledge Augmentation

In our knowledge augmentation demo, we'll look at how to incorporate structured data to help passengers book airline tickets on Kesha Air. If you're running this code for the very first time, there are several libraries that you'll need to install. Once you've installed those libraries, you'll also need to import them and make sure that you've generated an OpenAI API key and that you're pulling it in to authenticate to the API. Here in cell 13, I've created a function called get_completion. This function is responsible for taking in your prompt and sending it to the ChatCompletions API, which calls model GPT 3.5 turbo. The next helper function is called get_completion_from_messages. Here, instead of passing in a single prompt, we are passing in a series of messages to the ChatCompletions API. Let's look at this first example. This first example here in cell 26 uses the very simple get_completion function that takes a simple prompt and passes it in. For this prompt, tell me the flight routes on Kesha Air. The response from the model is I'm sorry, but as an AI language model, I do not have information on Kesha Air. Well, we can fix that by augmenting the prompt with additional knowledge, and that is what we are doing here in cell 37. There is a system message that sets the persona for the model and then also gives the model additional data, all of the flight routes on Kesha Air. So let me scroll back up. Here's the context. You are an AI assistant that helps customers book flights on the Kesha Air Airline. Line 5, follow these steps to answer the customer queries. Step one, first, decide whether the user is asking a question about a specific airline or flight route. Step two, if the user is asking about specific flight routes, identify whether the flight routes are in the following list. And then here is the list of the flight routes. There are 1, 2, 3, 4, 5 flight routes that also include the length, seat options, and the cost. The third step is if the message contains flight routes in the list, list any assumptions that the user is making. Step four, if the user made any assumptions, figure out whether the assumption is true based on your flight route's information. And the last step, step five, first, politely correct the customer's incorrect assumptions if applicable. Only mention or reference flight routes in the list of the five. More detail on the response output, step one, step two, step three, step four, and the response to the user. Now let's see this in action. Here in cell 38, the messages include the system message, which we just reviewed, and another prompt, which routes do you have that include ATL, which is Atlanta. I'm calling here on Line 8 the get_completion_from_messages function because we're passing in a series of messages. On Line 9, I'm printing the response. Here's the response.

Step one, the user is asking about flight routes involving ATL, step two, among the available flight routes mentioned, flight route 1 includes ATL which goes to Miami, MIA. Step three, the user assumes that there are multiple routes involving ATL and wants to know about them. Step four, as per the available flight routes list, there's only one flight route that includes Atlanta. Step five, we currently only offer one flight that includes Atlanta, which is from Atlanta to Miami. Would you like to know more information about this route and its flight options? Notice now that we have augmented the knowledge of the machine learning model. It is able to answer a question related to Kesha Air that it could not answer before. Cell 17, passing in the system_message and a new prompt, which is why is ATL to MIA cheaper than AUS, Austin, to ATL, Atlanta. Calling the get_completion_from_messages function here on Line 8, and I'm printing their response. So let's look at this output. Step one, the user is asking a question about the cost of two specific routes on the Kesha Air Airline. Step two, the flight routes in question are Atlanta to Miami, which is at 399 and Austin to Atlanta which is 299. Step three, the user is assuming that the cost of a flight is only based on the distance and not other factors like route popularity or seat availability. That is a great assumption made by the model. Step four, the assumption is not necessarily true as different airlines may have different pricing strategies based on seat availability, popularity of the route, seasonality, and other factors. And the response to the user, the cost of a flight is influenced by various factors such as seat availability, popularity of a route, etc. Hence, it is not always directly proportional to the distance covered. This is why the flight from Austin to Atlanta is cheaper than the flight from Atlanta to Miami. Great reasoning by the model. Cell 18, which seat options are available for Miami to Seattle. Step one in the response, the user is asking about a specific flight. The flight route, Miami to Seattle, is not available on Kesha Air, but with Pluralsight Air. And the response is I'm sorry that flight route is not available with Kesha Air. However, Pluralsight Air does offer the route first business and economy for $999. Would you like assistance with booking that flight? And let's scroll back up and look more closely at the flight routes. So yes, there are one, two, three flight routes for Kesha Air, another flight route for Kesha Air, and notice flight route four is for Pluralsight Air, so the model reasoned that correctly. And lastly, do you recommend flight route 1 or flight route 4 leaving from Atlanta? And the response the user is asking about a specific flight, and yes, flight routes 1 and 4 are available. The user is not making any assumptions in their message. Step four, as an AI assistant, I cannot recommend a specific flight route. However, I can tell you that flight route 1 goes to Miami and flight route 4 goes to Seattle. You can choose the route based on your destination, and the response to the user is both flight route 1 and 4 depart from Atlanta, but go to different destinations. Flight 1 to Miami, flight 4 to Seattle. Please choose the flight route that best suits your destination. So here in this example, the model is not equipped with making flight recommendations. Next, do you recommend flight route 100? And the model responds with sorry, but flight route 100 is not a listed flight route for Kesha Air. Can I help you with information about any of our available flights? Great reasoning there. Example two is a part of your hands-on challenge, so let's talk about that now. Now it's your turn to be hands-on. In this hands-on challenge, you will turn the salary negotiation prompt into a chatbot with multi-turn conversations. Here are a few tips. Prompt the user and collect their responses using Python's input function. It's important for you to note that each interaction with the AI assistant is a new session, so the entire history of prompts and responses must be included in each exchange. Let's wrap up. We looked at augmenting prompts as a way to improve the model's performance. You can include relevant keywords, you can also use structured data to augment the prompt, you can add additional context, or even use external data, and all of this helps to improve the model's output. Next up, we have prompt tuning.

Prompt Tuning

Let's talk about prompt tuning, which is an advanced prompting technique. First, though, you'll need to understand the differences between prompt tuning versus prompt engineering. The names are similar and can be confusing, but the differences are pretty straightforward. Prompt tuning focuses on refining and optimizing an existing prompt to improve the model's output. This is where you will tweak the prompt to achieve better results by changing the wording. Whereas, prompt engineering involves creating and designing a prompt from scratch. Prompt tuning can be manual or automatic and often uses techniques like scoring functions and applying optimization algorithms to update and refine the prompts based on evaluation results. The algorithm explores different prompt variations, such as modifying keywords, adding context, or adjusting the structure to find prompts that yield more relevant in diverse information. Let's say we have a language model that generates movie recommendations based on user preferences, and we want to automatically tune the prompts to optimize the relevance and diversity of the movies recommended. First, we would define the objective and that is to maximize the relevance and diversity of movie recommendations. So we'll start with a set of initial prompts or even use a prompt template. For example, we could have prompts like recommend a movie similar to the user's favorite movie which is Mission Impossible or suggest a movie from the genre action. Next, we would generate responses. This is where you use the language model with the given prompts to generate movie recommendations for different user preferences. The next step would be to evaluate the generated movie recommendations using predefined metrics. This could include measuring relevance by comparing recommended movies to the user's preferences or evaluating diversity by analyzing the range of genres, directors, or even actors in the recommendations. Next, we could apply an optimization algorithm to update and refine the prompts based on the evaluation results. This could involve techniques like reinforcement learning, using different algorithms, or the Bayesian optimization. The algorithms explore different prompt variations, such as modifying keywords, adding context, or adjusting the structure to find prompts that yield more relevant in diverse movie recommendations. The next step is to iterate. We would repeat the process of generating, evaluating, and updating prompts for a certain number of iterations or until the desired performance in terms of relevance and diversity is achieved. Through the iterative process of automatic prompt tuning, the algorithm learns to generate prompts that effectively elicit movie recommendations that align with user preferences, ensuring both relevance and diversity in the suggested movies. It's important to note that most automatic tuning techniques require that you have access to modify prompt embeddings through your own self-hosted large language model. The OpenAI API, which we're using, doesn't provide that level of access yet, it only surfaces the text inputs and outputs.

Demo: Prompt Tuning

You do have access to manual prompt tuning, so I'm going to show you a few techniques. Tuning your prompt can improve your performance, and it is quite powerful. You can tune your prompt by changing keywords and adding additional context. Let's look at several examples of prompt tuning. We'll look at the before prompt and the after prompt. If this is your first time running the code, you'll need to install the necessary libraries. Once you have those libraries installed, you'll need to import them and then ensure that you have generated your OpenAI API key and that you're pulling it in from an environment variable so that you can authenticate to the API. This first function you've seen before, it is the get_completion function. In cell 3, it takes on a prompt, and it sends that prompt to the ChatCompletion API. Cell 4, the get_completion_from_messages function. You've seen this before. Instead of taking in a single prompt, it takes in a series of messages and sends it to the ChatCompletions API. So let's look at this first example. The first version of this prompt is what are the benefits of exercise? Here in cell 11, the model responds with multiple benefits, improved cardiovascular health and circulation, reduced risk of chronic disease, muscle strength, improved sleep quality, etc, but now let's tune this prompt in cell 12 to get a more detailed response, and the new prompt reads and a detailed response provide at least five scientifically-backed benefits of regular exercise, including both physical and mental advantages. The response, the model talks about the physical benefits and calls out three physical benefits. The model then talks about mental benefits and calls out three examples, reduces stress, boosts cognitive function, increases self-esteem, and wraps up saying that regular exercise provides a wide range of physical and mental benefits. The response, this time around, is more detailed. The next example, Kesha is an expert Java developer that solves programming challenges easily with bug-free code. Karen is an expert GPT prompt engineer that writes prompts that improve the model performance. Karen's job is to make Kesha's prompts more effective at generating accurate and relevant responses, present a conversation between them where they suggest improvements and reach a final version of the prompt. The prompt should be tailored to optimize the performance of a large language model. The conversation ends with the final version of the prompt, that's the system message. And now the user message is here is the GPT prompt. What are the benefits of exercise? And if you look, the response is this back and forth dialogue between Karen and Keesha. Karen says, hi, Kesha. I was hoping to chat with you about the prompt. What are the benefits of exercise? Do you have a few minutes to discuss it? Kesha responds with yes, Karen, what's on your mind? Karen continues the conversation. I was hoping we could work on making this prompt more effective, and Karen offers up a suggestion. We might want to focus it more specifically on a particular type of exercise or a particular aspect of its benefits. Kesha agrees and rephrases what are the cardiovascular benefits of running? Karen: I like that. Another potential issue with the current prompt is that it doesn't specify the target audience. We might want to add a bit more context. Kesha says good point. We could say what are the cardiovascular benefits of running for middle-aged adults? Karen responds, that's perfect. It might also be helpful to include some specific examples of the benefits we're trying to get at. Kesha agrees, and she says, how about something like what are the cardiovascular benefits of running for middle-aged adults such as improved heart health, increased endurance, and reduced risk of heart disease. Karen says, I love it. That prompt is more specific and focused. And so in this example, we've actually used the model to help us tune our prompt. Now, let's look at the original prompt, which is what are the benefits of exercise and compare it to the finely-tuned prompt produced by the model. The original prompt, what are the benefits of exercise lists numerous benefits, improved cardiovascular health, weight management, increased muscle strength and endurance, stronger bones, a long list of benefits. Now, let's look at the tuned prompt. What are the cardiovascular benefits of running for middle-aged adults such as improved heart health, increased endurance, and reduced risk of heart disease. And the model responds with a specific focus on improved heart health, increased endurance, reduced risk of heart disease, weight management, blood circulation, and the list goes on, and it even offers up a tip. It is important for middle-aged adults to consult with their healthcare provider before starting a new exercise program. We see that the output from the second model is more detailed and focused because of the finely-tuned prompt. And now a hands-on challenge, write a prompt that is responsible for tuning your prompt.

Conclusion

This wraps up our module on advanced prompt tuning techniques. We talked about chain-of-thought prompting which generates a chain of thought or a series of intermediate reasoning steps that improves the model's ability. Next, we looked at knowledge or prompt augmentation. This is where you augment your prompts with additional context relevant knowledge for your specific use case. And we wrapped up with prompt tuning which focuses on refining and optimizing an existing prompt to improve the model's performance. Up next, we'll talk about optimizing prompts for specific use cases.

Optimize Prompts for Specific Use Cases

Introduction

Welcome to our new module, Optimize Prompts for Specific Use Cases. In this module, we'll look at ways to summarize and expand existing text. We'll look at text-to-image generation. We'll look at inferring sentiment in a given piece of text, and we'll wrap up with text transformation which includes language translation. So let's get started.

Demo: Summarizing Text

Let's start with summarizing text. This is where you have existing texts that you'd like to condense. We'll talk through how that works, and we'll have a quick demo. There is so much text in the world and a plethora of information. Wouldn't it be nice if you could quickly summarize a website, a list of product reviews, or a book? Do you remember cliff notes? Is that still a thing? It's a series of study guides providing summary and explanation of literary works. If you're trying to cheat on your exam, you may read the cliff notes as opposed to reading the book. Well, did you know there's no need to read the cliff notes for a book that you don't have time to read. You can use ChatGPT to do the summarization for you. Let's look at summarizing text and optimizing it for specific topics and audiences. In our first demo, we'll look at summarizing text for a specific audience with a word or sentence limit with a specific focus, so let's get into that now. If this is your first time pulling down the code, you'll need to make sure that you install the necessary library shown here. After you install the libraries, you'll need to import them and make sure that you've already generated your OpenAI API key and you're pulling that from a local environment variable in order to authenticate to the API. Here in cell 25, you've seen this function before. It's the get_completion_from_messages function which simply takes in a series of messages and sends it to the chat_completion API shown here on Line 3. The next function is a new function, it's called summarize. And in this example, we'll look solely at the summary process. Summarize takes two parameters, the content to summarize and the criteria. Notice for the messages here on Line 4, I'm passing in a system message which sets the persona for the AI assistant, and that persona is you are an AI assistant that is an expert at summarizing the content found on websites. Your goal is to summarize a website based on the given criteria. And the next message that we send in is a user message shown here on Line 7, and that message is your content to summarize is, that's the value passed in, here is the criteria. And the response returned is the response from the get_completion_from_messages function here on Line 14. So I'm going to skip over the expand function now and cover that when we talk about expanding text. So here is an example of summarizing a website for a student. The website that needs to be summarized is the Pluralsight.com product skills website, and the criteria is to summarize this website for a student. The skills website talks about the Pluralsight Skills platform. Notice here, this summary is tailored to a student. Pluralsight is an online learning platform that provides a wide range of technology courses and skill development resources targeted towards students, and you can pull the code down and review the response in more detail. The next summary is for a software engineer. Notice, the summary is now different. It says Pluralsight's Skills platform is an online learning resource that offers a wide range of technical training courses and tools for software engineers. It goes on to say how many courses are available, calls out technical training, and calls out that it's a valuable resource for software engineers. The next summary is for a sales person. In this summary, it calls out that Pluralsight is an online learning platform. It calls out 7,000 courses, and it has a special focus on businesses saying businesses can also use Pluralsight as a tool to help their employees stay up to date with the latest technology trends. Now, you can summarize the website with a sentence limit. So summarize it in one sentence. Pluralsight's product, Skills, offers businesses a comprehensive learning platform to upskill their workforce. Here, the criteria is summarizing it based on cost, so summarize with the focus on costs. Here, Pluralsight is a technology skills platform that offers online learning courses. It calls out the free trial, afterwhich, there is a paid subscription. We can also summarize in 50 words or less and notice the summary here is now condensed in 50 words or less. You can also summarize it to be in Tweet format. So summarize for a short tweet and notice now this summary is much shorter, and it also has tags to Pluralsight's skills, online learning, and a tech career. And that's a quick look at summarizing text from a website. Up next, let's take a look at expanding text.

Demo: Expanding Text

Now let's look at expanding text, which is the opposite of summarizing text. Let's have a quick demo where we look at expanding text based on a negative review and based on a positive review. This code is the same code that we looked at from the summarizing text example. I do want to scroll down and call out the expand function which we did not talk about earlier. The expand function takes in the content to expand shown here in cell five. Notice there are a series of messages that we'll pass to the model. The first is the system message shown here on Line 3, and that system message reads as a customer service AI assistant, your goal is to generate an email reply to a customer after reading their review of the Oregon Trail game. Thank the customer if their review is positive, apologize if the review is negative. Be concise and professional in your response. And the next message is a user message with the specific content, and the response returned from the expand message is the response from the get_completion_from_messages function. Now we looked at summarizing, so let's scroll down and find the expanding text. Here we go. So with expanding text here in cell 6, we have two examples. On Line 2, we have this negative review of the Oregon Trail, and I'm not going to read the entire thing, but this person is upset because they lost the Oregon Trail, and it was a frustrating experience, and the challenges in the game were insurmountable, and the decision seemed to lead to constant setbacks. This person is not happy about their experience with the Oregon Trail game. The next review is a positive review, and this person has nothing but nice things to say about the game. I conquered the Oregon Trail and emerged triumphant. The game's immersive challenges and strategic choices made for an unforgettable adventure. So we have two reviews, negative and positive. Here, we are sending to the expand function first the negative review shown here on Line 1 in cell 7, and then we're printing out the response from the model. And notice here, the response from the model. It takes in that negative review and expands it into an email response that says dear valued customer, we're saddened to hear that your experience with the Oregon Trail has fallen short of expectations. It goes on to apologize, it tells the customer to reach out to customer service with any additional questions or feedback, and it thanks the customer for sharing the feedback with us, and it says best regards signed with your name on behalf of the Oregon Trail. So this is an email that you can then send out automatically to a customer when they send in negative reviews. Now, let's look at the positive review. The response to the positive review is thanking the customer for taking the time to submit the review, telling them we appreciate their feedback. We're happy to hear they had an amazing time. We are humbled to know that our game has helped create a memorable and rewarding gaming experience. We thank them, and then there's a place to sign your name. And for your hands-on challenge, summarize the book reviews for a specific audience with word limit constraints. I've provided two book reviews for you to play around with. Now it's your turn. Good luck, and have fun. In summary, we looked at expanding text. In our examples, we took negative and positive reviews and expanded those into response emails. Next up, we will look at text-to-image generation.

Text-to-image Generation

Now let's talk about text-to-image generation. We'll look at image generation options, three parts to an effective prompt, and tips for image generation. With DALL-E, you can generate brand-new images from a text prompt, edit or retouch an existing image based on a text prompt, and create variations of an existing image. We'll focus on generating brand-new images from a text prompt ensuring those prompts are optimized to produce the most detailed images. When working with AI image generation, it's important to create effective prompts that will help the AI model learn to generate images that are realistic and accurate. Let's look at the three parts of an effective prompt for image generation. First, you have the subject, a noun that is the main subject of the rendered image. Then you have a description which answers questions about the subject and its surrounding scenery, use multiple adjectives. And lastly, we have style which represents the image style. Is it modern, artistic, Picasso-like, Van Gogh-like, wide shot, close up? Don't forget to let your creativity run wild to develop an idea. Let's look at the prompt. Generate a vibrant sunset seascape in impressionist style. In this prompt, the subject is sunset which specifies the central theme or focus of the image. The description is seascape indicating that the image should depict a scene involving the sea or ocean, and the style specified is impressionist suggesting that the image should be rendered in a distinctive style characterized by loose brushwork and an emphasis on capturing the essence of the scene rather than detailed realism. Let's see output of the prompt. Here's the first image, the second image, the third image, and the fourth image. Let's look at another prompt. Generate a majestic mountain landscape in realistic watercolor style. In this prompt, the subject is mountain landscape indicating that the image should focus on a scene featuring mountains. The description is majestic suggesting that the mountain should be portrayed in a grand and awe inspiring manner. The style specified is realistic watercolor implying that the image should be created using watercolor techniques that capture the essence of the scene with a realistic touch. Let's look at a few generated mountain images. Here's the first one, the second, the third, and the fourth. Now let's wrap up with a few final tips on how to optimize your image generation prompts. Text prompts should be at least 10 words or more, include the subject and multiple adjectives as the descriptors, beautiful, realistic, colorful, massive. Include style keywords like contemporary, minimalism, and the name of the artist whose style you'd like to mimic. Avoid using abstract concepts altogether.

Demo: Text-to-image Generation

Let's have a quick demo of text-to-image generation. We'll look at image generation prompts starting with very simple prompts moving up to optimize prompts, and our image of choice is a dog on a bike on the moon. If this is your first time downloading the code, there are several libraries that you'll need to install. Next up in cell 2, you'll need to authenticate to the API using your API key. Here in cell 3, I've created a helper function called generate_image. Notice here in Line 3, it takes in the prompt, so this is the text prompt that we'll use to generate the image. On Line 4, I'm calling out to the Images API which interacts with the DALL-E image generation tool. On Line 5, I'm passing in the prompt. On Line 6, I'm setting 1 for the n parameter which is the number of images you want generated. This is a very helpful parameter because it allows you to generate multiple images at once. Here, on Line 7, I'm setting the size of the image to generate to 256x256. Here on Line 10, the response returned, I'm pulling out the URL to that image and storing it in the image_url variable. On Line 11, I am returning that URL from the function. Here in cell 4, let's look at a sample prompt that we are going to pass in to generate our image. On Line 3, you see a dog on a bike on the moon, and I've already executed most of the cells in this notebook in order to save time. I'm passing in this prompt to generate_image. The URL is returned here and stored in the image_url. On Line 5, I am printing the image_url. And then on Line 7, I am using the image library or module to display the image inline within the notebook. So I've executed this code with the prompt a dog on a bike on the moon. This is the image generated, and what is really funny about this image, it looks like the moon is made out of cheese. So that's the first image. Now let's take that very same prompt and optimize it using all of the techniques that I just taught you about. Here in cell 5 on Line 1, notice the updated prompt, a dog on a bike on the moon, van gogh style, trending cinematographic artstation. And let's look at the image generated. That is definitely in line with the prompt. Let's see if we can refine the prompt even further. The next prompt is create a realistic portrait of a dog on a bike on the moon in the style of renaissance art, emphasizing naturalistic lighting and meticulous attention to detail. Let's see the image generated. This is the image generated. What do you all think? Let's try it again. Generate an image of a dog on a bike on the moon in the style of impressionism with vibrant brushstrokes and a focus on capturing the play of light. Let's see. I actually like this one. It's right in line with the prompt. Let's try another one. Create a digital artwork of a dog on a bike on the moon inspired by Cubism, featuring fragmented geometric forms and multiple perspectives. Let's see this one. This one is definitely in line with the prompt. I like this one too. And let's look at this one, produce a surrealistic image of a dog on a bike on the moon that combines dreamlike elements and unexpected juxtapositions reminiscent of Salvador Dali's style. Let's see this one. Okay. I like that one too. Now, there's a hands-on challenge that you'll try out. Let's talk through that now. Your hands-on challenge is to write an optimized prompt that generates a beautiful image. With Dall-E, users get full usage rights to commercialize the images they create with DALL-E, including the right to reprint, sell, and merchandise. In order to complete this hands-on challenge, you will need to subscribe to the DALL-E API. We looked at image generation options, we talked about the three parts to an effective prompt, which are the subject, description, and style. We also looked at tips for image generation. Make sure your prompt is 10 or more words, include the subject in multiple adjectives, and include style keywords. Up next, let's look at inferring sentiment.

Demo: Inferring Sentiment

Let's look at inferring sentiment. When it comes to inferring from text, topics can be inferred, emotions, and sentiment. Inferring involves extracting hidden or unspoken information from a given text. This hidden information can be applied to different purposes, such as sentiment, which is emotion. Topic inference. A model can look at a piece of text and infer sentiment, happy, angry, or neutral. Now let's have a quick demo. We'll look at inferring topics, emotion, and sentiment. In this demo, we'll look at inferring sentiment. If this is your first time pulling down the code, there are several libraries that you'll need to install. Next, you'll need to import the libraries, make sure you've already generated your OpenAI API key and you've stored it in an environment variable, pull it in, authenticate to the API using the API key, pulling it from your environment variables file. And here we have a helper function called get_completion. In cell 18, notice the prompt is passed into this get_completion helper function. On Line 3 here, I'm calling out to the ChatCompletion API which interacts with the gpt-3.5-turbo model, which is ChatGPT. I'm passing in the prompt as a list of messages. The response is stored in the response variable and then returned from the function here on Line 8. In cell 19, this is the piece of text that we want to infer topics from. I'm not going to read the entire thing to you, but it's about artificial intelligence and how it's revolutionized various industries through advanced algorithms and how it's enabled accurate diagnosis, financial recommendations, and just the overall integration of AI into society raises questions about data security, bias, the impact on human labor, and responsible and equitable AI is important. And here on Line 1, passing into the model, what topics are being discussed in the following text. So it's the text that we just looked at passing in the prompt plus the text to the get_completion function and then printing the response. I've already executed the cells in this notebook in order to save time, and notice here the model said that there are several topics being discussed, the revolution of artificial intelligence, the benefits of AI, concerns about privacy, ethical implications, questions about data security, and the importance of exploring the potential of AI. Sounds about right to me. Here in cell 23, let's look at the next prompt. Determine six topics that are being discussed in the provided text. Make each topic one word, format your response in

JSON. The text sample is here on Line 11. Again, I'm passing in the prompt and the text. Notice here, the output is in JSON format with specific topics, artificial intelligence, healthcare, finance, transportation, privacy, and ethics. Now let's look at inferring emotion. In this prompt, identify a list of emotions in this text. Again, I'm passing in the prompt and the text and printing it out. Notice here the model lists various topics and calls out the potential positive emotion. So revolutionize industry's, excitement, advanced algorithms and machine learning, awe, autonomous vehicles, fascination. Now let's look at a few of the negative sentiments. Ethical implications, concern, job displacement, worry, questions about data security, uncertainty, responsible and equitable AI-powered future, hope. So the model takes that text and infers sentiment and it's pretty spot on. The next prompt, identify a list of emotions that this text will invoke by a reader that doesn't like artificial intelligence and the current advancements. Determine no more than six emotions, format your answer in a pipe-delimited list, pass in here the prompt in the text on Line 9, and the output concern, worry, fear, skepticism, doubt, and unease. Now let's look at inferring sentiment, positive, negative, and neutral. The prompt is what is the sentiment of the following text? And we pass in our prompt and text on Line 3, and the model outputs that the sentiment of the text is positive overall, it acknowledges the benefits of AI, and it emphasizes the importance of addressing the challenges to ensure a responsible and equitable future. And the next prompt is, what is the sentiment of the provided text, provide the answer in one word as either positive, negative, or neutral. And for our text, it prints out positive. Next, infer if a given topic exists in the text. Here's topic_list2 machine learning, technology, security, equity, job replacement. Topic_list1, farming, climate change, weather, drought, organic seeds. Now, here's the prompt. Determine if each item in the following list of topics is a topic in the provided text. The list of topics, topic_list1, so farming, climate change, etc, and the text, and then provide your answer as a list with yes or no for each topic. So in our prompt about the future of AI and how it's revolutionizing the world, it's the list of topics that the model needs to compare, I'm printing out here, and for each one, the answer is no. Farming is not talked about, climate change, weather, drought, organic seeds, etc. Now let's look in topic_list2 to determine if that topic list is present in the text, and this time, we tell the model provide your answer as a list with yes or no for each topic. Because all of the topics were yes, the model simplified that to say yes to machine learning, technology, security, equity, and job replacement. Now it's time for your hands-on challenge, which I'll talk to you about now. Your hands-on challenge is to pass a piece of text to ChatGPT and have it infer the sentiment by identifying a list of emotions provided in the text. Today, we discussed how to use chat GPT to infer topics, emotion, sentiment, all in a piece of text. Up next is text transformation.

Text Transformation

Let's look at text transformation. When it comes to text transformation, ChatGPT can be used for language translation, spelling and grammar, tone adjustment, and conversation formatting or text conversion. It's important to use the techniques we've previously studied to ensure the best performance from the model. Let's start with language translation. ChatGPT is trained in English, Spanish, Italian, German, and many more languages. This means the model can easily act as a translator. Now let's look at spelling and grammar. To signal to the large language model that you want it to proofread your text, you instruct the model to proofread or check spelling and grammar. The model will then output the corrected text. It's really cool. I can't wait to show you this example.

Next is tone adjustment. Writing can vary based on the intended audience, and ChatGPT can produce different tones. For example, it can take 80's slang and translate it to business language. Lastly, text conversion. ChatGPT can translate between formats. The prompt should describe the input and the output format. For example, take this text and translate it to HTML. There are several options when it comes to translation. There are several options when it comes to text transformation. Let's see those in action.

Demo: Text Transformation

Now let's look at a demo of text transformation prompts. We'll look at language translation, spelling and grammar, tone adjustment, and conversion. If this is your first time pulling down the code, there are several modules or libraries that you'll need to install. You'll need to make sure that you've generated your Open API key and that you are authenticating to the API using that key. You've seen this helper function before called get_completion where we pass in a prompt, we call out to the ChatCompletions API using the model gpt_3.5_turbo or ChatGPT passing in the prompt as messages and returning the response. Let's look at language translation. Here, the prompt is translate the following sentence from English to Italian. I love learning new things. We pass in this prompt to the get_completion, store the response and the response variable, and then print out the content of that response. And notice here is I love learning new things in Italian. You Italians out there, let me know if this is right or wrong. Cell 9, detect the language in this piece of text. I simply did a cut and paste on the Italian, and I passed that in to the get_completion function and I'm printing the response, and the model was smart enough to detect that the language in this piece of text is Italian. Next up, in cell 10, the prompt is translate the following text into South American and European Spanish. Please do not drive the car fast. Passing it in to get_completion and printing the response. You Spanish speakers out there, let me know if this response from the model is correct. Because there's only one response, I'm assuming this is correct for South American and European Spanish. Next up is spelling and grammar. The prompt says proofread and correct the spelling and grammar in the following text. If no error is found, simply state no errors. And here's the text. I'm not going to read the entire text to you, but there are intentional errors. For example, me and my friend, we goes to the park yesterday. Incorrect grammar. We play ball and have lots of fun. Incorrect. Another one, my friend laughts here. So there are several errors in this text. Let's see if ChatGPT is smart enough to figure it out. Pass that prompt to get_completion, print the response. Me and my friend, we went to the park yesterday, so it corrected, we goes to the park yesterday, and we play ball and have lots of fun. It corrected it, we played ball and had lots of fun. So it looks like it, it caught all of the grammatical errors, which is really cool. Now, anytime that you have a piece of text that you need to check the spelling and grammar, you can use ChatGPT. Here is another example, proofread and correct the spelling and grammar in the following text. If no errors found simply say, no errors. Notice here, the text is actually the corrected text from the previous example, and notice ChatGPT found no errors. Now let's look at tone adjustment. Here is 80s slang that we want ChatGPT to translate to formal business language. Again, I'm not going to read the entire sentence to you. I'll call out a few important keywords. We start with hey dudes and dudettes. Guess what? That is definitely 80s slang. I just scored some totally rad concert tickets to see my fave band, The Electric Dreams. So we pass this whole prompt in and let's see it translated to business language, and ChatGPT responds, Hello, colleagues and associates. I wanted to share some exciting news. I think it did pretty good at translating 80s slang. What about 90s slang? Here's the prompt. Hey, peeps. Guess what? I just scored some wicked cool tickets to the dopest concert ever. Definitely 90s slang. And let's see the response. Dear colleagues. I'm thrilled to inform you that I have recently acquired tickets to an exceptional concert. That does sound very professional. Slang from the 2000s. Hey, guys, guess what? I just snagged some awesome tix to the sickest concert ever. And hello, colleagues. I have exciting news to share. And of course, I had to add current slang. Hey, fam, guess what? I just scored some lit tickets to the sickest concert ever. And ChatGPT says I'm excited to inform you that I have obtained tickets to an exceptional concert. And now let's wrap up with text conversion, converting from one format to another. Here, translate the following list to HTML, Item 1, Item 2, Item 3, and Item 4, and pass that prompt in, and here is the output using the unordered list and the list element tags of HTML. Can translate to JSON, translate the following list to JSON, cat, dog, horse, and cow. This is what the model returns. Translate to SQL. Translate the following list to a SQL insert statement, cat, dog, horse, cow. It does convert it, it makes some assumptions that we have a table called animals and creates the insert statement for the animals table. And translate to Java. Translate the following list to a Java function that inserts the following values into a database cat, dog, horse, cow, also provide a function code written in Java. And notice here it spits out a DatabaseInsert class in Java with all of the necessary import statements. It looks like you have to hard code your username and password. We have to talk to ChatGPT about that. And then here's the rest of the code combined in a try catch finally block and a main method for implementing it. Now it's your turn. Let's talk about your hands-on challenge. Your hands-on challenge is to write a prompt to perform tone adjustment and language translation. Have fun with it. In summary, we saw that ChatGPT can translate from one language to the next and it supports a wide range of languages. We looked at spelling and grammar examples where ChatGPT can fix spelling errors and repair grammatical mistakes. We saw examples of tone adjustment and conversion from one format to the next.

Conclusion

This was a fun module where we looked at summarizing or condensing and expanding text which allowed us to see the creativity of ChatGPT. We looked at text-to-image generation where from a text prompt, you can generate a custom image that you own. We then looked at inferring sentiment allowing ChatGPT to label text as happy, angry, or neutral. And we wrapped up with text transformation where we're able to translate from one language to the next, adjust the tone, and more. Up next, we'll discuss key factors that impact the latency and performance of text generation models, and we'll look at how you can design prompts that improve performance.

Design Prompts That Improve Performance

Introduction

There are strategies that you can use to improve the overall performance of the model and the quality of the output returned. In this module, we'll look at latency and the factors that influence latency, which in turn, impacts the model's performance. Once you understand the factors that improve latency, we'll look at ways to design your prompts so that latency is reduced and performance is improved. So let's get started.

Factors That Impact Latency

Latency is a well-known term in the tech industry. Let's look at common factors that impact it. Latency is the time it takes for a request to be processed and a response to be returned. Latency is one way to evaluate the performance of your model and most of it arises from the token generation process. The flow of a completion request looks like this. You have the end user to ChatGPT API latency, the time to process prompt tokens, the time to generate tokens, and the ChatGPT API to end user latency. What is a token? A token is a common sequence of characters found in text. ChatGPT processes text using token. For example, the string tokenization is decomposed as token and ization, while a short and common word like the is represented as a single token. When the model returns a response, it is actually predicting the next token, not the next word. In our example prompt, translate the following sentence from English to Italian: I love learning new things. There are 16 tokens. OpenAI provides a tool to help you count tokens and better understand how a piece of text would be tokenized by the API and the total count of tokens. Latency is controlled by two main factors, the model and the number of tokens generated. OpenAI, the creators of ChatGPT, offer different models with varying levels of complexity and generality. What you choose can impact latency. Requesting a large amount of generated tokens can lead to increased latencies. Keep that in mind as you're developing your prompts and optimizing them. Now that you understand what impacts performance, let's understand how to design prompts that improve model performance and inference time and discuss mitigation techniques to improve performance.

Improve Prompt Design

There are several steps you can take to improve the overall performance and mitigation strategies you can employ. Let's talk about those now. Select the best model for the problem you're trying to solve. You can choose the model that best suits your use case. The most capable models, the newer models, can generate more complex and diverse completions, but they also take longer to process your query and can be slower and more expensive. The older models can generate faster and cheaper chat completions, but they may generate results that are less accurate or relevant for your query. Make sure you understand the tradeoffs between speed and quality. As a prompt engineer, your goal is to design the prompt to reduce the number of completion tokens. You can do this by using stop sequences, the max_token attribute, and the n and stream attributes. Let's talk about these now. In this example, to prevent generating unneeded tokens, we put a limit, and in this case, recommend three books in the mystery genre, three is the stop sequence, and it limits the response from the model. You can also use the max_tokens parameter to reduce the number of tokens. This also incurs less latency. In this example, in our call to the ChatCompletions API, we're putting max_tokens at 15. This means the model will return at most 15 tokens in the completion response. Another option is the n parameter, which is how many ChatCompletion choices to generate for each input message. The default value is one. In our example, we're setting n to 3, which means that we want three chat completions to return, which could increase latency. The n parameter is often combined with max tokens. Next up, there is a stream parameter which returns tokens as soon as they are available. Setting it to true makes the model not wait for the full sequence of tokens to be generated, and as tokens are generated, they are returned. This does not change the time to get all of the tokens, but it reduces the wait time for that first initial token, and this can be a better user experience for users. You can discover better prompts over time by tracking and versioning your prompts. It's also great to experiment and test out different prompt ideas to see what works best for you. For a hands-on challenge, see if you can reduce the number of tokens in a given prompt while still maintaining response quality. Update one of the prompts from the summarizing and expanding text lesson to reduce the number of tokens returned. Congratulations for making it to the end of the course. Let's review everything you've learned and the next steps for putting what you've learned into practice.

Conclusion

Before we wrap up the course, let's wrap up this current module. We looked at factors that impact latency like the chosen model and the number of completion tokens. We also looked at ways to improve the prompt design by using stop sequences and the max_tokens n and stream attributes. Now let's officially close out the course. Congratulations. You gained the ability to harness the capabilities of language models effectively, ensuring they align with specific use cases, minimize biases, and deliver high quality outputs. First, we discussed advanced prompting techniques that elevate a model's performance like chain-of-thought prompting that discourages the model from generating quick answers. We looked at knowledge augmentation that provides additional training data to the model within the prompt, and prompt tuning which allows us to define and optimize an existing prompt. Next, we looked at the art of prompt optimization for specific use cases like summarizing and expanding text, text-to-image generation, inferring sentiment, and text transformation. Lastly, we discussed the key factors that impact the latency and performance like the chosen model and the number of completion tokens and improving prompt design by using stop sequences, max tokens, and n stream attributes. To continue your learning journey, continue to experiment hands-on with ChatGPT and look for courses in the Pluralsight library on the OpenAI API and large language models. I'm Kesha Williams. I'll see you next time.