OpenAI Prompt Engineering for Improved Performance

Course Overview

Hi there. Welcome to my course, OpenAI Prompt Engineering for Improved Performance. My name is Tim Warner. I'm a Pluralsight staff author, and I'm happy to be your instructor. In today's fast paced tech landscape, ensuring optimal performance while integrating OpenAI APIs is crucial. This course is designed for software developers who prioritize application efficiency, looking to master OpenAI APIs without sacrificing performance. In this course, you'll first get familiar with the basics of GPT and prompt engineering. Next, you'll practice with your newfound techniques to generate text, source code, data, and images efficiently. Finally, you'll explore advanced techniques, optimizing query response times, and steering responses to obtain more focused and desired output from ChatGPT. By the end of this course, you'll have honed the skills and acquired the knowledge of GPT prompt engineering needed to streamline your applications and improve your effectiveness as a working software developer. I hope you'll join me on this journey to master prompt engineering techniques with the Open AI Prompt Engineering for Improved Performance course at Pluralsight.

Optimize GPT Prompt Performance

Overview

Hello there, and welcome to Pluralsight. Welcome to our course, Open AI Prompt Engineering for Improved Performance. There's one content module in the course, and that's this one, Optimize GPT Prompt Performance. I'm your instructor. My name's Tim Warner. The four learning goals I have for you in this lesson are as follows. We're going to start by delving deeper into prompt engineering advanced techniques. We'll discuss some few-shot prompting methods with GPT. We'll leverage system messages, including role, to steer responses from the AI. And lastly, we'll evaluate methods for optimizing query response times. The main audience for this course is the application developer who's interested in integrating the OpenAI model, particularly the GPT, the Generative Pre-trained Transformer model, large language model to be more specific, into your line of business applications, and you want to ensure you don't create a bottleneck with that functionality. You're looking at keeping performance nice and high. The exercise files for this course can be found in the usual spot. In the full pluralsight.com browser experience, you can flip over from the Table of contents tab to the Exercise files tab, click Download exercise files, and away you go. Normally for me, at least with these OpenAI courses, there'll be a links list that contains helpful references, web references that supplement the training, and I may have a Python script for you as well. We'll see, let's let it be a surprise. With that, let's get into the content.

Prompt Engineering Advanced Techniques

Prompt Engineering Advanced Techniques. If you've studied with me before, you know that I like to level set before we get into the high weeds to make sure that we have a common vocabulary, so let's just take a step back for a moment and understand what GPT is. It is a machine learning model, specifically a large language model, or LLM, Generative Pre-trained Transformer, OpenAI's artificial intelligence tool, that what can it do? It processes and generates human-like text, aiding businesses and automating customer service, content creation, data analysis, and so on. GPT has two main versions. The two mainline most common versions are GPT version 3.5 Turbo and GPT-4. Choosing the correct model is really important as an application developer. For that, I'm going to draw you back to the Pluralsight library because I've done a mini course on choosing the right model, So we're going to continue from there. Prompt engineering is the process of crafting and refining your inputs to a large language model. These are the prompts that guide the response of GPT towards your desired outputs or behaviors. You should know if you don't already know that GPT-3.5 Turbo and GPT-4 are multimodal LLMs, which means they're equally adept at taking your plain text input, your prompt, and giving you back any combination of natural language, source code, or even images, thanks to the DALL-E 3 integration. Prompt engineering, this is something I mention all the time to my students and customers and beyond, that prompt engineering is really the main way you quote-unquote program GPT. So to that point, let's take a look at some core prompt engineering principles and so-called advanced techniques that you should have in your shirt pocket as you break ground integrating GPT into your applications. First of all, we have explicit instructions and formatting those instructions. This is where, as an application developer, you know what I would recommend? In addition to making sure you're sharp on your languages and your frameworks, I would say that you might want to pick up a guide on technical writing. And I'm serious when I say that, because the quality of your prompt is directly proportional to the quality of the completion, or the response from the AI. And all the knowledge of C#, Java, JavaScript, Python in the world isn't going to help you get better results, it's going to be the language in your prompt. So I'm serious when I say studying technical writing, because as explicit as you get is going to help you get the best results from the model. There's also controls, or parameters, that you can tune when you're addressing the model via API. One is a property called temperature. This is a degree of randomness. Lower values, it goes from 0 to 2, low is the lowest temperature, you're going to get the most predictable, deterministic response. If you cranked it all the way up to 2, you'll probably get, well, you'll definitely get more creative and expansive responses, but liable, you're more liable, I should say, to get garbage. So you're going to want to throttle that temperature so that you can control, to a degree the verbosity, but mostly the deterministic versus randomized behavior of the AI. We always want to keep in mind token control because we're paying OpenAI for the tokens that we spend. I'm going to make sure to include a little mini lesson on tokens in the demo upcoming, and I've discussed tokens in great detail in my other OpenAI courses here at Pluralsight. So I hope that if you like this course, you'll go and you'll study all the rest of my courses, so you can get as well-rounded at grounding in using these OpenAI models as possible. And I don't want to leave out my colleague Hampton Paulk. Amber Israelsen is another one who is also contributing content to this Pluralsight OpenAI partnership. Check out all of our stuff, okay? Anyway, we want to think about as a developer, not only optimizing our prompts for AI accuracy and the completion, but optimizing our spend as well, and that comes down to tuning temperature, but mostly tokens. Active learning is another prompt engineering method. This you can do right alongside. I like to look at active learning as learning alongside the AI. You can use the Advanced Data Analysis, or ADA plugin, upload data sets into the AI, and perform exploratory data analysis. Another thing that goes with active learning is fine-tuning the model on your own internal line-of-business data. You probably know that GPT has been trained on public open source data, but of huge value of consuming the OpenAI models directly via APIs is that you can run fine-tuning jobs and make your model smarter on your internal line-of-business data. Lastly, a lot of this prompt engineering stuff comes down to experimentation and iteration. To that point, there are the web playgrounds. OpenAI has a playground environment that we can play with. Then in your applications, there's ways that you can interact with the API and save money. For instance, you can use the tiktoken library, the Python library, in order to figure out how many tokens you'll be burning without actually doing the work. That's an important thing that tiktoken has as a helper.

Few-shot Prompting Methods

When you're looking at GPT in the news and you're reading up on prompt engineering techniques, you'll often hear the notion of zero-shot or few-shot prompting methods. Zero-shot is simply you send in one prompt, you get one completion back, and you're done. It's a one-for-one one and done, and that may be what you want. An equally popular method, especially during development, is few-shot prompting. Well, actually, this could work with the goal of getting a zero-shot response. Basically, the shot is how much source data are you giving the AI? Zero-shot, you're giving no examples. You're just saying, tell me this, do this, what is that? And the AI gives you a response. Few-shot is where you provide the model with a small number of examples. That's really what a shot is. So you say perform an analysis on this dataset and give me the results that look like this, and you'll do source with X result Y. So you're modeling in your prompt, using examples of how you want the AI to behave, the work you want it to do, and importantly, how you want it to format the results. You know as a developer that parsing and formatting output is part of our daily grind. It's part of our daily work. I think you'll be impressed on how much work you'll be able to save by delegating or offloading a lot of that data pruning and processing and output formatting to the AI. It's really quite wonderful. This is why I think OpenAI in their marketing literature likes to describe and position the GPT model as your personal chief of staff. I actually love that metaphor because when you think about what a chief of staff does, they're basically there for you to delegate work to. Whatever that work is, they'll handle it. It's a wonderful luxury. And once you start interacting with GPT that way, I think you'll find your productivity begins to soar, and you'll see, wow, I can really enhance the value of my line-of-business application by performing this integration with GPT. Let's look into more details on few-shot prompting methodology. Like I said, the idea with few-shot is that you're including examples of input and output in your prompt, and you can specify formatting as well. Now, at the same time, I don't want you to forget about what I mentioned earlier about optimizing for cost and performance. Remember, well, I shouldn't say remember because you might not yet know that GPT, they're all REST APIs. So you're at a big advantage if you've already done REST development, you know about the HTTP request response cycle and you know that the data payload in those transactions is in JavaScript Object Notation, or JSON. So it's a lean mean, efficient data serialization format, but you're going to pay for every character you have in your input, so you want to think about being as efficient as possible. And frankly, from a performance perspective, a larger JSON file is going to take longer, generally speaking, to go to and from the GPT endpoint than a smaller JSON fragment. That just stands to reason. Few-shot prompting is helpful when you're fine tuning because your examples can help the model get more comfortable with your line-of-business data. That's a good tip right there for sure. So that brings in two concepts, how to fine tune, which you can find, again, I'll boot you back into the Pluralsight library on that one. But given that you fine tuned your model, you're giving the AI even more assistance on how to understand and interact with your source data by providing these few-shot examples, all right? Another thing that’s important is addressing possible data bias. How can you as a developer uphold responsible AI principles? This is particularly important when you’re doing fine tuning on your own internal data. You can compare your dataset against known good exemplars. You also can use tooling. You can use statistical methods, bias detection tools like IBM AIF360 or FairComp on your custom training data. The idea there with AI and machine learning and it’s why the question of data engineering is so critical to AI is that we want to make sure that the source data that we’re training the model on is as free from bias as possible. You know, the old GIGO rule, garbage in, garbage out, that’s what I’m talking about.

System Messages

Next, let’s discuss system messages and how they contribute to performance efficiency with your applications. To do this, I have a screenshot of the OpenAI Playground where we can see a differentiation between what’s called the system prompt and the user prompt. Now, this is a really important point that’s not obvious if you’re just diving right into the AI. You can include or infer the system and user prompts together by just providing a single prompt, but I always like to err on the side of being more explicit. Well, first of all, what the heck do I mean by system prompt? The difference here is that the system prompt provides the AI with meta instructions where you can do things like roleplay with GPT. In this example, I’m saying you’re an expert, I’m talking to the AI here, you’re an expert in Windows Server administration, Active Directory, and PowerShell. You apply best practices to your PowerShell code and always explain what you write fully. And I’m also in my system prompt, describing my persona, I am technically proficient, so speak to me at technical depth. Make sure to give me the most current information you have. What we’re doing with the system prompt is reducing the number of assumptions that GPT needs to make. If you don’t tell the AI what your skill level is, it just has to guess, and you can wind up with completions that are too complex or too elementary for your application’s use case. You can absolutely, and should, consider these system prompts in your application code in addition to what you’re doing just ad hoc personally with the model. The user prompt is the actual task that you’re putting on the AI. In this example, I’m saying, Generate a PowerShell script that installs IIS Web Server on a Windows Server 2019 domain member server named mem1. Now side note, when you’re consuming the GPT APIs, OpenAI will not use your prompt data to train their model. Please understand that. So note that I am putting a real member server name in my environment, mem1. You do want to be careful about proprietary information in your prompts just in the name of general security and security hygiene. Note that in my user prompt, I go on to say, Also create a demo website that listens on TCP 443 and contains a simple placeholder homepage. And then we have the completion. This is what OpenAI calls the response. And here, the assistant role, those are actually named roles by the way, system, user, and assistant, so that’s a good tip for you. When you’re making your prompts, you can literally prefix your system and user prompts. And that, again, is more explicit and lets the AI have a leg up because it has a better idea of exactly what you’re telling it. The assistant role is the AI, and that’s the response that comes back. Now, the token count is going to be a sum of not only the tokens from the assistant, but also in your system and user prompt. So that exchange, system plus user plus assistant equals your token count for that exchange. And then if you’re using GPT-3.5 Turbo or GPT-4s chat completion endpoint rather than the garden variety completion endpoint, you should know that we've got conversation history. This is one of the benefits of ChatGPT, that you don't have to repeat yourself at least for a while. And the AI understands previous prompts and responses. And how that works behind the scenes, I can easily tell you this because you're a developer, is that ChatGPT or the Chat endpoint will simply include in the JSON payload all previous prompts and responses. So as you go through a multi-turn conversation with ChatGPT, that token count is gradually getting bigger and bigger. And depending upon the size of the model, you're eventually going to reach the token limit and then the oldest request and responses will fall off the payload and you may have to refresh the AI with further prompts to remind it of what it's forgotten. Some tips on implementing system messages. First of all, just the reiteration, that's basically what this slide is, that they're important to steer GPT's responses. And I mentioned the role playing techniques, the system and user prompts, the assistant and user roles. You can also make up your own role names like tutor and student, but fundamentally, they're arbitrary except for system, user, and assistant. Those are also understood internally by GPT. In summary, system prompt sets the conversation's tone and content focus.

Optimize GPT Query Response Times

Last subject before our brief demo is optimize query response times. I've split this into two. First, I have GPT API app latency reduction strategies, just in general because I mentioned that GPT is a RESTful service. You're going over the internet to communicate with OpenAI's endpoints, so you want to think about as a developer caching on where you can store the results of expensive API calls, request batching to minimize the number of calls. Most languages have asynchronous paradigms where you can do concurrency, particularly nonblocking await type concurrency, general principle of adding error handling, retry, and backoff logic. You will get throttled from time to time. In fact, I'll show you in the demo, OpenAI shows you your quotas and your limits. So in order to avoid HTTP 429s, you'll want to be proactive as a developer and include error handling, retry, and backoff logic in your code. Then in the name of efficiency as well, compress those responses to make them fly faster. The other set of latency reduction strategies are more to GPT specifically, like I mentioned, choosing the right model. Choosing the right model is going to come down most of the time to either GPT-3.5 Turbo or GPT-4. And then once you choose your model, you'll want to choose your context length. For example, GPT-4 has up to a 32K context length, whereas, GPT-3.5 Turbo is 4K, quite a difference. And there's some variation in between. You'll want to track your token expenditure using the Python tiktoken or a similar library. Python is definitely the lingua franca in AI. And, again, I'll include this in the demo, but you'll want to make sure you're sorted out with the best libraries and tools for the tasks. Next, you'll want to think about preprocessing your input data to ensure acceptability, consistency, and clarity. Now there's a lot we could talk about with this. So if you're allowing your application users to submit free text that will ultimately be formed into a prompt, you want to run it by a moderation API. Now OpenAI allows you to consume their moderation API. You can customize your own API, your own moderation API as well if you want it to flag particular words, concepts, etc. That's what I mean by acceptability. Consistency and clarity, you'll want to think about token expenditure and just is this prompt in the best fighting shape before it goes to the AI. Post processing the models output to meet your application requirements. I mentioned that you can have GPT do a lot of that for you. And then keep in mind that you can update, not just your prompts, but you can iterate on different model parameters like temperature and max tokens as new data requirements emerge. What about performance monitoring tools? Well, you probably have a performance monitoring stack, but let me just give you some common ones that are mentioned by OpenAI itself, Prometheus, Grafana, ELK Stack. If you're in the cloud, there's AWS CloudWatch, Google Cloud Monitoring and Logging, Azure Monitor and Logs. Let's get into that demo now.

Demo: Tour the OpenAI API Docs and Libraries

All right, let's get this party started. Our subject is how can you as an application developer interact with the GPT APIs with maximum performance, keeping all of these practices in mind? Now, I mentioned several times in the theory part of this lesson I was going to show you some stuff, and all that stuff can be found here at the OpenAI platform site, platform.openai.com. I'm signed in. I'm actually part of a ChatGPT Enterprise workspace, but I also have API access. So to show you around here, if you're not familiar with the site, let me go over to API reference, and this is definitely important for you to understand how to address the different endpoints. The main one that OpenAI promotes to their customers is using the chat completion endpoint, primarily because it's the most robust. Yes, the chat endpoint has a higher token count because of the conversational wrapping. When you're addressing the completions endpoint, you just get a straight up response from the model. Chat does have that extra layer of conversation to it, but also very importantly, the chat completion endpoint will store that history, that message history over time, as I've mentioned before. Let's now go over to Documentation, Libraries. And I mentioned that Python is the most highly supported language, so if you're not a Python programmer, I might want to invite you to get into that because it's definitely going to carry you forward the fastest. OpenAI maintains Python and Node.js libraries, as you can see here. Microsoft has the OpenAI service, the Azure OpenAI service, and they have client libraries for .NET, JavaScript, Java, and Go. I mention that because, as you can see, those libraries are compatible with both the OpenAI API as well as the Azure OpenAI service, so definitely take a look at those. And then you see that there's a whole bunch of third-party community libraries covering a whole bunch of different languages and frameworks. That's really important. I want to take you out to the Playground next, and this is where you can play with the model. It defaults to the chat endpoint, and at least as of this recording in fall 2023, I don't have the ability to adjust GPT-4 probably because, well, I don't know because I'm paying for anything I send into the Playground here. That comes down to your account properties here. If I go in to Manage account, you'll have your Billing information there and you can get to Rate limits and see what your usage is. Remember, I mentioned all that. Basically, platform.openai.com is your home base when you're an OpenAI developer. That's basically what I'm trying to say here. Let me think for a minute if there's anything else I wanted to say preliminarily. I think that's good. Oh, one more thing, yeah, about the tokens. If you go to platform.openai.com/tokenizer, you'll see this is not super duper useful because it's limited to GPT-3. I guess that's fine if you're using GPT-3, but the main thing about this is that it's a way to see the relationship between your prompt and the tokens.

Demo: Explore Tokenization and System Messages

So you can see as you type, GPT will convert your plain text input into these tokens. And what they say, what OpenAI says is that a token is generally about four characters of English text, and you pay for the token count in your conversations. You get a discount on your prompt. You do pay for both prompt and completion, but you get a really nice, I think it's upwards to 50% discount on the prompt. And notice that it says, If you need a programmatic interface for tokenizing text, check out our tiktoken package for Python. There's another library for JavaScript, and there's community ones for other languages. But what I want to just mention here briefly with tiktoken, what's cool about this is that, and libraries like it, what's cool about them is it allows you to model token consumption without burning up real API calls. So in this example here where this is Python, we need to do a pip install tiktoken and then import tiktoken into our session. What happens is each of the GPT models has its own encoding, and this is the algorithm that's responsible for converting the plain text into these tokens. And you can actually deconvert the tokenized text back to plain text. And so token, tiktoken rather, you either specify a specific encoding or I like this one here where you can run encoding for model and pass in the short name of your model, and this enc variable will automatically use the appropriate encoding algorithm for that model. And then once you're doing that, you can just run encode and decode methods to take that prompt. And you could even throw in a prompt completion pair here, it's just a string, and it'll give you back your token count. Again, the great benefit here is that you're not burning up real API calls, real tokens, you're doing this offline on the client side. Now, as far as a concrete example, I'm going to stay with Python because that's my jam. I'm here at chat.openai.com. I'm going to do the rest of this demo actually asking the AI to generate the code for us, figure it'll be a nice way to dual purpose. We can play with prompt engineering techniques, as well as see live code. I'm in, as I mentioned, part of a ChatGPT Enterprise workspace where I've got all features turned on, as you can see. And if you've wondered what the so-called GPT-V capabilities are, the little icon down here, Chat with images, means that I can upload an image and I can interact with the API and ask questions about it. If I open up the GPT-4 menu, I have access to Advanced Data Analysis and Browse with Bing first-party plugins, third-party plugins, DALL-E for image generation. I'm just showing you this just to show it to you to make sure you've seen it. But this, that is the ChatGPT Enterprise product and the ChatGPT+, the standalone individual product, you can look at those as Software as a Service, or SaaS solutions. As a developer, remember, we're going to be back at platform.openai.com addressing the API. Here, this is quote-unquote ChatGPT. That's a SaaS web application and a downside actually, unfortunately. Eventually, it won't be a problem, but right now, it's a downside is that we don't yet have API access to plugins and uploading images. Eventually, I'm sure, OpenAI will give us that capability. We have to remember that we're early days with this technology. All that having been said, let me go to a chat that I started recently, and I here's the prompt, and we're going to optimize this prompt. So I'm actually going to select this prompt, copy it to my clipboard, and we'll throw it in VS Code. I have GitHub Copilot Chat enabled. That's another subject for another time. I'm going to start by just creating a simple text file. I'll call this prompt.txt, and I'll throw in what I had in ChatGPT, and let's see if we can refactor. This is a pretty stout prompt. You might be wondering how many tokens is this going to consume? Well, we're actually going to get some help on that. We can use tiktoken as a matter of fact. But what am I saying? I'm saying, Please give me a Python script that performs these tasks. Now, I mentioned the system prompt user prompt. This would be a good example of where the user prompt comes in. What about the system prompt? All right, here we are back in ChatGPT. Again, some of this can get a little confusing. I hope I'm going to do a good job of clearing up that confusion if it's there for you. But if I come down in ChatGPT, there's something called Custom instructions there. And this is where you can role play with the API. Now, I'm actually going to select this. I'm not going to use custom instructions in the ChatGPT AI, I'm going to turn it off. And let's go back to VS Code now, and I'm going to add this to my API prompt. I just want to show you this end to end. So notice now that I'm telling in my prompt what the AI is and then I'm going to say in my user prompt, I'm not going to use my own persona here. I'm just going to go straight to a user prompt, and I'm using soft line returns here. You want to think about that as well, actually. Another thing you can play with back at the tokenizer, let me show you what I mean. Let me just grab some of this Python code here and throw this into the tokenizer back in Edge.

Demo: Generate Python Source Code

Let's come back to Edge, and let me open up a browser tab, go to platform.openai.com/tokenizer. And let me show you what tokenization happens with source code. It's one thing when you see text, it's another to see source code. And I want you to see here that we're getting dinged on these indents. You know, Python is all about readability. So you want to think about, ultimately, the prompt that shows up in your application, you'll want to minify that because you want to minimize your token consumption here. Let's see if there's anything else we might be able to trim besides the indents and the line returns. It looks like the line returns aren't costing us anything. Don't try to read too much into the encoding algorithm, at least that's my suggestion because sometimes it's a little bit nonsensical. For instance, this highlighted piece is a closing paren and a colon. This token up here is both parentheses and a colon. You see what I'm saying? So it's something, tokenization is something to absolutely keep at the forefront of your mind, but in terms of trying to make it make sense, I haven't gotten there yet. And we have to remember that where OpenAI does have for-profit and non-for-profit branches, their GPT models and the actual encoding algorithms are all proprietary. So there's only so much they can tell us being a for-profit business. Anyway, I'm saying in my user prompt, Please give me, in fact, that's something I can strip. Instead of, please, I can save some tokens and even give me create. This is what I mentioned before where I mentioned that prompt engineering is the secret sauce, is mastering your spoken language and your written language and being really concise. And I'm using few-shot here by saying, first of all, I'm generating a list of tasks that I want the AI to do. And then in step 6, I'm saying, present formatted results using the following format, and there we have that. You'll see sometimes people separating system and user with some hash characters. That's something you can do if you want, but that's just extra tokens. What I personally like is this notation of system and user, it's compact, it's really explicit. And as far as giving numbered lists and unordered list formats, I think of markdown really. I like to use the hyphen because we could get geeky about this if you wanted to. My friend at Microsoft, Sean Wheeler, taught me this that semantically, if you're doing an unordered list with an asterisk, you're kind of mixing the message here because the asterisk has a markup capability in mark down. Whatever, let's leave that alone. So ultimately, let's just take a scan of what the tasks are. We want the Python script to authenticate to the OpenAI using the GPT-3.5 Turbo model. We're going to clear the terminal screen. That's just something that I like so I don't have to clear the terminal before I run the script. Obviously, that's going to be jettisoned before we go to production. Display the welcome message to the user, prompt the user, wait for response. So we're going to actually get a response from the user here. And then we want to pass the response to GPT via the Chat endpoint, not the Completion endpoint. See how I'm using natural language here. I found in my experimentation, unless I'm being explicit, the code that GPT will generate will default to the Completion's endpoint. And ultimately, I want back the original prompt, the refactored prompt, the total tokens consumed, the estimated cost, and then my round trip time. Lastly, I put as an afterthought almost, use tiktoken and assume a cost of, and I got this cost from the OpenAI pricing website. And I specified the current price in USD per 1000 tokens for input and what it costs for output. Pretty interesting, right? So that's what we're looking at. Let me right-click, copy this. Let's go back to our application. And again, I'm going to ask the AI to do this for us. I'm going to start a new chat, and because I have the convenience of using plugins, I'm going to choose Browse with Bing. That is really helpful, at least, eventually, we'll be able to surface it in the API, but that's going to help us ensure that the AI can go beyond its knowledge cut off date. So let me come in here and paste this prompt in, send it in. Parenthetically, while we're waiting, another nice benefit of ChatGPT Enterprise is that your users get access to not only GPT-4, but it defaults to the longest 32K context length, so you'll be able to see more context over time. Okay, so we can scroll down and, again, this is an application, so notice that the code comes back formatted. We can easily copy it out. That's because the front end was created for us by OpenAI. When you're interacting with the API, it's all going to come back as pure JSON. Let's take a look at this code. Actually, before I do, I want to do one other thing here. I want to show you, I'm going to say, Generate a sample JSON response from the ChatCompletion endpoint that shows a multi-turn conversation between the User and Assistant roles. Now I was going to show you a live example, but in the interest of time, I find it so convenient and so reflexive nowadays to turn to GPT, especially if you're fortunate enough to have a ChatGPT Plus or ChatGPT Enterprise license.

Demo: Analyze Python Source Code

And actually, the results that came back from the AI aren't exactly what I want, so let me shift over here to Postman. Hopefully, you know what Postman is. It's a wonderful Software as a Service application. And what I want to say here, again, parenthetically, is that OpenAI has a really great Postman collection. I'll include a link in the course exercise files that has all these sample requests, as you can see down here, for all the different endpoints. We're looking at a chat message completion sample request, and we're looking at the sample response here. Actually, we're looking at the request, and this is really what I wanted to show you. Let me zoom in a little bit more if I can. No, I guess I can't. That the way that ChatGPT handles remembering the conversation history is really clever, but elementary at the same time. You have a messages array where you've got this distinction between user and assistant. Here, it's using a different nomenclature, but it's the same idea. This is showing just a multi-shot example, role, user, content, okay, I'm sorry, it shows user, assistant, user, and then we're going to get assistant back when we send this request to the API. I don't think I've got it wired up. Let me check.. No, I'm missing authorization. But that's basically what happens. So eventually, this messages array grows to the token limit of the model and then the oldest ones fall off the vine. Does that make sense? All right, good. Now that I've covered that, let's just take a quick look at the code that ChatGPT gave us. And remember, we're asking it to authenticate to OpenAI API, form a prompt for the user, get a response, etc. So let’s take a look here. So we’ve got in our imports, we’ve got the ones that are relevant for our purposes. We need tiktoken. I’m really surprised I’m not seeing an entry for import OpenAI, to be honest. That’s another discussion where we want to think about GPT and hallucination. It says this down here in the fine print that responses might contain inaccurate info. So we’ll need an import for OpenAI as well. You’ll not want to put your API key in plain text here. You’ll want to have that as an environment variable or from a secret store. We’re modeling creating variables for the endpoint, the model we want, the price for input and output. Clearing the screen, here we’re getting our token count by interacting with the tokenizer. We’re going to capture the original prompt. We’re going to strip whitespace from it. And then ultimately, we’re going to authenticate and send in that JSON to the API. So this code, it’s going to need some refactoring for sure, but that’s just part of life having GPT generate code for you. And it’s something I always tell customers and students, trust but verify. It’s going to perform the math to give us our costs, and then it’s going to use the good old print function in order to give us our output. So that’s at least a starting point for us to work with. I think it’s incomplete, but basically, it’s showing the lay of the land where we can see how we can model cost, how we can take input from the user, and then we’re in a position to parse, moderate, format that data, form really, ultimately, it’s just an HTTP request that we just as well could do in cURL or in Postman, but you’re going to do in your application. You’re going to get that response from the service, and you’re going to lay it into your formatted results.

Summary

Well, there you have it, the end of the proverbial day. I’m going to say one more time that you know now that the OpenAI APIs are all garden variety RESTful services. So you’ll find that if you’ve done REST development, your learning curve will really be most likely with learning the OpenAI API definitions, which as you’ve seen, is very garden variety. And you know, I’m going to throw on even though it’s not on the slide, I’m going to repeat what I said at the beginning, that a good leg up you can give yourself being an effective prompt engineer, as well as application developer is to get sharp in your native language, I’m talking your communication language, not programming language, and master how to write accurately and concisely because that will literally pay off in reduced expenditure with the OpenAI APIs. And then in terms of performance optimization, a reminder for you to think at every layer, not only in your app logic, but think about the network path between your source and destination and what you can do to tighten things up. Well, that’s the course. I hope you learned a lot. I appreciate you. You can check out all the rest of my OpenAI content, as well as the other stuff I’ve done over the years at Pluralsight. I created a short link, timw.info/ps or you can reach me via my work email address, tim-warner@pluralsight.com. Thanks again. Happy studying.