you saw how to evaluate an LLM output in an example where it had the right answer. And so we could write down a function to explicitly just tell us if the LLM outputs the right categories and list of products. But what if the LLM is used to generate text and there isn't just one right piece of text? Let's take a look at an approach for how to evaluate that type of LLM output. Here's my usual helper functions, and given a customer message, "tell me about the smartx pro phone and the fotosnap camera.", and so on. Here are a few utils to get me the assistant answer. This is basically the process that Isa has stepped through in earlier videos. And so here's the assistant answer, "Sure, I'd be happy to help!". Smartphone, the smartx pro phone, and so on and so forth. So, how can you evaluate if this is a good answer or not? Seems like there are lots of possible good answers. One way to evaluate this is to write a rubric, meaning a set of guidelines, to evaluate this answer on different dimensions, and then use that to decide whether or not you're satisfied with this answer. Let me show you how to do that. So, let me create a little data structure to store the customer message as well as the product info. So here, I'm going to specify a prompt for evaluating the assistant answer using what's called a rubric. I'll explain in a second what that means. But this prompt says in the system message, "You are an assistant that evaluates how well the customer service agent answers a user question by looking at the context that the customer service agent is using to generate its response.". So, this response is what we had further up in the notebook, that was the assistant answer. And we're going to specify the data in this prompt, what was the customer message, what was the context, that is, what was the product and category information that was provided, and then what was the output of the LLM. And then, this is a rubric. So, we want the LLM to, "Compare the factual content of the submitted answer with the context. Ignore differences in style, grammar, or punctuation. And then, we wanted to check a few things, like, "Is the assistant response based only on the context provided? Does the answer include information that is not provided in the context? Is there any disagreement between the response and the context?" So, this is called a rubric, and this specifies what we think the answer should get right for us to consider it a good answer. Then, finally, we wanted to print out yes or no, and so on. And now, if we were to run this evaluation. This is what you get. So it says, "the assistant response is based only on the context provided.". It does not, in this case, seem to make up new information. There isn't disagreements. User asked two questions. Answered question one and answered question two. So answered both questions. So we would look at this output and maybe conclude that this is a pretty good response. And one note, here I'm using the ChatGPT 3.5 Turbo model for this evaluation. For a more robust evaluation, it might be worth considering using GPT-4 because even if you deploy 3.5 Turbo in production and generate a lot of text, if your evaluation is a more sporadic exercise, then it may be prudent to pay for the somewhat more expensive GPT-4 API call to get a more rigorous evaluation of the output. One design pattern that I hope you can take away from this is that when you can specify a rubric, meaning a list of criteria by which to evaluate an LLM output, then you can actually use another API call to evaluate your first LLM output. There's one other design pattern that could be useful for some applications, which is if you can specify an ideal response. So here, I'm going to specify a test example where the customer message is, "tell me about the smartx pro phone", and so on. And here's an ideal answer. So this is if you have an expert human customer service representative write a really good answer. The expert says, this would be a great answer., "Of course! The SmartX ProPhone is a.". It goes on to give a lot of helpful information. Now, it is unreasonable to expect any LLM to generate this exact answer word for word. And in classical natural language processing techniques, there are some traditional metrics for measuring if the LLM output is similar to this expert human written outputs. For example, there's something called the BLEU score, BLEU, that you can search online to read more about. They can measure how similar one piece of text is from another. But it turns out there's an even better way, which is you can use a prompt, which I'm going to specify here, to ask an LLM to compare how well the automatically generated customer service agent output corresponds to the ideal expert response that was written by a human that I just showed up above. Here's the prompt we can use, which is. We're going to use an LLM and tell it to be an assistant that evaluates how well the customer service agent answers a user question by comparing the response, that was the automatically generated one, to the ideal (expert) human written response. So we're going to give it the data, which is what was the customer request, what is the expert written ideal response, and then what did our LLM actually output. And this rubric comes from the OpenAI open source evals framework, which is a fantastic framework with many evaluation methods contributed both by OpenAI developers and by the broader open source community. In fact, if you want you could contribute an eval to that framework yourself to help others evaluate their Large Language Model outputs. So in this rubric, we tell the LLM to, "Compare the factual content of the submitted answer with the expert answer. Ignore any differences in style, grammar, or punctuation.". And feel free to pause the video and read through this in detail, but the key is we ask it to carry the comparison and output a score from A to E, depending on whether the "submitted answer is a subset of the expert answer and is fully consistent", versus the "submitted answer is a superset of the expert answer and is fully consistent with it". This might mean it hallucinated or made up some additional facts. "Submitted answer contains all the details as the expert answer.", whether there's disagreement or whether "the answers differ, but these differences don't matter from the perspective of factuality". And the LLM will pick whichever of these seems to be the most appropriate description. So here's the assistant answer that we had just now. I think it's a pretty good answer, but now let's see what the things when it compares the assistant answer to test set ID. Oh, looks like it got an A. And so it thinks "The submitted answer is a subset of the expert answer and is fully consistent with it", and that sounds right to me. This assistant answer is much shorter than the long expert answer up top, but it does hopefully is consistent. Once again, I'm using GPT-3.5 Turbo in this example, but to get a more rigorous evaluation, it might make sense to use GPT-4 in your own application. Now, let's try something totally different. I'm going to have a very different assistant answer, "life is like a box of chocolates", quote from a movie called "Forrest Gump". And if we were to evaluate that it outputs D and it concludes that, "there is a disagreement between the submitted answer", life is like a box of chocolate and the expert answer. So it correctly assesses this to be a pretty terrible answer. And so that's it. I hope you take away from this video two design patterns. First is, even without an expert provided ideal answer, if you can write a rubric, you can use one LLM to evaluate another LLM's output. And second, if you can provide an expert provided ideal answer, then that can help your LLM better compare if, and if a specific assistant output is similar to the expert provided ideal answer. I hope that helps you to evaluate your LLM systems outputs. So that both during development as well as when the system is running and you're getting responses, you can continue to monitor its performance and also have these tools to continuously evaluate and keep on improving the performance of your system.