how to build an application using an LLM from evaluating the inputs to processing the inputs to then doing final output checking before you show the output to the user. After you've built such a system, how do you know how it's working? And maybe even as you deploy it and let users use it, how can you track how it's doing and find any shortcomings and continue to improve the quality of the answers of your system? In this video, I'd like to share with you some best practices for evaluating the outputs of an LLM and I want to share with you specifically what it feels like to build one of these systems. One key distinction between what you hear me talk about in this video and what you may have seen in more traditional machine learning supervised learning applications is because you can build such an application so quickly, the methods for evaluating it, it tends not to start off with a test set. Instead, you often end up gradually building up a set of test examples. Let me share with you what I mean by that. You may remember this diagram from the second video about how prompt-based development speeds up the core parts of model development from maybe months to just minutes or hours or at most a very small number of days. In the traditional supervised learning approach, if you needed to collect say, 10,000 labeled examples anyway, then the incremental cost of collecting another 1,000 test examples isn't that bad. So, in the traditional supervised learning setting, it was not unusual to collect a training set, collect a development set or holdout cross-validation set in the test set and then tap those at hand throughout this development process. But if you're able to specify a prompt in just minutes and get something working in hours, then it would seem like a huge pain if you had to pause for a long time to collect 1,000 test examples because you cannot get this working with zero training examples. So, when building an application using an LLM, this is what it often feels like. First, you would tune the prompts on just a small handful of examples, maybe one to three to five examples and try to get a prompt that works on them. And then as you have the system undergo additional testing, you occasionally run into a few examples that are tricky. The prompt doesn't work on them or the algorithm doesn't work on them. And in that case, you can take these additional one or two or three or five examples and add them to the set that you're testing on to just add additional tricky examples opportunistically. Eventually, you have enough of these examples you've added to your slowly growing development set that it becomes a bit inconvenient to manually run every example through the prompt every time you change the prompt. And then you start to develop metrics to measure performance on this small set of examples, such as maybe average accuracy. And one interesting aspect of this process is, if you decide at any moment in time, your system is working well enough, you can stop right there and not go on to the next bullet. And in fact, there are many deploy applications that stops at maybe the first or the second bullet and are running actually just fine. Now, if your hand-built development set that you're evaluating the model on isn't giving you sufficient confidence yet in the performance of your system, then that's when you may go to the next step of collecting a randomly sampled set of examples to tune the model to. And this would continue to be a development set or a hold-out cross-validation set, because it'd be quite common to continue to tune your prompt to this. And only if you need even higher fidelity estimate of the performance of your system, then might you collect and use a hold-out test sets that you don't even look at yourself when you're tuning the model. And so step four tends to be more important if, say, your system is getting the right answer 91% of the time, and you want to tune it to get it to give the right answer 92% or 93% of the time, then you do need a larger set of examples to measure those differences between 91% and 93% performance. And then only if you really need an unbiased, fair estimate of how was the system doing, then do you need to go beyond the development set to also collect a hold-out test set. One important caveat, I've seen a lot applications of large language models where there isn't meaningful meaning risk of harm if it gives not quite the right answer. But obviously, for any high-stakes applications, if there's a risk of bias or an inappropriate output causing harm to someone, then the responsibility to collect a test set to rigorously evaluate the performance of your system to make sure it's doing the right thing before you use it, that becomes much more important. But if, for example, if you are using it to summarize articles just for yourself to read and no one else, then maybe the risk of harm is more modest, and you can stop early in this process without going to the expense of bullets four and five and collecting larger data sets on which to evaluate your algorithm. So in this example, let me start with the usual helper functions. Use the utils function to get a list of products and categories. So, in the computers and laptops category, there's a list of computers and laptops, in the smartphones and accessories category, here's a list of smartphones and accessories, and so on for other categories. Now, let's say, the task we're going to address is, given a user input, such as, "what TV can I buy if I'm on a budget?", to retrieve the relevant categories and products, so that we have the right info to answer the user's query. So here's a prompt, feel free to pause the video and read through this in detail if you wish. But the prompt specifies a set of instructions, and it actually gives the language model one example of a good output. This is sometimes called a few-shot or technically one-shot prompting, because we're actually using a user message and a system message to give it one example of a good output. If someone says, "I want the most expensive computer.". Yeah, let's just return all the computers, because we don't have pricing information. Now, let's use this prompt on the customer message, "Which TV can I buy if I'm on a budget?". And so we're passing in to this both the prompt, customer message zero, as well as the products and category. This is the information that we have retrieved up top using the utils function. And here it lists out the relevant information to this query, which is under the category, televisions and whole theater systems. This is a list of TVs and whole theater systems that seem relevant. To see how well the prompt is doing, you may evaluate it on a second prompt. The customer says, "I need a charger for my smartphone.". It looks like it's correctly retrieving this data. Category, smartphones, accessories, and it lists the relevant products. And here's another one. So, "What computers do you have?". And hopefully you'll retrieve a list of the computers. So, here I have three prompts, and if you are developing this prompt for the first time, it would be quite reasonable to to have one or two or three examples like this, and to keep on tuning the prompt until it gives appropriate outputs, until the prompt is retrieving the relevant products and categories to the customer request for all of your prompts, all three of them in this example. And if the prompt had been missing some products or something, then what we would do is probably go back to edit the prompt a few times until it gets it right on all three of these prompts. After you've gotten the system to this point, you might then start running the system in testing. Maybe send it to internal test users or try using it yourself, and just run it for a while to see what happens. And sometimes you will run across a prompt that it fails on. So here's an example of a prompt, "tell me about the smartx pro phone and the fotosnap camera. Also, what TVs do you have?". So when I run it on this prompt, it looks like it's outputting the right data, but it also outputs a bunch of text here, this extra junk. It makes it harder to parse this into a Python list of dictionaries. So we don't like that it's outputting this extra junk. So when you run across one example that the system fails on, then common practice is to just note down that this is a somewhat tricky example, so let's add this to our set of examples that we're going to test the system on systematically. And if you keep on running the system for a while longer, maybe it works on those examples. We did tune the prompt to three examples, so maybe it will work on many examples, but just by chance you might run across another example where it generates an error. So this custom message for also causes the system to output a bunch of junk text at the end that we don't want. Trying to be helpful to give all this extra text, we actually don't want this. And so at this point, you may have run this prompt, maybe on hundreds of examples, maybe you have test users, but you would just take the examples, the tricky ones is doing poorly on, and now I have this set of five examples, index from 0 to 4, have this set of five examples that you use to further fine-tune the prompts. And in both of these examples, the LLM had output a bunch of extra junk text at the end that we don't want. And after a little bit of trial and error, you might decide to modify the prompts as follows. So here's a new prompt, this is called prompt v2. But what we did here was we added to the prompt, "Do not output any additional text that's not in JSON format.", just to emphasize, please don't output this JSON stuff. And added a second example using the user and assistant message for few-shot prompting where the user asked for the cheapest computer. And in both of the few-shot examples, we're demonstrating to the system a response where it gives only JSON outputs. So here's the extra thing that we just added to the prompt, "Do not output any additional text that's not in JSON formats.", and we use "few\_shot\_user\_1", "few\_shot\_assistant\_1", and "few\_shot\_user\_2", "few\_shot\_assistant\_2" to give it two of these few shot prompts. So let me hit Shift-Enter to find that prompt. And you were to go back and manually rerun this prompt on all five of the examples of user inputs, including this one that previously had given a broken output, you'll find that it now gives a correct output. And if you were to go back and rerun this new prompt, this is prompt version v2, on that customer message example that had results in the broken output with extra junk after the JSON output, then this will generate a better output. And I'm not going to do it here, but I encourage you to pause the video and rerun it yourself on customer message 4 as well on this prompt v2, see if it also generates the correct output. And hopefully it will, I think it should. And of course, when you modify the prompts, it's also useful to do a bit of regression testing to make sure that when fixing the incorrect outputs on prompts 3 and 4, it didn't break the output on prompt 0 either. Now, you can kind of tell that if I had to copy-paste 5 prompts, customers such as 0, 1, 2, 3, and 4, into my Jupyter notebook and run them and then manually look at them to see if they output in the right categories and products. You can kind of do it. I can look at this and go, "Yep, category, TV and home theater systems, products. Yep, looks like you got all of them.". But it's actually a little bit painful to do this manually, to manually inspect or to look at this output to make sure with your eyes that this is exactly the right output. So when the development set that you're tuning to becomes more than just a small handful of examples, it then becomes useful to start to automate the testing process. So here is a set of 10 examples where I'm specifying 10 customer messages. So here's a customer message, "Which TV can I buy if I'm on a budget?" as well as what's the ideal answer. Think of this as the right answer in the test set, or really, I should say development set, because we're actually tuning to this. And so we've collected here 10 examples indexed from 0 through 9, where the last one is if the user says, "I would like hot tub time machine.". We have no relevant products to that, really sorry, so the ideal answer is the empty set. And now, if you want to evaluate automatically, what the prompt is doing on any of these 10 examples, here is a function to do so. It's kind of a long function. Feel free to pause the video and read through it if you wish. But let me just demonstrate what it is actually doing. So let me print out the customer message, for customer message 0. So the customer message is,"Which TV can I buy if I'm on a budget?". And let's also print out the ideal answer. So the ideal answer is here are all the TVs that we want the prompt to retrieve. And let me now call the prompt. This is prompt V2 on this customer message with that user products and category information. Let's print it out and then we'll call the eval. We'll call the eval response of ideal function to see how well the response matches the ideal answer. And in this case, it did output the category that we wanted, and it did output the entire list of products. And so this gives it a score of 1.0. Just to show you one more example, it turns out that I know it gets it wrong on example 7. So if I change this from 0 to 7 and run it, this is what it gets. Oh, let me update this to 7 as well. So under this customer message, this is the ideal answer where it should output under gaming consoles and accessories. So list of gaming consoles and accessories. But whereas the response here has three outputs, it actually should have had 1, 2, 3, 4, 5 outputs. And so it's missing some of the products. So what I would do if I'm tuning the prompt now is I would then use a fold to loop over all 10 of the development set examples, where we repeatedly pull out the customer message, get the ideal answer, the right answer, call the arm to get a response, evaluate it, and then, you know, accumulate it in average. And let me just run this. So this will take a while to run, but when it's done running, this is the result. We're running through the 10 examples. Looks like example 7 was wrong. And so the fraction correct of 10 was 90% correct. And so, if you were to tune the prompts, you can rerun this to see if the percent correct goes up or down. What you just saw in this notebook is going through steps 1, 2, and 3 of this bulleted list, and this already gives a pretty good development set of 10 examples with which to tune and validate the prompts is working. If you needed an additional level of rigor, then you now have the software needed to collect a randomly sampled set of maybe 100 examples with their ideal outputs, and maybe even go beyond that to the rigor of a holdout test set that you don't even look at while you're tuning the prompt. But for a lot of applications, stopping at bullet 3, but there are also certainly applications where you could do what you just saw me do in this Jupyter notebook, and it gets a pretty performance system quite quickly. With again, the important caveat that if you're working on a safety critical application or an application where there's non-trivial risk of harm, then of course, it would be the responsible thing to do to actually get a much larger test set to really verify the performance before you use it anywhere. And so that's it. I find that the workflow of building applications using prompts is very different than a workflow of building applications using supervised learning, and the pace of iteration feels much faster. And if you have not yet done it before, you might be surprised at how well an evaluation method built on just a few hand-curated tricky examples. You think with 10 examples, and this is not statistically valid for almost anything. But you might be surprised when you actually use this procedure, how effective adding a handful, just a handful of tricky examples into development sets might be in terms of helping you and your team get to an effective set of prompts and effective system. In this video, the outputs could be evaluated quantitatively, as in there was a desired output and you could tell if it gave this desired output or not. So the next video, let's take a look at how you can evaluate output in that setting where what is the right answer is a bit more ambiguous.