Conversational AI

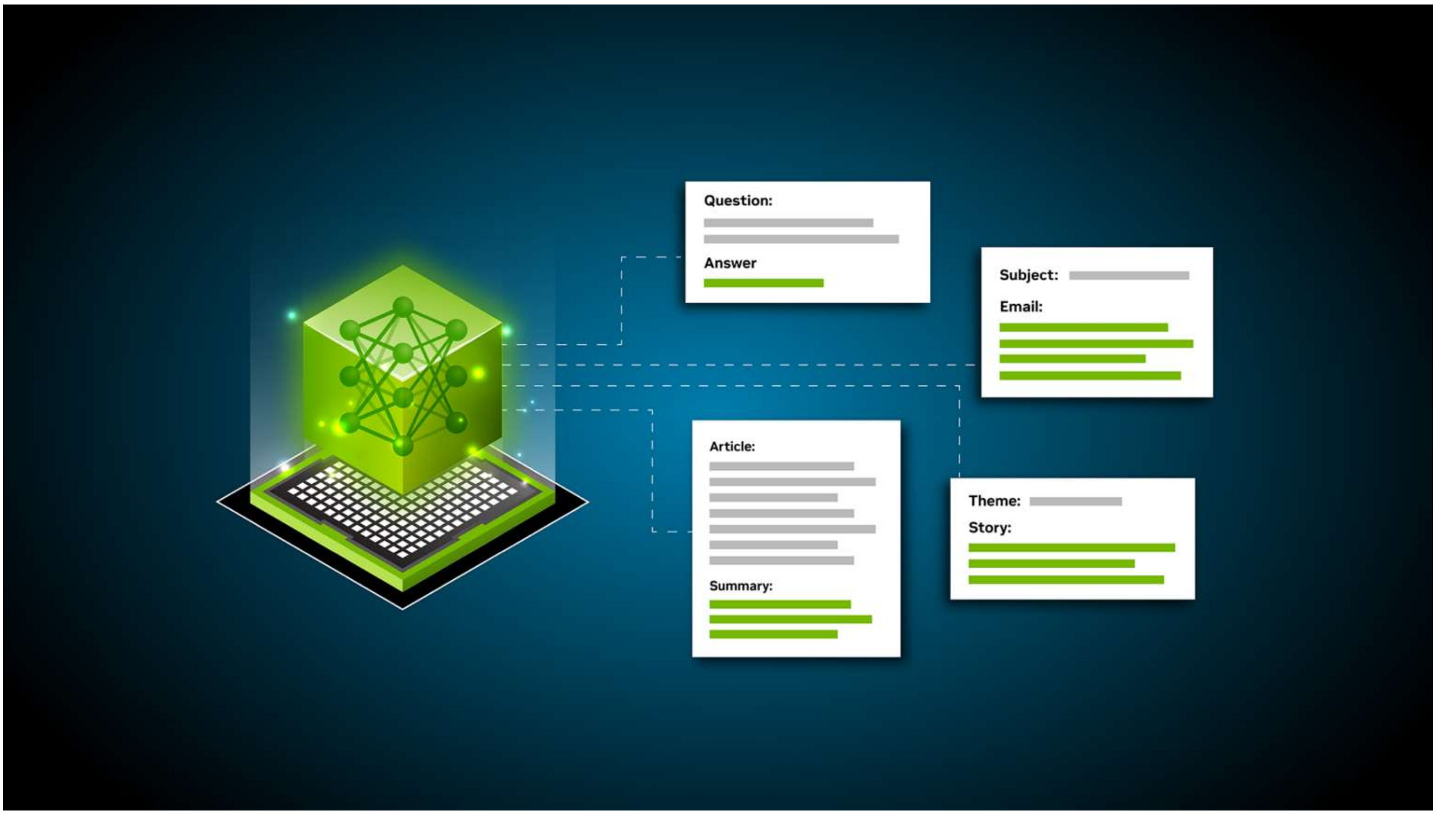
English

How to Get Better Outputs from Your Large Language Model

Jun 14, 2023

By Annie Surla

+19 Like Discuss (0)



Large language models (LLMs) have generated excitement worldwide due to their ability to understand and process human language at a scale that is unprecedented. It has transformed the way that we interact with technology.

Having been trained on a vast corpus of text, LLMs can manipulate and generate text for a wide variety of applications without much instruction or training. However, the quality of this generated output is heavily dependent on the instruction that you give the model, which is referred to as a prompt. What does this mean for you? Interacting with the models today is the art of designing a prompt rather than engineering the model architecture or training data.

Dealing with LLMs can come at a cost given the expertise and resources required to build and train your models. NVIDIA NeMo offers pretrained language models that can be flexibly adapted to solve almost any language processing task while we can focus entirely on the art of getting the best outputs from the available LLMs.

In this post, I discuss a few ways of getting around with LLMs, so that you can make the best out of them. For more information about getting started with LLMs, see An Introduction to Large Language Models: Prompt Engineering and P-Tuning.

Mechanism behind prompting

Before I get into the strategies to generate optimal outputs, step back and understand what happens when you prompt a model. The prompt is broken down into smaller chunks called tokens and is sent as input to the LLM, which then generates the next possible tokens based on the prompt.

Tokenization

LLMs interpret the textual data as tokens. Tokens are words or chunks of characters. For example, the word “sandwich” would be broken down into the tokens “sand” and “wich”, whereas common words like “time” and “like” would be a single token.

NeMo uses byte-pair encoding to create these tokens. The prompt is broken down into a list of tokens that are taken as input by the LLM.

Behind the curtains, the model first generates *logits* for each possible output token. Logits are a function that represents probability values from 0 to 1, and negative infinity to infinity. Those logits then are passed to a softmax function to generate probabilities for each possible output, giving you a probability distribution over the vocabulary. Here is the softmax equation for calculating the actual probability of a token:

$$P(token_k|token_context) = \frac{exp(logit_k)}{\sigma_j exp(logit_j)}$$

In the formula, $P(token_k|token_context)$ is probability of $token_k$ given the context from previous tokens ($token_1$ to $token_k - 1$ and $logit_k$ is the output of the neural network

The model would then select the most likely word and add it to the prompt sequence.

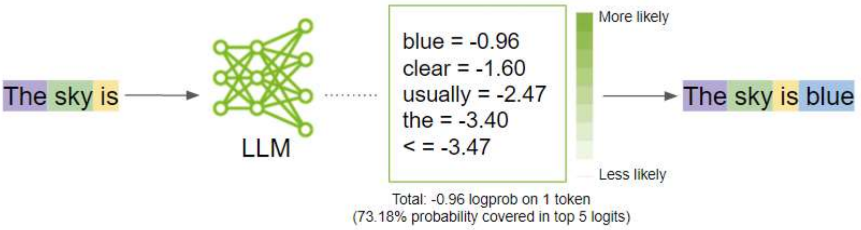


Figure 1. General working flow of an LLM predicting the next word

While the model decides what is the most probable output, you can influence those probabilities by turning some model parameter knobs up and down. In the next section, I discuss what those parameters are and how to tune them to get the best outputs.

Tweak the parameters

To unlock the full potential of LLMs, explore the art of refining the outputs. Here are the key parameter categories to consider tweaking:

- Let the model know when to stop
- Predictability vs. creativity
- Reducing repetition

Play around with these parameters and figure out the best combinations that work for your specific use case. In many cases, experimenting with the temperature parameter can get what you might need. However, if you have something specific and want more granular control over the output, start experimenting with the other ones.

Let the model know when to stop

There are parameters that can guide the model to decide when to stop generating any further text:

- Number of tokens
- Stop words

Number of tokens

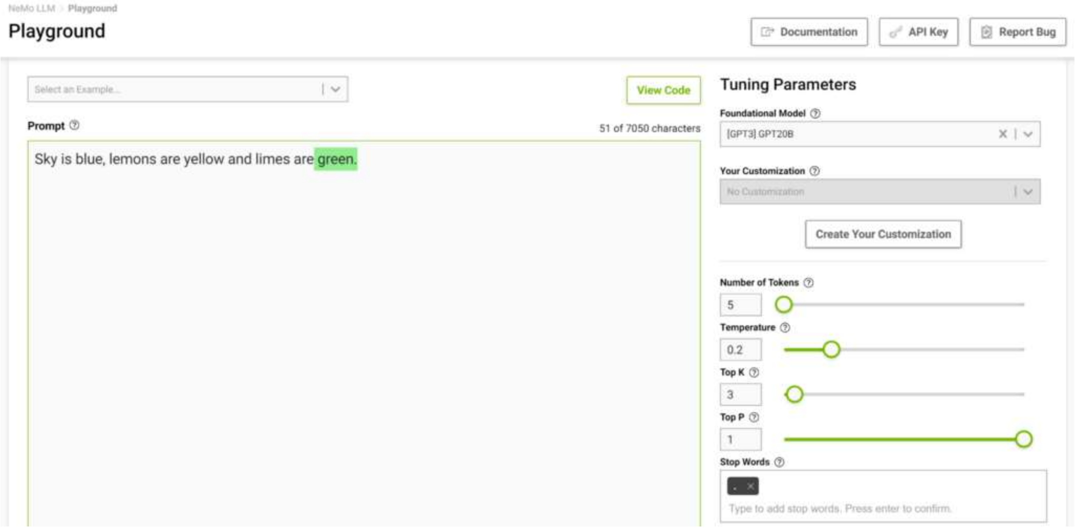
Earlier, I mentioned that the LLM is focused on generating the next token given the sequence of tokens. The model does this in a loop appending the predicted token to the input sequence. You wouldn't want the LLM to go on and on.

While there is a limit to the number of tokens ranging from 2048 to 4096 that NeMo models can accept for now, I don't recommend hitting these limits as the model may generate off responses.

Stop words

Stop words are a set of character sequences that tells the model to stop generating any additional text, even if the output length has not reached the specified token limit.

This is another way to control the length of the output. For example, if the model is prompted to complete the following sentence "Sky is blue, lemons are yellow and limes are" and you specify the stop word as just ".", the model stops after finishing just this sentence, even if the token limit is higher than the generated sequence (Figure 2).



Playground

Prompt

51 of 7050 characters

Sky is blue, lemons are yellow and limes are green.

Tuning Parameters

Foundational Model: [GPT3] GPT20B

Your Customization: No Customization

Number of Tokens: 5

Temperature: 0.2

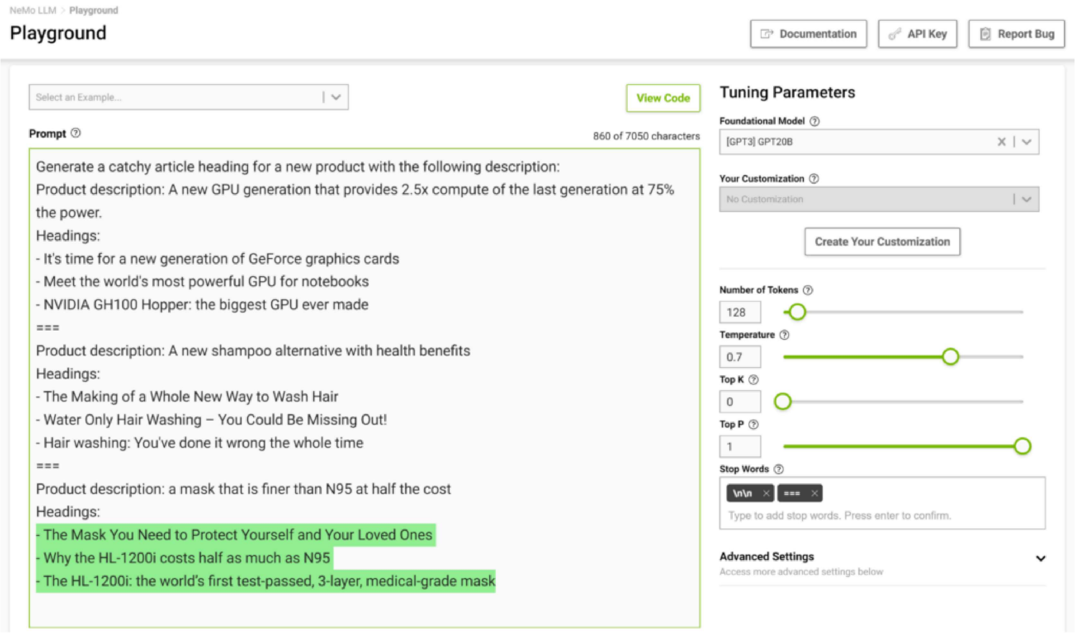
Top K: 3

Top P: 1

Stop Words: .

Type to add stop words. Press enter to confirm.

It is especially useful to design a stopping template in a few-shot setting so the model can learn to stop appropriately upon completing an intended task. Figure 3 shows separating examples with the string “===” and passing that as the stop word.



Prompt

860 of 7050 characters

Generate a catchy article heading for a new product with the following description:
Product description: A new GPU generation that provides 2.5x compute of the last generation at 75% the power.
Headings:
- It's time for a new generation of GeForce graphics cards
- Meet the world's most powerful GPU for notebooks
- NVIDIA GH100 Hopper: the biggest GPU ever made
===
Product description: A new shampoo alternative with health benefits
Headings:
- The Making of a Whole New Way to Wash Hair
- Water Only Hair Washing – You Could Be Missing Out!
- Hair washing: You've done it wrong the whole time
===
Product description: a mask that is finer than N95 at half the cost
Headings:
- The Mask You Need to Protect Yourself and Your Loved Ones
- Why the HL-1200i costs half as much as N95
- The HL-1200i: the world's first test-passed, 3-layer, medical-grade mask

Foundational Model: [GPT3] GPT20B
Your Customization: No Customization
Number of Tokens: 128
Temperature: 0.7
Top K: 0
Top P: 1
Stop Words: \n\n ===
Type to add stop words. Press enter to confirm.

Figure 3. Using stop words with a few-shot prompt

Predictability vs. creativity

Given a prompt, it is possible to generate different outputs based on the parameters you set. Based on the application of the LLM, you can choose to increase or decrease the creative ability of the model. Here are a few of these parameters that can help you do so:

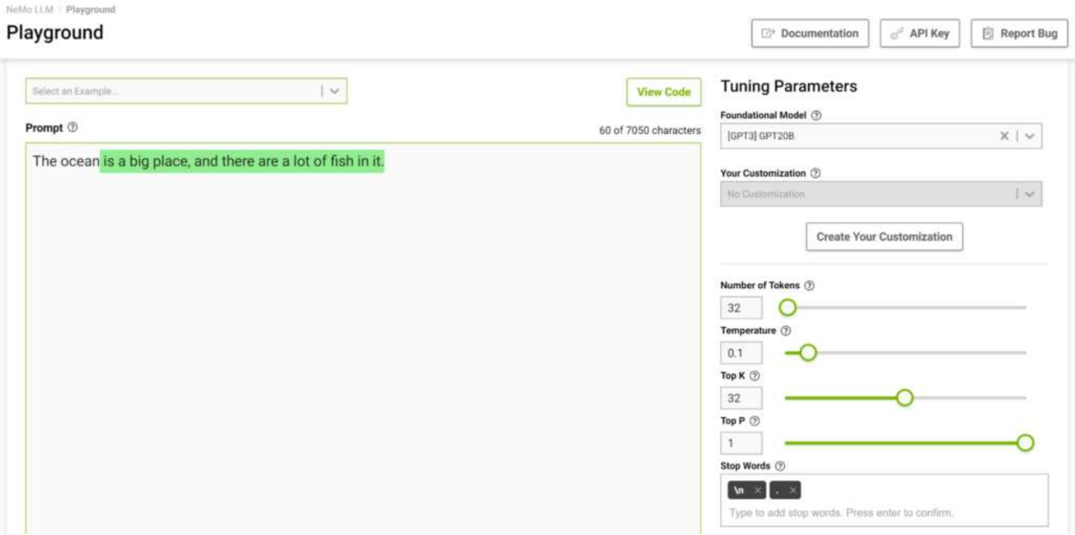
- Temperature
- Top-k and Top-p
- Beam search width

Temperature

This parameter controls the creative ability of your model. As discussed earlier, while generating the next token in the input sequence, the model comes up with a probability distribution. The temperature parameter adjusts the shape of this distribution, leading to more diversity in the generated text.

At a lower temperature, the model is more conservative and is limited to choosing tokens with higher probabilities. As you increase the temperature, that limit gets lenient, allowing the model to choose lesser probable words, resulting in more unpredictable and creative text.

Figure 4 shows tasking the model to complete the sentence starting with “The ocean” where you set the temperature to 0.1.



Prompt

60 of 7050 characters

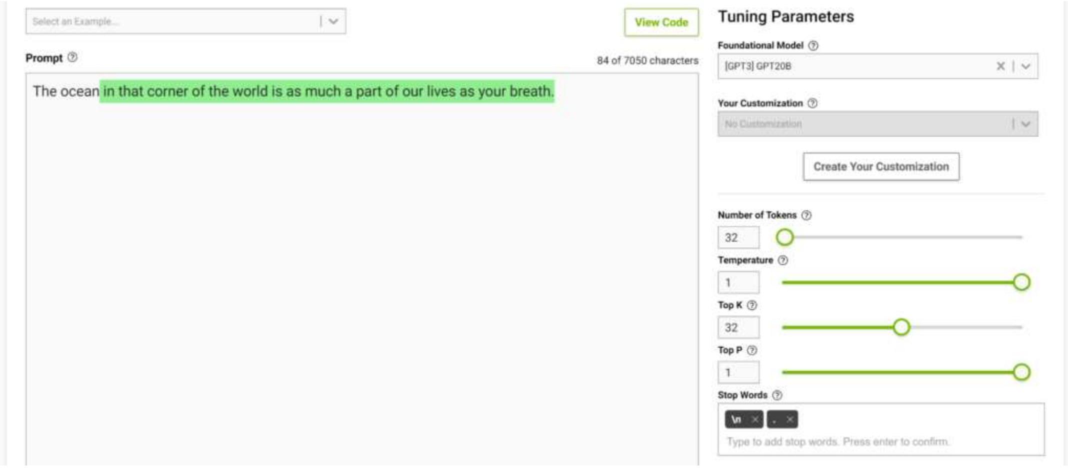
The ocean is a big place, and there are a lot of fish in it.

Foundational Model: [GPT3] GPT20B
Your Customization: No Customization
Number of Tokens: 32
Temperature: 0.1
Top K: 32
Top P: 1
Stop Words: \n .
Type to add stop words. Press enter to confirm.

Figure 4. Sentence generation at temperature = 0.1 using the NeMo service playground

When you think of completing such a phrase, you would probably think of phrases like “...is huge” or “...is blue”. The output is pretty much a simple fact that the ocean is big with lots of fish.

Now, try this again with the temperature setting at 1 (Figure 5).



Select an Example...

View Code

Prompt

84 of 7050 characters

The ocean in that corner of the world is as much a part of our lives as your breath.

Tuning Parameters

Foundational Model: [GPT3] GPT20B

Your Customization: No Customization

Create Your Customization

Number of Tokens: 32

Temperature: 1

Top K: 32

Top P: 1

Stop Words: \n , .

Type to add stop words. Press enter to confirm.

Figure 5. Sentence generation at temperature = 1 using the NeMo service playground

The model started to give you analogies that you commonly don't think of. Higher temperatures are suitable for tasks that require creative writing like poems and stories. But beware that the generated text can sometimes also turn out nonsensical. Lower temperatures are suitable for more definitive tasks like question-answering or summarization.

I recommend experimenting with different temperature values to find the best temperature for your use case. The range `[0.5, 0.8]` should be a good starting point in the NeMo service playground.

Top-k and Top-p

These two parameters also control the randomness of selecting the next token. Top-k tells the model that it has to keep the top *k* highest probability tokens, from which the next token is selected at random. Lower values reduce randomness as you are clipping off less likely tokens generating predictable text. If *k* is set to 0, Top-k is not used. When set to 1, it is always going to select the most probable token next.

There can be cases when the probability distribution for the possible token could be broad where there are so many tokens that are likely. There can also be cases where the distribution is narrow where there are only a few tokens that are more likely.

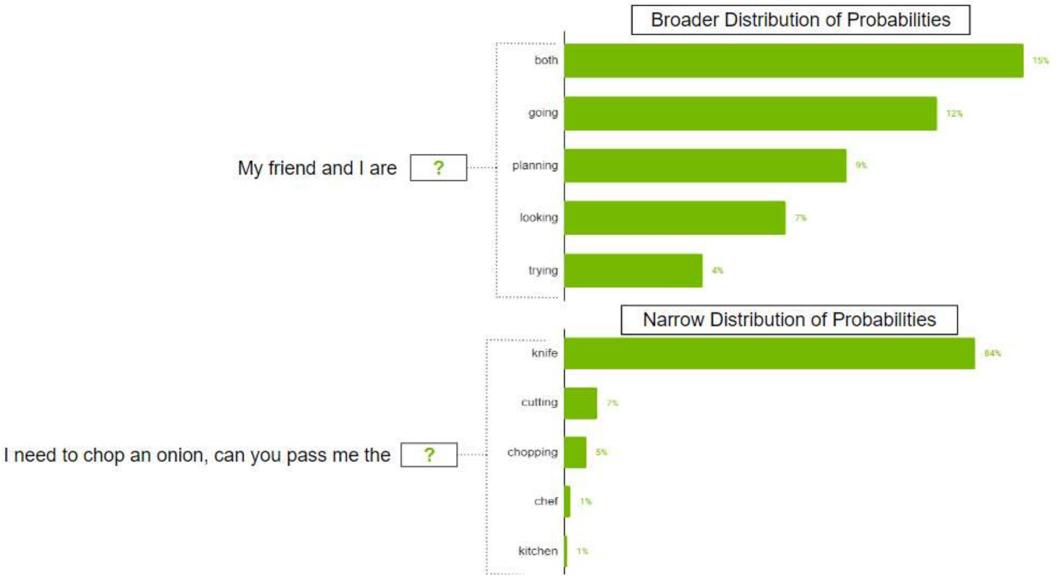


Figure 6. Probability distributions types for the generated LLM output

You probably don't want to strictly restrict the model to just select the top *k* tokens in the broader distribution scenario. To address this, parameter top-p can be used where the model picks at random from the highest probability tokens whose probabilities sum to or exceed the top-p value. If top-p is set to 0.9, one of the following scenarios may occur:

- In the broader distribution example, it may consider the top 50 tokens whose sum of probabilities that are equal to or exceed 0.9.
- In the narrow distribution scenario, you may exceed 0.9 with just the top two tokens. This way, you are avoiding picking from the random tokens, while still preserving the variety.

Beam search width

This is another helpful parameter that can control the diversity of outputs. Beam search is an algorithm commonly used in many NLP and speech recognition models as a final decision-making step to choose the best output given the possible options. Beam search width is a parameter that determines the number of candidates that the algorithm should consider at each step in the search.

Higher values increase the chance of finding a good output, but that also comes at the cost of more computation.

Reducing repetition

Sometimes, repeated text might not be desirable in the output. If this is the case, use the repetition penalty parameter to help reduce repetition.

This parameter can help penalize tokens based on how frequently they occur in the text, including the input prompt. A token that has already appeared five times is penalized more heavily than a token that has appeared only one time. A value of 1 means that there is no penalty and values larger than 1 discourage repeated tokens.

Few-shot strategies for effective prompt design

Prompt design is crucial for generating relevant and coherent outputs from the LLMs. Having strategies for effective prompt design can help create prompts that are relevant while avoiding common pitfalls like bias, ambiguity, or lack of specificity. In this section, I share some key strategies for effective prompt design.

Prompt with constraints

Constraining the model's behavior through careful prompt design can be quite useful. You know that language models at their core are trying to predict the next word in a sequence. A task description that makes perfect sense to a human might not be understood by the language model. This is why few-shot learning often works well: as you demonstrate a pattern to the model, it does a good job adhering to it.

Consider the following prompt, "Translate English to French: Today is a beautiful day."

With this prompt, the model would likely try to continue the sentence or add more sentences rather than performing the translation. Changing the prompt to, "Translate this English sentence to French: Today is a beautiful day." increases the likelihood of the model understanding this task as a translation task and generates a more reliable output.

Characters matter!

As you saw in the previous translation example, small changes can lead to varied outputs. Another thing to note is tokens are often generated with a leading space, so characters like space and next line can also affect your outputs. If a prompt is not working out, try changing the way that you structured it.

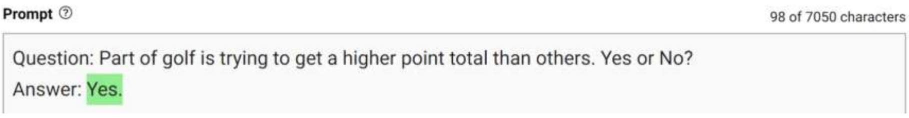
Consider certain phrases

Often when you want your model to answer your prompts logically and arrive at accurate conclusions or simply to make the model achieve a certain outcome, you can consider using the following phrases:

- **Let us think this through step by step:** This encourages the model to approach a problem logically and arrive at accurate answers. This style of prompting is also known as *chain-of-thought promoting* (CoT).
- **In the style of <notable person>:** This matches the style of the notable person's writing. For example, to generate text like Shakespeare or Edgar Allen Poe, add this to the prompt and the generation will closely match their writing style.
- **As a <profession/role>**: This helps the model understand the context of the question better. With a better understanding, the model often gives better answers.

Prompt with generated knowledge

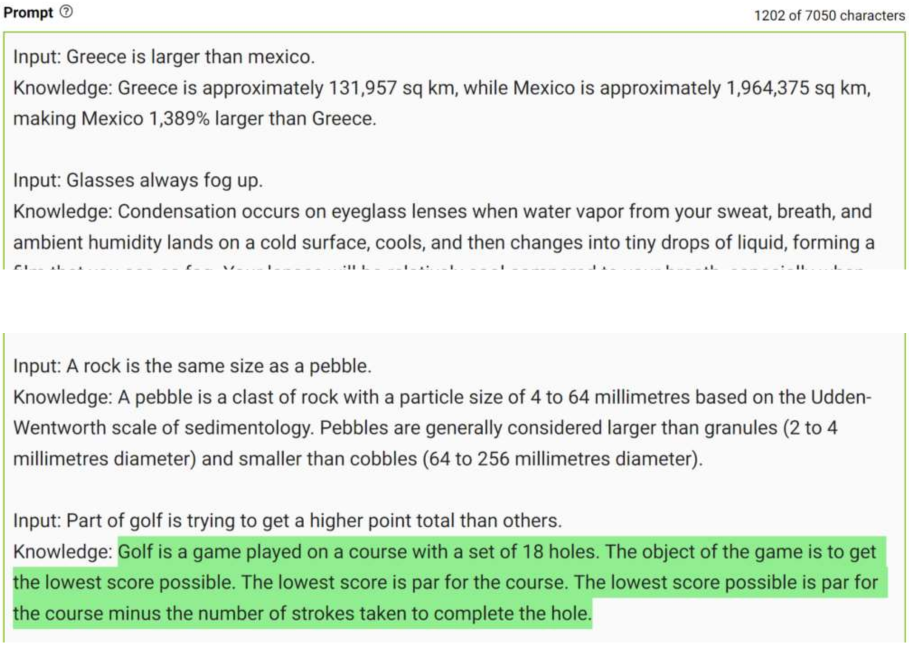
To obtain more accurate answers, you can prompt the LLM to generate potentially useful knowledge about a given question before generating a final answer (Figure 7).



Question: Part of golf is trying to get a higher point total than others. Yes or No?
Answer: Yes.

Figure 7. Question-answer prompt where the answer is incorrect

This type of mistake shows that LLMs sometimes require more knowledge to answer a question. The next examples show generating a few facts about golf scoring in a few-shot setting.



Input: Greece is larger than mexico.
Knowledge: Greece is approximately 131,957 sq km, while Mexico is approximately 1,964,375 sq km, making Mexico 1,389% larger than Greece.

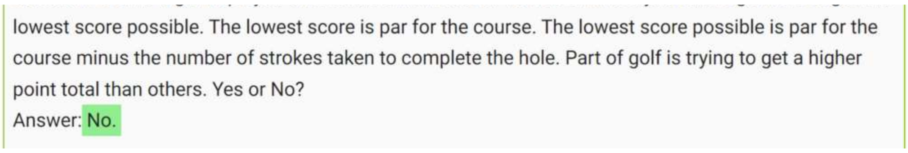
Input: Glasses always fog up.
Knowledge: Condensation occurs on eyeglass lenses when water vapor from your sweat, breath, and ambient humidity lands on a cold surface, cools, and then changes into tiny drops of liquid, forming a film that you see as fog. Your lenses will be relatively cool compared to your breath, especially when

Input: A rock is the same size as a pebble.
Knowledge: A pebble is a clast of rock with a particle size of 4 to 64 millimetres based on the Udden-Wentworth scale of sedimentology. Pebbles are generally considered larger than granules (2 to 4 millimetres diameter) and smaller than cobbles (64 to 256 millimetres diameter).

Input: Part of golf is trying to get a higher point total than others.
Knowledge: Golf is a game played on a course with a set of 18 holes. The object of the game is to get the lowest score possible. The lowest score is par for the course. The lowest score possible is par for the course minus the number of strokes taken to complete the hole.

Figure 8. Generating knowledge around the prompt

Integrate this knowledge into the prompt and ask the question again.



lowest score possible. The lowest score is par for the course. The lowest score possible is par for the course minus the number of strokes taken to complete the hole. Part of golf is trying to get a higher point total than others. Yes or No?
Answer: No.

Figure 9. Question-answer task with the correct answer post-knowledge integration

The model confidently answered "No" to the same question. This is a simple demonstration of this kind of prompting. However, there are some more details to consider before arriving at the final answer. For more information, see Generated Knowledge Prompting for Commonsense Reasoning.

In practice, you generate multiple answers and select the most frequently occurring answer as the final one.

Experiment with it!

The best way to write prompts that fit your use case is to experiment and play around. It is a learning experience to engineer a prompt that can get you the right outputs, whether it's how you write it or the way you set model parameters.

The NeMo service playground can help you test out your prompts and craft your use case. If you are interested in accessing the playground, see NVIDIA NeMo Service.

Conclusion

In this post, I shared ways to generate better outputs from LLMs. I discussed how model parameters could be tweaked to get desired outputs and some strategies to engineer your prompts.

Stay up to date on LLM technologies, learnings, and breakthroughs by signing up for the LLM newsletter.

Related resources

- **GTC session:** Optimizing Large Language Models: An Experimental Approach to Pruning and Fine-Tuning LLama2 7B
- **GTC session:** Considerations for Choosing LLM Serving Technologies (Presented by Run:ai)
- **GTC session:** Making Large Language Models and Retrieval-Augmented Generation Work With Ease (Presented by Softserve, Inc.)
- **Webinar:** Implementing Large Language Models
- **Webinar:** What AI Teams Need to Know About Generative AI
- **Webinar:** Harness the Power of Cloud-Ready AI Inference Solutions and Experience a Step-By-Step Demo of LLM Inference Deployment in the Cloud

Discuss (0) +19 Like

Tags

Conversational AI | Generative AI / LLMs | NeMo Framework | Beginner Technical | Featured | LLMs | NLP | Speech & Audio Processing | Text Generation | Tutorial

About the Authors

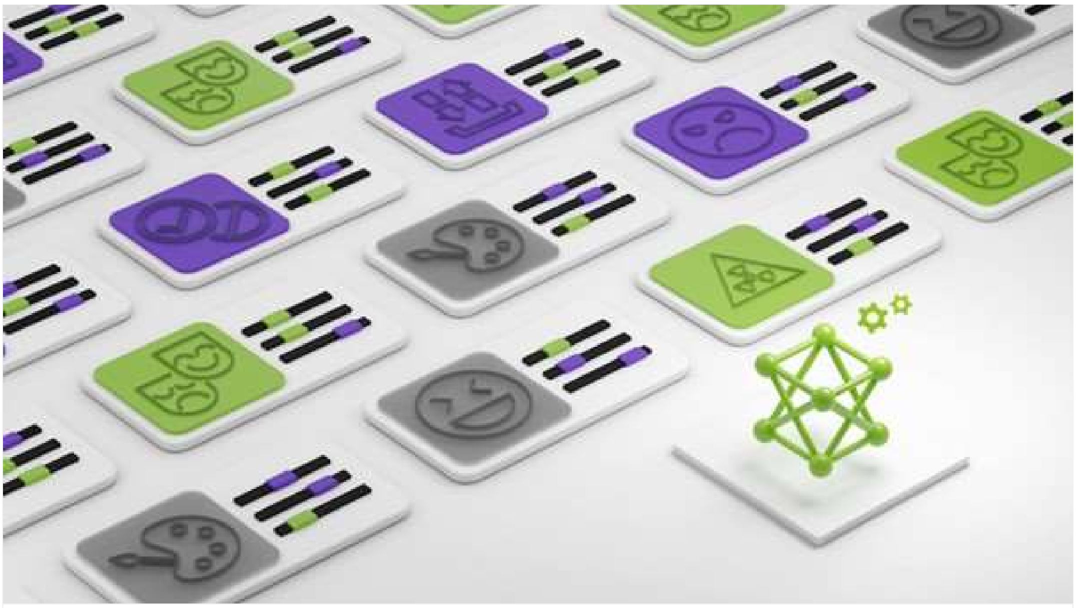
About Annie Surla

Annie Surla is a Developer Advocate Engineer at NVIDIA responsible for developing and presenting a wide range of deep learning software products. She comes with experience working in deep learning applications including vision and NLP. She holds a master's degree in Engineering Management from Duke University.

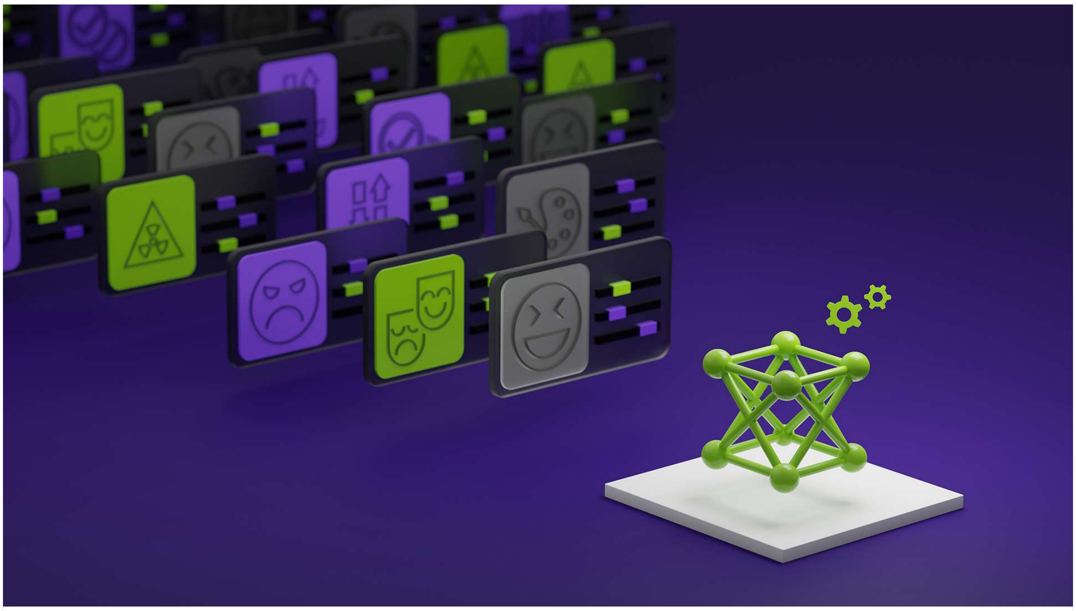
View all posts by Annie Surla

Comments

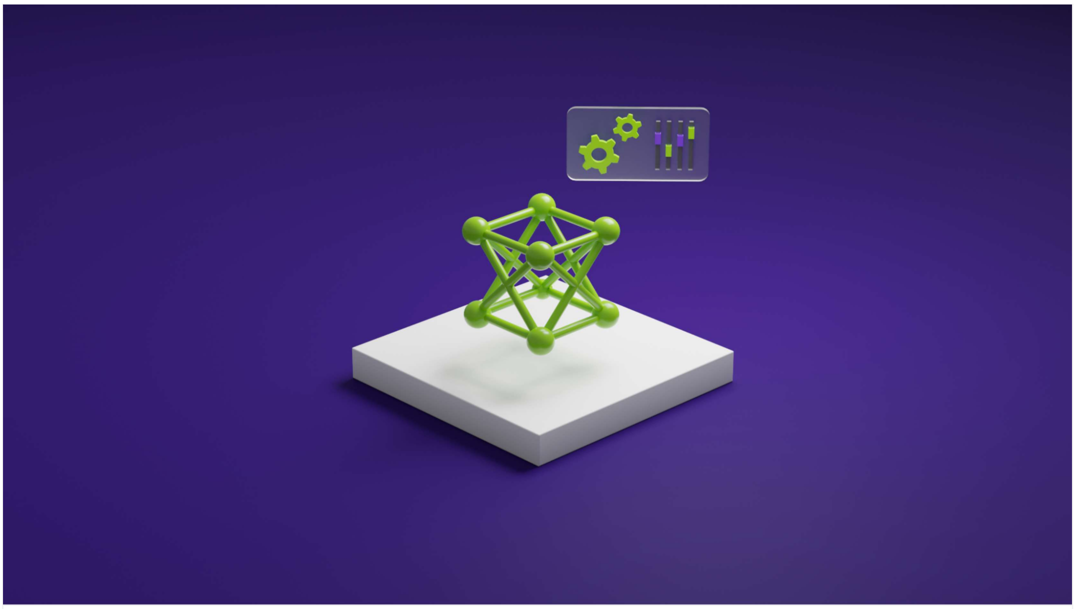
Start the discussion at forums.developer.nvidia.com

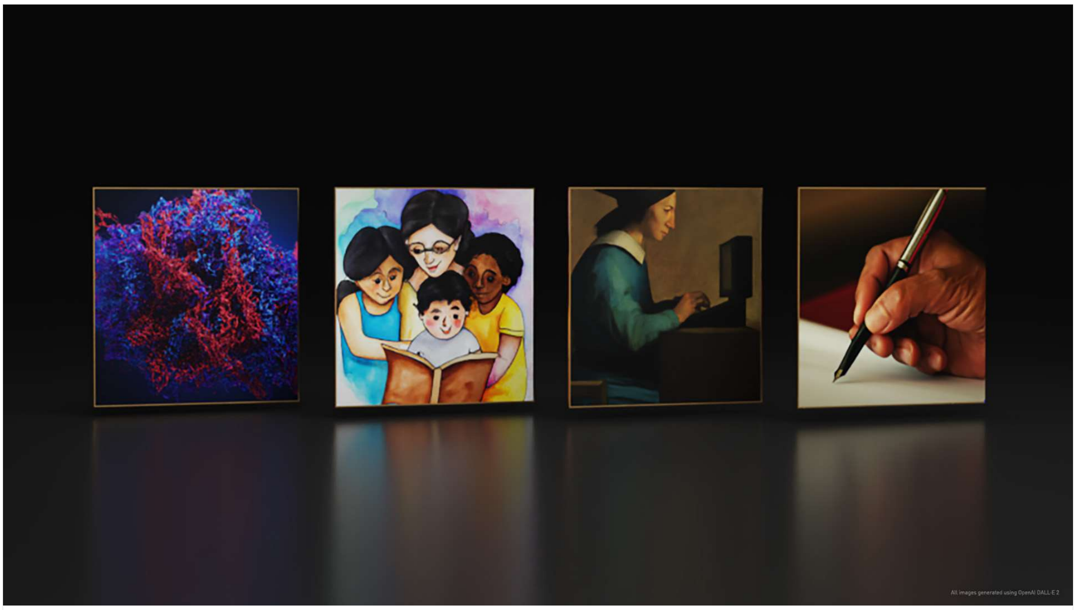


Announcing HelpSteer: An Open-Source Dataset for Building Helpful LLMs

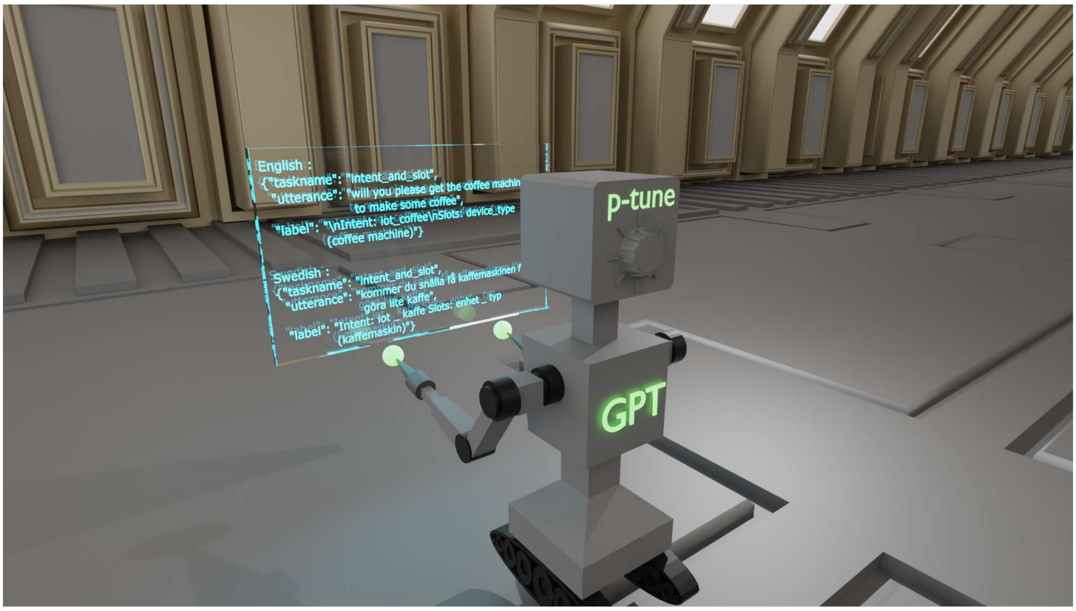


Announcing NVIDIA SteerLM: A Simple and Practical Technique to Customize LLMs During Inference





Simplifying Access to Large Language Models with the NVIDIA NeMo Framework and Services


English :
{"taskname": "intent_and_slot",
"utterance": "will you please get the coffee machin to make some coffee",
"label": "\nIntent: iot_coffee\nSlots: device_type (coffee machine)"}
Swedish :
{"taskname": "intent_and_slot",
"utterance": "kommer du snälla få kaffemaskinen t göra lite kaffe",
"label": "Intent: iot_kaffe Slots: enhet_typ (kaffemaskin)"}
p-tune
GPT

Adapting P-Tuning to Solve Non-English Downstream Tasks

Sign up for NVIDIA News Subscribe

Follow NVIDIA Developer

Privacy Policy | Manage My Privacy | Do Not Sell or Share My Data | Terms of Use | Cookie Policy | Contact