

Interacting with Llama 2 | Generative AI Series

Discover how to interact with Llama 2, Meta's powerful open-source language model. Learn key features, use cases, and best practices for optimal results.

Contents

| | | |
|----|--|----|
| 01 | Introduction | 3 |
| 02 | Prerequisites | 3 |
| 03 | Install the Python Modules | 4 |
| 04 | Launch the Huggingface Text Generation Container | 4 |
| 05 | Prompt Engineering | 5 |
| 06 | Definition of Tokens | 12 |
| 07 | Tweaking the API Parameters | 14 |
| 08 | Conclusion | 17 |

Introduction

The Hugging Face text generation inference is a production-ready Docker container that allows you to deploy and interact with Large Language Models (LLMs). For instance, you can use this container to run an API that exposes Llama 2 models programmatically.

Llama 2 is a collection of fine-tuned text models that you can use for natural language processing tasks.

In this guide, you are to implement a Hugging Face text generation Inference API on Vultr Cloud GPU.

Prerequisites

Before you begin:

- [Deploy a new Ubuntu 22.04](#) A100 Vultr Cloud GPU Server with at least:
 - 80 GB GPU RAM
 - 12 vCPUs
 - 120 GB Memory
- [Establish an SSH connection to the server.](#)
- [Create a non-root user with `sudo` rights and switch to the account.](#)
- [Create a HuggingFace account.](#)
- [Create a Hugging Face user access token.](#)
- Install JupyterLab.

Install the Python Modules

https://www.youtube.com/embed/g9PpYl_MGKs?si=_5zw30T53w9NG0Ma

After setting up a Docker container in this guide, you'll code some Python scripts to use the text generation inference API. These Python scripts rely on some Python libraries. Run the commands below to install the libraries.

CONSOLE

```
$ pip install huggingface-hub  
$ pip install transformers
```

Launch the Huggingface Text Generation Container

Hugging Face allows you to download and run a Docker container with all the necessary settings to access LLMs. Follow the steps below to download and run the Docker container:

1. Initialize the following variables. Replace `$HF_TOKEN` with your Hugging Face user access token. The `meta-llama/Llama-2-7b-chat-hf` is the model you will use to generate text.

CONSOLE

```
$ model=meta-llama/Llama-2-7b-chat-hf  
volume=$PWD/data  
token=$HF_TOKEN
```

2. Run the following command to download and run the Hugging Face container.

CONSOLE

```
$ sudo docker run -d \  
--name hf-tgi \  
--runtime=nvidia \  
--gpus all \  
-e HUGGING_FACE_HUB_TOKEN=$token \  
-p 8080:80 \  
-v $volume:/data \  
ghcr.io/huggingface/text-generation-inference:1.1.0 \  
--model-id $model
```

3. Wait for the command above to finish, and then check the Docker logs to ensure the container listens for incoming requests.

CONSOLE

```
$ sudo docker logs -f hf-tgi
```

The following output shows that the API is listening to all IP addresses.

```
src/main.rs:247: Connected  
.....Invalid hostname, defaulting to 0.0.0.0
```

4. Click **Python 3 ipykernel** under **Notebook**. This opens a new notebook where you can type and run the Python codes in the following sections.

Prompt Engineering

<https://www.youtube.com/embed/DFNVYJwmV0Q?si=Galp0MGaD-v5d-lV>

Every LLM understands a specific template, format, or prompt to ensure optimal output. Designing the correct format is called prompt engineering. While most LLMs are very flexible in handling such prompts, you may need some experimentation before perfecting crafting better prompts.

The Hugging Face LLM uses the following prompt template:

```
<s>
[INST]
  <<SYS>>
    {{ system_prompt }}
  <</SYS>>
  {{ user_message }}
[/INST]
```

In the following sections, you'll programmatically use the above template using some Python code to prompt the LLM.

Default Prompt

In this step, you'll query the LLM using a default prompt to answer a query.

- Run the following code on your Jupyter Notebook.

PYTHON

```
from huggingface_hub import InferenceClient

URI      = "http://127.0.0.1:8080"
client  = InferenceClient(model = URI)
Role    = """
    You are a helpful, respectful and honest assistant.
    Always answer as helpfully as possible, while being safe.
    Your answers should not include any harmful, unethical,
    racist, sexist, toxic, dangerous, or illegal content. Please
    ensure that your responses are socially unbiased and positive
    in nature.\\n\\nIf a question does not make any sense, or is
    not factually coherent, explain why instead of answering
    something not correct. If you don't know the answer to a
    question, please don't share false information.
    """

User    = """There's a python in my garden. What should I
do?"""

prompt = f"<s>[INST]<<SYS>>{Role}<</SYS>>{User}[/INST]"

print(client.text_generation(prompt = prompt, max_new_tokens
= 500).strip())
```

You should see the following output:

```
Thank you for reaching out! I'm here to help you in a responsible and safe
manner.
...
Remember, it is important to prioritize safety and respect the wildlife in your
area. If you have any further questions or concerns, please feel free to ask!
```

Zero-shot Prompt

In a zero-shot prompt, you should give the model a prompt that describes how you want it to respond.

- Execute the following Python code to ask the model to describe the trend of electronic sales.

PYTHON

```
from huggingface_hub import InferenceClient

URI = "http://127.0.0.1:8080"
client = InferenceClient (model = URI)

def chat_completion(system_prompt, user_prompt, length =
100):

    final_prompt = f"""\<s>[INST]<<SYS>>
                    {system_prompt}
                    <</SYS>>{user_prompt}
                    [/INST]
                    """

    return client.text_generation(prompt =
final_prompt,max_new_tokens = length).strip()

system_prompt = """
    You are a helpful AI assistant. Please be concise in
    your responses.
    """

user_prompt = """
    Imagine a dataset showing a significant increase in
    online retail electronics sales over the past three years,
    particularly in mobile devices and wearable technology.
```

```
Provide an analysis of this trend and potential reasons
behind it.
```

```
print(chat_completion(system_prompt, user_prompt, length =
1000))
```

Output:

Great, I'd be happy to help you with that!

The significant increase in online retail electronics sales over the past three years, particularly in mobile devices and wearable technology, is a clear trend that suggests several factors at play. Here are some potential reasons:

1. Smartphone adoption: The widespread adoption of smartphones has revolutionized the way people shop, communicate, and access information. As more people own smartphones, they are increasingly turning to online retailers for purchases, especially in the electronics category.

1. Convenience and accessibility: Online shopping offers unparalleled convenience and accessibility, allowing people to shop from anywhere, at any time. This is particularly appealing in the electronics category, where customers may need to compare prices, read reviews, and make informed purchasing decisions.

1. Competition and pricing: Online retailers have been able to offer competitive pricing and promotions, which has attracted price-sensitive customers. Additionally, the rise of marketplaces like Amazon has created a level playing field for small and large retailers alike, further driving competition and innovation.

1. Innovation and new product launches: The electronics industry has seen significant innovation in recent years, with new product launches and advancements in areas like augmented reality, artificial intelligence, and the Internet of Things (IoT). This has created new opportunities for online retailers to showcase and sell these cutting-edge products.

1. Shift to online shopping: As more people become comfortable shopping online, the electronics category has seen a corresponding shift towards online retail. This is partly due to the convenience and accessibility mentioned earlier, but also due to the growing popularity of online shopping in general.

1. Changing consumer behavior: The rise of e-commerce has also led to changes in consumer behavior, with more people preferring to shop online rather than in physical stores. This shift has been driven by factors like convenience, time-saving, and the desire for a more personalized shopping experience.

1. Increased access to credit: The growth of online retail has also been fueled by increased access to credit, which has enabled more people to make larger purchases, including electronics.

1. Improved logistics and delivery: The rise of e-commerce has led to

improvements in logistics and delivery, making it faster and more convenient for people to receive their purchases. This has helped to drive growth in the electronics category, particularly in mobile devices and wearable technology.

These are just a few potential reasons behind the significant increase in online retail electronics sales over the past three years. The trend is likely the result of a combination of factors, including technological advancements, changing consumer behavior, and the growth of e-commerce.

- Change the `user_prompt` to the following text and review the response.

```
PYTHON
```

```
...
user_prompt = """Discuss the ethical implications of using
artificial intelligence in decision-making processes that
directly affect human lives, such as employment, legal
judgment, and healthcare"""
...
```

Output:

Certainly! The use of artificial intelligence (AI) in decision-making processes that directly affect human lives raises important ethical considerations. Here are some of the key ethical implications:

1. Bias and Discrimination: AI algorithms can perpetuate existing biases and discrimination if they are trained on biased data or designed with a particular worldview. This can lead to unfair outcomes, particularly in areas like employment and healthcare, where decisions can have a significant impact on individuals and communities.
2. Lack of Transparency: AI algorithms can be complex and difficult to interpret, making it challenging to understand the reasoning behind decisions. This lack of transparency can erode trust in AI systems and make it difficult to identify and address biases or errors.
3. Accountability: As AI systems become more autonomous, it can be difficult to determine who is responsible when an AI decision goes wrong. This raises questions about accountability and the need for clearer lines of responsibility and oversight.
4. Privacy: AI systems often rely on vast amounts of personal data to make decisions, which raises concerns about privacy and data protection. Individuals may not want their personal information to be used in this way, and there may be a lack of control over how their data is collected, stored, and used.

5. Job Displacement: The increasing use of AI in decision-making processes could lead to job displacement, particularly in sectors where tasks are repetitive or can be easily automated. This raises questions about the need for a social safety net and retraining programs to help workers adapt to the changing job market.
6. Healthcare: AI systems are being used in healthcare to make decisions about diagnosis, treatment, and patient care. However, there are concerns about the potential for bias in AI-driven diagnoses, particularly in cases where there is limited data available for certain patient groups.
7. Legal Judgment: AI systems are being used to make legal judgments, such as predicting the outcomes of court cases. However, there are concerns about the potential for bias in AI-driven legal judgments, particularly in cases where there is limited data available for certain groups.

Overall, the ethical implications of using AI in decision-making processes that directly affect human lives are complex and multifaceted. It is important to consider these implications carefully and develop ethical frameworks that address these concerns and ensure that AI is used in a responsible and ethical manner.

One-shot Prompt

In this prompt, the model requires a sample input text and the corresponding output to generate other samples.

- Run the following Python code to ask the model to translate an English text to French.

PYTHON

```
from huggingface_hub import InferenceClient

URI = "http://127.0.0.1:8080"
client = InferenceClient(model = URI)

def chat_completion(system_prompt, user_prompt, length = 100):

    final_prompt = f"""\<s>[INST]<<SYS>>{system_prompt}
                    <</SYS>>{user_prompt} [/INST]
                    """

    return client.text_generation(prompt = final_prompt,
max_new_tokens = length).strip()

system_prompt = ""
```

```

        You are language translator specializing in English and
        French. Don't say anything else but the value.
        """
user_prompt = """
        Here is a sentence in English and its translation in
        French: [EN]'Hello, how are you?' [FR]'Bonjour, comment ça
        va?', Translate the following sentence into French: 'I am
        doing well, thank you!'
        """
print(chat_completion(system_prompt, user_prompt, length =
1000))

```

Output:

```
[FR]Je vais bien, merci!
```

Few-shot Prompt

In this strategy, the model accepts a few examples of the tasks that you want it to do.

- Run the following code to prompt the model to complete proverbs.

PYTHON

```

from huggingface_hub import InferenceClient

URI = "http://127.0.0.1:8080"
client = InferenceClient (model = URI)

def chat_completion(system_prompt, user_prompt, length =
100):

    final_prompt=f"""\<s>[INST]<<SYS>>
    {system_prompt}
    <</SYS>>

    {user_prompt} [/INST]"""

    return client.text_generation(prompt = final_prompt,
max_new_tokens = length).strip()

system_prompt = """
        You are a helpful AI assistant. Please be concise

```

```
in your responses.
"""
user_prompt = """
    Based on the below examples, complete the task.

    Example 1:

        Input : "The sun rises in the"
        Output: "east."

    Example 2:

        Input: "An apple a day keeps the"
        Output: "doctor away."

    Task:

        Input: "The pen is mightier"
    """
print(chat_completion(system_prompt, user_prompt, length =
1000))
```

Output:

```
Of course! I'm here to help. Here's the completion of the task based on the
examples provided:
```

```
Input: "The pen is mightier than the"
```

```
Output: "sword."
```

Definition of Tokens

<https://www.youtube.com/embed/dE-4XKBLPDw?si=yzgLE0f2U5iQaWe0>

When dealing with LLMs, tokens are numerical representations of words or characters. Tokens form the basic unit an LLM uses to process and generate output.

For instance, the Llama 2 model can accept a maximum of 4096 tokens.

- Run the following Python code to understand how LLMs generate and handle tokens.

PYTHON

```
from transformers import AutoTokenizer

model = "meta-llama/Llama-2-7b-chat-hf"
tokenizer = AutoTokenizer.from_pretrained(model, token =
"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")

prompt = "Apple is a fruit"

encoded_value = tokenizer(prompt)

print(encoded_value["input_ids"])

prompt = "Mango is a fruit"
encoded_value = tokenizer(prompt)
print(encoded_value["input_ids"])

prompt = "Bus is a vehicle"
encoded_value = tokenizer(prompt)
print(encoded_value["input_ids"])

def count_words (sentence):
    words = sentence.split()
    number_of_words = len(words)
    return number_of_words

prompt = "The quick brown fox jumps over the lazy dog"

print(count_words(prompt))
encoded_value = tokenizer(prompt)
print(len(encoded_value['input_ids']))
```

Output:

```
[1, 12113, 338, 263, 15774]
[1, 341, 4524, 338, 263, 15774]
[1, 8406, 338, 263, 19716]
```

```
9
12
```

Tweaking the API Parameters

```
https://www.youtube.com/embed/tlxnlkbgbhc?si=0kEEQUWSuwCylDjK
```

When running the Hugging Face text generation inference, you can change some model's parameters. For instance, to tweak the model parameters when running the text generation inference container, follow the steps below:

1. Initialize the container's variables.

CONSOLE

```
$ model=meta-llama/Llama-2-7b-chat-hf
volume=$PWD/data
token=$HF_TOKEN
```

- Run the Docker container. The `max-input-length` parameter declares the maximum number of tokens the LLM can accept. The `max-total-tokens` parameter defines the total tokens the LLM generates, including the input token.

CONSOLE

```
$ sudo docker run -d \
--name hf-tgi \
--runtime=nvidia \
--gpus all \
-e HUGGING_FACE_HUB_TOKEN=$token \
-p 8080:80 \
-v $volume:/data \
ghcr.io/huggingface/text-generation-inference:1.1.0 \
--model-id $model \
--max-input-length 2048 \
--max-total-tokens 4096
```

- Wait for the docker container to load, then run the following Python code on your Jupyter notebook.

```
PYTHON

from huggingface_hub import InferenceClient

URI = 'http://127.0.0.1:8080'
client = InferenceClient(model = URI)

def chat_completion(system_prompt, user_prompt, length =
100,temperature = None, repetition_penalty = 1.0, top_p =
None, top_k = None):

    final_prompt = f"<s>[INST]<<SYS>>{system_prompt}<</
SYS>> {user_prompt} [/INST]"

    return client.text_generation(prompt =
final_prompt,max_new_tokens = length, temperature =
temperature, top_p = top_p, top_k = top_k)

system_prompt = """
        You are an assistant to help me write short
stories.
        """
user_prompt = """
        Here's an example of a fairy tale opening: 'Once
upon a time, in a kingdom far, far away, there lived a gentle
dragon who loved to play in the meadows.'
        Now, start a new fairy tale story with the
sentence: 'In a land filled with whispers and magic, there
was a peculiar rabbit known to all.
        """

print(chat_completion(system_prompt, user_prompt).strip())

print(chat_completion(system_prompt, user_prompt, length =
1500).strip())

print(chat_completion(system_prompt, user_prompt, length =
1500,temperature = 0.2).strip())

print(chat_completion(system_prompt, user_prompt, length =
1500,temperature = 0.8).strip())
```

```
print(chat_completion(system_prompt, user_prompt, length =
1500,temperature = 0.1).strip())

print(chat_completion(system_prompt, user_prompt, length =
1500,repetition_penalty = 2.0).strip())

print(chat_completion(system_prompt, user_prompt, length =
1500,top_p = 0.2).strip())

print(chat_completion(system_prompt, user_prompt, length =
1500,top_k = 1).strip())
```

- Review the following output and note how your model behaves after tweaking the different API parameters.

Great! Here's my continuation of the story:

In a land filled with whispers and magic, there was a peculiar rabbit known to all. His name was Benny, and he was no ordinary rabbit.

...

And with that, Benny began to share the whispers of the land with Lily, revealing the hidden magic that lay just beneath the surface of the forest. From that day on, Lily and Benny became the best of friends, exploring the secrets of the land together and uncovering the wonders that lay hidden in plain sight.

In the previous code, the `chat_completion` function accepts the following parameters:

- `length`: Represents the `max_new_tokens` parameter for the model.
- `temperature`: Defines how creative the output will be for the model. The higher the temperature, the more the model randomizes the output. The default value is `none`.
- `repetition_penalty`: Instructs the model on how to deal with repetitive sequences.
- `top_p`: Limits the LLMs choices and prevents nonsensical outputs and unwanted diversity.
- `top_k`: Makes the model more focused and consistent.

Conclusion

In this guide, you've implemented a text generation Inference API on Vultr Cloud GPU. You've run a Hugging Face Docker container that exposes an API that generates content based on some prompts. Towards the end, you've learned how to tweak the model parameters to improve output.



VULTR

