Maven_Advanced_LLM_Module_2a_Mistral_7b_instruct

August 15, 2025

1 Installation of Libraries

If you use our code, please cite:

@misc{2024 title = {Mistral -7b hosting}, author = {Hamza Farooq, Darshil Modi, Kanwal Mehreen, Nazila Shafiei}, keywords = {LLM Hosting, Mistral-7b}, year = {2024}, copyright = {MIT, non-exclusive license} }

[1]:
```
from IPython.display import HTML, display

def set_css(*args, **kwargs):
  display(HTML('''
  <style>
    pre {
        white-space: pre-wrap;
    }
  </style>
  '''))
get_ipython().events.register('pre_run_cell', set_css)
```

[2]:
```
#!git add "git+https://github.com/huggingface/transformers" torch accelerate␣
↪bitsandbytes langchain
```

```
<IPython.core.display.HTML object>
```

[3]:
```
import locale
locale.getpreferredencoding = lambda: "UTF-8"
```

```
<IPython.core.display.HTML object>
```

[4]:
```
#!pip install pyngrok --quiet
#!pip install fastapi nest-asyncio --quiet
#!pip install uvicorn --quiet
#!pip install langchain-community --quiet
#!pip install -U langchain-huggingface --quiet
```

```
<IPython.core.display.HTML object>
```

2 Setup

Get you Hugging Face Token, here

[5]:
```
#from google.colab import userdata
#HUGGING_FACE_TOKEN = userdata.get('HUGGING_FACE_TOKEN')
```

<IPython.core.display.HTML object>

[6]:
```
#!huggingface-cli login --token $HUGGING_FACE_TOKEN
```

<IPython.core.display.HTML object>

[7]:
```
# Import libraries
from dotenv import load_dotenv
import os

load_dotenv()  # Make sure .env is in the same folder as this notebook
token = os.getenv("HUGGINGFACE_HUB_TOKEN")
assert token, "No HUGGINGFACE_HUB_TOKEN found in .env"

# ---- 2) Log in to Hugging Face ----
from huggingface_hub import login, whoami

login(token=token)  # Auth for this Python process
#print("HF user info:", whoami())  # Should print your username/orgs


from transformers import AutoModelForCausalLM, AutoTokenizer ,␣
 ↪BitsAndBytesConfig
import transformers
import torch
from langchain_huggingface import HuggingFacePipeline
from langchain_core.output_parsers import StrOutputParser
from langchain.prompts import PromptTemplate
from threading import Thread

device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

<IPython.core.display.HTML object>

[8]:
```
!nvidia-smi
```

<IPython.core.display.HTML object>

```
Fri Aug 15 15:47:32 2025
+-----------------------------------------------------------------------------
----------+
| NVIDIA-SMI 570.133.07             Driver Version: 570.133.07     CUDA Version:
12.8     |
```

```
|-----------------------------------------+------------------------+------------
----------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util
Compute M. |
|                                         |                        |
MIG M. |
|=========================================+========================+============
==========|
|   0  NVIDIA RTX A6000               Off |   00000000:68:00.0  On |
Off |
| 30%   39C    P8             26W /  300W |   39871MiB /  49140MiB |     32%
Default |
|                                         |                        |
N/A |
+-----------------------------------------+------------------------+------------
----------+

+-----------------------------------------------------------------------------------
----------+
| Processes:
|
|  GPU   GI   CI              PID   Type   Process name
GPU Memory |
|        ID   ID
Usage      |
|===================================================================================
==========|
|    0   N/A  N/A           2956      G   /usr/lib/xorg/Xorg
348MiB |
|    0   N/A  N/A           3174      G   /usr/bin/gnome-shell
153MiB |
|    0   N/A  N/A           3781      G   …exec/xdg-desktop-portal-gnome
67MiB |
|    0   N/A  N/A           4010      G   …chrome https://ubuntu.com/pro
4MiB |
|    0   N/A  N/A           4059      G   …ersion=20250714-050047.266000
296MiB |
|    0   N/A  N/A           5402      G   /usr/bin/gnome-control-center
30MiB |
|    0   N/A  N/A          10887      G   /usr/bin/nautilus
52MiB |
|    0   N/A  N/A          54905      G   …ess --variations-seed-version
37MiB |
|    0   N/A  N/A          60911      C   …stems/Week2/.venv/bin/python3
38688MiB |
|    0   N/A  N/A          69543      G   …nap-store/1270/bin/snap-store
```

```
55MiB |
+-----------------------------------------------------------------------------
----------+
```

[9]:
```python
import textwrap

def wrap_text(text, width=90): #preserve_newlines
    # Split the input text into lines based on newline characters
    lines = text.split('\n')

    # Wrap each line individually
    wrapped_lines = [textwrap.fill(line, width=width) for line in lines]

    # Join the wrapped lines back together using newline characters
    wrapped_text = '\n'.join(wrapped_lines)

    return wrapped_text
```

```
<IPython.core.display.HTML object>
```

3 Basic Prompt

Note: If you are on windows then bitsandbytes won't work. You can counter this with bitsandbytes-windows but that is and older version that only supports 8-bit quantization.

[10]:
```python
# Define the model ID for the desired model
%env HF_HOME=/mnt/sda1/huggingface
from dotenv import load_dotenv
load_dotenv()
import os

model_id = "mistralai/Mistral-7B-Instruct-v0.1"

# Define the quantization configuration for the model
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.bfloat16
)

# Load the model using the model ID and quantization configuration
model = AutoModelForCausalLM.from_pretrained(model_id,␣
 ↪quantization_config=quantization_config, device_map='auto', token=os.
 ↪getenv("HUGGINGFACE_HUB_TOKEN"))

# Load the tokenizer associated with the model
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

```
<IPython.core.display.HTML object>
```

```
env: HF_HOME=/mnt/sda1/huggingface
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

[11]:
```
vegetarian_recipe_prompt = """### Instruction: Act as a gourmet chef.
what is the best chicken dish for large groups

 ### Answer:
 """
prompt_template = """
### [INST]
Instruction: Act as a gourmet chef.


### QUESTION:
what is the best chicken dish for large groups

[/INST]
 """

# Encode the instruction using the tokenizer
encoded_instruction = tokenizer(prompt_template, return_tensors="pt",␣
 ↪add_special_tokens=True)

# Move the encoded instruction to the appropriate device (e.g., GPU)
model_inputs = encoded_instruction.to(device)

# Generate text based on the model inputs
generated_ids = model.generate(**model_inputs, max_new_tokens=1000,␣
 ↪do_sample=True, pad_token_id=tokenizer.eos_token_id)

# Decode the generated text
decoded = tokenizer.batch_decode(generated_ids)

# Print the decoded output
print(decoded)
```

```
<IPython.core.display.HTML object>
```

```
["<s> \n### [INST]\nInstruction: Act as a gourmet chef.\n\n\n### QUESTION:\nwhat
is the best chicken dish for large groups\n\n[/INST]\n \nWhen it comes to
cooking for a group, chicken is always a great option as it is both flavorful
and versatile, and can be prepared in a wide variety of ways. Here are a few
large group chicken dishes that are sure to impress:\n\n1. Chicken Fried
Rice:\nChicken Fried Rice is a classic Chinese dish that is easy to prepare for
a large group. Simply cook the chicken in a wok or large frying pan until it is
well-cooked, then set it aside. In the same wok or pan, sauté chopped vegetables
and scrambled eggs, then add the cooked chicken and cook it for an additional 2
minutes. Serve the chicken fried rice hot with your choice of garnishes.\n\n2.
```

Grilled Chicken Skewers:\nGrilled chicken skewers are a great choice for a summer outdoor event or a casual backyard BBQ. Marinate the chicken strips in a mixture of olive oil, lemon juice, herbs, and spices for a few hours before grilling. Once the chicken is cooked, thread it onto skewers, along with any vegetables you enjoy like bell peppers, mushrooms, and onions. Serve the grilled chicken skewers with a side of tzatziki or another refreshing dip.\n\n3. Chicken Parmesan:\nChicken Parmesan is an Italian classic that can be customized to suit the tastes of your group. Start by coating thin slices of chicken in panko crumbs, then bake or fry them until crispy. Serve the chicken on a bed of marinara sauce and topped with a generous layer of melted mozzarella cheese, followed by grated parmesan cheese. Bake until the cheese is bubbly and melted, then serve with pasta and garlic bread.\n\n4. Chicken Tacos:\nTacos are a fun and flavorful dish that can be customized to suit the tastes of your group. Start by cooking shredded chicken in a spicy tomato sauce, then add any toppings you like such as chopped vegetables, shredded cheese, and sour cream. Serve the chicken tacos in warm tortillas or over lettuce wraps, and don't forget the fresh salsa and guacamole on the side!\n\nOverall, chicken is a great option for large group meals, as it is versatile and flavorful, and can be prepared in a wide variety of ways. Whether you're hosting a summer BBQ or a winter cookout, one of these chicken dishes</s>"]

[12]:
```
response = decoded[0].split("[/INST]")[1]
print((response.split('</s>')[0]).strip())
```

<IPython.core.display.HTML object>

When it comes to cooking for a group, chicken is always a great option as it is both flavorful and versatile, and can be prepared in a wide variety of ways. Here are a few large group chicken dishes that are sure to impress:

1. Chicken Fried Rice:
Chicken Fried Rice is a classic Chinese dish that is easy to prepare for a large group. Simply cook the chicken in a wok or large frying pan until it is well-cooked, then set it aside. In the same wok or pan, sauté chopped vegetables and scrambled eggs, then add the cooked chicken and cook it for an additional 2 minutes. Serve the chicken fried rice hot with your choice of garnishes.

2. Grilled Chicken Skewers:
Grilled chicken skewers are a great choice for a summer outdoor event or a casual backyard BBQ. Marinate the chicken strips in a mixture of olive oil, lemon juice, herbs, and spices for a few hours before grilling. Once the chicken is cooked, thread it onto skewers, along with any vegetables you enjoy like bell peppers, mushrooms, and onions. Serve the grilled chicken skewers with a side of tzatziki or another refreshing dip.

3. Chicken Parmesan:
Chicken Parmesan is an Italian classic that can be customized to suit the tastes of your group. Start by coating thin slices of chicken in panko crumbs, then bake or fry them until crispy. Serve the chicken on a bed of marinara sauce and

```
topped with a generous layer of melted mozzarella cheese, followed by grated
parmesan cheese. Bake until the cheese is bubbly and melted, then serve with
pasta and garlic bread.

4. Chicken Tacos:
Tacos are a fun and flavorful dish that can be customized to suit the tastes of
your group. Start by cooking shredded chicken in a spicy tomato sauce, then add
any toppings you like such as chopped vegetables, shredded cheese, and sour
cream. Serve the chicken tacos in warm tortillas or over lettuce wraps, and
don't forget the fresh salsa and guacamole on the side!

Overall, chicken is a great option for large group meals, as it is versatile and
flavorful, and can be prepared in a wide variety of ways. Whether you're hosting
a summer BBQ or a winter cookout, one of these chicken dishes
```

4 Creating LLM Chain to run a RAG prompt

[13]:
```python
text_generation_pipeline = transformers.pipeline(
    model=model,
    tokenizer=tokenizer,
    task="text-generation",  # Specify the task as text generation
    temperature=0.3,  # Temperature parameter for controlling randomness in␣
 ↪sampling
    repetition_penalty=1.1,  # Repetition penalty parameter to avoid repeating␣
 ↪tokens
    return_full_text=True,  # Flag to return full text instead of a list of␣
 ↪generated tokens
    max_new_tokens=1000,  # Maximum number of tokens to generate
    do_sample=True  # Flag to use sampling during text generation
)

prompt_template = """
### [INST]
Instruction: I will ask you a QUESTION and give you a CONTEXT and you will␣
 ↪respond with an answer easily understandable.

### CONTEXT:
{context}

### QUESTION:
{question}

[/INST]
 """

# Create HuggingFacePipeline object wrapping the text generation pipeline
```

```
mistral_llm = HuggingFacePipeline(pipeline=text_generation_pipeline)

# Create prompt object from the prompt template with input variables as context␣
 ↪and question
prompt = PromptTemplate(
    input_variables=["context", "question"],  # Specify input variables for the␣
 ↪prompt
    template=prompt_template,  # Specify the template for the prompt
)

# Create language model chain (llm_chain) with HuggingFacePipeline and prompt
llm_chain = prompt | mistral_llm | StrOutputParser()
```

<IPython.core.display.HTML object>

Device set to use cuda:0

[14]:
```
prompt_base = """
### [INST]
Instruction: Act as a gourmet chef.


### QUESTION:
what is the best chicken dish for large groups

[/INST]
 """
```

<IPython.core.display.HTML object>

[15]:
```
response=text_generation_pipeline(prompt_base)
```

<IPython.core.display.HTML object>

Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

[16]:
```
response
```

<IPython.core.display.HTML object>

[16]:
```
[{'generated_text': '\n### [INST]\nInstruction: Act as a gourmet chef.\n\n\n###
QUESTION:\nwhat is the best chicken dish for large groups\n\n[/INST]\n \nAs a
gourmet chef, I would recommend roasting a whole chicken for a large group. This
classic dish is not only delicious but also easy to prepare and serve. To make
it even more special, you can add some herbs and spices to the chicken before
roasting it in the oven. You can also serve it with a side of roasted vegetables
or a fresh salad. Another option is to make a chicken pot pie, which is a
comforting and hearty dish that everyone will love.'}]
```

[17]:
```
r=response[0].get('generated_text').split("[/INST]")[1].split('</s>')[0]
```

```
<IPython.core.display.HTML object>
```

[18]:
```
print(r.strip())
```

```
<IPython.core.display.HTML object>

As a gourmet chef, I would recommend roasting a whole chicken for a large group.
This classic dish is not only delicious but also easy to prepare and serve. To
make it even more special, you can add some herbs and spices to the chicken
before roasting it in the oven. You can also serve it with a side of roasted
vegetables or a fresh salad. Another option is to make a chicken pot pie, which
is a comforting and hearty dish that everyone will love.
```

[19]:
```
# Input query to LLM
input_text = {
    "context": "Artificial Intelligence (AI) is the stream of data science that␣
 ↪can predict future based on past trends!",
    "question": "What is AI ?"
}

# Generate and print LLM response
response = llm_chain.invoke(input_text)

wrapped_text = wrap_text(response)
print(f'The answer is: \n {wrapped_text}')
```

```
<IPython.core.display.HTML object>

Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

The answer is:

### [INST]
Instruction: I will ask you a QUESTION and give you a CONTEXT and you will
respond with an
answer easily understandable.

### CONTEXT:
Artificial Intelligence (AI) is the stream of data science that can predict
future based
on past trends!

### QUESTION:
What is AI ?

[/INST]

Artificial Intelligence, commonly known as AI, refers to the development of
computer
systems that are able to perform tasks that would normally require human
```

```
intelligence,
such as visual perception, speech recognition, decision-making, and language
translation.
These systems use algorithms, machine learning techniques, and other advanced
methods to
analyze and process large amounts of data in order to make predictions or take
actions
based on that data. In essence, AI allows machines to learn from experience and
adapt to
new situations, making them increasingly capable of performing complex tasks
autonomously.
```

[20]:
```
###############
question_rag = "What are the differences in work expectations between␣
 ↪generations?"
###############


context_rag = """
'Generational differences in work expectations and values are a significant␣
 ↪topic of discussion in organizations today. Specifically, the differences␣
 ↪between baby boomers and millennials have garnered attention due to their␣
 ↪coexistence in the workplace. These generational differences can impact␣
 ↪management dynamics at various levels, including operational, team, and␣
 ↪strategic levels. Understanding these differences and finding ways to␣
 ↪navigate them is crucial for creating a harmonious and productive work␣
 ↪environment.

1. Contrasting Work Values, Attitudes, and Preferences:
Baby Boomers:
- Work Values: Baby boomers tend to prioritize loyalty, dedication, and hard␣
 ↪work. They often value stability and long-term commitment to their␣
 ↪organizations.
- Attitudes: Baby boomers may have a more traditional approach to work, valuing␣
 ↪hierarchy and authority. They may prefer a structured work environment and␣
 ↪clear guidelines.
- Preferences: Baby boomers may prefer face-to-face communication and value␣
 ↪personal relationships in the workplace. They may also prioritize financial␣
 ↪stability and job security.

Millennials:
- Work Values: Millennials often prioritize work-life balance, personal growth,␣
 ↪and purpose-driven work. They seek opportunities for learning and␣
 ↪development.
- Attitudes: Millennials may have a more collaborative and inclusive approach␣
 ↪to work. They value flexibility, autonomy, and a sense of purpose in their␣
 ↪roles.
```

```
- Preferences: Millennials are comfortable with technology and may prefer␣
 ↪digital communication channels. They value feedback and recognition and seek␣
 ↪a positive and engaging work environment.

2. Factors Contributing to Variations:
a) Socioeconomic Factors: Socioeconomic factors such as technological␣
 ↪advancements, economic conditions, and societal changes can shape the␣
 ↪expectations and values of different generations. For example, baby boomers␣
 ↪experienced significant economic growth and stability, while millennials␣
 ↪entered the workforce during times of economic uncertainty.
b) Cultural Influences: Cultural factors, including societal norms, values, and␣
 ↪beliefs, can also contribute to generational differences. Each generation␣
 ↪grows up in a unique cultural context that influences their perspectives on␣
 ↪work.
c) Life Experiences: Different life experiences, such as major historical␣
 ↪events or technological advancements, can shape the values and attitudes of␣
 ↪each generation. For example, baby boomers may have experienced significant␣
 ↪social and political changes, while millennials grew up in the digital age.

3. Practical Scenarios and Examples:
To illustrate these generational differences, consider the following scenarios:
- Baby boomers may prefer a structured work environment with clear hierarchies,␣
 ↪while millennials may thrive in a more flexible and collaborative setting.
- Baby boomers may prioritize job security and financial stability, while␣
 ↪millennials may prioritize personal growth and work-life balance.
- Baby boomers may prefer face-to-face communication, while millennials may be␣
 ↪more comfortable with digital communication tools.

4. Navigating Generational Differences:
To create a harmonious and productive work environment, organizations can␣
 ↪consider the following strategies:
- Foster open communication and create opportunities for intergenerational␣
 ↪dialogue and understanding.
- Provide flexible work arrangements and opportunities for personal growth and␣
 ↪development.
- Recognize and appreciate the diverse perspectives and strengths that each␣
 ↪generation brings to the table.
- Implement mentorship or reverse-mentoring programs to facilitate knowledge␣
 ↪sharing and collaboration between different generations.
- Create a culture of inclusivity and respect, valuing the contributions of all␣
 ↪employees regardless of their generational background.
```

```
In summary, generational differences in work expectations and values exist␣
 ↪between baby boomers and millennials. These differences can be attributed to␣
 ↪various factors such as socioeconomic influences, cultural factors, and life␣
 ↪experiences. Understanding and navigating these differences is crucial for␣
 ↪organizations to create a harmonious and productive work environment. By␣
 ↪fostering open communication, providing flexibility, and appreciating the␣
 ↪diverse perspectives of each generation, organizations can effectively␣
 ↪manage generational differences and leverage the strengths of their␣
 ↪multi-generational workforce.'
"""
```

<IPython.core.display.HTML object>

[21]:
```
# Input query to LLM
input_text = {
    "context": context_rag,
    "question": question_rag
}

# Generate and print LLM response
response = llm_chain.invoke(input_text)

wrapped_text = wrap_text(response)
print(f'The answer is: \n {wrapped_text}')
```

<IPython.core.display.HTML object>

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

The answer is:

### [INST]
Instruction: I will ask you a QUESTION and give you a CONTEXT and you will
respond with an
answer easily understandable.

### CONTEXT:

'Generational differences in work expectations and values are a significant
topic of
discussion in organizations today. Specifically, the differences between baby
boomers and
millennials have garnered attention due to their coexistence in the workplace.
These
generational differences can impact management dynamics at various levels,
including
operational, team, and strategic levels. Understanding these differences and
finding ways
to navigate them is crucial for creating a harmonious and productive work
```

environment.

1. Contrasting Work Values, Attitudes, and Preferences:
Baby Boomers:
- Work Values: Baby boomers tend to prioritize loyalty, dedication, and hard work. They
often value stability and long-term commitment to their organizations.
- Attitudes: Baby boomers may have a more traditional approach to work, valuing hierarchy
and authority. They may prefer a structured work environment and clear guidelines.
- Preferences: Baby boomers may prefer face-to-face communication and value personal
relationships in the workplace. They may also prioritize financial stability and job
security.

Millennials:
- Work Values: Millennials often prioritize work-life balance, personal growth, and
purpose-driven work. They seek opportunities for learning and development.
- Attitudes: Millennials may have a more collaborative and inclusive approach to work.
They value flexibility, autonomy, and a sense of purpose in their roles.
- Preferences: Millennials are comfortable with technology and may prefer digital
communication channels. They value feedback and recognition and seek a positive and
engaging work environment.

2. Factors Contributing to Variations:
a) Socioeconomic Factors: Socioeconomic factors such as technological advancements,
economic conditions, and societal changes can shape the expectations and values of
different generations. For example, baby boomers experienced significant economic growth
and stability, while millennials entered the workforce during times of economic uncertainty.
b) Cultural Influences: Cultural factors, including societal norms, values, and beliefs,
can also contribute to generational differences. Each generation grows up in a unique
cultural context that influences their perspectives on work.
c) Life Experiences: Different life experiences, such as major historical events or
technological advancements, can shape the values and attitudes of each generation. For

example, baby boomers may have experienced significant social and political changes, while
millennials grew up in the digital age.

3. Practical Scenarios and Examples:
To illustrate these generational differences, consider the following scenarios:
- Baby boomers may prefer a structured work environment with clear hierarchies, while
millennials may thrive in a more flexible and collaborative setting.
- Baby boomers may prioritize job security and financial stability, while millennials may
prioritize personal growth and work-life balance.
- Baby boomers may prefer face-to-face communication, while millennials may be more
comfortable with digital communication tools.

4. Navigating Generational Differences:
To create a harmonious and productive work environment, organizations can consider the
following strategies:
- Foster open communication and create opportunities for intergenerational dialogue and
understanding.
- Provide flexible work arrangements and opportunities for personal growth and development.
- Recognize and appreciate the diverse perspectives and strengths that each generation
brings to the table.
- Implement mentorship or reverse-mentoring programs to facilitate knowledge sharing and
collaboration between different generations.
- Create a culture of inclusivity and respect, valuing the contributions of all employees
regardless of their generational background.

In summary, generational differences in work expectations and values exist between baby
boomers and millennials. These differences can be attributed to various factors such as
socioeconomic influences, cultural factors, and life experiences. Understanding and
navigating these differences is crucial for organizations to create a harmonious and
productive work environment. By fostering open communication, providing flexibility, and
appreciating the diverse perspectives of each generation, organizations can effectively
manage generational differences and leverage the strengths of their multi-

```
generational
workforce.'


### QUESTION:
What are the differences in work expectations between generations?

[/INST]
  The differences in work expectations between generations can vary
significantly, with
baby boomers and millennials having distinct work values, attitudes, and
preferences. Baby
boomers tend to prioritize loyalty, dedication, and hard work, while millennials
prioritize work-life balance, personal growth, and purpose-driven work. These
differences
can impact management dynamics at various levels, including operational, team,
and
strategic levels. Organizations must understand these differences and find ways
to
navigate them to create a harmonious and productive work environment.
```

5 Streaming Output & Metrics for Evaluation

[22]:
```python
from transformers import TextIteratorStreamer
from threading import Thread
import time

first_token_time = 0
token_times = []


# Initialize a TextIteratorStreamer object for streaming text generation
streamer = TextIteratorStreamer(tokenizer, timeout=10., skip_prompt=True,␣
 ↪skip_special_tokens=True)

# Create a text generation pipeline using the Hugging Face transformers library
text_generation_pipeline = transformers.pipeline(
    model=model,
    tokenizer=tokenizer,
    task="text-generation",
    temperature=0.3,
    repetition_penalty=1.1,
    max_new_tokens=1000,
    do_sample=True,
    streamer=streamer  # Use the streamer for streaming text generation
)
```

```
prompt_template = """
### [INST]
Instruction: I will ask you a QUESTION and give you a CONTEXT and you will␣
 ↪respond with an answer easily understandable.

### CONTEXT:
{context}

### QUESTION:
{question}

[/INST]
 """

mistral_llm = HuggingFacePipeline(pipeline=text_generation_pipeline)

prompt = PromptTemplate(
    input_variables=["context", "question"],
    template=prompt_template,
)

llm_chain = prompt | mistral_llm | StrOutputParser()


# Input query to LLM
input_text = {
    "context": """
    Act as a gourmet chef.
    I have a friend coming over who is a vegetarian.
    I want to impress my friend with a special vegetarian dish.
    """,
    "question": "Give me two options, along with the whole recipe for each."
}
```

```
<IPython.core.display.HTML object>
```

```
Device set to use cuda:0
```

[23]:
```
# Initialize variables for time measurements
start_time = time.time()

# Start a new thread to invoke the language model chain with the input text
thread = Thread(target=llm_chain.invoke, args=[input_text])
thread.start()

# Initialize a variable to store the model output
model_output = ""
```

```python
# Iterate over the streamer to get the generated text in chunks
for i, new_text in enumerate(streamer):
    model_output += new_text
    print(new_text, end='')

    # Measure time for the first token
    if i == 0:
        first_token_time = time.time()
    # Measure time for each token
    token_times.append(time.time())

# Calculate end-to-end latency
end_time = time.time()
end_to_end_latency = end_time - start_time

# Calculate time to first token
ttft = first_token_time - start_time

# Calculate inter-token latency
itl = sum(x - y for x, y in zip(token_times[1:], token_times[:-1])) /␣
 ↪(len(token_times) - 1)

# Calculate throughput
throughput = len(tokenizer.encode(model_output)) / end_to_end_latency

print("\nTime To First Token (TTFT):", ttft)
print("Inter-token latency (ITL):", itl)
print("End-to-end Latency:", end_to_end_latency)
print("Throughput:", throughput)
```

```
<IPython.core.display.HTML object>
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
```

```
1. Vegetable lasagna:

Ingredients:

* 1 pound lasagna noodles
* 2 tablespoons olive oil
* 1 onion, chopped
* 3 cloves garlic, minced
* 8 ounces mushrooms, sliced
* 8 ounces zucchini, sliced
* 8 ounces eggplant, sliced
* 1 15-ounce can tomato sauce
* 1 cup ricotta cheese
* 1 egg, beaten
* 1 teaspoon dried basil
```

* 1 teaspoon dried oregano
* Salt and pepper to taste
* 1 1/2 cups grated mozzarella cheese
* 1/4 cup grated parmesan cheese

Instructions:

1. Preheat oven to 375 degrees F (190 degrees C).
2. Cook lasagna noodles according to package instructions until al dente. Drain and set aside.
3. In a large skillet, heat olive oil over medium heat. Add onion and garlic and cook until softened, about 5 minutes.
4. Add mushrooms, zucchini, and eggplant to the skillet and cook until vegetables are tender, about 10 minutes.
5. Stir in tomato sauce, ricotta cheese, egg, basil, oregano, salt, and pepper.
6. Spread a thin layer of the vegetable mixture in the bottom of a 9x13 inch baking dish. Place a layer of cooked lasagna noodles on top. Repeat layers until all ingredients are used, ending with a layer of vegetable mixture on top.
7. Sprinkle mozzarella and parmesan cheeses over the top of the lasagna.
8. Cover with foil and bake in preheated oven for 25 minutes. Remove foil and bake for an additional 25 minutes, or until cheese is melted and bubbly.
9. Let cool for 10 minutes before serving.

2. Quinoa-stuffed bell peppers:

Ingredients:

* 4 large bell peppers, any color
* 1 cup quinoa
* 2 cups vegetable broth
* 1 onion, chopped
* 3 cloves garlic, minced
* 1 red bell pepper, chopped
* 1 green bell pepper, chopped
* 1 yellow bell pepper, chopped
* 1 cup corn kernels
* 1 cup black beans, drained and rinsed
* 1 teaspoon cumin
* 1 teaspoon smoked paprika
* Salt and pepper to taste
* Juice of 1 lime
* 1/4 cup chopped fresh cilantro
* 1 avocado, diced (optional)

Instructions:

1. Preheat oven to 375 degrees F (190 degrees C).
2. Cut off the tops of the bell peppers and remove the seeds and membranes.

```
Place the peppers in a baking dish and set aside.
3. In a saucepan, bring quinoa and vegetable broth to a boil. Reduce heat,
cover, and simmer for 15-20 minutes, or until quinoa is cooked and liquid is
absorbed.
4. In a large skillet, heat oil over medium heat. Add onion and garlic and cook
until softened, about 5 minutes.
5. Add red bell pepper, green bell pepper, and yellow bell pepper to the skillet
and cook until vegetables are tender, about 10 minutes.
6. Stir in corn kernels, black beans, cumin, smoked paprika, salt, and pepper.
7. Add lime juice and cilantro to the skillet and stir to combine.
8. Add the cooked quinoa mixture to the bell peppers and bake in preheated oven
for 25-30 minutes, or until peppers are tender and filling is hot.
9. Serve with diced avocado on top, if desired.
Time To First Token (TTFT): 0.12711381912231445
Inter-token latency (ITL): 0.03599916295758609
End-to-end Latency: 33.53449082374573
Throughput: 27.702821100899982
```

Let's create it into a function

[24]:
```python
def streaming_inference(context, question):
    # Initialize a TextIteratorStreamer object for streaming text generation
    streamer = TextIteratorStreamer(tokenizer, timeout=10., skip_prompt=True,␣
 ↪skip_special_tokens=True)

    # Create a text generation pipeline using the Hugging Face transformers␣
 ↪library
    text_generation_pipeline = transformers.pipeline(
        model=model,
        tokenizer=tokenizer,
        task="text-generation",
        temperature=0.3,
        repetition_penalty=1.1,
        max_new_tokens=1000,
        do_sample= True,
        streamer=streamer  # Use the streamer for streaming text generation
    )

    prompt_template = """
    ### [INST]
    Instruction: I will ask you a QUESTION and give you a CONTEXT and you will␣
 ↪respond with an answer easily understandable in bullet points.

    ### CONTEXT:
    {context}

    ### QUESTION:
    {question}
```

```
[/INST]
"""

mistral_llm = HuggingFacePipeline(pipeline=text_generation_pipeline)

prompt = PromptTemplate(
    input_variables=["context", "question"],
    template=prompt_template,
)

llm_chain = prompt | mistral_llm | StrOutputParser()

input_text = {
    "context": context,
    "question": question
}

# Initialize variables for time measurements
start_time = time.time()

# Start a new thread to invoke the language model chain with the input text
thread = Thread(target=llm_chain.invoke, args=[input_text])
thread.start()

# Initialize a variable to store the model output
model_output = ""

# Iterate over the streamer to get the generated text in chunks
for i, new_text in enumerate(streamer):
    model_output += new_text
    print(new_text, end='')

    # Measure time for the first token
    if i == 0:
        first_token_time = time.time()
    # Measure time for each token
    token_times.append(time.time())

# Calculate end-to-end latency
end_time = time.time()
end_to_end_latency = end_time - start_time

# Calculate time to first token
ttft = first_token_time - start_time

# Calculate inter-token latency
```

```
    itl = sum(x - y for x, y in zip(token_times[1:], token_times[:-1])) /␣
↪(len(token_times) - 1)

    # Calculate throughput
    throughput = len(tokenizer.encode(model_output)) / end_to_end_latency

    print("\nTime To First Token (TTFT):", ttft)
    print("Inter-token latency (ITL):", itl)
    print("End-to-end Latency:", end_to_end_latency)
    print("Throughput:", throughput)

    # Return the metrics
    return {
        "TTFT": ttft,
        "ITL": itl,
        "End-to-end Latency": end_to_end_latency,
        "Throughput": throughput
    }
```

```
<IPython.core.display.HTML object>
```

[25]:
```
streaming_inference(context_rag,question_rag)
```

```
<IPython.core.display.HTML object>

Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.


   The differences in work expectations between generations are primarily driven
by the values, attitudes, and preferences of each generation. Here are some key
differences between baby boomers and millennials:

   - Baby boomers tend to prioritize loyalty, dedication, and hard work, whereas
millennials prioritize work-life balance, personal growth, and purpose-driven
work.
   - Baby boomers may have a more traditional approach to work, valuing
hierarchy and authority, while millennials may have a more collaborative and
inclusive approach to work.
   - Baby boomers may prefer face-to-face communication and value personal
relationships in the workplace, while millennials are comfortable with
technology and may prefer digital communication channels.
   - Baby boomers may prioritize financial stability and job security, while
millennials may prioritize personal growth and work-life balance.

These differences can impact management dynamics at various levels, including
operational, team, and strategic levels. Organizations need to understand these
differences and find ways to navigate them to create a harmonious and productive
work environment.
```

```
Time To First Token (TTFT): 0.26491475105285645
Inter-token latency (ITL): 0.03634322060845715
End-to-end Latency: 8.55327558517456
Throughput: 26.65645432905186
```

[25]:
```
{'TTFT': 0.26491475105285645,
 'ITL': 0.03634322060845715,
 'End-to-end Latency': 8.55327558517456,
 'Throughput': 26.65645432905186}
```

Time To First Token (TTFT): 5.885674238204956 Inter-token latency (ITL): 0.25101944495291606 End-to-end Latency: 29.334325551986694 Throughput: 7.124759000496103

6 Setting Up a FastAPI Wrapper

[26]:
```python
from fastapi import FastAPI, Request, BackgroundTasks
from fastapi.responses import StreamingResponse
from threading import Thread

app = FastAPI()

# Variables for time measurements
start_time = 0
first_token_time = 0
token_times = []

# Invoke the LLM chain using the input text
def invoke_llm_chain(input_text):
    llm_chain.invoke(input_text)


# Function to calculate metrics
def calculate_metrics(start_time, first_token_time, token_times, model_output):
    end_time = time.time()
    end_to_end_latency = end_time - start_time
    ttft = first_token_time - start_time
    itl = sum(x - y for x, y in zip(token_times[1:], token_times[:-1])) /␣
 ↪(len(token_times) - 1)

    throughput = len(model_output) / end_to_end_latency
    return {
        "End-to-end Latency": end_to_end_latency,
        "Time To First Token (TTFT)": ttft,
        "Inter-token latency (ITL)": itl,
        "Throughput": throughput
    }
```

```
# Generate output text using the streamer
def generate_output(streamer):
    global start_time, first_token_time, token_times, model_output
    model_output = ""
    start_time = time.time()

    for i, new_text in enumerate(streamer):
        model_output += new_text

        # Measure time for the first token
        if i == 0:
            first_token_time = time.time()

        # Measure time for each token
        token_times.append(time.time())
        yield new_text

    metrics = calculate_metrics(start_time, first_token_time, token_times,␣
 ↪model_output)
    print("Metrics:", metrics)
    return metrics


@app.get("/")
async def root():
    return {"message": "Hello, World!"}


@app.post("/mistral-inference")
async def mistral_inference(input_text: dict, background_tasks:␣
 ↪BackgroundTasks):
    # Start a separate thread to run the LLM chain asynchronously
    thread = Thread(target=invoke_llm_chain, args=[input_text])
    thread.start()

    # Add the generate_output function to the background tasks with the streamer
    background_tasks.add_task(generate_output, streamer)

    return StreamingResponse(generate_output(streamer))
```

```
<IPython.core.display.HTML object>
```

7 Using Ngrok to Create Public URL

Sign up for your own key on Ngrok

[27]:
```
NGROK_KEY = os.getenv('NGROK_KEY')
```

```
<IPython.core.display.HTML object>
```

[28]:
```
!ngrok config add-authtoken $NGROK_KEY
```

```
<IPython.core.display.HTML object>

Authtoken saved to configuration file: /home/richard/.config/ngrok/ngrok.yml
```

[29]:
```
import nest_asyncio
from pyngrok import ngrok
import uvicorn
ngrok.kill()
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
ngrok_tunnel = ngrok.connect(8000)
print('Public URL:', ngrok_tunnel.public_url)
nest_asyncio.apply()

uvicorn.run(host="127.0.0.1", port=8000, app=app)
```

```
<IPython.core.display.HTML object>

INFO:     Started server process [73877]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)

Public URL: https://34586ccad689.ngrok-free.app
INFO:     2600:4040:ae09:de00:69f:fde9:b12a:5609:0 - "GET / HTTP/1.1" 200 OK
INFO:     2600:4040:ae09:de00:69f:fde9:b12a:5609:0 - "POST /mistral-inference
HTTP/1.1" 200 OK

Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

Metrics: {'End-to-end Latency': 1.2409858703613281, 'Time To First Token
(TTFT)': 0.14339852333068848, 'Inter-token latency (ITL)': 0.040641316661128295,
'Throughput': 91.05663706477067}
```

[ ]:

[ ]:

[ ]: