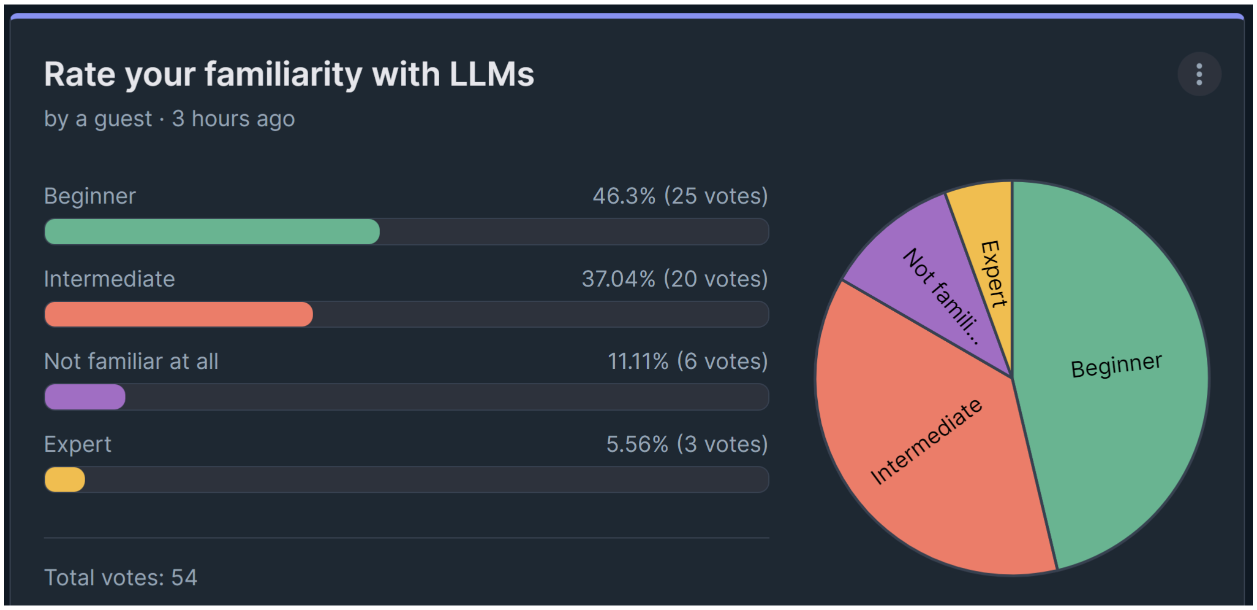
Understanding the Audience

Provide your input on familiarity with LLMs:
https://strawpoll.com/XmZRQxPeWgd

Or scan :

Responses :



Rate your familiarity with LLMs

by a guest · 3 hours ago

Beginner	46.3% (25 votes)
Intermediate	37.04% (20 votes)
Not familiar at all	11.11% (6 votes)
Expert	5.56% (3 votes)

Total votes: 54

The purpose of this manual is to make you familiar with different frameworks and various ways of running a LLM model.

We will majorly talk about :
Popular Frameworks for running and downloading LLMs :
LMStudio(GUI based), Ollama(Command based)

Popular frameworks for creating a LLM application :
Langchain, LlamaIndex

And also get familiar with Hugging Face which is a machine learning (ML) and data science platform and community that helps users build, deploy and train various ai models.

Popular libraries/frameworks for creating frontend :
Streamlit, Gradio

About Hugging Face :

Hugging Face is a company and community known for its open-source contributions to natural language processing (NLP) and machine learning. It provides a wide range of tools, models, and libraries to facilitate the development and deployment of machine learning models, particularly in the field of NLP.

Key Components:

- Transformers Library: A popular open-source library that provides pre-trained models for various NLP tasks such as text classification, text generation, translation, and more. It includes models like BERT, GPT, T5, and many others.

- Datasets Library: Provides tools and pre-built datasets for machine learning research and applications. It allows easy access to a variety of datasets for training and evaluation.

- Tokenizers Library: A fast and efficient library for tokenizing text, which is a crucial step in processing text data for NLP models.

- Hugging Face Hub: A platform for sharing and discovering machine learning models and datasets. It provides a central repository where users can upload their models and access others' contributions.

- Inference API: Allows users to easily deploy models and access them via a RESTful API. This is useful for integrating pre-trained models into applications without needing to manage the underlying infrastructure.

Hugging Face Pipelines are high-level abstractions provided by the Transformers library to simplify the process of using pre-trained models for various tasks. Pipelines streamline the workflow by handling tokenization, model inference, and post-processing in a single call, making it easier for developers to apply state-of-the-art NLP models without deep expertise in machine learning.

Pipelines support a wide range of predefined tasks such as:
- Text Generation: Generate coherent text based on a prompt.
- Sentiment Analysis: Determine the sentiment of a piece of text.
- Named Entity Recognition (NER): Identify and classify entities in text.
- Text Classification: Categorize text into predefined labels.
- Translation: Translate text from one language to another.
- Question Answering: Answer questions based on a given context.

Provide output for the following :

```
# Using pipeline class to make predictions from models available in the Hub in an easy way
from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis")
data = ["I love you", "I am ok with you"]
sentiment_pipeline(data)
```

```
# Using a specific model for sentiment analysis
specific_model = pipeline(model="finiteautomata/bertweet-base-sentiment-analysis")
specific_model(data)
```

```
from transformers import pipeline
```

```
detector = pipeline(task="object-detection")
preds = detector(
   "https://pes.irins.org/assets/profile_images/163446.jpg"
)
preds = [{"score": round(pred["score"], 4), "label": pred["label"], "box": pred["box"]} for
pred in preds]
preds
```

Try Out more at : https://huggingface.co/docs/transformers/en/main_classes/pipelines

Tools to run LLM locally (there are other tools too, following are popular):

LM Studio
Ollama

Using LM Studio :

Download LM Studio from : https://lmstudio.ai/

Open LM Studio, search for any model, chat with it.

You can also Run a server on LM studio and have a client application interact with it.

Using Ollama :

Ollama is a small program that operates quietly in the background, allowing you to handle and deploy large open-source language models such as llama3, llama2, meta, and others.

Using Ollama

```
%pip install -U langchain-ollama
```

Download and Install Ollama :

https://ollama.com/

It is actually a command-line application, so you can interact with it in the terminal directly. Open the terminal and type this command:

```
$ollama
```

Then pull the required model :

```
$ollama pull mistral
Or
$ollama pull llama3
```

To run a model on terminal/cmd prompt:

```
$ollama run <model name>
>>> Ask your query here

Type /bye to exit
```

Popular Frameworks : Langchain, Llamaindex

Following are the two robust frameworks designed for developing applications powered by large language models, each with distinct strengths and areas of focus.

- LangChain, a generic framework for developing stuff with LLM.(https://python.langchain.com/v0.2/docs/introduction/)
- LlamaIndex, a framework dedicated for building RAG systems.

```
%pip install llama-index
%pip install langchain_community
```

Using a model downloaded from LM Studio via CTransformers:

```
%pip install langchain
%pip install langchain_community
%pip install ctransformers
```

For execution : either download LM studio and download the required model. The model will be stored in C drive → users → <username> → .cache→ lm studio

Or download the model using the link :
https://drive.google.com/file/d/1r3X2I47JmfwVHVwWz96dyyZrQ9BL5Kjn/view?usp=sharing

Upload this file in google drive
In Google Colab, connect to Google drive,
Copy the path to the model and assign this path to model_file variable

```
from langchain_community.llms import CTransformers
import time

#It is a local path, it will be different for you
model_file = r"C:\Users\Aiml
cse\.cache\lm-studio\models\TheBloke\Mistral-7B-Instruct-v0.1-GGUF\mistral-7b-instru
ct-v0.1.Q2_K.gguf"

MAX_NEW_TOKENS = 4096
CONTEXT_LENGTH = 4096

def load_llm(max_new_tokens=MAX_NEW_TOKENS,
context_length=CONTEXT_LENGTH, temp=0.0, layers=50):
   llm_config = {
       "temperature": temp,
       'max_new_tokens': max_new_tokens,
       "context_length": context_length,
       'gpu_layers': layers,
   }

   llm = CTransformers(
       model = model_file,
      # model_type = 'mistral',
       config = llm_config
   )

   return llm
```

Try out the following :

```
my_llm = load_llm()
```

```
my_llm("write a poem on Machine")
```

```
response = my_llm.invoke("write a poem on go")
print(response)
```

```
response = my_llm.invoke("AI is going to")
print(response)
```

Create a Prompt :

```
from langchain.chains import LLMChain
from langchain_core.prompts import PromptTemplate

template = """ [INST] Question: {question}

Answer:[/INST]"""

prompt = PromptTemplate.from_template(template)

llm_chain = LLMChain(prompt=prompt, llm=my_llm)

response = llm_chain.invoke("What is AI?")
print(response)
```

```
response = llm_chain.run("What is AI?")
print(response)
```

```
response = llm_chain("What is AI?")
print(response)
```

Take the prompt as input :

```
from langchain import PromptTemplate
from langchain.schema.runnable import RunnablePassthrough
from langchain.schema import StrOutputParser
template = """ [INST] Question: {question}

Answer:[/INST]"""

prompt = PromptTemplate(template=template, input_variables = ['question'])
variable = {"question": RunnablePassthrough()}
# prompt = prompt.format(**variable)
llm_chain = LLMChain(prompt=prompt, llm=my_llm)
llm_chain.run(question = "What is square root of 2")
```

Take the question as input during run-time

```
template = """ [INST] Question: {question}

Answer:[/INST]"""

prompt = PromptTemplate(template=template, input_variables = ['question'])
chain = (
         {"question": RunnablePassthrough()}
         | prompt
         |my_llm
         |StrOutputParser()
   )
query = ""
while query != "quit":
   query = input("Your Query : ")
   output = chain.invoke(query)
   print(output)
```

Using Ollama from Langchain :

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_community.llms.ollama import Ollama

template = """Question: {question}

Answer: Let's think step by step."""

prompt = ChatPromptTemplate.from_template(template)

model = Ollama(model="mistral")

chain = prompt | model

print(chain.invoke({"question": "What is System Maintenance?"}))
```

Using Ollama from llama-index :

```

from llama_index.llms.ollama  import Ollama

llm = Ollama(model="mistral")

response = llm.complete("What is the history of LEGO?")
print(response)
```

Learn more : https://docs.llamaindex.ai/en/stable/examples/llm/ollama/

Using Huggingface :

```
%pip install llama-index-llms-huggingface
%pip install llama-index-llms-huggingface-api
```

There are many ways to interface with LLMs from Hugging Face.

Step I: Log in to Hugging face website and generate Access Tokens.
Profile → Settings → Access Tokens → Create New Token → Choose write (third option after finegrained and read) → give a name to a token → copy the access token

Step II : Multiple ways to connect to Hugging face using the token generated in step I :

Option 1 :

```
import os
HF_TOKEN: Optional[str] = os.getenv("Your_Access_Token_from_Step_I")
```

Option 2 :

```
from huggingface_hub import login
login(token="Your_Access_Token_from_Step_I")
```

Option 3 :

```
HF_TOKEN = "Your_Access_Token_from_Step_I"
!huggingface-cli login --token $HF_TOKEN
```

Step III : Use langchain or llama-index to use LLMs from Huggingface

Using Langchain :
%pip install
%pip install langchain
%pip install langchain_community

Example 1 :

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

access_token = "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"
tokenizer = AutoTokenizer.from_pretrained("google/gemma-2b-it", token=access_token)
model = AutoModelForCausalLM.from_pretrained("google/gemma-2b-it",
device_map="auto",                                    torch_dtype=torch.bfloat16,
token=access_token)

input_text = "Write me a poem about Machine Learning."
input_ids = tokenizer(input_text, return_tensors="pt").to("cuda")

outputs = model.generate(**input_ids)
print(tokenizer.decode(outputs[0]))
```

Why is the output not complete ?

Try the following :

```
input_text = "Write me a poem about cat."
input_ids = tokenizer(input_text, return_tensors="pt").to("cuda")

# Adjust the parameters as needed
outputs = model.generate(**input_ids, max_length=50, num_return_sequences=3,
do_sample=True)

# Decode the outputs
for i, output in enumerate(outputs):
    print(f"Generated text {i + 1}: {tokenizer.decode(output, skip_special_tokens=True)}")
```

Example 2 :

```
from langchain.llms.huggingface_pipeline import
HuggingFacePipeline

hf = HuggingFacePipeline.from_model_id(
    model_id="microsoft/DialoGPT-medium",
task="text-generation", pipeline_kwargs={"max_new_tokens":
200, "pad_token_id": 50256},
)

from langchain.prompts import PromptTemplate

template = """Question: {question}

Answer: Let's think step by step."""
prompt = PromptTemplate.from_template(template)

chain = prompt | hf

question = "What is electroencephalography?"

print(chain.invoke({"question": question}))
```

Example 3 :

```
import torch
from transformers import BitsAndBytesConfig
from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline
from huggingface_hub import login
from langchain.llms import HuggingFacePipeline
from langchain import PromptTemplate, LLMChain

quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
)
login(token="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")

model_4bit = AutoModelForCausalLM.from_pretrained(
"mistralai/Mistral-7B-Instruct-v0.1", device_map="auto"
,quantization_config=quantization_config,)
tokenizer = AutoTokenizer.from_pretrained("mistralai/Mistral-7B-Instruct-v0.1")

pipeline_inst = pipeline(
        "text-generation",
        model=model_4bit,
        tokenizer=tokenizer,
        use_cache=True,
        device_map="auto",
        max_length=2500,
        do_sample=True,
        top_k=5,
        num_return_sequences=1,
        eos_token_id=tokenizer.eos_token_id,
        pad_token_id=tokenizer.eos_token_id,
)
llm = HuggingFacePipeline(pipeline=pipeline_inst)
```

What is the output for the following :

```
print(llm("Once upon a time"))
```

Run the following :

```
template = """<s>[INST] You are an respectful and helpful assistant, respond always be precise, assertive and politely answer in few words conversational english.
Answer the question below:
{question} [/INST] </s>
"""

def generate_response(question):
  prompt = PromptTemplate(template=template, input_variables=["question"])
  llm_chain = LLMChain(prompt=prompt, llm=llm)
  response = llm_chain.run({"question":question})
  return response


generate_response("What is love?")
```

https://www.linkedin.com/pulse/chatbot-7-simple-steps-using-open-source-llm-via-inference-mitra-334of/

1. Downloading the required Packages:

To get started, we will have to install the necessary libraries by running the command below. These include huggingface_hub for accessing Hugging Face's models, transformers for pre-trained model and langchain for LLM utilities.

```
!pip install langchain huggingface_hub transformers
```

2. Importing Libraries and Setting API Key:

Import necessary modules from langchain for working with Hugging Face LLMs and set your Hugging Face API key as an environment variable for secure access. Replace 'YOUR_HUGGINGFACE_API_KEY' with your actual API key from Hugging Face's settings.

```
import os
from langchain import HuggingFaceHub, LLMChain, PromptTemplate
```

3. Selecting a Model and Initializing a langhchain llm object with custom LLMs parameters:

```
model_id = 'mistralai/Mixtral-8x7B-Instruct-v0.1'

custom_llm = HuggingFaceHub(huggingfacehub_api_token=os.environ['HF_API_KEY'],
                       repo_id=model_id,
                       model_kwargs={"temperature":0.5,"max_new_tokens":2000})
```

4. Setting up the Prompt Template:

Define a prompt template for structuring queries to the LLM model, ensuring it knows its role as a helpful assistant. The template is prepared to dynamically include user questions.

```
template = """
As a highly knowledgeable and articulate AI assistant, your primary objective is to
provide concise, accurate,
and helpful responses to user queries. Each answer should be framed with clarity,
aiming to offer insightful
information or solutions.

Consider the user's perspective to ensure your responses are not only informative but
also easy to understand
and apply. Your knowledge spans a wide array of subjects, enabling you to address
queries with depth and precision.

Question: {question}
"""

prompt = PromptTemplate(template=template, input_variables=['question'])
```

5. Create a Langchain LLMChain:

Initialize an LLMChain with the chosen model and prompt setup, enabling verbose output for detailed interaction feedback.

This LLMChain is composed of a PromptTemplate (prompt, the one which you created right above) and a language model (custom_llm). It works by formatting the prompt template with the given input key values (and memory key values, when applicable). This formatted string is then passed to the LLM, which processes it and returns the output. LLMChain is probably the most widely used Chain. There are other options like ConversationalRetrievalQAChain or DocumentStuffChain. Check out the Langchain documentations for more info.

```
customllm_chain = LLMChain(llm=custom_llm,
                  prompt=prompt,
                  verbose=True)
```

6. Executing the Model:

```
print(customllm_chain.run("Who directed the movie 'My neighbor Totoro'?"))
```

Using llama-index :

```
%pip install llama-index
%pip install llama-index-llms-huggingface
%pip install llama-index-llms-huggingface-api
```

```python
import os
from llama_index.llms.huggingface import HuggingFaceLLM
from llama_index.llms.huggingface_api import HuggingFaceInferenceAPI

# We just need a token with read permissions for this demo
HF_TOKEN: Optional[str] = os.getenv("Enter your token")

#This will locally download the model to local Hugging face model cache and runs the model on local machines hardware

locally_run = HuggingFaceLLM(model_name="google/gemma-2b-it")

completion_response = locally_run.complete("To infinity, and")
print(completion_response)

# Run the model remotely on Hugging Face's servers,
# accessed via the Hugging Face Inference API
# Note that using your token will not charge you money,
# the Inference API is free it just has rate limits

remotely_run = HuggingFaceInferenceAPI(
   model_name="google/gemma-2b-it", token=HF_TOKEN
)

completion_response = remotely_run.complete("To infinity, and")
print(completion_response)
```

Front End Creation :

Use : Link 1 : https://docs.streamlit.io/develop/tutorials/llms/llm-quickstart or Link 2 : https://docs.streamlit.io/develop/tutorials/llms/build-conversational-apps to Build a simple GUI using streamlit for a chatbot

Install streamlit.

In visual studio : open terminal and run the following :
$pip install streamlit

Followed by :
$pip install langchain, langchain_community, langchain_core

Open a new Python file (.py extension for example file.py)

Write the code to load the model.

```
import streamlit as st
```

Define a generate_response() function which takes the user query as input.

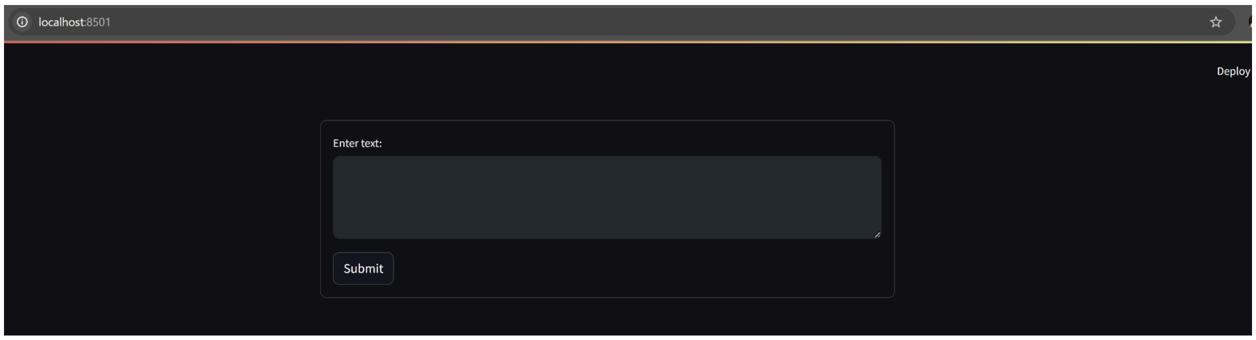
```
def generate_response(input_text):
    #make the prompt, get the model output in a
    # variable called response
    st.info(response)
```

Create simple form with a text area:

```
with st.form("my_form"):
    text = st.text_area(
        "Enter text:"
    )
    submitted = st.form_submit_button("Submit")
    if submitted:
        generate_response(text)
```

Run the python file by going to the terminal (make sure you are in the folder where your file.py is:
$streamlit run ./file.py

Following tab opens up in browser :



You can enter the query, click on submit, you will see the output from LLM.

Exercises :

1) Download Quora Dataset using the link : https://drive.google.com/file/d/1SLzBcwzc48SCW7M7T2Z2QvcSORMCP8bc/view?usp=sharing

   Print the dataset distribution
   If the dataset is imbalanced, make it uniform using LLMs

2) Make a simple sentiment analysis application using LLMs, use streamlit for frontend