L5: Chat with any LLM! 💬

Load your HF API key and relevant Python libraries

In [ ]:

```
import os
import io
import IPython.display
from PIL import Image
import base64
import requests
requests.adapters.DEFAULT_TIMEOUT = 60

from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file
hf_api_key = os.environ['HF_API_KEY']
```

In [ ]:

```
# Helper function
import requests, json
from text_generation import Client

#FalcomLM-instruct endpoint on the text_generation library
client = Client(os.environ['HF_API_FALCOM_BASE'], headers={"Authorization": f"Basi
```

Building an app to chat with any LLM!

Here we'll be using an [Inference Endpoint (https://huggingface.co/inference-endpoints)](https://huggingface.co/inference-endpoints) for `falcon-40b-instruct` , one of best ranking open source LLM on the [🤗 Open LLM Leaderboard (https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard)](https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard).

To run it locally, one can use the [Transformers library (https://huggingface.co/docs/transformers/index)](https://huggingface.co/docs/transformers/index) or the [text-generation-inference (https://github.com/huggingface/text-generation-inference)](https://github.com/huggingface/text-generation-inference)

In [ ]:

```
prompt = "Has math been invented or discovered?"
client.generate(prompt, max_new_tokens=256).generated_text
```

In [ ]:

```
#Back to Lesson 2, time flies!
import gradio as gr
def generate(input, slider):
    output = client.generate(input, max_new_tokens=slider).generated_text
    return output

demo = gr.Interface(fn=generate, inputs=[gr.Textbox(label="Prompt"), gr.Slider(lab
gr.close_all()
demo.launch(share=True, server_port=int(os.environ['PORT1']))
```

`gr.Chatbot()` to the rescue!

In [ ]:

```
import random

def respond(message, chat_history):
        #No LLM here, just respond with a random pre-made message
        bot_message = random.choice(["Tell me more about it",
                                     "Cool, but I'm not interested",
                                     "Hmmmm, ok then"])
        chat_history.append((message, bot_message))
        return "", chat_history

with gr.Blocks() as demo:
    chatbot = gr.Chatbot(height=240) #just to fit the notebook
    msg = gr.Textbox(label="Prompt")
    btn = gr.Button("Submit")
    clear = gr.ClearButton(components=[msg, chatbot], value="Clear console")

    btn.click(respond, inputs=[msg, chatbot], outputs=[msg, chatbot])
    msg.submit(respond, inputs=[msg, chatbot], outputs=[msg, chatbot]) #Press ente
gr.close_all()
demo.launch(share=True, server_port=int(os.environ['PORT2']))
```

In [ ]:

```
def format_chat_prompt(message, chat_history):
    prompt = ""
    for turn in chat_history:
        user_message, bot_message = turn
        prompt = f"{prompt}\nUser: {user_message}\nAssistant: {bot_message}"
    prompt = f"{prompt}\nUser: {message}\nAssistant:"
    return prompt

def respond(message, chat_history):
        formatted_prompt = format_chat_prompt(message, chat_history)
        bot_message = client.generate(formatted_prompt,
                                     max_new_tokens=1024,
                                     stop_sequences=["\nUser:", "<|endoftext|>"]).
        chat_history.append((message, bot_message))
        return "", chat_history

with gr.Blocks() as demo:
    chatbot = gr.Chatbot(height=240) #just to fit the notebook
    msg = gr.Textbox(label="Prompt")
    btn = gr.Button("Submit")
    clear = gr.ClearButton(components=[msg, chatbot], value="Clear console")

    btn.click(respond, inputs=[msg, chatbot], outputs=[msg, chatbot])
    msg.submit(respond, inputs=[msg, chatbot], outputs=[msg, chatbot]) #Press ente
gr.close_all()
demo.launch(share=True, server_port=int(os.environ['PORT3']))
```

Adding other advanced features

In [ ]:

```
def format_chat_prompt(message, chat_history, instruction):
    prompt = f"System:{instruction}"
    for turn in chat_history:
        user_message, bot_message = turn
        prompt = f"{prompt}\nUser: {user_message}\nAssistant: {bot_message}"
    prompt = f"{prompt}\nUser: {message}\nAssistant:"
    return prompt

def respond(message, chat_history, instruction, temperature=0.7):
    prompt = format_chat_prompt(message, chat_history, instruction)
    chat_history = chat_history + [[message, ""]]
    stream = client.generate_stream(prompt,
                                      max_new_tokens=1024,
                                      stop_sequences=["\nUser:", "<|endoftext|>"],
                                      temperature=temperature)
                                      #stop_sequences to not generate the user ans
    acc_text = ""
    #Streaming the tokens
    for idx, response in enumerate(stream):
            text_token = response.token.text

            if response.details:
                return

            if idx == 0 and text_token.startswith(" "):
                text_token = text_token[1:]

            acc_text += text_token
            last_turn = list(chat_history.pop(-1))
            last_turn[-1] += acc_text
            chat_history = chat_history + [last_turn]
            yield "", chat_history
            acc_text = ""

with gr.Blocks() as demo:
    chatbot = gr.Chatbot(height=240) #just to fit the notebook
    msg = gr.Textbox(label="Prompt")
    with gr.Accordion(label="Advanced options",open=False):
        system = gr.Textbox(label="System message", lines=2, value="A conversatior
        temperature = gr.Slider(label="temperature", minimum=0.1, maximum=1, value
    btn = gr.Button("Submit")
    clear = gr.ClearButton(components=[msg, chatbot], value="Clear console")

    btn.click(respond, inputs=[msg, chatbot, system], outputs=[msg, chatbot])
    msg.submit(respond, inputs=[msg, chatbot, system], outputs=[msg, chatbot]) #Pı
gr.close_all()
demo.queue().launch(share=True, server_port=int(os.environ['PORT4']))
```

In [ ]:

```
gr.close_all()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```