Appandix_B_Gradio

February 4, 2025

1 Appendix B: Building a UI Demo for Your LLM as a Chatbot

Welcome to **Appendix B**! In this section, we're going to take your LLM to the next level by building a **chatbot UI** that's not only functional but fun to interact with!

1.0.1 What we'll do:

1. **Interact with Falcon 7B**:
   - We'll start by using the **Hugging Face API** to connect to the **Falcon 7B model** , a powerful language model that can generate intelligent, context-aware responses. You'll get to chat with it just like any other AI assistant!
2. **Build Multi-Turn Conversations**
   - We'll begin with a simple one-turn chatbot. But that's just the start! We'll gradually build it up to handle **multi-turn conversations**, so your chatbot can remember context and keep the chat flowing naturally.
3. **Add a Voice with Kokoro**
   - Next, we'll spice things up by integrating **Kokoro**, a **text-to-speech model**, to give your AI assistant a voice. No more just reading responses! Your chatbot will speak , making interactions even more immersive and fun!

By the end of this notebook, you'll have a **fully functional chatbot** that can talk and chat, with the ability to hold engaging, multi-turn conversations. Let's dive in and bring your AI assistant to life!

Installing Required Libraries To get started, we'll need to install some essential libraries:

transformers for interacting with Hugging Face models gradio for building the chatbot UI text_generation for managing text generation tasks huggingface_hub for accessing models and datasets from Hugging Face python-dotenv for securely handling environment variables Run the following command to install these libraries:

[1]:
```
! pip install transformers gradio text_generation huggingface_hub python-dotenv
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-
packages (4.47.1)
Requirement already satisfied: gradio in /usr/local/lib/python3.11/dist-packages
(5.14.0)
Requirement already satisfied: text_generation in
/usr/local/lib/python3.11/dist-packages (0.7.0)
```

```
Requirement already satisfied: huggingface_hub in
/usr/local/lib/python3.11/dist-packages (0.27.1)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.11/dist-
packages (1.0.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-
packages (from transformers) (3.17.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-
packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-
packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-
packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.21.0)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.5.2)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-
packages (from transformers) (4.67.1)
Requirement already satisfied: aiofiles<24.0,>=22.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (23.2.1)
Requirement already satisfied: anyio<5.0,>=3.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (3.7.1)
Requirement already satisfied: fastapi<1.0,>=0.115.2 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.115.8)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.11/dist-packages
(from gradio) (0.5.0)
Requirement already satisfied: gradio-client==1.7.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (1.7.0)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-
packages (from gradio) (0.28.1)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-
packages (from gradio) (3.1.5)
Requirement already satisfied: markupsafe~=2.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (2.1.5)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-
packages (from gradio) (3.10.15)
Requirement already satisfied: pandas<3.0,>=1.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (11.1.0)
Requirement already satisfied: pydantic>=2.0 in /usr/local/lib/python3.11/dist-
packages (from gradio) (2.10.6)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages
(from gradio) (0.25.1)
```

```
Requirement already satisfied: python-multipart>=0.0.18 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.0.20)
Requirement already satisfied: ruff>=0.9.3 in /usr/local/lib/python3.11/dist-
packages (from gradio) (0.9.4)
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.1.6)
Requirement already satisfied: semantic-version~=2.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (2.10.0)
Requirement already satisfied: starlette<1.0,>=0.40.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.45.3)
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.13.2)
Requirement already satisfied: typer<1.0,>=0.12 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.15.1)
Requirement already satisfied: typing-extensions~=4.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (4.12.2)
Requirement already satisfied: uvicorn>=0.14.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.34.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages
(from gradio-client==1.7.0->gradio) (2024.10.0)
Requirement already satisfied: websockets<15.0,>=10.0 in
/usr/local/lib/python3.11/dist-packages (from gradio-client==1.7.0->gradio)
(14.2)
Requirement already satisfied: aiohttp<4.0,>=3.8 in
/usr/local/lib/python3.11/dist-packages (from text_generation) (3.11.11)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0,>=3.8->text_generation) (2.4.4)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0,>=3.8->text_generation) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-
packages (from aiohttp<4.0,>=3.8->text_generation) (25.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0,>=3.8->text_generation) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0,>=3.8->text_generation) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0,>=3.8->text_generation) (0.2.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0,>=3.8->text_generation) (1.18.3)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-
packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-
```

```
packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-
packages (from httpx>=0.24.1->gradio) (2024.12.14)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-
packages (from httpx>=0.24.1->gradio) (1.0.7)
Requirement already satisfied: h11<0.15,>=0.13 in
/usr/local/lib/python3.11/dist-packages (from
httpcore==1.*->httpx>=0.24.1->gradio) (0.14.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas<3.0,>=1.0->gradio) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas<3.0,>=1.0->gradio) (2025.1)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.0->gradio) (0.7.0)
Requirement already satisfied: pydantic-core==2.27.2 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.0->gradio) (2.27.2)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-
packages (from typer<1.0,>=0.12->gradio) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in
/usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-
packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0->gradio) (1.17.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from
rich>=10.11.0->typer<1.0,>=0.12->gradio) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from
rich>=10.11.0->typer<1.0,>=0.12->gradio) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-
packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0,>=0.12->gradio)
(0.1.2)
```

Logging into Hugging Face Before interacting with Hugging Face models, we need to log in to the Hugging Face Hub. This will allow us to access and utilize the models and datasets hosted on their platform. Run the code below to log in:

[2]:
```
from huggingface_hub import notebook_login

notebook_login()
```

```
VBox(children=(HTML(value='<center> <img\nsrc=https://huggingface.co/front/
↪assets/huggingface_logo-noborder.sv…
```

Storing and Verifying the Hugging Face API Key To interact with Hugging Face's API, we'll store the API key securely in environment variables. This way, we can authenticate our requests without hardcoding sensitive information.

This code will:

Load your .env file using python-dotenv (if it's present) Prompt you to enter your Hugging Face API key Store the API key in an environment variable for future use Verify that the API key has been successfully stored

[1]:
```
import os
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file
hf_api_key = input("Enter your Hugging Face API Key: ")
os.environ["HF_API_KEY"] = hf_api_key  # Store it in environment

# Check if it is stored correctly
print("API Key stored successfully!" if "HF_API_KEY" in os.environ else "API␣
↪Key not found!")
```

```
API Key stored successfully!
```

Interacting with Falcon-7B-Instruct Model Now that we have set up the Hugging Face API key, we can interact with the Falcon-7B-Instruct model (or any other Hugging Face model) to generate text. In this step, we'll initialize a client to communicate with the model and make a request to generate some text based on a prompt.

This code will:

Retrieve the Hugging Face API key from the environment variable Set up the Falcon-7B-Instruct model endpoint for text generation Initialize the Client with the authentication headers, ensuring we're connected to Hugging Face's API Generate text by sending a prompt ("Tell me a joke.") to the model Print the generated response Run the code below to generate text with the Falcon-7B model:

[5]:
```
from text_generation import Client

# Make sure the API key is set in the environment
hf_api_key = os.getenv("HF_API_KEY")  # Retrieve stored API key

# Falcon-7B-Instruct Endpoint (or replace with another HF model)
HF_API_FALCOM_BASE = "https://api-inference.huggingface.co/models/tiiuae/
↪falcon-7b-instruct"

# Initialize the client with correct authentication
client = Client(HF_API_FALCOM_BASE, headers={"Authorization": f"Bearer␣
↪{hf_api_key}"}, timeout=120)
```

```
# Generate text using FalconLM
response = client.generate("Tell me a joke.", max_new_tokens=50)

print("  Generated Text:", response)
```

```
 Generated Text: generated_text='\nWhy did the tomato turn red? Because it saw
the salad dressing!'
details=Details(finish_reason=<FinishReason.EndOfSequenceToken: 'eos_token'>,
generated_tokens=16, seed=None, prefill=[], tokens=[Token(id=193, text='\n',
logprob=-0.011985779, special=False), Token(id=4479, text='Why',
logprob=-0.87109375, special=False), Token(id=826, text=' did',
logprob=-0.77734375, special=False), Token(id=248, text=' the',
logprob=-0.00504303, special=False), Token(id=16604, text=' tomato',
logprob=-0.7138672, special=False), Token(id=1411, text=' turn',
logprob=-0.028640747, special=False), Token(id=2400, text=' red',
logprob=-0.006298065, special=False), Token(id=42, text='?',
logprob=-0.0036525726, special=False), Token(id=4790, text=' Because',
logprob=-0.15026855, special=False), Token(id=334, text=' it',
logprob=-0.00096416473, special=False), Token(id=2759, text=' saw',
logprob=-0.0138549805, special=False), Token(id=248, text=' the',
logprob=-0.0025596619, special=False), Token(id=12830, text=' salad',
logprob=-0.0047302246, special=False), Token(id=15529, text=' dressing',
logprob=-0.0023555756, special=False), Token(id=12, text='!',
logprob=-0.036102295, special=False), Token(id=11, text='<|endoftext|>',
logprob=-0.024353027, special=True)], top_tokens=None, best_of_sequences=None)
```

Lets try another prompt

[6]:
```
prompt = "So is math get discoverd or invented?"
response = client.generate(prompt, max_new_tokens=50)
print("  Generated Text:", response)
```

```
 Generated Text: generated_text='\nMath is a discovery, not an invention. It is
a set of logical rules and procedures that have been discovered and developed
over time by humans.'
details=Details(finish_reason=<FinishReason.EndOfSequenceToken: 'eos_token'>,
generated_tokens=31, seed=None, prefill=[], tokens=[Token(id=193, text='\n',
logprob=-0.012840271, special=False), Token(id=25864, text='Math',
logprob=-0.1361084, special=False), Token(id=304, text=' is',
logprob=-0.4074707, special=False), Token(id=241, text=' a', logprob=-1.7392578,
special=False), Token(id=12070, text=' discovery', logprob=-2.1835938,
special=False), Token(id=23, text=',', logprob=-0.95166016, special=False),
Token(id=416, text=' not', logprob=-0.46020508, special=False), Token(id=267,
text=' an', logprob=-0.58496094, special=False), Token(id=13217, text='
invention', logprob=-0.002986908, special=False), Token(id=25, text='.',
logprob=-0.08300781, special=False), Token(id=605, text=' It',
logprob=-0.68359375, special=False), Token(id=304, text=' is',
logprob=-1.1044922, special=False), Token(id=241, text=' a', logprob=-1.0644531,
special=False), Token(id=889, text=' set', logprob=-1.6826172, special=False),
```

```
Token(id=275, text=' of', logprob=-0.0016498566, special=False), Token(id=14813,
text=' logical', logprob=-1.2578125, special=False), Token(id=4213, text='
rules', logprob=-0.9213867, special=False), Token(id=273, text=' and',
logprob=-0.39624023, special=False), Token(id=7547, text=' procedures',
logprob=-1.6875, special=False), Token(id=325, text=' that',
logprob=-0.42944336, special=False), Token(id=413, text=' have',
logprob=-1.5322266, special=False), Token(id=650, text=' been',
logprob=-0.04269409, special=False), Token(id=6524, text=' discovered',
logprob=-0.8540039, special=False), Token(id=273, text=' and',
logprob=-1.0830078, special=False), Token(id=4027, text=' developed',
logprob=-0.9580078, special=False), Token(id=648, text=' over',
logprob=-0.32226562, special=False), Token(id=601, text=' time',
logprob=-0.484375, special=False), Token(id=431, text=' by', logprob=-1.3222656,
special=False), Token(id=7305, text=' humans', logprob=-0.90722656,
special=False), Token(id=25, text='.', logprob=-0.9501953, special=False),
Token(id=11, text='<|endoftext|>', logprob=-0.7363281, special=True)],
top_tokens=None, best_of_sequences=None)
```

Cleaning the output wuth less verbosity

[7]:
```
prompt = "Has math been invented or discovered?"
client.generate(prompt, max_new_tokens=256).generated_text
```

[7]:
```
'\nMath has been discovered, not invented. It is a system of rules and formulas
that are used to describe the natural world and its behavior.'
```

Creating the Chatbot UI with Gradio Let's make things interactive! Using Gradio, we'll create a simple user interface where users can input a prompt, and the model will generate a response. This makes it easy to interact with our Falcon-7B-Instruct model in real-time.

[22]:
```
import gradio as gr

def generate(prompt):
    response = client.generate(prompt, max_new_tokens=256).generated_text
    return response

gr.Interface(generate, "textbox", "textbox").launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://5911ef6f27b876de44.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

<IPython.core.display.HTML object>

[22]:

Adding More Customization with Gradio: Max Tokens Slider In this step, we'll enhance the user interface by adding more customization. Users will now be able to input not only a prompt but also control the number of new tokens generated in the response using a slider. This gives more control over the length of the output text.

This code will:

Define the generate function to accept both a prompt and max_new_tokens as inputs Use a Gradio slider to allow users to control the number of tokens generated for each response Set up the Gradio interface with two inputs: a textbox for the prompt and a slider for the max tokens, and one output: a textbox to display the generated text Launch the interface with the option to share it publicly

[25]:
```
def generate(prompt, max_new_tokens):
    response = client.generate(
        prompt=prompt,
        max_new_tokens=int(max_new_tokens)
    ).generated_text
    return response

demo = gr.Interface(
    fn=generate,
    inputs=[
        gr.Textbox(label="Prompt"),
        gr.Slider(label="Max new tokens", value=20, minimum=1, maximum=1024)
    ],
    outputs=gr.Textbox(label="Completion")
)

gr.close_all()
demo.launch(share=True)
```

```
Closing server running on port: 7860
Closing server running on port: 7864
Closing server running on port: 7860
Closing server running on port: 7861
Closing server running on port: 7864
Closing server running on port: 7864
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://4326df173473a1cc8b.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

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

[25]:

Having more control in the output b controlling the generation parameters , we dicussed these before on the Infrence notebpook (notebook 6.1)

[26]:
```
def generate_text(prompt, temperature, top_k, top_p, max_new_tokens):
    generated_text = client.generate(
        prompt,
        max_new_tokens=int(max_new_tokens),  # Convert to int for safety
        temperature=temperature,
        top_k=int(top_k),
        top_p=top_p
    ).generated_text
    return generated_text
```

Now we add them to the gradio UI using Slider method

[27]:
```
# Create Gradio UI
iface = gr.Interface(
    fn=generate_text,  # Function to call
    inputs=[
        gr.Textbox(label="Enter your prompt"),
        gr.Slider(0.1, 1.5, value=1.0, label="Temperature"),
        gr.Slider(1, 100, value=50, step=1, label="Top-k"),
        gr.Slider(0.0, 1.0, value=0.9, label="Top-p"),
        gr.Slider(10, 512, value=256, step=1, label="Max New Tokens")  # Added␣
 ↪slider
    ],
    outputs=gr.Textbox(label="Generated Text"),
    title="Text Generation App",
    description="Adjust parameters to control text generation randomness,␣
 ↪diversity, and length."
)

# Launch app
iface.launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically
setting `share=True` (you can turn this off by setting `share=False` in
`launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://eab1323787566f5dac.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

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

[27]:

Introducing Multi-Turn Conversations with Gradio In this section, we're enhancing our chatbot to handle multi-turn conversations. Instead of just responding once, the chatbot will keep track of the conversation history and provide context-aware responses across multiple turns. This will create a more interactive and dynamic conversation flow, where the bot can reply intelligently based on what was said before.

We will simulate this with random pre-made responses for now, but you can imagine how a powerful LLM could make this much more sophisticated!

This code will:

Define a respond function that keeps track of the conversation history, ensuring multi-turn interactions. The bot responds with random pre-set messages based on each new input Set up a Gradio chatbot UI to display the conversation in a chat-like format Use a Textbox for input, a Button to submit the prompt, and a Clear Button to reset the conversation Allow users to press "Enter" to submit messages, making the conversation flow more naturally

[28]:
```
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␣
 ↪enter to submit

gr.close_all()
demo.launch(share=True)
```

```
/usr/local/lib/python3.11/dist-packages/gradio/components/chatbot.py:282:
UserWarning: You have not specified a value for the `type` parameter. Defaulting
to the 'tuples' format for chatbot messages, but this is deprecated and will be
removed in a future version of Gradio. Please set type='messages' instead, which
uses openai-style dictionaries with 'role' and 'content' keys.
  warnings.warn(

Closing server running on port: 7865
```

```
Closing server running on port: 7860
Closing server running on port: 7864
Closing server running on port: 7860
Closing server running on port: 7861
Closing server running on port: 7864
Closing server running on port: 7864
Closing server running on port: 7864
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://9e959423e7854c998f.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

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

[28]:

Adding Multi-Turn Conversations with an LLM In this section, we will upgrade our chatbot to handle multi-turn conversations using an actual language model. Instead of random responses, the bot will now generate context-aware replies by referencing previous conversation history. This is achieved by formatting the conversation and sending it to the model.

The key addition here is the function format_chat_prompt, which prepares the conversation history so that the model can understand the context of the ongoing chat. Based on this, the model generates relevant responses, making the interaction feel more natural and intelligent.

This code will:

Format the chat history to include previous messages from both the user and the assistant Send this formatted conversation as a prompt to the Falcon-7B model to generate a coherent response based on the entire conversation history Use Gradio to create an interactive interface with a Textbox for input and a Chatbot display for the ongoing conversation Enable users to interact with the chatbot over multiple turns while maintaining context throughout the conversation Run the code below to try out the upgraded chatbot that uses an LLM to generate thoughtful, context-aware responses:

[29]:
```
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|>"]).generated_text
        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␣
 ↪enter to submit

gr.close_all()
demo.launch(share=True)
```

```
/usr/local/lib/python3.11/dist-packages/gradio/components/chatbot.py:282:
UserWarning: You have not specified a value for the `type` parameter. Defaulting
to the 'tuples' format for chatbot messages, but this is deprecated and will be
removed in a future version of Gradio. Please set type='messages' instead, which
uses openai-style dictionaries with 'role' and 'content' keys.
  warnings.warn(

Closing server running on port: 7865
Closing server running on port: 7860
Closing server running on port: 7864
Closing server running on port: 7860
Closing server running on port: 7861
Closing server running on port: 7864
Closing server running on port: 7864
Closing server running on port: 7864
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://3121e8d33726061366.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)

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

[29]:

Put it all together:

[30]:
```
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, temperature, top_k, top_p, max_new_tokens):
    formatted_prompt = format_chat_prompt(message, chat_history)
    bot_message = client.generate(
        formatted_prompt,
        max_new_tokens=int(max_new_tokens),
        temperature=temperature,
        top_k=int(top_k),
        top_p=top_p,
        stop_sequences=["\nUser:", "<|endoftext|>"]
    ).generated_text

    chat_history.append((message, bot_message))
    return "", chat_history

with gr.Blocks() as demo:
    gr.Markdown("##  AI Chatbot with Adjustable Parameters")

    chatbot = gr.Chatbot(height=240)
    msg = gr.Textbox(label="Prompt")

    # Parameter sliders
    temperature = gr.Slider(0.1, 1.5, value=1.0, label="Temperature")
    top_k = gr.Slider(1, 100, value=50, step=1, label="Top-k")
    top_p = gr.Slider(0.0, 1.0, value=0.9, label="Top-p")
    max_new_tokens = gr.Slider(10, 1024, value=256, step=1, label="Max New␣
 ↪Tokens")

    btn = gr.Button("Submit")
    clear = gr.ClearButton(components=[msg, chatbot], value="Clear Chat")

    # Connect everything
    btn.click(respond, inputs=[msg, chatbot, temperature, top_k, top_p,␣
 ↪max_new_tokens], outputs=[msg, chatbot])
    msg.submit(respond, inputs=[msg, chatbot, temperature, top_k, top_p,␣
 ↪max_new_tokens], outputs=[msg, chatbot])

gr.close_all()
demo.launch(share=True)
```

```
/usr/local/lib/python3.11/dist-packages/gradio/components/chatbot.py:282:
UserWarning: You have not specified a value for the `type` parameter. Defaulting
```

```
to the 'tuples' format for chatbot messages, but this is deprecated and will be
removed in a future version of Gradio. Please set type='messages' instead, which
uses openai-style dictionaries with 'role' and 'content' keys.
  warnings.warn(

Closing server running on port: 7865
Closing server running on port: 7860
Closing server running on port: 7864
Closing server running on port: 7860
Closing server running on port: 7861
Closing server running on port: 7864
Closing server running on port: 7864
Closing server running on port: 7864
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://c0aaddae81fdc6353f.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)

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

[30]:

More refined UI layout and styling, with improvements to make the interface more user-friendly and visually appealing.

[31]:
```python
import gradio as gr

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, temperature, top_k, top_p, max_new_tokens):
    formatted_prompt = format_chat_prompt(message, chat_history)
    bot_message = client.generate(
        formatted_prompt,
        max_new_tokens=int(max_new_tokens),
        temperature=temperature,
        top_k=int(top_k),
        top_p=top_p,
        stop_sequences=["\nUser:", "<|endoftext|>"]
    ).generated_text
```

```
    chat_history.append((message, bot_message))
    return "", chat_history

with gr.Blocks(css="body {background-color: #f7f7f7;}") as demo:
    gr.Markdown("#  AI Chatbot with Adjustable Parameters")

    with gr.Row():  # Create a two-column layout
        with gr.Column(scale=1):  # Left column for input and parameters
            msg = gr.Textbox(label="Enter Your Prompt")

            # Sliders for parameters
            temperature = gr.Slider(0.1, 1.5, value=1.0, label="Temperature")
            top_k = gr.Slider(1, 100, value=50, step=1, label="Top-k")
            top_p = gr.Slider(0.0, 1.0, value=0.9, label="Top-p")
            max_new_tokens = gr.Slider(10, 1024, value=256, step=1, label="Max␣
 ↪New Tokens")

            btn = gr.Button("Generate")
            clear = gr.ClearButton(components=[msg], value="Clear Input")

        with gr.Column(scale=2):  # Right column for chatbot UI
            chatbot = gr.Chatbot(height=500)
            clear_chat = gr.ClearButton(components=[chatbot], value="Clear␣
 ↪Chat")

    # Button & input triggers response
    btn.click(respond, inputs=[msg, chatbot, temperature, top_k, top_p,␣
 ↪max_new_tokens], outputs=[msg, chatbot])
    msg.submit(respond, inputs=[msg, chatbot, temperature, top_k, top_p,␣
 ↪max_new_tokens], outputs=[msg, chatbot])

gr.close_all()
demo.launch(share=True)
```

```
/usr/local/lib/python3.11/dist-packages/gradio/components/chatbot.py:282:
UserWarning: You have not specified a value for the `type` parameter. Defaulting
to the 'tuples' format for chatbot messages, but this is deprecated and will be
removed in a future version of Gradio. Please set type='messages' instead, which
uses openai-style dictionaries with 'role' and 'content' keys.
  warnings.warn(
```

```
Closing server running on port: 7865
Closing server running on port: 7860
Closing server running on port: 7864
Closing server running on port: 7860
Closing server running on port: 7861
Closing server running on port: 7864
Closing server running on port: 7864
```

```
Closing server running on port: 7864
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://913efb5957d3c44b7e.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

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

[31]:

Now instead of waiting for the model to response we can see the stream in real time:

[32]:
```
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
```

[33]:
```
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␣
 ↪answer
    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 = ""
```

[35]:
```
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␣
 ↪conversation between a user and an LLM-based AI assistant. The assistant␣
 ↪gives helpful and honest answers.")
        temperature = gr.Slider(label="temperature", minimum=0.1, maximum=1,␣
 ↪value=0.7, step=0.1)
    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])␣
 ↪#Press enter to submit

gr.close_all()
demo.queue().launch(share=True)
```

```
/usr/local/lib/python3.11/dist-packages/gradio/components/chatbot.py:282:
UserWarning: You have not specified a value for the `type` parameter. Defaulting
to the 'tuples' format for chatbot messages, but this is deprecated and will be
removed in a future version of Gradio. Please set type='messages' instead, which
uses openai-style dictionaries with 'role' and 'content' keys.
  warnings.warn(

Closing server running on port: 7865
Closing server running on port: 7860
Closing server running on port: 7864
Closing server running on port: 7860
Closing server running on port: 7861
Closing server running on port: 7864
Closing server running on port: 7864
Closing server running on port: 7864
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://1e842090a832e62e82.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)

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

[35]:

[8]:
```
import gradio as gr

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, top_k=50,␣
 ↪top_p=0.9, max_new_tokens=256):
    prompt = format_chat_prompt(message, chat_history, instruction)
    chat_history = chat_history + [[message, ""]]
    stream = client.generate_stream(prompt,
                                    max_new_tokens=max_new_tokens,
                                    stop_sequences=["\nUser:",␣
 ↪"<|endoftext|>"],
                                    temperature=temperature,
                                    top_k=top_k,
                                    top_p=top_p)
    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)  # Adjust the chatbot height
    msg = gr.Textbox(label="Prompt")

    with gr.Accordion(label="Advanced options", open=False):
        system = gr.Textbox(label="System message", lines=2, value="A␣
 ↪conversation between a user and an LLM-based AI assistant. The assistant␣
 ↪gives helpful and honest answers.")
```

```
        temperature = gr.Slider(label="Temperature", minimum=0.1, maximum=1,␣
↪value=0.7, step=0.1)
        top_k = gr.Slider(label="Top-k", minimum=1, maximum=100, value=50,␣
↪step=1)
        top_p = gr.Slider(label="Top-p", minimum=0.0, maximum=1.0, value=0.9,␣
↪step=0.01)
        max_new_tokens = gr.Slider(label="Max New Tokens", minimum=10,␣
↪maximum=1024, value=256, step=1)

    btn = gr.Button("Submit")
    clear = gr.ClearButton(components=[msg, chatbot], value="Clear console")

    # Trigger response generation
    btn.click(respond, inputs=[msg, chatbot, system, temperature, top_k, top_p,␣
↪max_new_tokens], outputs=[msg, chatbot])
    msg.submit(respond, inputs=[msg, chatbot, system, temperature, top_k,␣
↪top_p, max_new_tokens], outputs=[msg, chatbot])  # Press enter to submit

gr.close_all()
demo.queue().launch(share=True)
```

```
/usr/local/lib/python3.11/dist-packages/gradio/components/chatbot.py:282:
UserWarning: You have not specified a value for the `type` parameter. Defaulting
to the 'tuples' format for chatbot messages, but this is deprecated and will be
removed in a future version of Gradio. Please set type='messages' instead, which
uses openai-style dictionaries with 'role' and 'content' keys.
  warnings.warn(

Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://1c33187a738a122ca7.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

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

[8]:

Lets move on the voice part

[9]:
```
from transformers.utils import logging

logging.set_verbosity_error()
```

This script demonstrates how to generate text-to-speech (TTS) audio using the kokoro package. It installs necessary dependencies, initializes a TTS pipeline for multiple languages, and generates speech from a provided text. The script displays the generated audio and saves each chunk as a

.wav file. The user can adjust the language and voice, and the output will be a set of audio files for each generated segment of speech.

1.1 got this from the model HF page

[10]:
```
# 1 Install kokoro
!pip install -q kokoro>=0.3.4 soundfile
# 2 Install espeak, used for English OOD fallback and some non-English␣
 ↪languages
!apt-get -qq -y install espeak-ng > /dev/null 2>&1
#    'e' => Spanish es
#    'f' => French fr-fr
#    'h' => Hindi hi
#    'i' => Italian it
#    'p' => Brazilian Portuguese pt-br

# 3 Initalize a pipeline
from kokoro import KPipeline
from IPython.display import display, Audio
import soundfile as sf
#    'a' => American English,    'b' => British English
#    'j' => Japanese: pip install misaki[ja]
#    'z' => Mandarin Chinese: pip install misaki[zh]
pipeline = KPipeline(lang_code='a') # <= make sure lang_code matches voice

# This text is for demonstration purposes only, unseen during training
text = '''
The sky above the port was the color of television, tuned to a dead channel.
"It's not like I'm using," Case heard someone say, as he shouldered his way␣
 ↪through the crowd around the door of the Chat. "It's like my body's␣
 ↪developed this massive drug deficiency."
It was a Sprawl voice and a Sprawl joke. The Chatsubo was a bar for␣
 ↪professional expatriates; you could drink there for a week and never hear␣
 ↪two words in Japanese.
```

```
These were to have an enormous impact, not only because they were associated␣
 ↪with Constantine, but also because, as in so many other areas, the decisions␣
 ↪taken by Constantine (or in his name) were to have great significance for␣
 ↪centuries to come. One of the main issues was the shape that Christian␣
 ↪churches were to take, since there was not, apparently, a tradition of␣
 ↪monumental church buildings when Constantine decided to help the Christian␣
 ↪church build a series of truly spectacular structures. The main form that␣
 ↪these churches took was that of the basilica, a multipurpose rectangular␣
 ↪structure, based ultimately on the earlier Greek stoa, which could be found␣
 ↪in most of the great cities of the empire. Christianity, unlike classical␣
 ↪polytheism, needed a large interior space for the celebration of its␣
 ↪religious services, and the basilica aptly filled that need. We naturally do␣
 ↪not know the degree to which the emperor was involved in the design of new␣
 ↪churches, but it is tempting to connect this with the secular basilica that␣
 ↪Constantine completed in the Roman forum (the so-called Basilica of␣
 ↪Maxentius) and the one he probably built in Trier, in connection with his␣
 ↪residence in the city at a time when he was still caesar.

[Kokoro](/k Okə O/) is an open-weight TTS model with 82 million parameters.␣
 ↪Despite its lightweight architecture, it delivers comparable quality to␣
 ↪larger models while being significantly faster and more cost-efficient. With␣
 ↪Apache-licensed weights, [Kokoro](/k Okə O/) can be deployed anywhere from␣
 ↪production environments to personal projects.
'''
# text = '                                                    '
# text = '                                                '
# text = 'Los partidos políticos tradicionales compiten con los populismos y␣
 ↪los movimientos asamblearios.'
# text = 'Le dromadaire resplendissant déambulait tranquillement dans les␣
 ↪méandres en mastiquant de petites feuilles vernissées.'
# text = '                              ,                          '
# text = "Allora cominciava l'insonnia, o un dormiveglia peggiore␣
 ↪dell'insonnia, che talvolta assumeva i caratteri dell'incubo."
# text = 'Elabora relatórios de acompanhamento cronológico para as diferentes␣
 ↪unidades do Departamento que propõem contratos.'

# 4  Generate, display, and save audio files in a loop.
generator = pipeline(
    text, voice='af_heart', # <= change voice here
    speed=1, split_pattern=r'\n+'
)
for i, (gs, ps, audio) in enumerate(generator):
    print(i)  # i => index
    print(gs) # gs => graphemes/text
    print(ps) # ps => phonemes
    display(Audio(data=audio, rate=24000, autoplay=i==0))
```

```
    sf.write(f'{i}.wav', audio, 24000) # save each audio file
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/rnn.py:123:
UserWarning: dropout option adds dropout after all but last recurrent layer, so
non-zero dropout expects num_layers greater than 1, but got dropout=0.2 and
num_layers=1
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torch/nn/utils/weight_norm.py:143:
FutureWarning: `torch.nn.utils.weight_norm` is deprecated in favor of
`torch.nn.utils.parametrizations.weight_norm`.
  WeightNorm.apply(module, name, dim)
```

```
0
The sky above the port was the color of television, tuned to a dead channel.
ðə sk I əb v ðə p  t w z ðə k lə  v t ləv  ən, t und tə  d d  æn l.
```

```
<IPython.lib.display.Audio object>
```

```
1
"It's not like I'm using," Case heard someone say, as he shouldered his way
through the crowd around the door of the Chat. "It's like my body's developed
this massive drug deficiency."
" ts n t l Ik  Im j uz ŋ," k As h  d s mw n s A, æz hi  Oldə d h z w A  u
ðə k Wd ə Wnd ðə d  v ðə  æt. " ts l Ik mI b diz dəv ləpt ð s m æs v
d  dəf  ənsi."
```

```
<IPython.lib.display.Audio object>
```

```
2
It was a Sprawl voice and a Sprawl joke. The Chatsubo was a bar for professional
expatriates; you could drink there for a week and never hear two words in
Japanese.
 t w z  sp  l v Ys ænd  sp  l  Ok. ðə  æts ubO w z  b  f  p əf  nəl
 ksp At iəts; ju k d d  ŋk ð  f  w ik ænd n və  h  t u w  dz  n
 æpən iz.
```

```
<IPython.lib.display.Audio object>
```

```
3
These were to have an enormous impact, not only because they were associated
with Constantine, but also because, as in so many other areas, the decisions
```

taken by Constantine (or in his name) were to have great significance for centuries to come. One of the main issues was the shape that Christian churches were to take, since there was not, apparently, a tradition of monumental church buildings when Constantine decided to help the Christian church build a series of truly spectacular structures.
ð iz w tə hæv n n məs mp ækt, n t Onli bək z ðA w əs Osi AT d w ð k nstənt in, b t lsO bək z, æz n s O m ni ðə iəz, ðə dəs nz t Akən bI k nstənt in (n h z n Am) w tə hæv At sə n fək ns f s n ə iz tə k m. w n v ðə m An juz w z ðə Ap ðæt k s ən z w tə t Ak, s ns ð w z n t, əp əntli, t əd ən v m njəm nt l b ld ŋz w n k nstənt in dəs Id d tə h lp ðə k s ən b ld s iz v t uli sp kt ækjələ st k ə z.

<IPython.lib.display.Audio object>

4
The main form that these churches took was that of the basilica, a multipurpose rectangular structure, based ultimately on the earlier Greek stoa, which could be found in most of the great cities of the empire. Christianity, unlike classical polytheism, needed a large interior space for the celebration of its religious services, and the basilica aptly filled that need.
ðə m An f m ðæt ðiz z t k w z ðæt v ðə bəs ləkə, m ltip pəs kt æŋ jələ st k ə , b Ast ltəmətli n ði liə ik st Oə, w k d bi f Wnd n m Ost v ðə At s Tiz v ði mp Iə . k s i ænəTi, nl Ik kl æsək l p li i zəm, n id d l nt iə sp As f ðə s ləb A ən v ts əl əs s vəs z, ænd ðə bəs ləkə æptli f ld ðæt n id.

<IPython.lib.display.Audio object>

5
We naturally do not know the degree to which the emperor was involved in the design of new churches, but it is tempting to connect this with the secular basilica that Constantine completed in the Roman forum (the so-called Basilica of Maxentius) and the one he probably built in Trier, in connection with his residence in the city at a time when he was still caesar.
w i n æ ə əli d u n t n O ðə də i tə w ði mpə ə w z nv lvd n ðə dəz In v n u z, b t t z t mpt ŋ tə kən kt ð s w ð ðə s kjələ bəs ləkə ðæt k nstənt in kəmpl iT d n ðə Omən f əm (ðə s Ok ld bəs ləkə v mæks ntiəs) ænd ðə w n hi p bəbli b lt n t , n kən k ən w ð h z zəd ns n ðə s Ti æt t Im w n hi w z st l s izə .

<IPython.lib.display.Audio object>

6
Kokoro is an open-weight TTS model with 82 million parameters. Despite its lightweight architecture, it delivers comparable quality to larger models while being significantly faster and more cost-efficient. With Apache-licensed weights, Kokoro can be deployed anywhere from production environments to personal projects.
k Okə O z n Op nw At t it i s m d l w ð ATi t u m lj n pə æməTə z.
dəsp It ts l Itw At kət k ə , t dəl və z k mpə əb l kw ləTi tə l ə

```
m d lz w Il b i ŋ sə n fəkəntli f æstə  ænd m   k stəf  ənt. w ð
əp æ il Is nst w Ats, k Okə O kæn bi dəpl Yd  niw   f m p əd k ən
ənv I ənm nts tə p  s nəl p   kts.
```

```
<IPython.lib.display.Audio object>
```

To combine the two pipelines — one for text generation using the Falcon-7B-Instruct model and the other for voice assistance using Kokoro — we will create a unified workflow where:

Text Generation: The input prompt is passed to the Falcon-7B-Instruct model to generate a response. Text-to-Speech (TTS): The generated response is then converted into speech using the Kokoro TTS system. Voice Assistance: The speech is played back to the user while also displaying the text. Here's how you can combine the pipelines into a single flow:

[21]:
```
import os
import gradio as gr
from text_generation import Client
from kokoro import KPipeline
import soundfile as sf
from IPython.display import display, Audio

# 1 Initialize the Falcon-7B-Instruct Model Client
hf_api_key = os.getenv("HF_API_KEY")  # Make sure you set the HF_API_KEY in␣
 ↪your environment
HF_API_FALCOM_BASE = "https://api-inference.huggingface.co/models/tiiuae/
 ↪falcon-7b-instruct"
client = Client(HF_API_FALCOM_BASE, headers={"Authorization": f"Bearer␣
 ↪{hf_api_key}"}, timeout=120)

# 2 Initialize the Kokoro TTS Pipeline
tts_pipeline = KPipeline(lang_code='a')  # Use American English (you can change␣
 ↪the language here)

# 3 Unified function to generate text from Falcon and TTS from Kokoro
def generate_audio_response(text_input):
    # Step 1: Get the text from Falcon-7B-Instruct model
    response = client.generate(text_input, max_new_tokens=50)

    # Check the structure of the response and print it to debug
    print("Response from Falcon:", response)

    # If response is an object, check if it has an attribute or method for␣
 ↪generated text
    # This is an example assumption, modify based on your response structure
    generated_text = response.generated_text if hasattr(response,␣
 ↪'generated_text') else str(response)

    # Step 2: Use Kokoro to generate audio from the text
```

```
    generator = tts_pipeline(
        generated_text,
        voice='af_heart',  # You can change the voice here if needed
        speed=1,
        split_pattern=r'\n+'
    )

    # Collect the audio
    audio_data = None
    for i, (gs, ps, audio) in enumerate(generator):
        audio_data = audio  # Get the generated audio

    # Step 3: Save the audio and return it for Gradio playback
    audio_filename = "response_audio.wav"
    sf.write(audio_filename, audio_data, 24000)  # Save the audio as .wav file

    # Return the saved audio file and the generated text
    return audio_filename, generated_text

# 4  Create the Gradio Interface
with gr.Blocks() as demo:
    # Chatbox to enter the prompt
    text_input = gr.Textbox(label="Enter Prompt", placeholder="Type something␣
 ↪here...")

    # Output components
    output_audio = gr.Audio(label="Generated Audio")  # This will play the audio
    output_text = gr.Textbox(label="Generated Text", interactive=False)  # This␣
 ↪will display the generated text

    # Button to trigger the process
    submit_button = gr.Button("Generate Response")

    # Set the action for the button
    submit_button.click(generate_audio_response, inputs=text_input,␣
 ↪outputs=[output_audio, output_text])

# Launch the Gradio interface
demo.launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://afd948d21f7d5fc233.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

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

[21]:

1.1.1 Final Fun Exercise: Multi-Turn Conversation with Audio Output!

Now that you've learned how to combine text generation and text-to-speech in a simple one-turn conversation, let's make it even more interesting! Your task is to extend the current pipeline into a **multi-turn conversation** where the model keeps talking to you, and you can respond with a new input to continue the conversation.

Here's what you need to do: 1. **User Input**: After each response from the model, you need to prompt the user for the next piece of input. 2. **Conversation Loop**: The model should generate a response based on the user's input and output both the generated text and audio. 3. **Interactive Dialogue**: Each time the user submits a new input, the conversation continues — as if you're chatting with an AI assistant.

To get started, modify the existing code and add a **conversation history** that stores past interactions, so the model has context for the next response.

1.1.2 What You'll Need to Do:

- Create a loop that allows the user to input new text after each response.
- Update the `generate_audio_response` function to handle multi-turn interactions, where the model remembers past exchanges.
- Make sure to continue the text generation and audio output for each new turn.

Good luck, and may your conversations with the AI assistant be lively and fun!

1.1.3 Wrapping Up

Congratulations! You've just created an interactive, multi-turn dialogue system using state-of-the-art **Falcon-7B** for text generation and **Kokoro** for text-to-speech. You learned how to generate responses, convert them into speech, and integrate them into a Gradio interface to create a seamless user experience.

Here's what we've accomplished together: - Set up and used **Falcon-7B-Instruct** for text generation. - Built a text-to-speech pipeline using **Kokoro**. - Integrated everything into a fun Gradio app where you can chat with an AI assistant.

Before you wrap up, feel free to: 1. Tweak the voices in **Kokoro** to give your assistant a new personality. 2. Enhance the conversation with richer dialogues or special voices for a unique experience. 3. Take the conversation loop challenge and make your assistant even more engaging!

Remember: AI is all about **experimenting**, **learning**, and having **fun**. Keep pushing boundaries, and who knows what awesome things you'll create next!

Thank you for building with me. The world of AI is yours to explore!

Happy coding, and may your assistant always have the best answers!