

# day5

January 12, 2025

## 1 Project - Airline AI Assistant

We'll now bring together what we've learned to make an AI Customer Support assistant for an Airline

```
[1]: # imports

import os
import json
from dotenv import load_dotenv
from openai import OpenAI
import gradio as gr
```

```
[2]: # Initialization

load_dotenv()

openai_api_key = os.getenv('OPENAI_API_KEY')
if openai_api_key:
    print(f"OpenAI API Key exists and begins {openai_api_key[:8]}")
else:
    print("OpenAI API Key not set")

MODEL = "gpt-4o-mini"
openai = OpenAI()
```

OpenAI API Key exists and begins sk-proj-

```
[3]: system_message = "You are a helpful assistant for an Airline called FlightAI. "
system_message += "Give short, courteous answers, no more than 1 sentence. "
system_message += "Always be accurate. If you don't know the answer, say so."
```

```
[4]: # This function looks rather simpler than the one from my video, because we're
    ↪ taking advantage of the latest Gradio updates

def chat(message, history):
    messages = [{"role": "system", "content": system_message}] + history +
    ↪ [{"role": "user", "content": message}]
```

```

    response = openai.chat.completions.create(model=MODEL, messages=messages)
    return response.choices[0].message.content

gr.ChatInterface(fn=chat, type="messages").launch()

```

\* Running on local URL: `http://127.0.0.1:7860`

To create a public link, set ``share=True`` in ``launch()``.

<IPython.core.display.HTML object>

[4]:

## 1.1 Tools

Tools are an incredibly powerful feature provided by the frontier LLMs.

With tools, you can write a function, and have the LLM call that function as part of its response.

Sounds almost spooky.. we're giving it the power to run code on our machine?

Well, kinda.

[5]: *# Let's start by making a useful function*

```

ticket_prices = {"london": "$799", "paris": "$899", "tokyo": "$1400", "berlin":
    ↪"$499"}

def get_ticket_price(destination_city):
    print(f"Tool get_ticket_price called for {destination_city}")
    city = destination_city.lower()
    return ticket_prices.get(city, "Unknown")

```

[6]: `get_ticket_price("London")`

Tool get\_ticket\_price called for London

[6]: '\$799'

[7]: *# There's a particular dictionary structure that's required to describe our*  
*↪function:*

```

price_function = {
    "name": "get_ticket_price",
    "description": "Get the price of a return ticket to the destination city.
    ↪Call this whenever you need to know the ticket price, for example when a
    ↪customer asks 'How much is a ticket to this city'",
    "parameters": {
        "type": "object",
        "properties": {

```

```

        "destination_city": {
            "type": "string",
            "description": "The city that the customer wants to travel to",
        },
    },
    "required": ["destination_city"],
    "additionalProperties": False
}
}

```

[8]: *# And this is included in a list of tools:*

```
tools = [{"type": "function", "function": price_function}]
```

## 1.2 Getting OpenAI to use our Tool

There's some fiddly stuff to allow OpenAI "to call our tool"

What we actually do is give the LLM the opportunity to inform us that it wants us to run the tool.

Here's how the new chat function looks:

```

[9]: def chat(message, history):
    messages = [{"role": "system", "content": system_message}] + history +
    ↪ [{"role": "user", "content": message}]
    response = openai.chat.completions.create(model=MODEL, messages=messages,
    ↪ tools=tools)

    if response.choices[0].finish_reason=="tool_calls":
        message = response.choices[0].message
        response, city = handle_tool_call(message)
        messages.append(message)
        messages.append(response)
        response = openai.chat.completions.create(model=MODEL,
    ↪ messages=messages)

    return response.choices[0].message.content

```

[10]: *# We have to write that function handle\_tool\_call:*

```

def handle_tool_call(message):
    tool_call = message.tool_calls[0]
    arguments = json.loads(tool_call.function.arguments)
    city = arguments.get('destination_city')
    price = get_ticket_price(city)
    response = {
        "role": "tool",
        "content": json.dumps({"destination_city": city, "price": price}),
    }

```

```

        "tool_call_id": tool_call.id
    }
    return response, city

```

```
[11]: gr.ChatInterface(fn=chat, type="messages").launch()
```

\* Running on local URL: `http://127.0.0.1:7861`

To create a public link, set ``share=True`` in ``launch()``.

<IPython.core.display.HTML object>

```
[11]:
```

Tool `get_ticket_price` called for Berlin

## 2 Let's go multi-modal!!

We can use DALL-E-3, the image generation model behind GPT-4o, to make us some images

Let's put this in a function called `artist`.

### 2.0.1 Price alert: each time I generate an image it costs about 4c - don't go crazy with images!

```
[12]: # Some imports for handling images
```

```

import base64
from io import BytesIO
from PIL import Image

```

```
[13]: def artist(city):
        image_response = openai.images.generate(
            model="dall-e-3",
            prompt=f"An image representing a vacation in {city}, showing
↪tourist spots and everything unique about {city}, in a vibrant pop-art
↪style",
            size="1024x1024",
            n=1,
            response_format="b64_json",
        )
        image_base64 = image_response.data[0].b64_json
        image_data = base64.b64decode(image_base64)
        return Image.open(BytesIO(image_data))

```

```
[16]: image = artist("Helsinki")
display(image)
```



[ ]:

## 2.1 Audio (NOTE - Audio is optional for this course - feel free to skip Audio if it causes trouble!)

And let's make a function talker that uses OpenAI's speech model to generate Audio

### 2.1.1 Troubleshooting Audio issues

If you have any problems running this code below (like a FileNotFound error, or a warning of a missing package), you may need to install FFmpeg, a very popular audio utility.

#### For PC Users

Detailed instructions are [here](#) and summary instructions:

1. Download FFmpeg from the official website: <https://ffmpeg.org/download.html>
2. Extract the downloaded files to a location on your computer (e.g., `C:\ffmpeg`)
3. Add the FFmpeg bin folder to your system PATH:
  - Right-click on 'This PC' or 'My Computer' and select 'Properties'
  - Click on 'Advanced system settings'
  - Click on 'Environment Variables'
  - Under 'System variables', find and edit 'Path'
  - Add a new entry with the path to your FFmpeg bin folder (e.g., `C:\ffmpeg\bin`)
  - Restart your command prompt, and within Jupyter Lab do Kernel -> Restart kernel, to pick up the changes
4. Open a new command prompt and run this to make sure it's installed OK `ffmpeg -version`

### For Mac Users

1. Install homebrew if you don't have it already by running this in a Terminal window and following any instructions:  
`/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
2. Then install FFmpeg with `brew install ffmpeg`
3. Verify your installation with `ffmpeg -version` and if everything is good, within Jupyter Lab do Kernel -> Restart kernel to pick up the changes

Message me or email me at [ed@edwarddonner.com](mailto:ed@edwarddonner.com) with any problems!

## 2.2 To check you now have ffmpeg and can access it here

Execute the next cell to see if you get a version number. (Putting an exclamation mark before something in Jupyter Lab tells it to run it as a terminal command rather than python code).

If this doesn't work, you may need to actually save and close down your Jupyter lab, and start it again from a new Terminal window (Mac) or Anaconda prompt (PC), remembering to activate the llms environment. This ensures you pick up ffmpeg.

And if that doesn't work, please contact me!

```
[17]: !ffmpeg -version
      !ffprobe -version
      !ffplay -version
```

```
ffmpeg version 7.1 Copyright (c) 2000-2024 the FFmpeg developers
built with Apple clang version 15.0.0 (clang-1500.1.0.2.5)
configuration: --prefix=/usr/local/Cellar/ffmpeg/7.1_4 --enable-shared --enable-
pthreads --enable-version3 --cc=clang --host-cflags= --host-
ldflags='-Wl,-ld_classic' --enable-ffplay --enable-gnutls --enable-gpl --enable-
libaom --enable-libaribb24 --enable-libbluray --enable-libdav1d --enable-
libharfbuzz --enable-libjxl --enable-libmp3lame --enable-libopus --enable-
librav1e --enable-librist --enable-librubberband --enable-libsrt --enable-
libssh --enable-libsvtav1 --enable-libtesseract --enable-
libtheora --enable-libvidstab --enable-libvmaf --enable-libvorbis --enable-
```

```

libvpx --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxml2
--enable-libxvid --enable-lzma --enable-libfontconfig --enable-libfreetype
--enable-frei0r --enable-libass --enable-libopencore-amrnb --enable-libopencore-
amrwb --enable-libopenjpeg --enable-libspeex --enable-libsoxr --enable-libzmq
--enable-libzimg --disable-libjack --disable-indev=jack --enable-videotoolbox
--enable-audiotoolbox
libavutil      59. 39.100 / 59. 39.100
libavcodec     61. 19.100 / 61. 19.100
libavformat    61.  7.100 / 61.  7.100
libavdevice    61.  3.100 / 61.  3.100
libavfilter    10.  4.100 / 10.  4.100
libswscale     8.  3.100 /  8.  3.100
libswresample  5.  3.100 /  5.  3.100
libpostproc   58.  3.100 / 58.  3.100
ffprobe version 7.1 Copyright (c) 2007-2024 the FFmpeg developers
built with Apple clang version 15.0.0 (clang-1500.1.0.2.5)
configuration: --prefix=/usr/local/Cellar/ffmpeg/7.1_4 --enable-shared --enable-
pthreads --enable-version3 --cc=clang --host-cflags= --host-
ldflags='-Wl,-ld_classic' --enable-ffplay --enable-gnutls --enable-gpl --enable-
libaom --enable-libaribb24 --enable-libbluray --enable-libdav1d --enable-
libharfbuzz --enable-libjxl --enable-libmp3lame --enable-libopus --enable-
librav1e --enable-rist --enable-librubberband --enable-libsrt --enable-
libssh --enable-libsvtav1 --enable-libtesseract --enable-
libtheora --enable-libvidstab --enable-libvmaf --enable-libvorbis --enable-
libvpx --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxml2
--enable-libxvid --enable-lzma --enable-libfontconfig --enable-libfreetype
--enable-frei0r --enable-libass --enable-libopencore-amrnb --enable-libopencore-
amrwb --enable-libopenjpeg --enable-libspeex --enable-libsoxr --enable-libzmq
--enable-libzimg --disable-libjack --disable-indev=jack --enable-videotoolbox
--enable-audiotoolbox
libavutil      59. 39.100 / 59. 39.100
libavcodec     61. 19.100 / 61. 19.100
libavformat    61.  7.100 / 61.  7.100
libavdevice    61.  3.100 / 61.  3.100
libavfilter    10.  4.100 / 10.  4.100
libswscale     8.  3.100 /  8.  3.100
libswresample  5.  3.100 /  5.  3.100
libpostproc   58.  3.100 / 58.  3.100
ffplay version 7.1 Copyright (c) 2003-2024 the FFmpeg developers
built with Apple clang version 15.0.0 (clang-1500.1.0.2.5)
configuration: --prefix=/usr/local/Cellar/ffmpeg/7.1_4 --enable-shared --enable-
pthreads --enable-version3 --cc=clang --host-cflags= --host-
ldflags='-Wl,-ld_classic' --enable-ffplay --enable-gnutls --enable-gpl --enable-
libaom --enable-libaribb24 --enable-libbluray --enable-libdav1d --enable-
libharfbuzz --enable-libjxl --enable-libmp3lame --enable-libopus --enable-
librav1e --enable-rist --enable-librubberband --enable-libsrt --enable-
libssh --enable-libsvtav1 --enable-libtesseract --enable-
libtheora --enable-libvidstab --enable-libvmaf --enable-libvorbis --enable-

```



```

libvpx --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxml2
--enable-libxvid --enable-lzma --enable-libfontconfig --enable-libfreetype
--enable-frei0r --enable-libass --enable-libopencore-amrnb --enable-libopencore-
amrwb --enable-libopenjpeg --enable-libspeex --enable-libsoxr --enable-libzmq
--enable-libzimg --disable-libjack --disable-indev=jack --enable-videotoolbox
--enable-audiotoolbox
libavutil      59. 39.100 / 59. 39.100
libavcodec     61. 19.100 / 61. 19.100
libavformat    61.  7.100 / 61.  7.100
libavdevice    61.  3.100 / 61.  3.100
libavfilter    10.  4.100 / 10.  4.100
libswscale     8.  3.100 /  8.  3.100
libswresample  5.  3.100 /  5.  3.100
libpostproc   58.  3.100 / 58.  3.100

```

### 3 For Mac users - and possibly many PC users too

This version should work fine for you. It might work for Windows users too, but you might get a Permissions error writing to a temp file. If so, see the next section!

As always, if you have problems, please contact me! (You could also comment out the audio talker() in the later code if you're less interested in audio generation)

```

[18]: from pydub import AudioSegment
      from pydub.playback import play

      def talker(message):
          response = openai.audio.speech.create(
              model="tts-1",
              voice="onyx",      # Also, try replacing onyx with alloy
              input=message
          )

          audio_stream = BytesIO(response.content)
          audio = AudioSegment.from_file(audio_stream, format="mp3")
          play(audio)

```

```

[19]: talker("Well, hi there")

```

```

Input #0, wav, from
'/var/folders/l4/484f84ms4w76b45tpq_149zw0000gn/T/tmp4azu_gc8.wav':
  Duration: 00:00:00.77, bitrate: 384 kb/s
  Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 24000 Hz, 1 channels,
s16, 384 kb/s
    0.54 M-A:  0.000 fd=   0 aq=   OKB vq=   OKB sq=   0B

    0.64 M-A: -0.000 fd=   0 aq=   OKB vq=   OKB sq=   0B

```



## 4 For Windows users (or any Mac users with problems above)

### 4.1 First try the Mac version above, but if you get a permissions error writing to a temp file, then this code should work instead.

A collaboration between students Mark M. and Patrick H. and Claude got this resolved!

Below are 4 variations - hopefully one of them will work on your PC. If not, message me please!

And for Mac people - all 3 of the below work on my Mac too - please try these if the Mac version gave you problems.

### 4.2 PC Variation 1

```
[20]: import base64
from io import BytesIO
from PIL import Image
from IPython.display import Audio, display

def talker(message):
    response = openai.audio.speech.create(
        model="tts-1",
        voice="onyx",
        input=message)

    audio_stream = BytesIO(response.content)
    output_filename = "output_audio.mp3"
    with open(output_filename, "wb") as f:
        f.write(audio_stream.read())

    # Play the generated audio
    display(Audio(output_filename, autoplay=True))

talker("Well, hi there")
```

<IPython.lib.display.Audio object>

### 4.3 PC Variation 2

```
[21]: import tempfile
import subprocess
from io import BytesIO
from pydub import AudioSegment
import time

def play_audio(audio_segment):
    temp_dir = tempfile.gettempdir()
    temp_path = os.path.join(temp_dir, "temp_audio.wav")
    try:
        audio_segment.export(temp_path, format="wav")
```

```

        time.sleep(3) # Student Dominic found that this was needed. You could
↪also try commenting out to see if not needed on your PC
        subprocess.call([
            "ffplay",
            "-nodisp",
            "-autoexit",
            "-hide_banner",
            temp_path
        ], stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
    finally:
        try:
            os.remove(temp_path)
        except Exception:
            pass

def talker(message):
    response = openai.audio.speech.create(
        model="tts-1",
        voice="onyx", # Also, try replacing onyx with alloy
        input=message
    )
    audio_stream = BytesIO(response.content)
    audio = AudioSegment.from_file(audio_stream, format="mp3")
    play_audio(audio)

talker("Well hi there")

```

#### 4.4 PC Variation 3

```

[22]: import os
from pydub import AudioSegment
from pydub.playback import play
from io import BytesIO

def talker(message):
    # Set a custom directory for temporary files on Windows
    custom_temp_dir = os.path.expanduser("~/Documents/temp_audio")
    os.environ['TEMP'] = custom_temp_dir # You can also use 'TMP' if necessary

    # Create the folder if it doesn't exist
    if not os.path.exists(custom_temp_dir):
        os.makedirs(custom_temp_dir)

    response = openai.audio.speech.create(
        model="tts-1",
        voice="onyx", # Also, try replacing onyx with alloy
        input=message
    )

```

```

    )

    audio_stream = BytesIO(response.content)
    audio = AudioSegment.from_file(audio_stream, format="mp3")

    play(audio)

talker("Well hi there")

```

```

Input #0, wav, from
'/var/folders/l4/484f84ms4w76b45tpq_149zw0000gn/T/tmpnrpmavii.wav':
  Duration: 00:00:00.74, bitrate: 384 kb/s
  Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 24000 Hz, 1 channels,
s16, 384 kb/s
    0.58 M-A:  0.000 fd=   0 aq=   OKB vq=   OKB sq=   0B

    0.64 M-A:  0.000 fd=   0 aq=   OKB vq=   OKB sq=   0B

```

## 4.5 PC Variation 4

### 4.5.1 Let's try a completely different sound library

First run the next cell to install a new library, then try the cell below it.

```
[23]: !pip install simpleaudio
```

```

Collecting simpleaudio
  Downloading simpleaudio-1.0.4.tar.gz (2.0 MB)
                                2.0/2.0 MB
3.2 MB/s eta 0:00:00a 0:00:01
  Preparing metadata (setup.py) ... one
Building wheels for collected packages: simpleaudio
  Building wheel for simpleaudio (setup.py) ...done
  Created wheel for simpleaudio:
filename=simpleaudio-1.0.4-cp311-cp311-macosx_10_15_x86_64.whl size=2036185
sha256=d766936d7bb09dfc26fe20ee43f3940f25917246d76546a847b0fee7fcfc3d66
  Stored in directory: /Users/markku/Library/Caches/pip/wheels/44/74/11/9d2cf6dd
f5d0dcc7750484a371b92fdb036f5496d1d4b42d08
Successfully built simpleaudio
Installing collected packages: simpleaudio
Successfully installed simpleaudio-1.0.4

```

```
[24]: from pydub import AudioSegment
from io import BytesIO
import tempfile
import os
import simpleaudio as sa

```

```

def talker(message):
    response = openai.audio.speech.create(
        model="tts-1",
        voice="onyx", # Also, try replacing onyx with alloy
        input=message
    )

    audio_stream = BytesIO(response.content)
    audio = AudioSegment.from_file(audio_stream, format="mp3")

    # Create a temporary file in a folder where you have write permissions
    with tempfile.NamedTemporaryFile(suffix=".wav", delete=False, dir=os.path.
    expanduser("~/Documents")) as temp_audio_file:
        temp_file_name = temp_audio_file.name
        audio.export(temp_file_name, format="wav")

    # Load and play audio using simpleaudio
    wave_obj = sa.WaveObject.from_wave_file(temp_file_name)
    play_obj = wave_obj.play()
    play_obj.wait_done() # Wait for playback to finish

    # Clean up the temporary file afterward
    os.remove(temp_file_name)

talker("Well hi there")

```

## 4.6 PC Users - if none of those 4 variations worked!

Please get in touch with me. I'm sorry this is causing problems! We'll figure it out.

Alternatively: playing audio from your PC isn't super-critical for this course, and you can feel free to focus on image generation and skip audio for now, or come back to it later.

## 5 Our Agent Framework

The term 'Agentic AI' and Agentization is an umbrella term that refers to a number of techniques, such as:

1. Breaking a complex problem into smaller steps, with multiple LLMs carrying out specialized tasks
2. The ability for LLMs to use Tools to give them additional capabilities
3. The 'Agent Environment' which allows Agents to collaborate
4. An LLM can act as the Planner, dividing bigger tasks into smaller ones for the specialists
5. The concept of an Agent having autonomy / agency, beyond just responding to a prompt - such as Memory

We're showing 1 and 2 here, and to a lesser extent 3 and 5. In week 8 we will do the lot!

```
[25]: def chat(history):
    messages = [{"role": "system", "content": system_message}] + history
    response = openai.chat.completions.create(model=MODEL, messages=messages,
    ↪tools=tools)
    image = None

    if response.choices[0].finish_reason=="tool_calls":
        message = response.choices[0].message
        response, city = handle_tool_call(message)
        messages.append(message)
        messages.append(response)
        image = artist(city)
        response = openai.chat.completions.create(model=MODEL,
    ↪messages=messages)

    reply = response.choices[0].message.content
    history += [{"role": "assistant", "content": reply}]

    # Comment out or delete the next line if you'd rather skip Audio for now..
    talker(reply)

    return history, image
```

```
[26]: # More involved Gradio code as we're not using the preset Chat interface!
# Passing in inbrowser=True in the last line will cause a Gradio window to pop
    ↪up immediately.

with gr.Blocks() as ui:
    with gr.Row():
        chatbot = gr.Chatbot(height=500, type="messages")
        image_output = gr.Image(height=500)
    with gr.Row():
        entry = gr.Textbox(label="Chat with our AI Assistant:")
    with gr.Row():
        clear = gr.Button("Clear")

    def do_entry(message, history):
        history += [{"role": "user", "content": message}]
        return "", history

    entry.submit(do_entry, inputs=[entry, chatbot], outputs=[entry, chatbot]).
    ↪then(
        chat, inputs=chatbot, outputs=[chatbot, image_output]
    )
    clear.click(lambda: None, inputs=None, outputs=chatbot, queue=False)

ui.launch(inbrowser=True)
```

\* Running on local URL: `http://127.0.0.1:7862`

To create a public link, set ``share=True`` in ``launch()``.

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

[26]:

Tool `get_ticket_price` called for Berlin

## 6 Exercises and Business Applications

Add in more tools - perhaps to simulate actually booking a flight. A student has done this and provided their example in the community contributions folder.

Next: take this and apply it to your business. Make a multi-modal AI assistant with tools that could carry out an activity for your work. A customer support assistant? New employee onboarding assistant? So many possibilities! Also, see the week2 end of week Exercise in the separate Notebook.

I have a special request for you

My editor tells me that it makes a HUGE difference when students rate this course on Udemy - it's one of the main ways that Udemy decides whether to show it to others. If you're able to take a minute to rate this, I'd be so very grateful! And regardless - always please reach out to me at [ed@edwarddonner.com](mailto:ed@edwarddonner.com) if I can help at any point.