

Your Personalized Interview Coach

Estimated time needed: 45 minutes

Picture yourself sitting across from an interviewer. The camera light turns on, and a calm voice asks, "Tell me about a time you faced a challenge at work." You answer confidently, but how do you know if your tone sounds engaging, your pauses feel natural, or your story highlights the right skills?

THE INTERVIEW



Interviews can be intimidating, not just because of the questions, but because candidates rarely get structured, objective feedback. Most people practice alone or with friends, without knowing how hiring managers actually perceive their answers. Even AI interview tools that exist today often feel robotic or lack personalization—they evaluate, but they don't coach.

This project, **Interview Coach**, changes that. It combines speech analysis, large language models (LLMs), and audio interaction to simulate realistic interviews and provide detailed, human-like feedback. The system listens to your spoken answers, analyzes your delivery and content, and generates follow-up questions just like a real interviewer.

You'll also build a clean **Gradio-based web interface** that automatically plays each question aloud, records user responses, and summarizes their performance in natural language. Behind the scenes, the app integrates a speech-to-text pipeline, LLM-based analysis, and text-to-speech synthesis to create a seamless, one-on-one coaching experience.

By the end, you'll have **a working AI interview simulator** that doesn't just test your responses—it teaches you how to improve, preparing you to speak with confidence and clarity in any real interview setting.

Personalized Interview Coach

Upload your pdf resume/CV and copy paste the job description you are applying to:

Decide the length of your mock interview (from 1 question to 10 questions):

Number of Questions

Click "Start Interview" below to start the Mock Interview!

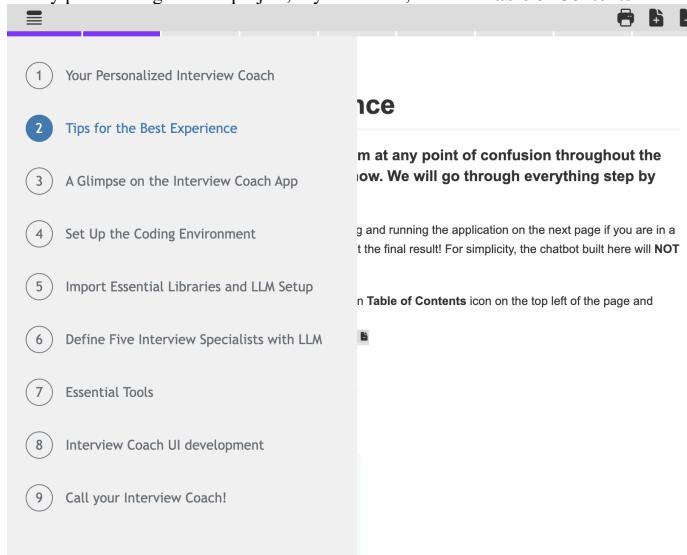
Evaluating your performance along the way ...

Performance Evaluation

Tips for the Best Experience

Keep the following tips handy and refer to them at any point of confusion throughout the tutorial. Do not worry if they seem irrelevant now. We will go through everything step by step later.

- You can choose to follow the **Glimpse version** of building and running the application on the next page if you are in a hurry or are too excited to test the final result! For simplicity, the chatbot built here will **NOT** require Serper API Key.
- At any point throughout the project, if you are lost, click on **Table of Contents** icon on the top left of the page and navigate to your desired content.



- Cloud IDE automatically saves any changes you make to your files immediately. You can also save from the toolbar.
- To stop execution of the chatbot app in addition to closing the application tab, hit **Ctrl+C** in terminal.

A Glimpse on the Interview Coach App

On this page, you can build and run the chatbot in **less than 10 minutes!**

If you like to build the app from the scratch, you may directly go to the next page!

Execution Instructions

To execute the code snippet, click directly on the run button >_.

```
1 git clone https://github.com/jianpingy/Interview-Coach.git
2 cd Interview-Coach
```



Step 1: Clone the Github repository:

```
git clone https://github.com/jianpingy/Interview-Coach.git
cd Interview-Coach
```

Step 2: Install all required libraries:

```
pip install -r requirements.txt
```

Step 2.5: Wait 3-5 minutes for libraries to be installed...

Step 3: Run the UI

```
gradio myapp.py
```

Step 3.5: Wait 20-30 seconds...

Step 4: CLICK the button and LAUNCH!

Try below! There are example resume and job descriptions on the web app you can use for a quick start!

[Start Interview Coach](#)

For this particular project, since we are going to use microphone, please click “open in a new window” to run your app in a separate web window.

The screenshot shows a web browser window with the following details:

- Header:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Address Bar:** Welcome | myapp.py | application X | https://jpye00-7861.theianext-0-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai/
- Title:** Personalized Interview Coach
- Content:**
 - Upload Area:** "Upload your pdf resume/CV and copy paste the job description you are applying to:"
 - File Input:** "Upload Resume (PDF)" with a file icon.
 - Drop Zone:** "Drop File Here" with an upward arrow icon.
 - Text:** "- OR - Click to Upload"
 - Job Description Area:** "Job Description" with a large text input field.
- Action:** A red box highlights the "Open in New Window" button in the top right corner of the page.

Set Up the Coding Environment

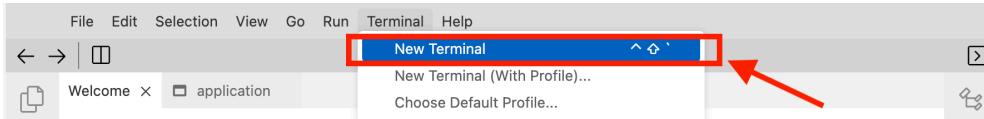
Great to have you here! I believe you are very excited about building the app on your own!

From this page, you will start your journey to develop the chatbot for the interview coach.

Step 0: Open a new terminal

We will use a new **terminal** to execute all the codes.

You can start a new terminal by clicking new terminal in the dropdown from Terminal tab.



Step 1: Create the directories (a.k.a. folders) to put your codes and files (e.g. databases)

Run the following codes to create the directories and the empty files.

Click on the Execute button >_ to run the lines.

```
mkdir interview_coach # Main directory for the app and files
cd interview_coach # Enter the created directory
wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/9REpXggZYZwpQAx5-J5r2A/example-resume.pdf # Get the example resume
wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/VLGt3FoQlGbcprgzyzi-Hg/example-job-description.txt # Get the example job description
touch requirements.txt # A text file to store the required Python dependencies
touch myapp.py # Main interview coach ui python file
```

Step 2: Populate requirements.txt file:

Navigate to the `requirements.txt` file that is located inside the `interview_coach` folder and **COPY** the following content into it. Open the text file in the required directory by clicking on the button below;

[Open requirements.txt in IDE](#)

To easily **COPY** the following content you can click on the copy icon at the bottom right side of the code box.

```
ibm_watsonx_ai==1.1.26
gTTS==2.5.4
faster-whisper==1.2.1
PyPDF2==3.0.1
gradio==5.49.1
pyarrow==22.0.0
```

Step 3: Set up a Python virtual environment

Initialize a new Python virtual environment to keep the required library versions tidy. Note, you can run the snippet directly by clicking on the run button >_.

```
python3.11 -m venv venv
source venv/bin/activate
```

Step 4: Install the required libraries

With your virtual environment activated, install all the required libraries via >_:

```
pip install -r requirements.txt
```

Approximately it will take **3-5 minutes** to install all the required libraries.

Now your environment is set up, you're ready to start building your personalized Interview Coach!

Import Essential Libraries and LLM Setup

Before building the interactive AI Interview Coach, we first import a few key libraries that power language understanding, voice synthesis, audio transcription, and the web interface. Each library serves a distinct purpose in enabling smooth communication between the user and the AI interviewer.

Then we will load the base LLM for our interview coach. In this project, we will use `Llama 3` as the LLM backend.

Open `myapp.py` by clicking the button below so that we can edit the python file.

[Open myapp.py in IDE](#)

Important: Copy and paste the codes in the following steps to fill in `myapp.py`.

Step 1: Import Libraries

This step imports all the necessary libraries for the project. In essence, we will import:

- **ModelInference:** Enables interaction with IBM Watsonx's foundation models, allowing the system to send prompts and receive intelligent text-based responses from an enterprise-grade LLM.
- **TextChatParameters:** Defines the parameters for how the model should behave during chat inference — including temperature, token limits, and other decoding controls.
- **Credentials:** Manages secure authentication and connection to IBM Cloud's Watsonx services before making API calls.
- **gTTS:** Stands for *Google Text-to-Speech*. It converts the interviewer's text-based questions into natural-sounding audio that plays automatically for the user.
- **WhisperModel:** From the *Faster Whisper* library, it transcribes the candidate's spoken answers into text with high accuracy and low latency.
- **PyPDF2:** Reads and extracts text from uploaded resume PDFs. This allows the system to process the candidate's resume content for automated analysis.
- **gradio:** Powers the web-based user interface. It provides components such as text boxes, buttons, and audio recorders that allow users to interact seamlessly with the AI interviewer.

```
from ibm_watsonx_ai.foundation_models import ModelInference
from ibm_watsonx_ai.foundation_models.schema import TextChatParameters
from ibm_watsonx_ai import Credentials
from gtts import gTTS
from faster_whisper import WhisperModel
import PyPDF2
import gradio as gr
```

Together, these libraries connect every part of the **AI Interview Coach**:

- IBM Watsonx powers the intelligence and reasoning.
- gTTS and Faster Whisper handle speech synthesis and transcription.
- PyPDF2 prepares the resume input.
- Gradio delivers the interactive front-end.

In combination, they form the foundation of an intelligent, voice-enabled, and highly interactive AI-driven interview experience.

Step 2: Setup the base LLM

In this step, we configure and initialize the **Large Language Model (LLM)** that serves as the brain of the AI Interview Coach. This setup ensures that the interviewer, evaluator, and all supporting agents share the same intelligent foundation.

Here's what happens in this block:

- **chat_histories, interview_step, resume_summary, and job_summary:** These variables act as the **global memory** of the system, tracking the ongoing interview progress — including the conversation history, current step number, and summarized insights from the candidate's resume and the job description.
- **project_id="skills-network":** Identifies your Watsonx project environment where model calls and analytics are managed.
- **Credentials:** Connects securely to **IBM Watsonx.ai** using your cloud endpoint (<https://us-south.ml.cloud.ibm.com>). This ensures authenticated and authorized access to IBM's foundation models.
- **TextChatParameters:** Defines how the LLM behaves when generating responses.

- `max_tokens = 1e5` allows for long-form, detailed outputs.
 - `response_format = None` ensures free-form, natural conversation responses.
- This configuration balances creativity with control over verbosity and coherence.

- **ModelInference:**

Initializes the core **LLM interface** by specifying:

- `model_id='meta-llama/llama-3-3-70b-instruct'` → a powerful instruction-tuned version of Meta's LLaMA 3.3 model.
 - `credentials` and `project_id` for secure access.
 - `params` for defining conversational style and constraints.
- ```
Global variable of the system: track the chat histories, number of interview questions,
the resume and job summary.
chat_histories = {}
interview_step = 0
resume_summary = None
job_summary = None
project_id="skills-network"
credentials = Credentials(
 url = "https://us-south.ml.cloud.ibm.com"
)
Get sample parameter values
sample_params = TextChatParameters.get_sample_params()
sample_params['max_tokens'] = int(1e5)
sample_params['response_format'] = None
Initialize the TextChatParameters object with the sample values
params = TextChatParameters(**sample_params)
Define LLM
llm_base = ModelInference(
 model_id='meta-llama/llama-3-3-70b-instruct',
 credentials=credentials,
 project_id=project_id,
 params=params,
)
```

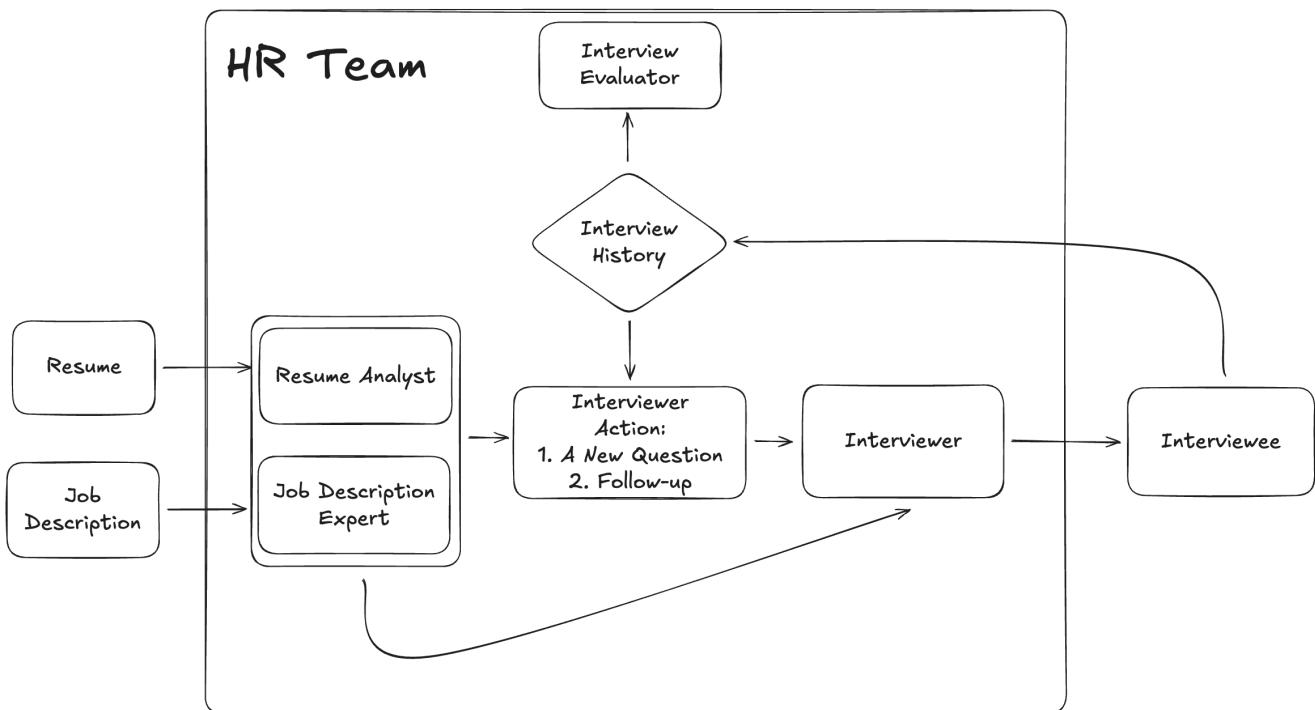
## Define Five Interview Specialists with LLM

This project uses a team of specialized **LLM-based HR agents** that work together to simulate a realistic interview experience: from resume screening to post-interview evaluation.

Each agent has a distinct role, mirroring how real-world HR departments operate: a screener, a recruiter, a strategist, an interviewer, and an evaluator.

Together, they create an intelligent, end-to-end mock interview pipeline that analyzes resumes, understands job descriptions, conducts adaptive questioning, and delivers personalized performance feedback.

Here is the flowchart for this LLM HR Team.



Let's keep furnishing `myapp.py` by adding these core LLM agents that power the interview simulation.

[Open myapp.py in IDE](#)

**Important:** Copy and paste the codes in each of the following steps to `myapp.py`.

## Step 1: Resume Analyst Agent — “The Resume Screener”

- **Role:** Acts as a resume reviewer who reads and summarizes a candidate’s CV in a structured, human-readable report.
- **Purpose:** This agent mimics how HR screeners or ATS systems extract key details from resumes, such as: The candidate’s identity and demographics (if present), Core skills and expertise, and Summary of past roles and achievements.
- **Output:** A concise 3-paragraph report summarizing the candidate’s background, forming the foundation for later interview questions.

```
def Resume_Analyst(resume):
 prompt = f"""
 Write a detailed REPORT, in several paragraphs, on the candidate.
 Three paragraphs: candidate's name and demographic info, key_skills, and the summary of the past experiences.
 Resume:
 {resume}
 """
 response = llm_base.chat(
 messages=[
 {"role": "system", "content": "You are an HR expert in reviewing resumes."},
 {"role": "user", "content": prompt}
]
)
 response_output = response['choices'][0]['message']['content']
return response_output
```

## Step 2: Job Description Expert — “The Recruiter”

- **Role:** Analyzes the provided job description to identify key skills, technical requirements, and preferred experiences.
- **Purpose:** This agent helps the system “understand” the employer’s expectations and construct a profile of the ideal candidate. It captures essential insights such as:
  - Core competencies required for the role
  - Preferred qualifications and experience levels
  - Key themes or priorities (e.g., teamwork, problem-solving, leadership)
- **Output:** A clear summary highlighting the required and desired skills and experiences from the job posting.

```
def Job_Description_Expert(job_description):
 prompt = f"""
 Write a summary of the job description.
 Identify the skills required and the experiences preferred.
 Job Description:
 {job_description}
 """
 response = llm_base.chat(
 messages=[
 {"role": "system", "content": "You are job expert."},
 {"role": "user", "content": prompt}
]
)
 response_output = response['choices'][0]['message']['content']
return response_output
```

## Step 3: Interview Question Action Planner — “The Strategist”

- **Role:** Decides what the interviewer should ask next, based on the chat history, the candidate’s resume summary, and the job summary.
- **Purpose:** This agent simulates a human interviewer’s strategic thinking — deciding whether to move on to a new topic or ask a follow-up question for deeper insight. It ensures the interview feels coherent, natural, and contextually aware.
- **Output:** A directive such as:
  - “Ask a follow-up about teamwork experience.”
  - “Ask about another technical skill mentioned in the resume.”

```
def Interview_Question_Action(chat_histories, resume_summary, job_summary):
 prompt = f"""
 Based on the histories of the answers, the resume summary and the job summary,
 pick one of the following two actions for the next question:
 - (1) Ask about another past experience or skills on the resume.
 - (2) Ask follow-up questions of the current topic.
 Answer Histories:
 {chat_histories}
 Resume Summary:
 {resume_summary}
 Job Summary:
 {job_summary}
 """
 response = llm_base.chat(
 messages=[
 {"role": "system", "content": "You are job expert."},
 {"role": "user", "content": prompt}
]
)
```

```

)
response_output = response['choices'][0]['message']['content']
return response_output

```

## Step 4: Interviewer Agent — “The Interview Host”

- **Role:** Generates natural, realistic interview questions based on the planner’s directive and interacts directly with the candidate.
- **Purpose:** This agent is the **front-facing interviewer** that drives the conversation. It ensures the tone feels human and conversational while maintaining a professional HR style.
  - Starts the session with: “Tell me about yourself.”
  - Continues dynamically based on previous answers.
  - Ends gracefully with a thank-you message once the interview concludes.
- **Output:** A human-like, spoken interview question (later converted to audio in the app).

```

def Interviewer(resume_summary, job_summary, action=None, last=False):
 if not last:
 if action is not None:
 prompt = f"""
 Directly ask the question based on the given action instruction,
 resume summary and the job summary.
 DO NOT GIVE ANY EXPLANATIONS WHY YOU ASK THE QUESTION.
 Action:
 {action}
 Resume Summary:
 {resume_summary}
 Job Summary:
 {job_summary}
 """
 response = llm_base.chat(
 messages=[
 {"role": "system", "content": "You are an expert interviewer."},
 {"role": "user", "content": prompt}
]
)
 response_output = response['choices'][0]['message']['content']
 else:
 response_output = "Tell me about yourself."
 else:
 prompt = f"""
 The interview ends. Wrap up and express gratitude towards the candidate based on the resume.
 Be CONCISE.
 Resume Summary:
 {resume_summary}
 """
 response = llm_base.chat(
 messages=[
 {"role": "system", "content": "You are an expert interviewer."},
 {"role": "user", "content": prompt}
]
)
 response_output = response['choices'][0]['message']['content']

 return response_output

```

## Step 5: Evaluator Agent — “The HR Decision Maker”

- **Role:** Assesses the entire conversation after the interview and provides an evaluation of the candidate’s suitability.
- **Purpose:** This agent mirrors the post-interview HR analysis — weighing the candidate’s technical strengths, personality fit, and communication style. It provides feedback that helps users reflect and improve their interview performance.
- **Output:** A detailed evaluation report summarizing the candidate’s performance, highlighting both strengths and areas for improvement.

```

def Evaluator(chat_histories, job_summary):
 prompt = f"""
 Based on the histories of the answers and the job summary,
 evaluate if the candidate is a good match, by personality and skills,
 and give reasons.
 Answer Histories:
 {chat_histories}
 Job Summary:
 {job_summary}
 """
 response = llm_base.chat(
 messages=[
 {"role": "system", "content": "You are an expert to judge the performance of an interviewee."},
 {"role": "user", "content": prompt}
]
)
 response_output = response['choices'][0]['message']['content']
 return response_output

```

## Essential Tools

Before building our multi-agent HR system, we need a few core utilities that allow the app to *read resumes, talk, and listen*.

These functions act as the building blocks for our interactive AI interviewer — handling file reading, voice synthesis, and speech recognition seamlessly within the Gradio interface.

Again, open the `myapp.py` by clicking the button below so that we can add these essential tools to the file.

[Open myapp.py in IDE](#)

**Important:** Copy and paste the codes in the following steps to fill in `myapp.py`.

### Step 1. Resume Reader — Extracting Text from PDF

When candidates upload their resume, the first task is to read and parse its content.

This function uses `PyPDF2` to extract text from all pages of the uploaded PDF:

- Input: A PDF resume file uploaded by the user.
- Output: A single text string containing the full resume content.
- Used by: `Resume_Analyst()` agent to generate a structured summary of the candidate's profile

It ensures that even multi-page resumes are converted into clean, structured text for further analysis by the `Resume Analyst` Agent.

```
Function to read the pdf file (for resume)
def extract_text_from_pdf(pdf_file_path):
 reader = PyPDF2.PdfReader(pdf_file_path.name)
 text = ""
 for page in reader.pages:
 page_text = page.extract_text()
 if page_text:
 text += page_text + "\n"
 return text.strip()
```

### Step 2: Text-to-Speech Engine — Giving the Interviewer a Voice

To make the AI interviewer feel more human, this function converts text-based questions into speech.

We use the `gTTS (Google Text-to-Speech) library`, a simple open-source solution that generates natural English audio directly from text.

We define the function `text_to_speech_file(text_input)` with:

- Input: The interview question text generated by the Interview Host Agent.
- Output: A playable .mp3 file automatically rendered in the Gradio app.
- Used by: The interviewer interface to speak each question aloud.

```
Function to generate the audio file
def text_to_speech_file(text_input):
 # 1. Generate the audio and save it
 audio_file_path = "temp_voice.mp3"

 # Using gTTS (as demonstrated earlier) to create the audio file
 tts = gTTS(text=text_input, lang='en')
 tts.save(audio_file_path)

 # 2. Return the file path
 # Gradio will automatically display an audio player for this file.
 return audio_file_path
```

### Step 3: Speech-to-Text Engine — Understanding the Candidate

The interviewer also needs to listen. This function `transcribe_audio_faster_whisper` leverages `Faster Whisper`, a lightweight open-source version of OpenAI's Whisper model, to transcribe spoken answers into text:

- Input: The recorded candidate answer (audio file).
- Output: Transcribed text representing what the candidate said.

- Used by: The Interview and Evaluation agents to interpret and assess responses.

It runs efficiently on CPU or GPU, making it suitable for local deployment.

```
Function to convert the audio file to text
def transcribe_audio_faster_whisper(
 audio_file_path: str,
 model_size: str = "base",
 device: str = "auto",
 compute_type: str = "auto"
) -> str:
 # ... (function body remains the same as previously defined)

 if audio_file_path is None:
 return "Please provide an audio input."

 if compute_type == "auto":
 compute_type = "float16" if device == "cuda" else "int8"

 device = "cpu"

 try:
 model = WhisperModel(model_size, device=device, compute_type=compute_type)
 segments, info = model.transcribe(audio_file_path, beam_size=5)
 full_transcript = [segment.text for segment in segments]
 return "".join(full_transcript).strip()
 except Exception as e:
 return f"🔴 An error occurred during transcription: {e}"
```

## Interview Coach UI development

Now, we have prepared all the ingredients for building the interview coach.

[Gradio](#) is a Python library that allows developers to easily create **interactive web interfaces** for Python code.

You can think of it as a way to turn Python scripts into simple web apps with buttons, text boxes, and image uploaders.

Open the `myapp.py` again to code for the interface.

[Open myapp.py in IDE](#)

**Important:** Copy and paste the codes in each of the following steps to complete `myapp.py`.

### Step 1: Define the core function for interview questions

This function, `next_question()`, is the **central engine** of the AI Interview Coach.

It coordinates all agents — from reading the resume to generating new questions, analyzing responses, and evaluating the candidate.

Here are the inputs of the function:

- `resume_path`: Path to the uploaded resume PDF file.
- `job_str`: Job description text entered by the user.
- `total_number`: The total number of questions to ask in the simulated interview.
- `question_previous`: The last question that was asked.
- `answer_previous`: The candidate's last recorded (spoken) answer.

```
def next_question(resume_path, job_str, total_number, question_previous="", answer_previous=""):
 # Refer to the global variables defined at the front
 global chat_histories, interview_step, resume_summary, job_summary

 # Generate resume_summary if it hasn't been done
 if resume_summary is None:
 resume_summary = extract_text_from_pdf(resume_path)

 # Generate job_summary if it hasn't been done
 if job_summary is None:
 job_summary = Job_Description_Expert(job_str)

 # Converts the user's last voice answer into text using the Whisper speech recognition model
 try:
 answer_previous = transcribe_audio_faster_whisper(answer_previous)
 except:
 answer_previous = ""

 # Update the interview history after the interview starts.
 # The chat history is formatted as a dictionary with keys of interview questions
 # and their values of interviewee's answers.
 if interview_step > 0:
 chat_histories[f"Q{interview_step+1}": {question_previous}] = answer_previous

 # If it's the first question, it defaults to "Tell me about yourself."
 # Otherwise, the Interview Question Action Agent decides whether to:
 # Ask about another resume topic, or Ask a deeper follow-up question.
 # The Interviewer Agent then formulates the next question naturally.
 if interview_step < total_number:
 if interview_step == 0:
 action = None
 else:
 chat_hist_str = str(chat_histories)
 action = Interview_Question_Action(chat_hist_str, resume_summary, job_summary)

 Question_next = Interviewer(resume_summary, job_summary, action, last=False)
else:
```

```

Question_next = Interviewer(resume_summary, job_summary, action=None, last=True)

If the interview ends, the evaluator will evaluate interviewee's performance.
if interview_step >= total_number:
 evaluation = Evaluator(str(chat_histories), job_summary)
 chat_histories = {}
 interview_step = 0
 resume_summary = None
 job_summary = None
else:
 evaluation = "Evaluation Ongoing"

Convert the next interview question to the audio.
question_audio_path = text_to_speech_file(Question_next)
interview_step += 1
return gr.update(value=question_audio_path), gr.update(value=None), gr.update(value="Submit!"), gr.update(value=evaluation)

```

## Step 2: Design the app interface

This step builds the **interactive front-end** of the AI Interview Coach using **Gradio**.

Gradio allows us to create a user-friendly web interface where users can upload resumes, input job descriptions, and participate in voice-based mock interviews, all without writing any HTML or JavaScript.

Overview of the Gradio app development:

1. First you will define all the visual and interactive elements in your web app, including
  - o texts (titles, instructions etc.),
  - o input/output boxes (image, audio and text inputs/outputs), and
  - o actionable buttons (to run the core interview function defined before)
2. Link all buttons you need with the corresponding functions defined.
3. Launch the app.

```

gradio ui
with gr.Blocks() as demo:
 gr.Markdown("# Personalized Interview Coach")
 gr.Markdown('## Upload your pdf resume/CV and copy paste the job description you are applying to:')

 # The first row contains two main inputs side-by-side: resume_input and job_desc_input.
 with gr.Row():
 resume_input = gr.File(label="Upload Resume (PDF)", type='filepath')
 job_desc_input = gr.Textbox(label="Job Description", lines=15)

 # Input for the length of the interview
 gr.Markdown('## Decide the length of your mock interview (from 1 question to 10 questions):')
 num_q_input = gr.Slider(label="Number of Questions", minimum=1, maximum=10, value=5, step=1)

 # Define the window for interviewer question audio
 gr.Markdown('## Click "Start Interview" below to start the Mock Interview!')
 interviewer_question = gr.Audio(label="Interviewer Question", type="filepath")

 # Define the window for interviewee's audio input
 user_answer = gr.Audio(
 sources=["microphone"], # Only allows microphone input
 type="filepath", # Returns the path to the temporary recorded file
 label="Your turn! Record Your Answer."
)

 # Define the start interview button
 start_btn = gr.Button("Start Interview", scale=2, min_width=200)

 # Define the evaluation textbox that shows the final feedback
 gr.Markdown('## Evaluating your performance along the way ...')
 evaluation_textbox = gr.Textbox(label="Performance Evaluation", lines=20)

 # Integrate the next_question() function with the start button
 start_btn.click(
 fn=next_question,
 inputs=[resume_input, job_desc_input, num_q_input, interviewer_question, user_answer],
 outputs=[interviewer_question, user_answer, start_btn, evaluation_textbox]
)
Launch the app
if __name__ == "__main__":
 demo.launch(share=True)

```

## Call your Interview Coach!

Everything is ready, run the app by executing the code:

```
gradio myapp.py
```

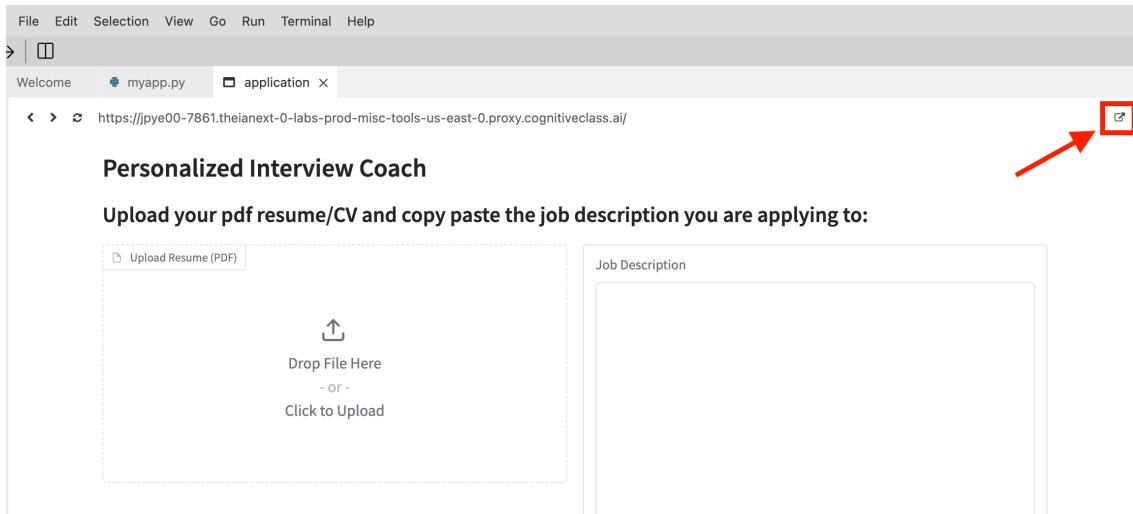
Enter the web app by clicking the button:

[Start Interview Coach](#)

In case the button above doesn't work, try this button:

[Start Interview Coach](#)

**For this particular project, since we are going to use microphone, please click “open in a new window” to run your app in a separate web window.**



You may also enter your web app by clicking the public URL you generated! You can find it in your console, like in the following screenshot (NOTE: your URL will NOT be the same!):

Watching: '/home/project/Interview-Coach'

```
* Running on local URL: http://127.0.0.1:7861
* Running on public URL: https://c46692a47b615192e3.gradio.live
```

Congrats, you've built a fully functional interview coach that could help you succeed in your career!

## Author(s)

[Jianping Ye | Data Scientist Intern @ IBM](#)