```
from google.colab import drive
drive.mount('/content/drive')
```

This imports the Google Colab Drive module and mounts Google Drive to the Colab environment, allowing the user to access and manipulate files stored in Google Drive.

```
%%capture
from google.colab import userdata
hftoken=userdata.get('hftoken')
```

This captures and suppresses the output of a cell to prevent displaying unnecessary results, imports the Colab userdata module, and retrieves a user-specific token stored in Colab's settings.

```
%%capture
!pip install gradio
!pip install huggingface_hub
```

This captures and suppresses the output of the cell, installs the Gradio library to create user interfaces, and installs the huggingface_hub library to interact with Hugging Face's model hub for managing and sharing borrowed models.

```
!pip install gradio
!pip install transformers
!pip install torchaudio
!pip install fasttext
```

This installs Gradio, a library to create user interfaces, Transformers, Hugging Face's library for NLP modes, Torchaudio, an audio processing library for loading, transforming, and manipulating audio files, and fastText, a library for text classification and language detection.

```
!wget
https://dl.fbaipublicfiles.com/fasttext/supervised-models/lid.176.bin
```

This downloads the pre-trained fastText language detection model from Facebook's repository. This model supports detection of 176 languages.

```
import gradio as gr
from transformers import WhisperProcessor,
WhisperForConditionalGeneration, pipeline, EncoderDecoderCache
import torchaudio
import warnings
import fasttext
import pandas as pd
import csv
import os
```

This imports libraries Gradio for user interface, WhisperProcessor and WhisperForConditionalGeneration, Hugging Face models for transcribing and processing audio using OpenAI's Whisper, pipeline, Hugging Face's pipeline for tasks like translation, EncoderDecoderCache for caching for Whisper model to handle past key values, Torchaudio for audio processing tools, Warnings to suppress specific warnings, fastText for language detection, and Pandas, CSV, and OS for handling and saving data.

```
warnings.filterwarnings("ignore", category=UserWarning,
message="PySoundFile failed.*")
```

This suppresses warnings related to PySoundFile when it is unavailable or fails.

```
whisper_model_name = "openai/whisper-large"
processor = WhisperProcessor.from_pretrained(whisper_model_name)
whisper_model =
WhisperForConditionalGeneration.from_pretrained(whisper_model_name)
```

This loads the Whisper model for transcription (speech-to-text) using WhisperProcessor and WhisperForConditionalGeneration.

```
translation_model = pipeline("translation",
model="Helsinki-NLP/opus-mt-ROMANCE-en")
```

This loads the Helsinki-NLP model's opus-mt-ROMANCE-en for translating.

```
lang_model = fasttext.load_model('lid.176.bin')
```

This loads the model fastText for detecting the language of text.

```
def saveData(text, language, translated_text, confidence_score):
   file_path = '/content/drive/MyDrive/IT164/a2prompt.csv'
   file_exists = os.path.isfile(file_path)
with open(file_path, 'a', newline='', encoding='utf-8') as f:
       w = csv.writer(f)
       if not file_exists:
           w.writerow(['Text', 'Language', 'Translation', 'Confidence
Score'])
       w.writerow([text, language, translated_text, confidence_score])
```

This function appends transcription results (text, detected language, translation, and confidence score) to a CSV file located in Google Drive.

```
def transcribe_audio(audio_file, sampling_rate=48000):
    waveform, sr = torchaudio.load(audio_file, normalize=True)
    if sr != 16000:
       transform = torchaudio.transforms.Resample(orig_freq=sr,
new_freq=16000)
       waveform = transform(waveform)
       sr = 16000
    inputs = processor(waveform.squeeze(0).numpy(), return_tensors="pt",
sampling_rate=sr)
    past_key_values = None
    generated_ids = whisper_model.generate(
       inputs["input_features"],
       past_key_values=past_key_values
    )
    if past_key_values is not None:
       past_key_values =
EncoderDecoderCache.from_legacy_cache(past_key_values)
    return processor.decode(generated_ids[0], skip_special_tokens=True)
```

This loads an audio file, resamples it to 16 kHz (for Mac, if needed), processes it with the Whisper model, and returns the transcription.

```
def detect_language(text):
    result = lang_model.predict(text)
    language = result[0][0].replace('__label__', '')
    score = result[1][0]
    return language, score
```

This uses the fastText language detection model to predict the language of the input text and return both the detected language and the confidence score.

```
def translate_text_to_english(text, source_lang="fr"):
    translation = translation_model(text, src_lang=source_lang,
tgt_lang="en")
    return translation[0]['translation_text']
```

This translates text from a specified source language to English using the Hugging Face translation model.

```
def save_to_history(text, language, translation, confidence_score):
    history_data.append([text, language, translation, confidence_score])
    saveData(text, language, translation, confidence_score)
```

This adds the results (transcription, language, translation, confidence score) to a history list and saves it to a CSV file.

```
def process_audio(audio_file):
    transcription = transcribe_audio(audio_file, sampling_rate=48000)
    language, score = detect_language(transcription)
    translated_text = translate_text_to_english(transcription,
source_lang=language)
    save_to_history(transcription, language, translated_text, score)
    return transcription, language, score, translated_text
```

This function handles the full process for an audio file, transcribing it, detecting the language of transcription, translating it to English, and saving the results.

```
def update_vis(radio_value):
    if radio_value == 'show':
        return gr.DataFrame(pd.DataFrame(history_data, columns=["Text",
"Language", "Translation", "Confidence Score"]), visible=True)
    else:
        return gr.DataFrame(pd.DataFrame(history_data, columns=["Text",
"Language", "Translation", "Confidence Score"]), visible=False)
```

This enables the visibility of the history table in the Gradio interface based on user input, and can be either shown or hidden.

```
with gr.Blocks() as demo:
    with gr.Row():
        with gr.Column():
            audio_input = gr.Audio(label="Record your voice",
type="filepath")
            transcription_output = gr.Textbox(label="Transcription")
            language_output = gr.Textbox(label="Detected Language")
            score_output = gr.Textbox(label="Confidence Score")
            translated_output = gr.Textbox(label="Translated Text to
English")
            process_button = gr.Button("Process Audio")
        with gr.Column():
            history = gr.Radio(['show', 'hide'], label="App usage history")
            dataframe = gr.DataFrame(pd.DataFrame(history_data,
columns=["Text", "Language", "Translation", "Confidence Score"]),
visible=False)
        process_button.click(fn=process_audio, inputs=[audio_input],
outputs=[transcription_output, language_output, score_output,
translated_output])
        history.change(fn=update_vis, inputs=history, outputs=dataframe)
demo.launch(debug=True)
```

This block creates a user interface where users can record or upload audio, the system processes the audio, returns transcription, detected language, confidence score, and translation, the history of app use is shown or hidden based on user interaction with a radio button, and all actions are tied to buttons and events for processing and updating the user interface.