

1. Title: TranscribeFlow: Audio Transcript Summarizer

The screenshot shows a dark-themed web application. On the left, there's a sidebar titled "Files / Links" with a table header "# File name / URL Duration". One row is visible: "1 2.Joel_Embiid_interview.mp3 00:04:12". Below this is a "Check automated transcription accuracy" button. The main content area features a large "TranscribeFlow" logo and the subtext "Audio Transcript Summarizer". A transcript section shows two speakers: "Speaker 1" and "Speaker 2". Speaker 1's transcript discusses their life in America, mentioning Canadian and U.S. citizenship, and how they feel about being American. Speaker 2's transcript is a summary of Speaker 1's words. To the right is an "Order summary" box showing "Duration: 00:04:12", "Files: 1", "Total price: \$0.84", and buttons for "Check AI Accuracy", "Checkout", and "Order Human Transcript". At the bottom, there's a "General settings" section with a "Language" dropdown set to "English".

2. **Project Statement:** Today's knowledge workers and students often need to quickly review long audio content, such as lectures, podcasts, or meeting recordings. Listening to an entire recording is inefficient when all that's needed is a summary of the key points. Manually transcribing and then summarizing is a time-consuming two-step process that is a significant barrier to productivity. "TranscribeFlow" aims to solve this by building a web application that takes an audio file, automatically transcribes it using an AI model, and then generates a concise summary of the transcript. This two-stage AI pipeline will allow users to rapidly extract the most important information from audio content, saving valuable time and making information more accessible.

3. Outcomes:

- **Audio-to-Text Transcription:** Accurately transcribe spoken audio from a file.
- **Text Summarization:** Generate a concise summary of the transcribed text.
- **User-Friendly Interface:** A simple web page for uploading an audio file and viewing the transcript and summary.
- **Input Flexibility:** Capable of processing common audio formats (e.g., MP3, WAV).

4. Modules to be Implemented:

- **Audio Upload & Processing Module:**
 - A web form on the front end for users to upload an audio file.
 - A backend API endpoint to receive and temporarily store the audio file.
- **Core AI Inference Module (ASR & NLP):**
 - A pre-trained **Automatic Speech Recognition (ASR)** model (e.g., Wav2Vec2 or Whisper from Hugging Face).

- A pre-trained **text summarization model** (e.g., T5 or BART).
 - Logic to run the ASR model to get the transcript, then run the summarization model on the transcript.
 - **Web Application Server:**
 - A Python web framework (e.g., Flask) to serve the front end and API endpoints.
 - A front end to display the transcript and the summary.
5. **Week-wise module implementation and high-level requirements with output screenshots:**

Milestone 1: Weeks 1-2

- **Module 1: Audio Upload & Backend Setup**
- **High-Level Requirements:**
 - Set up a basic Flask web application.
 - Create a file upload form on a single HTML page.
 - Implement a backend route (`/upload`) to receive the audio file and save it to a temporary location.

Module 1: Audio Upload & Backend Setup

The screenshot shows a user interface for 'TranscribeFlow'. At the top, there's a blue header bar with the 'TranscribeFlow' logo. Below it is a large white area containing a dashed blue border. Inside this border, there's a blue icon of a microphone, followed by the text 'Upload your audio file' and 'Supported formats: MP3, WAV'. Below this is a blue button labeled 'Choose File' with an upward arrow icon. At the bottom of this section is a light blue footer bar with the text 'Milestone 1: Weeks 1-2'.

High-Level Requirements

- Set up basic Flask web application
- Create file upload form on HTML page
- Implement /upload endpoint to receive audio file
- Save file to temporary location on server

Implementation Details

- Flask app initialization
- HTML form with file input
- File validation (MP3, WAV)
- Temporary file storage

Expected Output

- File upload functionality working
- File saved to server directory
- Success confirmation to user

Milestone 2: Weeks 3-4

- **Module 2: ASR Model Integration**
- **High-Level Requirements:**
 - Download a pre-trained ASR model from Hugging Face.
 - Write a Python function to load the model and use it to transcribe a sample audio file.
 - Ensure the transcription is accurate and the model can be successfully loaded.

Module 2: ASR Model Integration

TranscribeFlow - Transcription

Audio Transcript

Today's knowledge workers and students often need to quickly review long audio content, such as lectures, podcasts, or meeting recordings. Listening to an entire recording is inefficient when all that's needed is a summary of the key points...

High-Level Requirements

- Download **pre-trained ASR model** from Hugging Face
- Write function to **load model** and transcribe audio
- Ensure **transcription accuracy**
- Verify successful **model loading**

Implementation Details

- Whisper or Wav2Vec2 model selection
- Audio preprocessing for ASR
- Transcription function

```
def transcribe_audio(audio_path):
    model = whisper.load_model("base")
    result = model.transcribe(audio_path)
    return result["text"]
```

Expected Output

- Accurate text transcription
- Processing time optimization
- Error handling for audio issues

Transcribed using Whisper model from Hugging Face

Milestone 2: Weeks 3-4

Processing audio...

Milestone 3: Weeks 5-6

- **Module 3: Summarization & API Logic**
- **High-Level Requirements:**
 - Download a pre-trained summarization model from Hugging Face.
 - Update the /upload endpoint. After transcribing the audio, pass the transcript to the summarization model.
 - Return both the full transcript and the summary as a JSON response.

Module 3: Summarization & API Logic

TranscribeFlow - Results

Audio Transcript

Today's knowledge workers and students often need to quickly review long audio content, such as lectures, podcasts, or meeting recordings. Listening to an entire

Summary

TranscribeFlow helps knowledge workers and students quickly extract key information from audio content by automatically transcribing and summarizing lectures, podcasts, and

Summarized using T5 model from Hugging Face

Milestone 3: Weeks 5-6

High-Level Requirements

- Download summarization model from Hugging Face
- Update /upload endpoint to use both models
- Return transcript and summary as JSON
- Implement error handling for model failures

Implementation Details

- T5 or BART model selection
- Text preprocessing for summarization
- API response formatting

```
def summarize_text(text):
    model = T5ForConditionalGeneration.from_pretrained("t5-small")
    inputs = tokenizer("summarize: " + text, return_tensors="pt")
    outputs = model.generate(**inputs)
    return tokenizer.decode(outputs[0])
```

Expected Output

- Concise summary generation
- JSON API response
- Key points extraction

```
{ "transcript": "Full text of audio...", "summary": "Key points extracted..." }
```

Milestone 4: Weeks 7-8

- **Module 4: Front-End & Finalization**
- **High-Level Requirements:**
 - Update the front-end JavaScript to send the audio file to the /upload endpoint.
 - Display the received transcript and summary on the web page in separate sections.
 - Add a "processing" indicator to provide user feedback.
 - Prepare the project for a final demonstration.

Module 4: Front-End & Finalization

TranscribeFlow - Complete Application

Upload Audio: lecture_sample.mp3

Processing audio file...

Audio Transcript

Today's knowledge workers and students often need to quickly review long audio content, such as lectures, podcasts, or meeting recordings. Listening to an entire recording is inefficient when all that's needed is a summary of the key points. Manually transcribing and then summarizing is a time-consuming two-step process that is a significant barrier to productivity.

Summary

TranscribeFlow helps knowledge workers and students quickly extract key information from audio content by automatically transcribing and summarizing lectures, podcasts, and meetings, saving time and improving productivity.

Milestone 4: Weeks 7-8

High-Level Requirements

- Update front-end JavaScript to send audio file
- Display transcript and summary in separate sections
- Add processing indicator for user feedback
- Prepare for final demonstration

Implementation Details

- AJAX file upload handling
- Dynamic content rendering
- User experience improvements

File validation | Progress tracking | Error handling
Responsive design

Expected Output

- Complete functional web application
- Seamless user experience
- Clear presentation of results

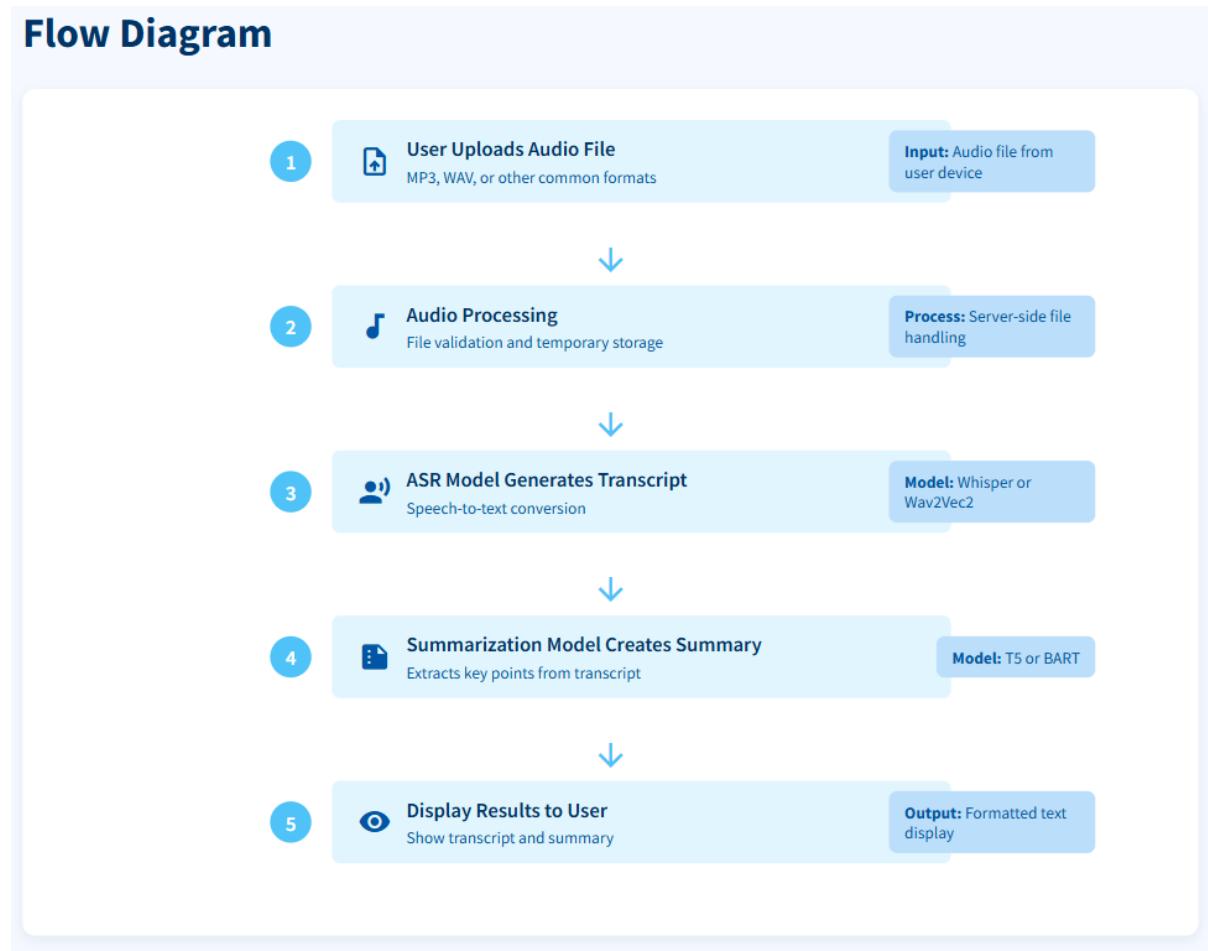
Upload → Process → Display | Real-time feedback
Clean UI

6. Evaluation Criteria:

- **Milestone 1 Evaluation (End of Week 2):**
 - Audio file upload functionality is working and the file is saved to the server.
- **Milestone 2 Evaluation (End of Week 4):**
 - The ASR model is integrated and can successfully transcribe a sample audio file.

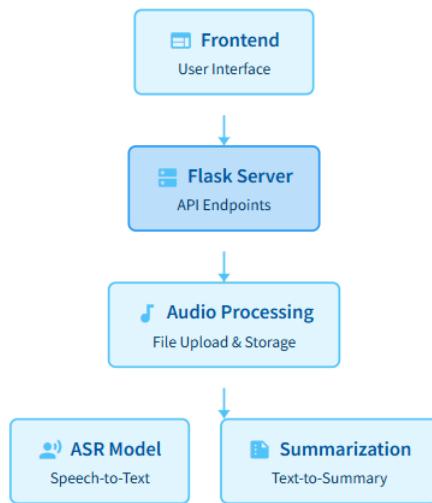
- **Milestone 3 Evaluation (End of Week 6):**
 - The summarization model works correctly and the API returns both the transcript and summary.
- **Milestone 4 Evaluation (End of Week 8):**
 - The full web application is functional, displaying the transcript and summary from an uploaded audio file.

7. Workflow Diagram:



8. Architecture Diagram:

Architecture Diagram



9. Database Schema:

