LLM HACKATHON

TEAM NAME: SILVER ARROWS

TEAM: ISHAAN M C PES1UG22AM071

KISHOR REDDY PES1UG22AM085

MANOJ G B PES1UG22AM092

MOHITH D K PES1UG22AM097

1. Introduction

- **Purpose:** To develop an end-to-end system that generates accurate and structured lecture notes.
- **Input:** Educational video content and accompanying PowerPoint presentations.
- **Output:** Comprehensive lecture notes.
- **Key Technologies:** AI models for speech-to-text conversion and content extraction.
- **Benefit:** Transforms multimedia educational resources into accessible study materials.

2. System Overview

The notebook provides a Python implementation of the lecture notes generation system. It processes a directory containing various educational materials, including video files (e.g., MP4, MKV, AVI, MOV), PowerPoint presentations (PPTX), and PDF documents, to produce organized lecture notes in both PDF and text formats. The system's architecture comprises several key components:

- **Content Extraction Module:** This module extracts textual and structural information from PowerPoint presentations and PDF documents, ensuring that key information from slide decks and supplementary readings is incorporated into the notes.

- **Audio Processing and Transcription Module:** This module extracts audio tracks from video files and transcribes the spoken content using the Whisper ASR model, capturing the verbal delivery of the lecture.
- **LLM-Based Note Synthesis Module:** This module leverages the Gemini 1.5 Flash LLM to synthesize the extracted text and transcripts into coherent and structured lecture notes, organizing the information logically and highlighting key concepts.
- **Output Formatting Module:** This module formats the generated notes into user-friendly PDF and text files, ensuring readability and accessibility.

3. Detailed Processing Pipeline

The system's processing pipeline involves a series of sequential steps:

1. **Data Ingestion:** The main function initiates the process by receiving the path to a directory containing the lecture materials as input.
2. **Content Extraction:**
    - **PPTX Processing:** The extract_pptx_content function extracts text, titles, and presenter notes from PowerPoint presentations. It iterates through each slide, identifying titles, content within shapes, and notes, preserving the hierarchical structure of the presentation.
    - **PDF Processing:** The extract_pdf_text function extracts all text from PDF files, providing a fallback or supplementary source of information.
3. **Audio Processing:**
    - **Audio Extraction:** The extract_audio function utilizes FFmpeg to extract audio tracks from video files in mono 16kHz WAV format. It includes error handling to ensure successful extraction and verifies the existence of both input and output files.
    - **Audio Transcription:** The transcribe_audio function employs the Whisper ASR model (accessed via the transformers library) to transcribe the extracted audio into text. It also incorporates error handling to manage potential transcription failures.
4. **Note Generation:**
    - The extracted text from slides and transcripts is combined and fed into the generate_notes function. This function intelligently chunks the combined text to handle potential LLM input

limitations and uses the Gemini LLM to generate structured lecture notes. The prompting strategy is designed to produce notes with clear headings, subheadings, bullet points, and highlighted key concepts.

5. **Output:**
    - The create_pdf function formats the generated notes into PDF files, using the fpdf library. It handles basic text formatting, including headings and paragraph breaks, to enhance readability. The notes are also saved as plain text files for backup and accessibility.

4. AI Model Implementation Details

- **Speech-to-Text Conversion:**
    - The system leverages the OpenAI Whisper model, accessed through the transformers library, for automatic speech recognition (ASR).
    - The notebook initializes the Whisper model with the "openai/whisper-small" configuration, utilizing a chunk length of 30 seconds and a stride length of 5 seconds to optimize transcription accuracy and memory usage.
- **Language Model for Note Synthesis:**
    - The Gemini 1.5 Flash LLM from Google Generative AI is employed for content summarization, organization, and note generation.
    -
    - The script configures the Gemini API using an API key and employs specific generation parameters (temperature, top_p, top_k, max_output_tokens) to guide the LLM's output.

5. Detailed Technologies and Libraries

The system relies on the following Python libraries and technologies:

- transformers: Provides access to the Whisper ASR model for speech-to-text conversion.
- pypdf: Used for extracting text from PDF files.
- python-pptx: Facilitates the extraction of content from PowerPoint files, including text, titles, and notes.

- ffmpeg-python: Enables the extraction of audio tracks from video files.
- google.generativeai: Provides access to the Gemini 1.5 Flash LLM for note synthesis.
- fpdf: Used for generating PDF files from the synthesized lecture notes.
- os: For interacting with the operating system, such as file path manipulation and directory creation.
- re: For regular expression operations, used in formatting the PDF output.
- matplotlib.pyplot: Though present, it's not directly used in the core logic.
- IPython.display: For displaying Markdown output within the notebook environment.
- tqdm.notebook: For displaying progress bars during processing.

6. Evaluation Methodology (Detailed)

The evaluation of the generated lecture notes should be conducted with a focus on the following key aspects:

- **Accuracy:** This involves a detailed comparison of the transcripts against the original spoken content in the videos to ensure that the text accurately reflects what was said. Special attention should be given to technical terms and proper nouns.
- **Completeness:** The evaluation should verify that all relevant information from both the video (spoken content) and the slides (text, images, charts) is included in the generated notes. This ensures that no crucial details are omitted.
- **Organization:** The logical structure of the notes should be assessed to determine if they are organized in a way that is easy to follow. Headings, subheadings, bullet points, and other formatting elements should be used effectively to create a clear and coherent flow of information.

7. Comparative Analysis (Further Elaboration)

The document requires a comparative analysis of notes generated from at least three different videos. This analysis should consider variations in:

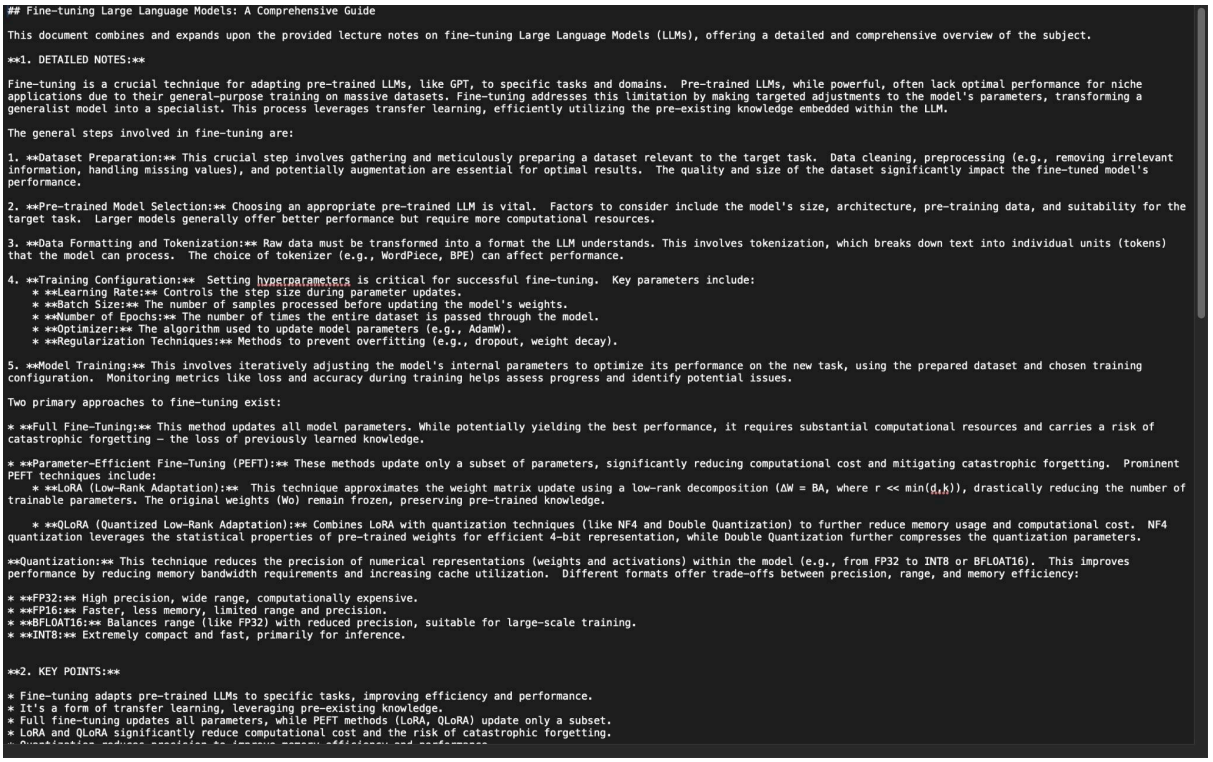
- **Video Length**: Comparing notes from short and long videos to assess the system's scalability and ability to handle varying content volumes.
- **Subject Matte**r: Evaluating notes from lectures on different topics (e.g., mathematics, history, computer science) to determine if the system effectively adapts to different terminologies and content structures.
- **Presentation Style**: Comparing notes from lectures with different delivery styles (e.g., fast-paced, slow-paced, interactive) to assess the impact of speaker behavior on transcription accuracy and note coherence.

8 Limitations and Future Improvements (Detailed)

The notebook provides a solid foundation for a lecture notes generation system, but there are several areas for potential improvement and expansion:

- **Multi-Folder Processing:** The current system is designed to process a single folder at a time. Enhancing it to handle multiple folders would enable batch processing and facilitate the comparative analysis required by the document.
- **Advanced Error Handling:** While the script includes basic error handling, it could be made more robust by implementing more specific exception handling for different types of video processing errors (e.g., codec issues, file corruption).
- **Prompt Optimization:** The prompting strategies used to guide the LLM could be further optimized to improve the quality, accuracy, and structure of the generated notes. Techniques like few-shot learning and prompt engineering could be explored.
- **Speaker Identification:** Integrating speaker identification capabilities would allow the system to differentiate between different speakers in the video, improving the clarity and organization of the notes, especially for lectures with multiple presenters.

9. Sample output



```
## Fine-tuning Large Language Models: A Comprehensive Guide

This document combines and expands upon the provided lecture notes on fine-tuning Large Language Models (LLMs), offering a detailed and comprehensive overview of the subject.

**1. DETAILED NOTES:**

Fine-tuning is a crucial technique for adapting pre-trained LLMs, like GPT, to specific tasks and domains.  Pre-trained LLMs, while powerful, often lack optimal performance for niche applications due to their general-purpose training on massive datasets. Fine-tuning addresses this limitation by making targeted adjustments to the model's parameters, transforming a generalist model into a specialist. This process leverages transfer learning, efficiently utilizing the pre-existing knowledge embedded within the LLM.

The general steps involved in fine-tuning are:

1. **Dataset Preparation:** This crucial step involves gathering and meticulously preparing a dataset relevant to the target task.  Data cleaning, preprocessing (e.g., removing irrelevant information, handling missing values), and potentially augmentation are essential for optimal results.  The quality and size of the dataset significantly impact the fine-tuned model's performance.

2. **Pre-trained Model Selection:** Choosing an appropriate pre-trained LLM is vital.  Factors to consider include the model's size, architecture, pre-training data, and suitability for the target task.  Larger models generally offer better performance but require more computational resources.

3. **Data Formatting and Tokenization:** Raw data must be transformed into a format the LLM understands. This involves tokenization, which breaks down text into individual units (tokens) that the model can process.  The choice of tokenizer (e.g., WordPiece, BPE) can affect performance.

4. **Training Configuration:**  Setting hyperparameters is critical for successful fine-tuning.  Key parameters include:
    * **Learning Rate:** Controls the step size during parameter updates.
    * **Batch Size:** The number of samples processed before updating the model's weights.
    * **Number of Epochs:** The number of times the entire dataset is passed through the model.
    * **Optimizer:** The algorithm used to update model parameters (e.g., AdamW).
    * **Regularization Techniques:** Methods to prevent overfitting (e.g., dropout, weight decay).

5. **Model Training:** This involves iteratively adjusting the model's internal parameters to optimize its performance on the new task, using the prepared dataset and chosen training configuration.  Monitoring metrics like loss and accuracy during training helps assess progress and identify potential issues.

Two primary approaches to fine-tuning exist:

* **Full Fine-Tuning:** This method updates all model parameters. While potentially yielding the best performance, it requires substantial computational resources and carries a risk of catastrophic forgetting – the loss of previously learned knowledge.

* **Parameter-Efficient Fine-Tuning (PEFT):** These methods update only a subset of parameters, significantly reducing computational cost and mitigating catastrophic forgetting.  Prominent PEFT techniques include:
    * **LoRA (Low-Rank Adaptation):**  This technique approximates the weight matrix update using a low-rank decomposition (ΔW = BA, where r << min(d,k)), drastically reducing the number of trainable parameters. The original weights (Wo) remain frozen, preserving pre-trained knowledge.

    * **QLoRA (Quantized Low-Rank Adaptation):** Combines LoRA with quantization techniques (like NF4 and Double Quantization) to further reduce memory usage and computational cost.  NF4 quantization leverages the statistical properties of pre-trained weights for efficient 4-bit representation, while Double Quantization further compresses the quantization parameters.

**Quantization:** This technique reduces the precision of numerical representations (weights and activations) within the model (e.g., from FP32 to INT8 or BFLOAT16).  This improves performance by reducing memory bandwidth requirements and increasing cache utilization.  Different formats offer trade-offs between precision, range, and memory efficiency:

* **FP32:** High precision, wide range, computationally expensive.
* **FP16:** Faster, less memory, limited range and precision.
* **BFLOAT16:** Balances range (like FP32) with reduced precision, suitable for large-scale training.
* **INT8:** Extremely compact and fast, primarily for inference.


**2. KEY POINTS:**

* Fine-tuning adapts pre-trained LLMs to specific tasks, improving efficiency and performance.
* It's a form of transfer learning, leveraging pre-existing knowledge.
* Full fine-tuning updates all parameters, while PEFT methods (LoRA, QLoRA) update only a subset.
* LoRA and QLoRA significantly reduce computational cost and the risk of catastrophic forgetting.
* Quantization reduces precision to improve memory efficiency and performance.
```

10. Individual contributions

Ishaan M C:

- `extract_audio` function
- `transcribe_audio` function
- `transformers` library (Whisper ASR)

Kishor Reddy:

- `extract_pptx_content` function
- `extract_pdf_text` function
- `python-pptx` library
- `pypdf` library

Mohith D K:

- `generate_notes` function
- `google.generativeai` library (Gemini LLM)

MAnoj G B:

- `create_pdf` function
- `fpdf` library
- `re` library (for PDF formatting)