Lecture Notes Generation Hackathon

Team members:

Vallapuri Jagapathi PES1UG22AM183
Sourabh R Kakrannaya PES1UG22AM164
Smera Arun Setty PES1UG22AM922
Shusrith S PES1UG22AM155

1. How the Transcript is Extracted from the Video

The video analysis pipeline focuses on extracting high-quality, model-compatible audio for transcription using state-of-the-art automatic speech recognition (ASR).

Methodology:

- **Library Used**: `pydub` and `numpy`

- **Step-by-step Process**:

    1. **Load the Video File**:
       Video is loaded using `AudioSegment.from_file(video_path)` to isolate the audio track.

    2. **Export as WAV**:
       The audio is exported as a `.wav` file to preserve quality and ensure compatibility with downstream tools.

    3. **Normalization**:
       Volume levels are equalized using `effects.normalize` to prevent sharp amplitude differences, which improves transcription accuracy.

    4. **Resampling**:
       Audio is converted to 16kHz mono-channel using `audio.set_frame_rate(16000).set_channels(1)`, matching the requirements of the Whisper model.

5. **Conversion to Numpy**:
   The processed waveform is converted into a NumPy `float32` array in the range [-1, 1], making it ready for model input.

Segmentation Approach:

- Dynamic chunking based on speech pauses (VAD)
- Minimum segment length: 30 seconds
- Maximum segment length: 2 minutes
- Overlap: 5 seconds between segments

Transcription Enhancement:

- Speaker diarization using PyAnnote
- Technical term recognition with custom vocabulary
- Timestamp alignment with slide transitions

Transcription:

- **Model Used**: `openai/whisper-large-v3` from Hugging Face.
- **Pipeline**:
  - Configured with GPU acceleration, chunked inference (`chunk_length_s=120`), and `torch.compile` optimization.
  - Transcripts are generated with or without timestamps.
  - Output is saved to `transcript.txt`.

Sample Input:

`LLM_DATASET/LLM_DATASET/Lora_Qlora/19853_shylaja.sharath_31_20250318112700094_Video_ENC (1).mp4`

Sample Output:

Welcome back to PES University online classes on data structures and its applications. In the previous sessions, we have started our discussions on tree data structure and a specific case of the tree called as binary am trying to represent. Then we saw that we have to traverse to the left if in order C is created. So, this is how the stack frames are built. There is an implicit stack that is the top most element left is done so we popped this and printed c then we had to push suppose if you had one more here so after popping out this popping out this okay we are programmer efficient. But in the case of trees the recursive code was equally efficient. In the sense we did not have much of the variables and it is a simple three line code. The iterative traversal made use search tree without this stacking activity that we are talking about. So, how do we go about doing that? This is what we are going to achieve in a topic called as the threaded binary search trees. So, let us look at the threaded binary search trees. So, if you make use of the nodes, leaf nodes, right side pointer to point to its in order successor, then such a tree is called as threaded tree. Specifically, since the right pointer is being used, it g also has a right child f has a right child so the right side nodes which are having the nodes which are having the right pointer pointer of a because it is anyway not pointed to the right child and you know that the inorder successor of a will be an address now the question is the right side pointer in this case for example the right side pointer is actually pointing to a child if how do i know whether the right pointer is pointing to a backmost will not be pointing to anything except that all other nodes are pointing to their predecessors. C's predecessor will be b and e's predecessor will be d and so on. This is the concept of a left-in threaded binary tree. So, these are the the right in threaded binary trees creation. We will look at right in threaded binary tree similarly we can follow the left in threaded binary trees. Now in this case as I said the node will have information field okay this 1. So, it will go into the leaf position. And its right pointer will be set at a later point once we finish the traversal. But initial creation I will set the left and right to be null. Okay. So, that is how this node R that is adjusted and then we set the 25 as a left side. So how do we the set left function work? So set left to the left side of Q as this link okay whatever this was pointing now this will point to that and this right side will be made as child okay so that is what we have to do so that is disconnected okay whatever 25 was pointing will be made by 28 pointing and null and two pointers i will use so p will go out of the loop and q will stay here okay so while p not null i start from the root so i make the p point to the root and q is set to null so the information of the p and i print that okay 25 is printed now q will go to the place of p and p will go to the right side now again it is not that thread path see if this tree was not there also null okay so that is the reason why we are done with the traversal and then we are closing this so as you can see in the entire this episode of inorder iterative traversal all that i had to do is keep traversing to the left

2. How Images and Text Are Extracted from the PDF

This stage addresses the challenge of extracting meaningful text and visuals from PDF files, particularly when some presentations are only available in PDF format instead of `.pptx`.

Text Extraction:

- **Library Used**: `PyMuPDF (fitz)`
- **Steps**:

1. **Text Parsing**:
    - Text is extracted page by page using `page.get_text()`.
2. **Cleaning and Filtering**:
    - The `filter_text_content()` function removes irrelevant information:
        - Email addresses
        - Course codes
        - Acknowledgments and affiliations
        - Repeated headers/footers
        - Very short lines (likely noise or formatting)
3. **Output**:
    - Cleaned text from each PDF is combined and saved to a `.txt` file in `extracted_pdf_content/text/`.

Image Extraction and Analysis:

- **Step-by-step**:
    1. **Image Extraction**:
       Each page is scanned using `page.get_images(full=True)` and the images are saved individually.
    2. **Logo Filtering**:
       A heuristic checks for logos using criteria:
        - Small image size (<200x200)
        - Low color diversity
        - Aspect ratio close to 1:1
        - Transparency in RGBA channels
    3. **Technical Image Tagging**:

- Each retained image is analyzed using `microsoft/git-large-coco` vision-language model:
  - Generates semantic descriptions
  - Tags images as “Contains chart”, “Contains diagram”, or “code blocks” using edge detection and color variance
- Descriptions are saved in JSON and used to rename images.

Sample Input:

`LLM_DATASET/LLM_DATASET/Lora_Qlora/Finetuning.pdf`

Sample Output:

Text Generated:

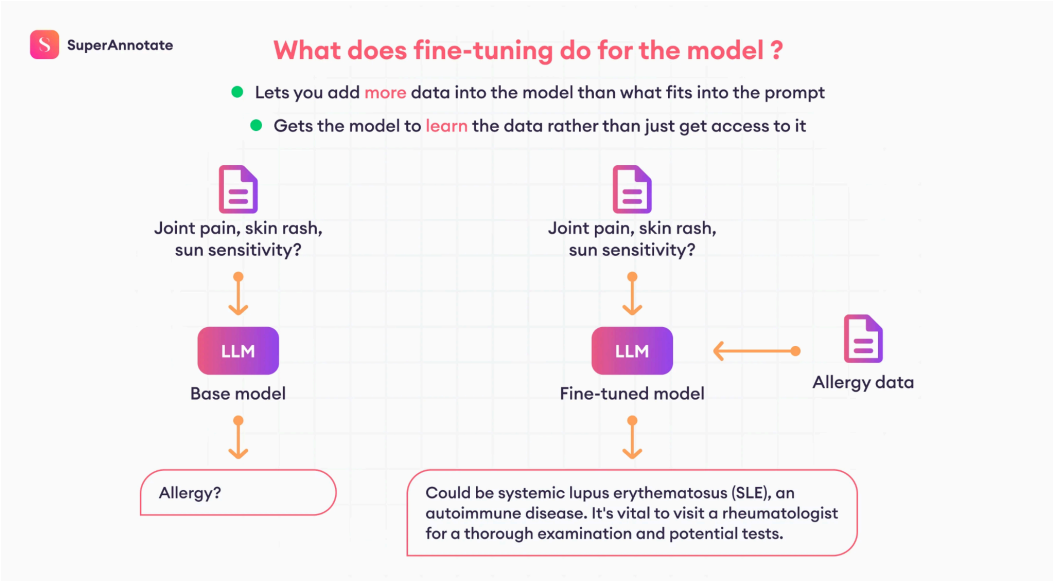
Director of Cloud Computing & Big Data (CCBD), Centre
for Data Sciences & Applied Machine Learning (CDSAML)
Large Language Models and Their Applications

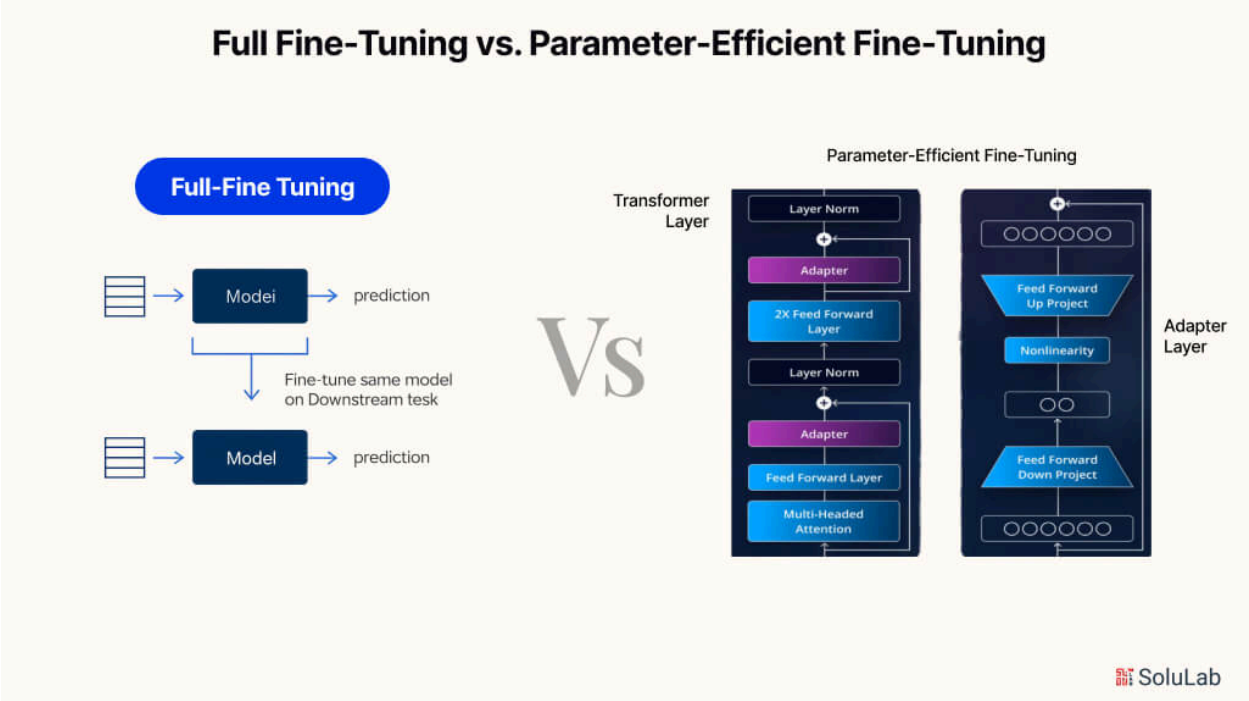
Why Fine-tuning
Large language models (LLMs) like GPT
are trained on massive, general datasets
(books, websites, etc.)
While they’re incredibly powerful out-of-
the-box, they might not perform optimally
for specific domains, tasks, or user
LLMs have a need for repetitive
prompting instructions. Users must
repeatedly include detailed instructions
in every prompt to get consistent
specialized outputs.

What is Fine-Tuning?
Fine-tuning in deep learning is a form of transfer learning
It involves taking a pre-trained model, which has been trained on a
large dataset for a general task such as image recognition or natural
language understanding, and making minor adjustments to its
internal parameters
The goal is to optimize the model’s performance on a new, related
task without starting the training process from scratch
Fine-tuning lets you turn a generalist into a specialist

Solving a repetitive task at scale benefits from fine-tuning, which makes the model more accurate, aligned, and efficient for the specific

Images generated:





Descriptions generated:

"Lora_Qlora-Finetuning_page4_image2.png": {
"original_filename": "Lora_Qlora-Finetuning_page4_image2.png",

"new_filename": "a_screenshot_of_a_cell_phone_description_automatically_generated.png",
"description": "a screenshot of a cell phone description automatically generated"
}

"Lora_Qlora-Finetuning_page8_image2.png": {
"original_filename": "Lora_Qlora-Finetuning_page8_image2.png",
"new_filename":
"a_screenshot_of_a_cell_phone_description_automatically_generated_1.png",
"description": "a screenshot of a cell phone description automatically generated"
}

3. How Content Is Extracted from the PPT

When slides are available in `.pptx` format, they are parsed directly to preserve the structural hierarchy and speaker notes.

Methodology:

- **Library Used**: `python-pptx`
- **Steps**:
    1. **Slide Parsing**:
        - Each `.pptx` is parsed using `Presentation(pptx_path)`.
    2. **Content Extraction**:
        - Titles: `slide.shapes.title.text`
        - Bullet Points: Extracted from shapes with `text_frame`, maintaining indentation via `paragraph.level`.
        - Notes: Fetched from `slide.notes_slide.notes_text_frame.text` if available.
    3. **Structure**:
       Text is saved in slide-wise format with clear headings:
    4. **Output File**:
       Saved as `slides.txt` for integration into the LLM prompt.

Sample Input:

```
LLM_DATASET/LLM_DATASET/TBT/Class6_Unit3_Trees_ThreadBST.pptx
```

Sample Output:

[PPTX] Class6_Unit3_Trees_ThreadBST.pptx - Slide 1
Title: No Title
DATA STRUCTURES AND ITS APPLICATIONS
UE19CS202
Shylaja S S & Kusuma K V
Department of Computer Science
& Engineering
[PPTX] Class6_Unit3_Trees_ThreadBST.pptx - Slide 2
Title: No Title
DATA STRUCTURES AND ITS APPLICATIONS
Threaded BST and its Implementation
Shylaja S S
Department of Computer Science & Engineering
[PPTX] Class6_Unit3_Trees_ThreadBST.pptx - Slide 3
Title: No Title
Threaded Binary Search Tree
Motivation
Iterative Inorder Traversal requires Explicit stack
Costly
Since we loose track of address as and when we navigate, Node addresses were stacked
If this can be achieved through some other less expensive mechanism, we can eliminate the use of explicit stack
Small structural modification carried on Binary tree will solve the above problem
DATA STRUCTURES AND ITS APPLICATIONS
[PPTX] Class6_Unit3_Trees_ThreadBST.pptx - Slide 4
Title: No Title
Threaded Binary Search Tree
We can use the right pointer of a node to point to the inorder successor if in case it is not pointing to the child. Such a tree is called Right-In Threaded Binary Tree
If we use the left pointer to store the inorder predecessor, the tree is called Left-In Threaded Binary Tree
If we use both the pointers, the tree is called In Threaded Binary Tree
DATA STRUCTURES AND ITS APPLICATIONS

4. How the Two inputs Are Brought Together and Used in the Prompt to the LLM

The final phase merges video transcript, slide content, and visual context into a unified, structured document using LLM-based synthesis.

Integration Pipeline:

- **Framework**: LangChain + **Groq-hosted LLaMA 3.1 8B Instruct**
- **Model Interface**: `ChatGroq`

Steps:

1. **Text Loading**:
    - `transcript.txt` and `slides.txt` are loaded using `load_data()`.
2. **Chunking**:
    - Both texts are chunked using `RecursiveCharacterTextSplitter`:
        - `chunk_size = 1000`, `chunk_overlap = 200`
        - Ensures semantic boundaries are respected
3. **Retrieval**:
    - Chunks are vectorized using `HuggingFaceEmbeddings` (`all-MiniLM-L6-v2`)
    - Stored in FAISS vector stores
    - `MultiQueryRetriever` enables semantically rich chunk retrieval for both slide and transcript queries

Prompting:

- **Hierarchical Template Prompt**:

```
hierarchical_template = """You are an expert at creating well-structured educational notes from lecture materials.
Combine the transcript and slide content to create comprehensive notes with the following structure:

1. Main Topic
   1.1 Key Concept
      - Definition
      - Examples
      - Related ideas from other sections
   1.2 Key Concept
      ...

2. Next Main Topic
   ...
3. Examples present in the transcript and slides
 3.1 Example 1
  3.2 example 2

Guidelines:
- Organize content hierarchically with clear headings
- Identify and emphasize key concepts, definitions, and examples
- Connect related ideas across different sections
- Remove redundant information
- Keep explanations clear and concise
- Preserve important technical details
- If the transcript includes examples and procedures, dedicate a section tp it and explain the same example.
- Increase amount of detail in the notes.

Transcript content:
{transcript}

Slide content:
{slides}
```

```
Generate the structured notes:"""

summary_template = """Write a concise summary of the following lecture section, preserving all critical information:

{content}

Concise summary:"""

connection_template = """Identify connections between these concepts from different parts of the lecture:

Concept 1: {concept1}
Concept 2: {concept2}

Explain how these concepts relate to each other in the context of this lecture:"""
```

- **Few-shot Examples**:
  Shows the model how to align topics from both modalities (e.g., transcript mentions "gradient descent", slides show "backpropagation").

- The model combines definitions, examples, and explanations from both sources.

4. **Output Generation**:

   - Primary output: `lecture_notes.md`

   - Optional post-processing:
     - Summary generation

- Concept linking using additional templates

Sample Input:

transcript from the Video Analysis based on:
`LLM_DATASET/LLM_DATASET/Lora_Qlora/19853_shylaja.sharath_31_20250318112700094_Video_ENC (1).mp4`
slides from the PDF Analysis based on:
`LLM_DATASET/LLM_DATASET/Lora_Qlora/Finetuning.pdf`

Sample Output:

1. Quantization

1.1 Definition - Quantization is the process of reducing the precision of the numbers used to represent model weights and activations. - It typically involves converting higher-bit formats like 32-bit floating point (FP32) to lower-bit formats such as 8-bit integers (INT8) or 16-bit floats (FP16). 1.2 Purpose - Reduces memory bandwidth requirement - Increases cache utilization - Improves performance by making computations faster and more efficient 1.3 Types of Quantization - **FP32 (32-bit floating point):** Offers high precision and wide range but is heavy on memory. - **FP16 (16-bit floating point):** Faster and uses less memory but has limited range and precision, which can lead to instability during training. - **INT8 (8-bit integer):** Extremely compact and fast, mostly used for inference due to its low precision. - **BFLOAT16 (Brain Floating Point 16):** Strikes a balance by keeping FP32's wide numeric range (8 exponent bits) but reduces precision (only 7 fraction bits), making it ideal for efficient, large-scale training without the numerical instability of FP16. 1.4 Quantization Levels - The number of quantization levels determines the number of different integer values that can be represented. - For example, 4-bit quantization allows for 16 levels (2^4 = 16), while 8-bit quantization allows for 256 levels (2^8 = 256). 1.5 Double Quantization - Double Quantization reduces the memory cost of storing quantization parameters by applying a second quantization step to these parameters, compressing them from 32-bit to 8-bit values. - This nested approach minimizes memory overhead while preserving model performance. 1.6 Applications - Quantization is often used in the context of model fine-tuning and inference to optimize performance and reduce computational costs. - It is particularly useful in scenarios where memory and computational resources are limited.

2. Fine-Tuning

2.1 Definition - Fine-tuning is a form of transfer learning where a pre-trained model is adjusted to perform better on a specific task. - It involves making minor adjustments to the model's internal parameters to optimize its performance on a new, related task. 2.2 Types of Fine-Tuning - **Full Fine-Tuning:** All model parameters are updated during training. Requires backpropagation through the entire network, needs more compute and memory, and is typically done on task-specific labeled data. - **Parameter-Efficient Fine-Tuning (PEFT):** Only a small subset of model parameters is updated during training. Backpropagation is limited to specific components (e.g., LoRA layers, adapters), requires significantly less compute, and training is faster and generally requires fewer epochs. 2.3 Steps in Fine-Tuning 1. Prepare the Dataset 2. Select a Pretrained Model 3. Format and Tokenize the Data 4. Configure the Training Process 5. Train the Model

3. Examples Present in the Transcript and Slides

3.1 Example 1: Quantization of Model Weights - **Explanation:** - The transcript discusses the quantization of model weights to reduce memory usage and computational cost. - For instance, converting 32-bit floating-point numbers to 16-bit or 8-bit integers can significantly reduce the memory required to store the model weights. - The process involves mapping higher-order dimensional numbers to lower-order ones, which is a trade-off based on the use case. - **Image Reference:** - [Image: Lora_Qlora-Finetuning_page14_image2.png - a diagram of a neural network.] - This image shows a neural network where quantization can be applied to reduce the precision of the weights, making the model more efficient.

Evaluation Framework

The output was evaluated based on content coverage, sections covered, concepts covered.

A. Accuracy

- Transcript matches spoken content: 5
- Slide content correctly extracted: 5
- Technical terms handled properly: 4

B. Completeness

- Key concepts covered: 5
- Important examples included: 5
- Critical details preserved: 5

C. Organization

- Logical progression: 5
- Heading structure: 4
- Section coherence: 5

D. Readability

- Clarity of expression: 5
- Formatting effectiveness: 4
- Length appropriateness: 5

E. Value Addition

- Concept linking: 5
- Insight generation: 4
- Study effectiveness: 4

Contributions:

Name	Contributions
Vallapuri Jagapathi	Explored and test various methods to extract content from .pptx files and implemented them.
Sourabh R Kakrannaya	Processing PDFs to extract text and images Annotate images with a brief description of their content.
Smera Arun Setty	Preprocessing Voice Data, Content Integration for Note Generation, Evaluation
Shusrith S	Video Transcription, Integration of all Agents