
FAST National University of Computer and Emerging Sciences

Department of Data Science

# Ola - All You Need in a Teacher's Best Assistant

Project Report

Mannan Ul Haq 22L-7556
Huzaifa Amir 22L-7518
Muhammad Abdullah 22L-7522

*Supervisor:* Ms. Esha Tur Razia Babar

A project report submitted in fulfilment of the requirements of
the **National University of Computer and Emerging Sciences (FAST-NUCES)**
for the degree of
**Bachelor of Science** in *Data Science*

May 9, 2025

# Abstract

The education sector is undergoing a rapid digital transformation, where artificial intelligence (AI) is increasingly playing a pivotal role in supporting teachers and enhancing the learning experience. However, educators still face significant challenges in managing lectures, assignments, and student engagement efficiently. This project presents **Ola – All You Need in a Teacher's Best Assistant**, an AI-powered assistant designed to automate key teaching tasks and reduce administrative burden.

As part of this term project, we focus on developing and evaluating two foundational modules of Ola: (1) **Automated Lecture Note Generation**, and (2) **Whiteboard Text Refinement**. For the lecture note generation module, we extract text from lecture slides using OCR and convert audio into text using a fine-tuned Whisper model. We then integrate both sources and apply a T5-based model fine-tuned on synthetic data generated via LLaMA to summarize and enhance the final notes. For whiteboard refinement, we implement a drawing canvas that captures handwritten input and apply a deep Capsule Network trained on the EMNIST dataset to recognize individual characters. The refined text is then rendered back to the digital whiteboard.

Together, these components demonstrate the feasibility of integrating AI in classroom environments to assist educators in real-time. The architecture and modules developed in this project lay the groundwork for future expansion.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background

The education sector is experiencing a significant transformation due to the integration of Artificial Intelligence (AI), which offers powerful tools to enhance both teaching and learning experiences. With the growing prevalence of digital content, hybrid classrooms, and virtual education, teachers are facing various challenges. These include managing lectures, grading assignments, organizing materials, and ensuring student engagement. Traditionally, these tasks require substantial time and effort, often leading to errors and diminishing the time available for more meaningful teaching activities.

In response to these challenges, we introduce *Ola*, an AI-powered tool designed to assist educators by automating key aspects of lecture management and classroom interaction. Ola aims to provide real-time lecture transcription, refinement of whiteboard content, and the automatic generation of comprehensive notes using advanced AI and Natural Language Processing (NLP) techniques. By reducing the manual workload of educators and improving classroom dynamics, Ola seeks to foster a more efficient and engaging teaching ecosystem.

## 1.2 Problem Statement

Teachers dedicate a significant amount of their time to administrative and repetitive tasks such as preparing lecture notes, clearly writing on whiteboards, and providing feedback to students. Despite the availability of various digital tools, current systems offer limited automation, especially when it comes to integrating different types of lecture content, such as slides and spoken audio. Additionally, handwritten content on whiteboards is often unclear or inconsistent, making it difficult to store, retrieve, or share with students in a clear and structured format.

There is an urgent need for an intelligent assistant that can automate several aspects of the teaching process. This includes transcribing lecture audio, extracting content from slides, recognizing and refining whiteboard text, and generating well-structured, readable lecture notes with contextual explanations. By leveraging cutting-edge deep learning and NLP techniques to automate these processes, the system can not only enhance teacher productivity but also improve students' access to high-quality, easily digestible learning materials.

## 1.3 Solution Approach

To address these challenges, we propose **Ola**, an AI-powered teacher's assistant. This project is a part of our **Final Year Project (FYP)** for the upcoming year, but for now, we aim to develop its foundational architecture and define its core components as a term project.

Ola will assist educators by automating key tasks, including **lecture transcription, quiz and assignment grading, plagiarism detection, and student performance analytics**. The goal is to enhance the teaching experience, reduce the teacher's workload, and improve student engagement through AI-driven automation and insights.

For the **term project**, we will focus on implementing the following foundational features:

1. **Lecture Notes Generation** – Automatically generating structured notes from lecture audio and slide content, helping teachers streamline note-taking and ensure consistency.
2. **Whiteboard Text Refinement** – Recognizing and refining handwritten text on a real-time drawing board to improve clarity and presentation for students.

## 1.4 Aims and Objectives

**Aims:** The primary aim of this project is to design and develop an AI-powered teacher assistant system that automates the generation of lecture notes and refines handwritten whiteboard content. This will allow educators to focus more on teaching and student interaction, while the system handles routine tasks.

**Objectives:**

- Develop a module that extracts and combines text from lecture slides and recorded audio.
- Fine-tune a T5-based model for summarizing and enriching the extracted content into coherent, structured lecture notes.
- Design an intuitive whiteboard interface that allows educators to input handwritten content, which can then be recognized and refined by the system.
- Train a deep capsule network for recognizing and refining handwritten whiteboard text, improving legibility and ensuring accuracy in the generated notes.

# Chapter 2

# Literature Review

This chapter reviews the key technological foundations and relevant studies that form the basis for our proposed system: an AI-driven educational assistant called **Ola**. The system integrates lecture note generation and real-time whiteboard text refinement. This review explores prior work and tools in audio transcription, slide text extraction, keyword and explanation generation, summarization, and handwritten text recognition.

## 2.1 Lecture Notes Generation

### 2.1.1 Audio Transcription

**Tool: Whisper AI**

Automatic Speech Recognition (ASR) has significantly improved with models like Whisper AI, an open-source deep learning model developed by OpenAI. Whisper leverages a large multilingual and multitask supervised dataset for training, achieving transcription accuracies exceeding 95% on clean audio (1). It is particularly robust to background noise and low-resource languages, making it ideal for noisy classroom environments (2). Its ability to process long-form lecture audio with minimal latency makes it highly effective for real-time applications.

### 2.1.2 Slide Text Extraction

**Tool: Tesseract OCR**

Tesseract OCR, developed by Google, remains one of the most reliable tools for printed text recognition. It can achieve up to 96.7% accuracy under optimal conditions (3). However, its performance decreases with low contrast slides or skewed content (4). To improve OCR results in Ola, preprocessing techniques such as grayscale conversion, skew correction, and adaptive thresholding are applied prior to OCR, thereby enhancing text clarity and segmentation.

### 2.1.3 Keyword Extraction and Explanation Generation

**Model: T5 fine-tuned on synthetic datasets**

Transformer-based models such as T5 (Text-to-Text Transfer Transformer) have shown strong performance in generating natural language explanations when fine-tuned on task-specific data (5). In Ola, we extract headings and subheadings using keyword detection methods and generate corresponding explanations using a fine-tuned T5 model trained on synthetic QA-style lecture datasets. Similar work has been done in the ELI5 dataset, where models generate long-form explanations from web data (6).

### 2.1.4 Context-Aware Merging of Slide Text, Audio, and Explanations

**Technique: Embedding Similarity with Paraphrase-MiniLM**

To merge slide content, audio transcription, and generated explanations contextually, we compute sentence embeddings using the paraphrase-mpnet-base-v2 model and compare similarity scores using cosine distance. Similar strategies are used in multi-modal fusion tasks in NLP to ensure contextual alignment (7). This embedding-based merging ensures that explanations are logically linked with their corresponding slide content and spoken references.

### 2.1.5 Summarization

**Model: T5 fine-tuned on synthetic datasets**

Abstractive summarization has been greatly improved through transformer models like T5 and BART, which outperform traditional extractive techniques by understanding deeper context and generating fluent text (5). These models are particularly valuable for lecture summarization where contextual flow is important. However, studies have shown that flat summarizers lack hierarchical note structuring (8). Ola addresses this by introducing topic segmentation prior to summarization, a method aligned with practices in hierarchical document summarization research (9).

## 2.2 Whiteboard Text Refinement

### 2.2.1 Handwritten Text Recognition (HTR)

**Model: Deep Capsule Networks**

Traditional CNNs are widely used in handwritten text recognition but fail to retain spatial relationships, especially in cursive or non-uniform handwriting (10). Capsule Networks, proposed by Hinton et al., maintain part-whole relationships and spatial hierarchies, making them better suited for character-level recognition in complex handwriting (11). We implement a custom Deep Capsule Network trained on the EMNIST dataset, targeting high accuracy in educational whiteboard scenarios.

### 2.2.2 Auto-Detection of Handwritten Input

**Model: Inactivity-Based Timeout Mechanism**

Gesture-based handwriting detection has been explored in IoT-enabled systems, but often relies on explicit triggers (12). To streamline usability in Ola, we implement an inactivity-based timeout detector. This approach monitors user pauses and automatically initiates handwriting recognition, aligning with time-series motion tracking studies for handwriting segmentation (13).

### 2.2.3 Image Preprocessing for Handwriting Recognition

**Techniques: Grayscale, Binarization, and Noise Removal**

Effective preprocessing is critical for improving HTR accuracy. Studies demonstrate that grayscale conversion, adaptive thresholding, and Gaussian noise removal can improve OCR performance by 25–30% in noisy environments (14). CNN-based preprocessing pipelines have also been proposed, improving HTR accuracy from 74% to over 88% on challenging datasets (15). Ola uses a combination of classical and deep-learning-based techniques to enhance whiteboard input clarity.

### 2.2.4 Spell Correction for Handwritten Text

**Tool: PySpellChecker**

Spelling correction significantly improves the legibility of OCR outputs. Dictionary-based spell correction systems have achieved 20–30% accuracy improvements in post-OCR tasks, especially when used alongside context-aware language models (16). However, inconsistencies in cursive handwriting or connected letters remain challenging. Ola integrates PySpellChecker with domain-specific vocabulary augmentation to adapt to academic writing.

### 2.2.5 AI-Based Text Refinement on Whiteboard

**Tool: Tkinter GUI with AI-Powered Text Replacement**

While most digital whiteboards focus on content amplification or digital ink rendering, few systems offer real-time AI-based text replacement. Skurzok et al. propose a remote learning whiteboard with enhancement tools, but without semantic processing (17). In Ola, we replace low-resolution handwritten text with clean digital fonts using an AI pipeline while preserving spatial layout and visual context—improving readability and aesthetics.

# Chapter 3

# Methodology

This chapter outlines the methodological approach adopted for the development and implementation of each core feature of the system. The methodology is divided by individual features, detailing the data sources, system design, and implementation strategy used for each.

## 3.1 Lecture Notes Generator

### 3.1.1 Overview

The Lecture Notes Generator is designed to automate the process of producing structured notes from lecture slides and audio recordings. It utilizes speech recognition, document parsing, and transformer-based summarization techniques to generate comprehensive, human-readable notes. The system takes PDF slides and real-time audio as inputs, processes them through a pipeline, and outputs lecture summaries aligned with the slide context.

### 3.1.2 Data Collection

The system processes two primary input sources:

- **PDF Slides:** Provided by the user before the lecture. For now, we assume a fixed slide format that follows the below structure.

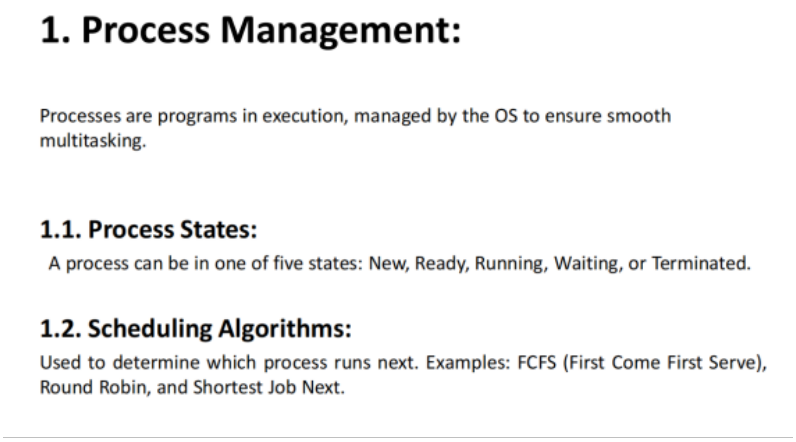


Figure 3.1: Sample slide

- **Lecture Audio:** Recorded during the lecture using the system's built-in microphone interface. This will be a live recording.

**Process Management:**

*"A process is a program in execution, and the operating system manages it to enable multitasking. A process moves through different states:* ***New*** *when created,* ***Ready*** *when waiting for CPU time,* ***Running*** *when executing,* ***Waiting*** *if it needs input or resources, and* ***Terminated*** *once completed. To manage multiple processes efficiently, scheduling algorithms like* ***First Come, First Serve (FCFS)****,* ***Round Robin****, and* ***Shortest Job Next*** *decide the execution order to optimize performance."*

**Memory Management:**

*"Efficient memory management ensures that processes get the required memory while minimizing waste. Two common techniques are* ***Paging****, which divides memory into fixed-size blocks to reduce fragmentation, and* ***Segmentation****, which divides memory into variable-sized logical segments. Additionally,* ***Process Scheduling*** *determines the order in which processes execute on the CPU to maximize efficiency and resource utilization."*

Figure 3.2: Audio Speech

### 3.1.3 System Architecture

The feature is implemented as a modular pipeline, consisting of the following components:

1. **Slides Parser:** Utilizes OCR tools to extract text and structural information (e.g., headings, subheadings) from PDF files.
2. **Audio Transcriber:** Converts audio to text using a pre-trained model, which supports robust transcription across accents and noise conditions.
3. **Generate Synthetic Data:** Use LLM to create a high-quality dataset that includes detailed explanations and concise summaries for various topics, ensuring comprehensive coverage of domain knowledge. For now, we have focused on content related to Operating Systems.


```
[
    {
        "topic_name":"What Operating Systems Do",
        "explanation":"What Operating Systems Do: A Comprehensive Explanation  An Operating System (OS)
        "summary":"In summary, an Operating System performs three main functions: process management, m
    },
    {
        "topic_name":"Computer-System Organization",
        "explanation":"Computer-System Organization  Computer-System Organization refers to the way in
        "summary":"Computer-System Organization is the design and structure of a computer system, compr
    },
    {
        "topic_name":"Computer-System Architecture",
        "explanation":"Computer-System Architecture  Computer-System Architecture refers to the design
        "summary":"Computer-System Architecture is the design and organization of the components that m
    },
    {
        "topic_name":"Operating-System Operations",
        "explanation":"Operating-System Operations: A Comprehensive Explanation  An operating system (O
        "summary":"In summary, operating-system operations involve the management of processes, memory,
    },
    {
        "topic_name":"Resource Management",
        "explanation":"Resource Management is the process of planning, organizing, and controlling the
        "summary":"Resource Management is a critical process that involves planning, organizing, and co
    },
    {
        "topic_name":"Security and Protection",
```


Figure 3.3: AI Generated Synthetic Dataset

- **Dataset Overview:** Topics related to Operating Systems containing *Topic Name*, *Explanation*, and *Summary*.
- **Dataset Size:** 3885 rows of primarily textual data.
- **Preprocessing:**
  - Removed unwanted characters such as asterisks (*), bullet points (•), backslashes (\), and numeric prefixes (e.g., `1.`, `2.`).
  - Dropped records with missing or empty fields.
  - Retained only samples with word counts between 15 and 700.
- **Data Visualization:**
  - *Histogram:* Showed text length distribution.

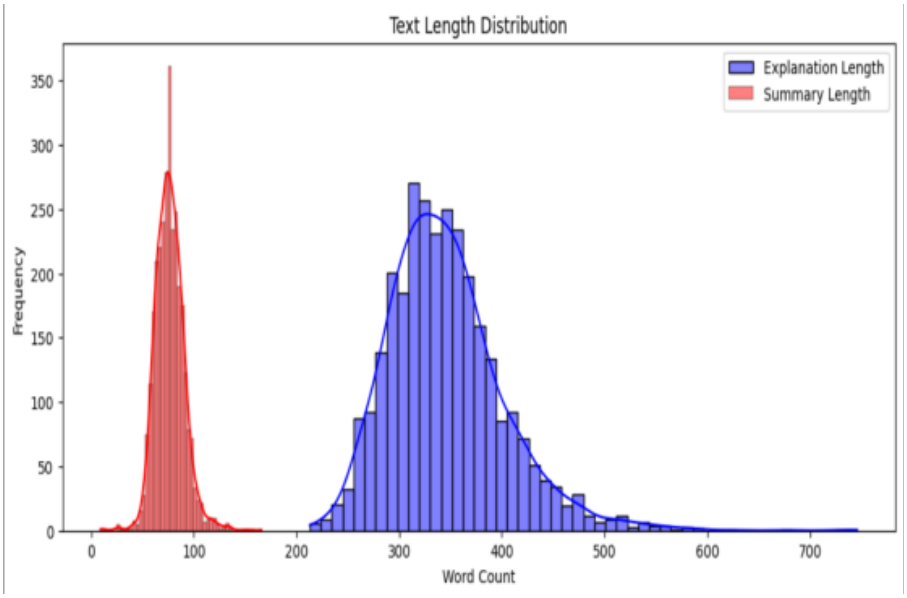


Figure 3.4: Text Length Distribution

  - *Word Cloud:* Highlighted frequently occurring words in the explanations.

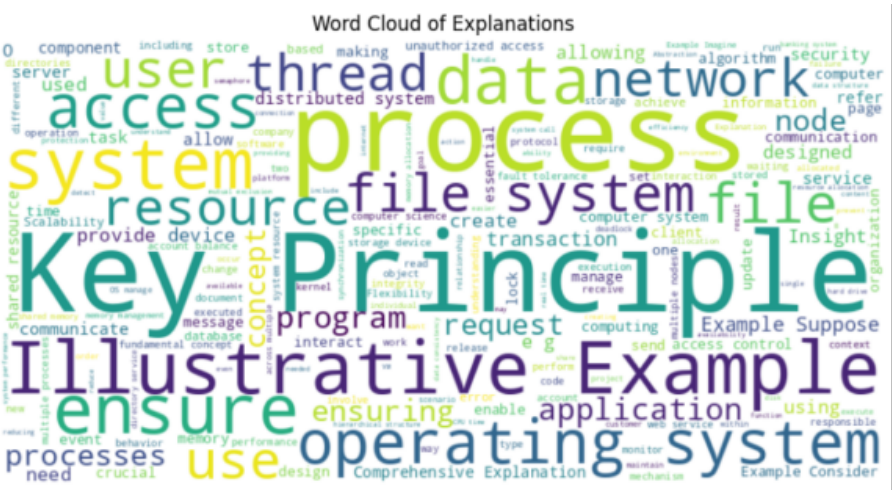


Figure 3.5: Frequently Occurring Words

4. **Add Explanation:** Use fine-tuned transformer model on synthatic data to add topic explanations to the content of extracted slides.
5. **Contextual Aligner:** Segments transcribed text and aligns it with the most relevant slide content based on the context.
6. **Summarizer:** Applies a fine-tuned transformer model to generate summaries of the combined lecture audio and slide content.
7. **Note Formatter:** Formats the final content into structured notes, preserving the order of topics and improving readability.

## 3.2 Whiteboard for Text Refinement

This section describes the approach taken to design, develop, and evaluate the whiteboard-based handwritten text recognition and refinement system using Deep Capsule Networks.

### 3.2.1 Handwritten Text Capture from Digital Whiteboard

To simulate the natural classroom writing experience, we developed a digital canvas for handwriting input using Tkinter. The handwritten text captured from this whiteboard undergoes a preprocessing pipeline to standardize it for model inference.

#### Canvas-Based Handwriting Interface

A GUI-based handwriting interface will be used with the following stack:

- Implement the canvas where users can draw using the mouse.
- Capture canvas images and convert them to compatible formats for pre-processing.
- Save the image for later recognition process.

#### Canvas Workflow

The following steps summarize the canvas system:

1. **Canvas Initialization:** A whiteboard is initialized, capable of capturing strokes based on mouse events.
2. **Stroke Recording:** User strokes are captured in real time and drawn in black. A bounding box is calculated around the written content.
3. **Auto-Capture Mechanism:** If the user is inactive for 4 seconds, the system captures the bounding box region containing the text.
4. **Preprocessing Pipeline:** The captured image is preprocessed using:
    - Grayscale conversion
    - Normalization
    - Resizing
5. **Post-Capture Behavior:** The handwritten text is replaced with an image of refined text on the same canvas.

### 3.2.2 Dataset Collection and Preprocessing

We used the **EMNIST ByClass** Dataset:

- **Classes:** 62 (Digits 0–9, Uppercase A–Z, Lowercase a–z).
- **Samples:** 814,255 grayscale images (28×28).
- **Preprocessing:**
  - Normalization to range [0, 1].
  - Reshaping into 4D tensor format.
  - One-hot encoding of labels.
  - Balanced selection of 800 samples per class.

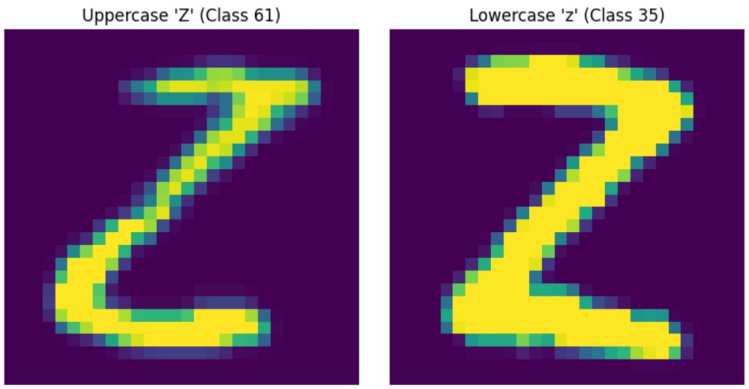


Figure 3.6: EMNIST Dataset

### 3.2.3 Deep Capsule Network for Handwritten Text Recognition

To accurately interpret handwritten characters written on a digital whiteboard, we designed a deep capsule network. The objective was to ensure robustness against handwriting variability, including distortions, different writing styles, and orientation shifts. The methodology began with a CNN baseline for performance comparison and was later extended to a Deep Capsule Network (DCaps) to better preserve spatial hierarchies in the data.

#### CNN Baseline

Initially, we experimented with a Convolutional Neural Network (CNN) as a baseline model, achieving an accuracy of only 87%. The CNN model consisted of multiple convolutional layers with ReLU activation, batch normalization, max-pooling, and dropout regularization, followed by fully connected layers for classification. Despite its reasonable performance, CNNs struggle with capturing spatial hierarchies effectively and are sensitive to affine transformations such as rotation, scaling, and viewpoint variations. These limitations can lead to misclassifications when letters exhibit distortions or variations in writing styles.

To address these challenges, we transitioned to Deep Capsule Networks, which leverage dynamic routing between capsules to preserve spatial relationships and encode pose information more effectively. Our Deep Capsule Network implementation outperformed the CNN baseline.

Table 3.1: CNN Architecture

| **CNN Architecture** | |
|---|---|
| **Layer** | **Description** |
| Input | $28 \times 28 \times 1$ (Grayscale Image) |
| Conv2D-1 | 32 filters, $3 \times 3$ kernel, ReLU, Same Padding |
| BatchNorm-1 | Batch Normalization |
| MaxPool-1 | $2 \times 2$ Pooling |
| Dropout-1 | 0.25 Dropout |
| Conv2D-2 | 64 filters, $3 \times 3$ kernel, ReLU, Same Padding |
| BatchNorm-2 | Batch Normalization |
| MaxPool-2 | $2 \times 2$ Pooling |
| Dropout-2 | 0.25 Dropout |
| Flatten | Flatten Layer |
| Dense-1 | 128 Neurons, ReLU |
| Dropout-3 | 0.5 Dropout |
| Output | 62 Neurons, Softmax Activation |

Table 3.2: 10-Fold Cross-Validation Results

| **Fold** | **Accuracy (%)** |
|---|---|
| 1 | 87.56 |
| 2 | 85.89 |
| 3 | 87.20 |
| 4 | 87.06 |
| 5 | 87.72 |
| 6 | 87.24 |
| 7 | 87.74 |
| 8 | 86.11 |
| 9 | 86.11 |
| 10 | 88.06 |
| **Average** | 87.07 |
| **Standard Deviation** | $\pm$0.73 |

### DCaps Architecture

The Deep Capsule Network was selected to overcome the limitations observed in CNNs. Capsule Networks represent features as vectors (capsules), enabling them to model not only the presence of features but also their pose, orientation, and spatial relationships. This property makes them more suitable for recognizing characters that may appear similar in local structure but differ globally due to variations in style or deformation.

### Input Layer

- The input to the network consists of grayscale images of size $28 \times 28 \times 1$.

**Convolutional Layers**

- A stack of five convolutional layers is used to progressively extract low-level to high-level features.
- Each layer uses activation functions to introduce non-linearity while preventing the dying neuron problem.
- Increasing the number of filters in deeper layers allows the network to learn more complex visual patterns that contribute to character identity.

**Primary Capsule Layer**

- The output of the convolutional layers is reshaped into capsules of a fixed vector dimension (e.g., 8D).
- A non-linear squashing function is applied to normalize the vector lengths between 0 and 1.
- This step enables encoding both the presence and instantiation parameters of features, such as orientation and scale.

**Digit Capsule Layer**

- The Digit Capsule Layer models higher-level entities (i.e., whole characters) by dynamically routing information from the Primary Capsules.
- Transformation matrices are learned to map lower-level capsule outputs to higher-level capsules, preserving spatial relationships.
- Dynamic routing by agreement iteratively adjusts the contribution of each capsule based on how well their predictions align, improving part-whole reasoning.

**Length Layer**

- The length (Euclidean norm) of each digit capsule's output vector is interpreted as the probability of the corresponding class being present.
- This probabilistic interpretation aligns well with classification tasks, allowing us to directly compare output strengths.

**Masking and Decoder Network**

- During training, the output of the correct class capsule is masked to encourage class-specific learning.
- A decoder network attempts to reconstruct the original input image from the capsule vector.
- This reconstruction loss acts as a regularizer, ensuring that the capsules retain rich and interpretable representations of the input.

### 3.2.4 Real-Time Text Replacement

The ultimate goal of this module is to facilitate clean and accessible digitization of handwritten content on the whiteboard. This process goes beyond character recognition and includes segmentation, text correction, and visual replacement, all integrated seamlessly in real time. Once the Deep Capsule Network achieves sufficient recognition accuracy, it will be integrated into the smart teaching assistant application to:

- Detect and segment handwritten regions from the whiteboard.
- Recognize characters within those regions using the trained DCaps model.
- Recognized letters are combined to form words, and words are assembled into complete sentences, ensuring spatial and lexical consistency with the user's original input.
- The raw predicted text is post-processed.
- After all corrections, the final sentence is rendered as a clean, high-quality image. This image replaces the original handwritten region on the canvas, maintaining its original position and spatial context. The result is a professional, neat text overlay that improves clarity, readability, and accessibility.

# Chapter 4

# Implementation

## 4.1 Lecture Notes Generation

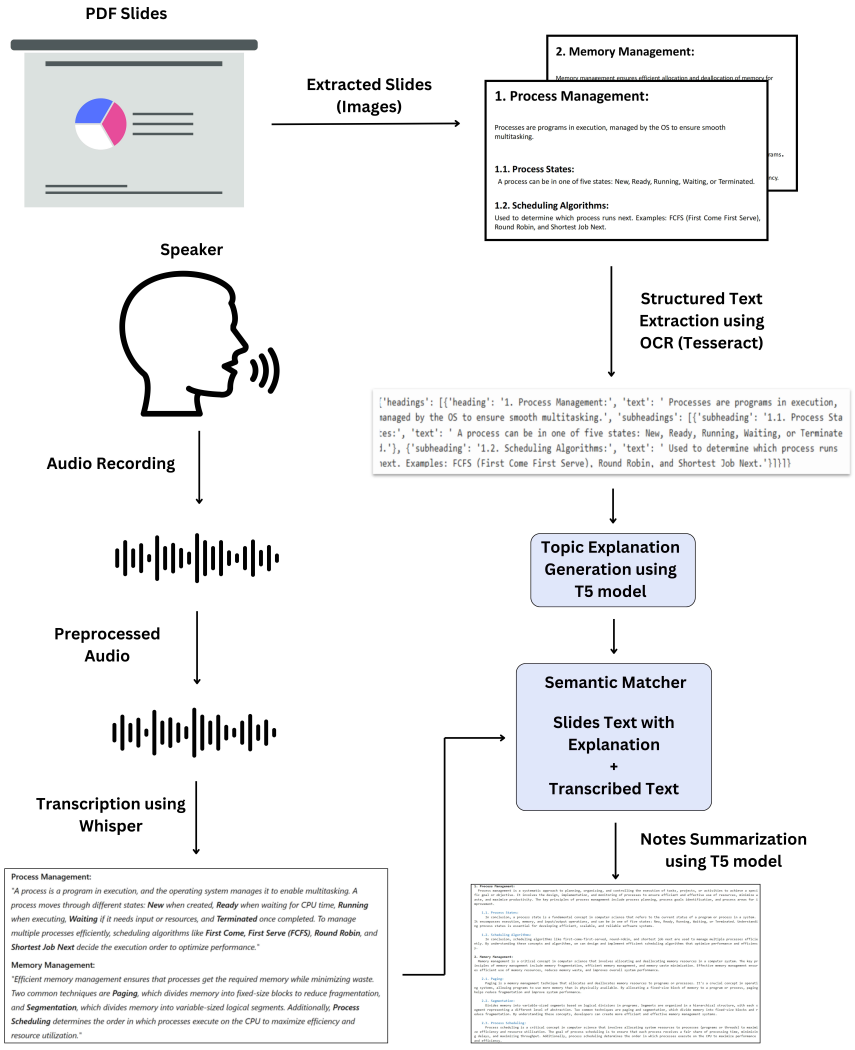


Figure 4.1: Lecture Notes Generation Pipeline

### 4.1.1 Extracting Text from Slides

We implemented a file selection dialog box using `Tkinter`, allowing the user to choose a Slides PDF file. The `pdf2image` library is used to extract slide images from the selected PDF. Each slide image is converted to grayscale, and Optical Character Recognition (OCR) is applied using `pytesseract`.

After OCR, we fix errors such as spacing issues and incorrect newlines using the `re` library:

- Ensure proper spacing after multi-level numbers (e.g., "1.1.Subheading" → "1.1. Subheading")
- Ensure spacing after single-level numbers (e.g., "1.Heading" → "1. Heading")
- Remove incorrect spaces before punctuation (e.g., "Hello , world ." → "Hello, world.")
- Fix incorrect newlines breaking words or sentences

The extracted text is then structured into:

- Headings (e.g., “1. Introduction”)
- Subheadings (e.g., “1.1 Background”)
- Text content under each heading/subheading

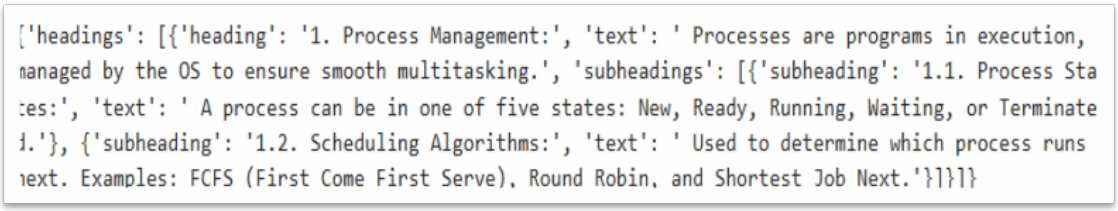
['headings': [{'heading': '1. Process Management:', 'text': ' Processes are programs in execution,
nanaged by the OS to ensure smooth multitasking.', 'subheadings': [{'subheading': '1.1. Process Sta
:es:', 'text': ' A process can be in one of five states: New, Ready, Running, Waiting, or Terminate
i.'}, {'subheading': '1.2. Scheduling Algorithms:', 'text': ' Used to determine which process runs
iext. Examples: FCFS (First Come First Serve), Round Robin, and Shortest Job Next.'}]}]}

Figure 4.2: Extracted Structured Text

### 4.1.2 Capturing and Processing Lecture Audio

We capture live speech input using the `sounddevice` library. The lecture is recorded slide by slide:

- The user records audio for the first slide, then moves to the next slide and record the corresponding explanation.
- This ensures that each slide has separate, synchronized audio.

The recorded audio is normalized for volume, and background noise is reduced using the `noisereduce` library. The processed audio is saved as a `.wav` file.

### 4.1.3 Transcribing Audio Using Whisper

After extracting slide content and recording slide-by-slide explanations, we use the **Whisper large-v3** model to transcribe speech from the saved audio files. This process is executed in parallel using threads, ensuring efficient and faster transcription at the end of the recording session.

### 4.1.4 Generating Topic Explanations for Slide Content

Before merging the slide text with the transcribed audio, we enhance the slide content by generating explanations using fine-tuned transformer model (T5) trained on synthatic data for each topic extracted during OCR. This step improves the completeness of the notes, even if some topics are not discussed in the recorded audio.

#### Dataset Preparation

The dataset used was sourced from a preprocessed `JSON` file containing structured topic explanations.

- Extracted **topic names** with the prefix `"explain:"`
- Mapped each topic to its explanation
- Split dataset into **90%** training and **10%** validation
- Converted to the `Hugging Face Dataset` format

#### Tokenization

Tokenization was applied using the `T5Tokenizer`:

- Both inputs and targets were tokenized
- Text was truncated and padded to **512** tokens
- Labels were assigned from target tokens

#### Fine-Tuning the T5 Model

The `t5-large` model was fine-tuned using the following hyperparameters:

- Learning rate: 0.004
- Weight decay: 0.01
- Epochs: 2 (more would lead to excessive model size)
- Batch size per device: 1
- Evaluation strategy: Per epoch

#### Explanation Generation After Fine-Tuning

The system processes structured lecture content, where each section contains a heading, relevant text, and subheadings. The model takes each heading and subheading as input and generates a concise yet informative explanation. These explanations are appended to the extracted slide content to ensure contextual completeness.

1. Prefix headings/subheadings with `"explain:"`
2. Tokenize and pass to the model
3. Generate explanation of maximum **120** tokens

4. To improve the diversity and fluency of generated explanations, decoding parameters such as **num_beams = 8** and **no_repeat_ngram_size = 3** were used during generation. These settings ensure that explanations are well-formed and avoid repetition.

5. Post-process the explanation:

   - Remove unnecessary prefixes like "Generated Explanation:", "Explanation:", etc.
   - If the explanation starts with a heading followed by a colon (e.g., `"Operating Systems:"`), remove that part to avoid redundancy
   - Ensure that the final sentence ends properly—if it does not end with punctuation like a full stop, question mark, or exclamation mark, it is removed to maintain sentence completeness

6. Append explanation to corresponding section in notes

### 4.1.5 Merging Slide Text with Transcribed Audio

Once we have both the text from slides and the transcribed audio, we merge them systematically:

**1. Keyword Extraction and Abbreviation Generation**

- Extracts main keywords from headings and subheadings, removing numbering (e.g., "1. Introduction" $\rightarrow$ "Introduction").
- Finds synonyms using the `WordNet` dictionary.
- Generates abbreviations by taking the first letter of each word while ignoring common stopwords (e.g., "Machine Learning" $\rightarrow$ "ML").

**2. Embedding Creation and Contextual Similarity Matching**

- Generate keyword and sentence embeddings using the `paraphrase_mpnet_model`.
- Compute cosine similarity between:
  - Slide text keywords, text, generated explanation text
  - Transcribed audio text
- Match the transcribed sentences to the most relevant slide section using highest similarity scores.
- Dynamically update slide sections by appending matched sentences for richer context.

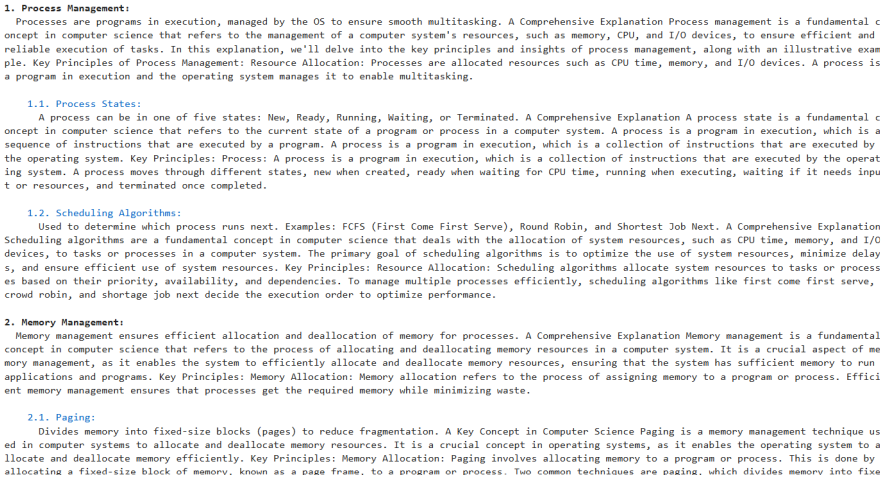


Figure 4.3: Merged Content

### 4.1.6 Summarized Notes Generation

A dedicated summarization module was implemented using a fine-tuned T5-based language model. This module transforms lengthy explanations (generated from both slides and lecture audio) into brief summaries to help students quickly grasp the core concepts.

#### Model Training

The T5 summarization model was fine-tuned using a synthatic dataset containing pairs of explanations and their corresponding summaries. The following steps were involved in the training process:

- The dataset was formatted into input-output pairs: the explanation was prefixed with `"summarize:"` and used as input, while the target summary served as the output label.
- The dataset was split into training (90%) and evaluation (10%) sets.
- Tokenization was performed using the `t5-large` tokenizer, with a maximum input length of 512 tokens and a target summary length of 128 tokens.
- The model was trained using the Hugging Face `Trainer` API with the following hyperparameters:
  - **Model:** T5-Large
  - **Epochs:** 2
  - **Batch Size:** 1 (train and eval)
  - **Learning Rate:** 3e-4
  - **Weight Decay:** 0.01

#### Summarization Pipeline

The summarization process involves the following steps:

1. Combined content (i.e., extracted text from slides and transcribed lecture audio) is passed to a T5 summarization model.

2. The model is guided using a task-specific prefix: `"summarize:"` is prepended to every input.
3. During decoding, the following parameters are used to ensure high-quality, coherent summaries:
   - **num_beams = 8** for effective beam search
   - **no_repeat_ngram_size = 3** to prevent redundancy
   - **repetition_penalty = 2.5** to penalize repeated tokens
   - **length_penalty = 1.0** to control the summary length
4. After generation, a post-processing step is applied to:
   - Remove generic prefixes such as "In summary," or "Conclusion:"
   - Trim incomplete trailing sentences
5. This summarization is applied to both headings and subheadings to ensure a well-structured and digestible note format.

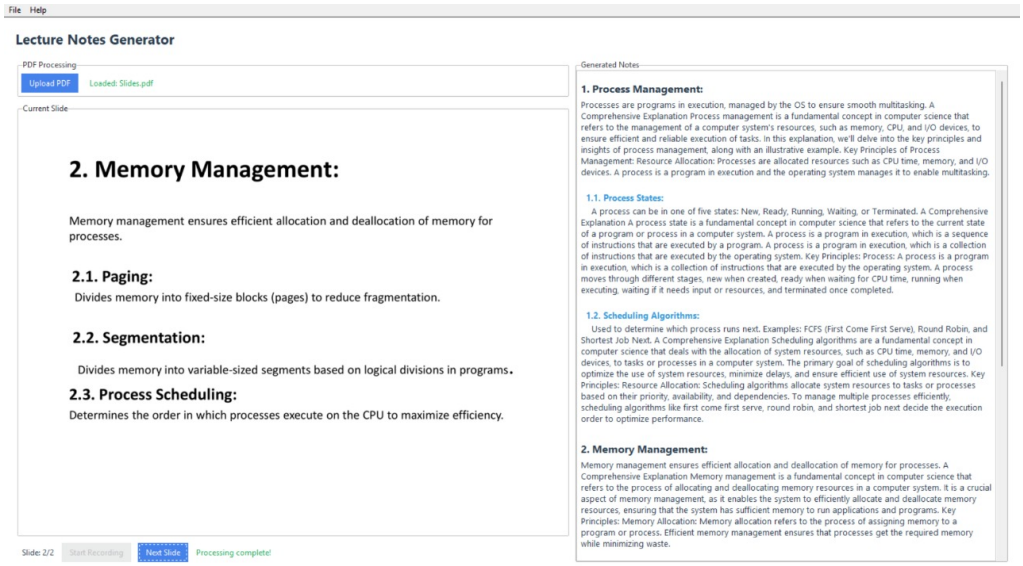


Figure 4.4: Lecture Notes Generation App

## 4.2 Whiteboard for Text Refinement

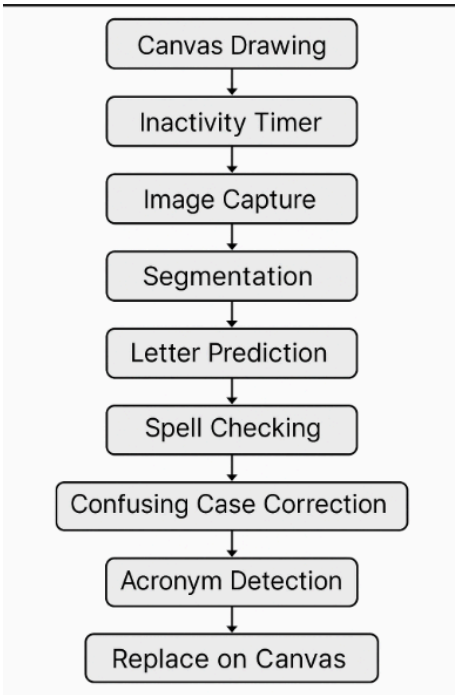


Figure 4.5: Whiteboard for Text Refinement Pipeline

This module implements a real-time digital whiteboard system capable of recognizing and refining handwritten text. The implementation combines graphical input capture, image processing, character recognition via a Capsule Network, and intelligent post-processing techniques to produce clean and readable digital text on the canvas. The pipeline operates end-to-end from stroke capture to text replacement.

### 4.2.1 Drawing Interface and Stroke Capture

The whiteboard interface is implemented using `Tkinter`, Python's standard GUI toolkit. A canvas widget is used to allow users to draw freehand using a mouse.

- Mouse events such as `<Button-1>` and `<B1-Motion>` are bound to drawing functions that record and render strokes in real time.
- A timer is initialized on each drawing action. If no movement is detected for 4 seconds, the system assumes the input is complete and automatically captures the canvas region containing handwriting.
- The drawn canvas area is saved as a grayscale image using the `PIL.ImageGrab` module for further processing.

### 4.2.2 Image Preprocessing and Segmentation

Once a handwritten image is captured, it undergoes a series of preprocessing steps to segment it into individual words and characters:

- **Binarization:** Adaptive thresholding is applied using OpenCV to convert the grayscale image into a binary format, enhancing contrast between ink and background.
- **Word Segmentation:** Dilation is used with a rectangular kernel to group nearby letters into word-level regions. Contours are detected around these regions to extract bounding boxes for each word.
- **Letter Segmentation:** Within each word bounding box, erosion is applied to separate tightly written letters. Individual character contours are extracted using OpenCV's `findContours()`.
- **Padding and Resizing:** Each detected character is padded to ensure a square aspect ratio, resized to 28×28 pixels, and saved as a temporary image for classification.

### 4.2.3 Character Recognition via Deep Capsule Network

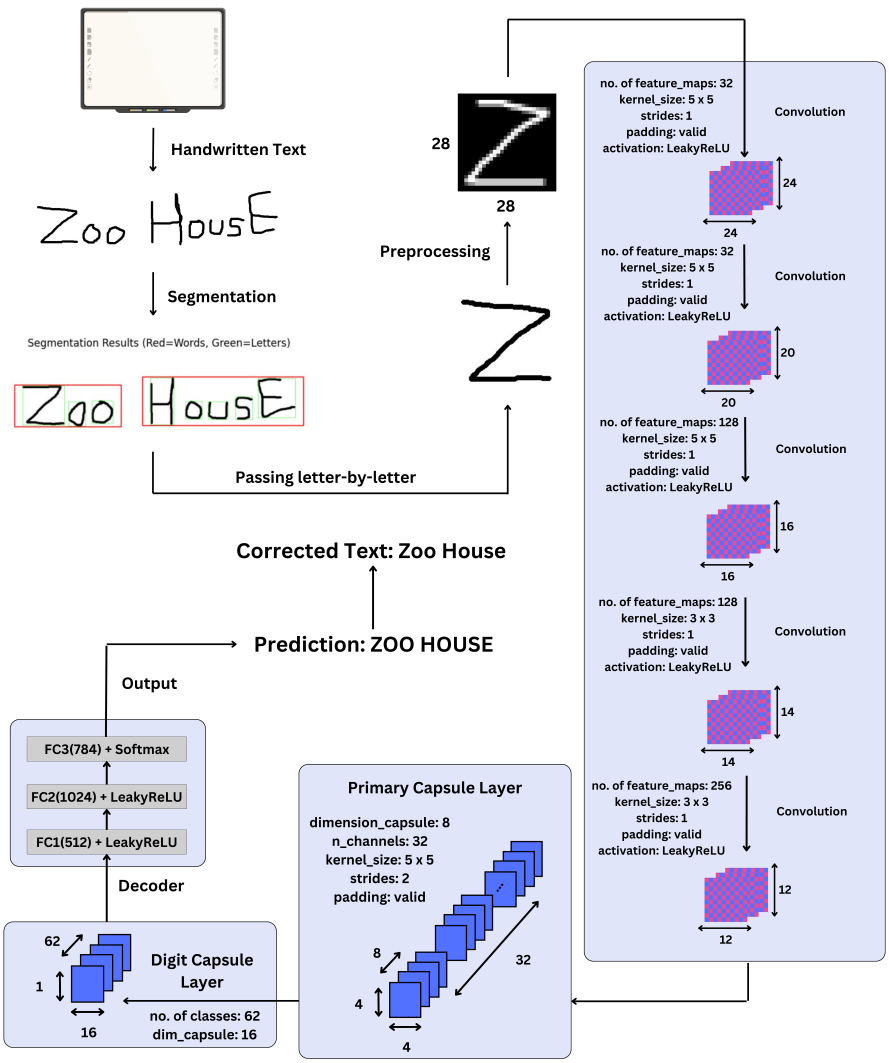


Figure 4.6: Handwritten Text Recognition using DCapsNet

Each segmented letter image is passed through a Deep Capsule Network (DCaps) model trained for character classification.

- The model expects input images of size 28×28×1. Each letter image is preprocessed to match this format.
- The `predict_letter()` function loads the image, processes it into a tensor, and feeds it into the trained DCaps model.
- The predicted class (corresponding to an alphabet or digit) is returned for each character.
- These characters are concatenated in sequence to reconstruct full words, preserving the left-to-right reading order of the input.

### DCaps Network Architecture

The DCaps network is designed for image classification by integrating convolutional feature extraction with capsule-based representation and dynamic routing. Below, we describe the architecture layer by layer from the input to the output.

**Input Layer** Accepts grayscale images of size $28 \times 28$ with a single channel. The input tensor is defined as:

$$\text{Input Shape} = (28, 28, 1)$$

with a fixed batch size of 64 during training.

**Convolutional Layers** The following table summarizes the configuration of the convolutional layers used in our DCaps model.

Table 4.1: Convolutional Layers Configuration

| Layer | Filters | Kernel Size | Stride | Activation |
|---|---|---|---|---|
| 1 | 32 | $5 \times 5$ | 1 | LeakyReLU |
| 2 | 32 | $5 \times 5$ | 1 | LeakyReLU |
| 3 | 128 | $5 \times 5$ | 1 | LeakyReLU |
| 4 | 128 | $3 \times 3$ | 1 | LeakyReLU |
| 5 | 256 | $3 \times 3$ | 1 | LeakyReLU |

### Primary Capsule Layer

- **Operation:** Applies a 2D convolution to the feature maps from the previous layer.
- **Output:** The convolution output is reshaped into capsules. For instance, if each capsule is of dimension 8 and there are 32 channels, the reshaping groups the features into vectors of size 8.
- **Normalization:** Each capsule vector is passed through the `squash` function to ensure its length is between 0 and 1. The squashing is mathematically defined as:

$$\mathbf{v}_{\text{squashed}} = \frac{\|\mathbf{v}\|^2}{1 + \|\mathbf{v}\|^2} \frac{\mathbf{v}}{\|\mathbf{v}\|}, \quad \text{with } \|\mathbf{v}\| = \sqrt{\sum_i v_i^2}.$$

**Digit Capsule Layer**

- **Transformation:** Each capsule from the previous layer is transformed using a weight matrix $\mathbf{W}$, where:

$$\mathbf{W} \in \mathbb{R}^{\text{num\_capsule} \times \text{input\_num\_capsule} \times \text{dim\_capsule} \times \text{input\_dim\_capsule}}.$$

- **Dynamic Routing:** Over several iterations (e.g., 3 iterations), dynamic routing updates coupling coefficients between capsules. At each iteration, a softmax function is applied to the coefficients and the capsule outputs are updated by:

$$\text{output} = \text{squash}\left(\sum_i c_{ij}\hat{\mathbf{u}}_{j|i}\right),$$

  where $c_{ij}$ are the coupling coefficients and $\hat{\mathbf{u}}_{j|i}$ are the predicted outputs.
- **Output:** This layer produces $n_{\text{class}}$ capsules (one per class, e.g., 62 for EMNIST), each with a vector dimension (e.g., 16).

**Length Layer**

**Operation:** Computes the Euclidean norm of each capsule vector:

$$\text{Length}(\mathbf{v}) = \sqrt{\sum_i v_i^2}.$$

**Purpose:** The resulting scalar for each capsule is interpreted as the probability of the corresponding class.

**Mask Layer**

**Operation:** Selectively masks the capsule outputs.

- **Supervised Mode:** Uses the true labels (provided as one-hot vectors) to select the capsule corresponding to the target class.
- **Unsupervised Mode:** Chooses the capsule with the maximum length.

**Purpose:** This masked output is used by the decoder network for reconstruction, thereby encouraging the capsules to encode detailed information about the input.

**Decoder Network**

**Structure:**

1. A dense (fully connected) layer with 512 neurons and LeakyReLU activation.
2. A dense layer with 1024 neurons and LeakyReLU activation.
3. A final dense layer that outputs a vector whose size equals the total number of pixels in the input image, i.e., $28 \times 28 \times 1$. A softmax activation is applied here.
4. Reshaping of the output vector to the original image shape.

**Purpose:** The decoder reconstructs the input image from the capsule representation, serving as a regularizer to improve the quality of the learned features.

#### Loss Function

**Margin Loss:** The network is trained using a margin loss defined as:

$$L = y_{\text{true}} \cdot \max(0, 0.9 - y_{\text{pred}})^2 + 0.5 \cdot (1 - y_{\text{true}}) \cdot \max(0, y_{\text{pred}} - 0.1)^2,$$

where $y_{\text{true}}$ is the one-hot encoded ground truth and $y_{\text{pred}}$ is the predicted probability from the Length layer. This loss encourages the correct class capsule to have a high probability while penalizing incorrect ones.

#### Training Strategy

The complete model, combining the capsule network and the decoder, is trained using 10-fold cross-validation. During each fold:

- The model is compiled with the Adam optimizer.
- Both the margin loss and a reconstruction loss (mean-squared error) are minimized.
- Batch sizes are carefully managed (e.g., 64) to ensure proper training dynamics.

Table 4.2: 10-Fold Cross-Validation Results using DCapsNet

| Fold | CapsNet Loss | CapsNet Accuracy |
|---|---|---|
| 1 | 0.1658 | 0.7484 |
| 2 | 0.1513 | 0.7884 |
| 3 | 0.1126 | 0.8584 |
| 4 | 0.0613 | 0.9347 |
| 5 | 0.0361 | 0.9663 |
| 6 | 0.0242 | 0.9799 |
| 7 | 0.0221 | 0.9840 |
| 8 | 0.0152 | 0.9911 |
| 9 | 0.0170 | 0.9870 |
| 10 | 0.0086 | 0.9951 |
| **Average** | - | 0.9233 |

### 4.2.4 Custom Enhancement for Confusing Pairs

This preprocessing enhancement is designed to help the model better distinguish between letters that appear very similar in their uppercase and lowercase forms (e.g., C/c, Z/z). There are 12 such challenging pairs.

#### Identification of Confusing Pairs

- The system first checks if an image corresponds to one of these confusing letters by using specific numeric codes (e.g., 12 for 'C', 38 for 'c', etc.).
- In total, 12 problematic uppercase/lowercase pairs are identified. For example, pairs such as (12, 38), (15, 41), ..., (35, 61) are flagged for additional processing.

**Enhancement Techniques**

- **Uppercase Letters (e.g., 'Z'):**
  - *Brighten the Top:* The upper part of the image is increased in brightness by 20% to emphasize the taller characteristics of uppercase letters.
  - *Add a White Line at the Bottom:* A solid white line is drawn along the bottom (rows 18–20) to mark the baseline of large letters.
- **Lowercase Letters (e.g., 'z'):**
  - *Brighten the Bottom:* The lower half of the image is brightened by 30% to accentuate the distinct shape of lowercase letters.
  - *Add a Gray Bar in the Middle:* A light gray bar is added across the middle (rows 10–12) to indicate the typical height of small letters.

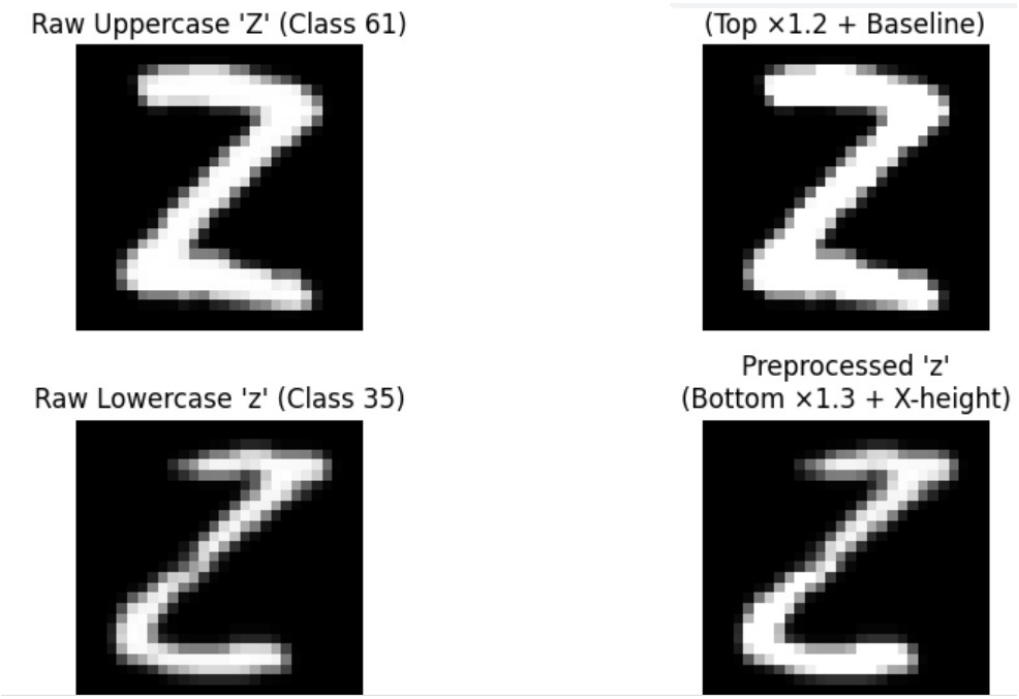


Figure 4.7: Handling Confusing Pairs

**Purpose**

This custom preprocessing for confusing pairs enhances the model's ability to distinguish similar uppercase and lowercase letters, leading to an accuracy increase of approximately 4% (from 92% to 96%).

Table 4.3: 10-Fold Cross-Validation Results After Handling Confusing Pairs

| Fold | CapsNet Loss | CapsNet Accuracy |
|---|---|---|
| 1 | 0.1155 | 0.8454 |
| 2 | 0.0835 | 0.8918 |
| 3 | 0.0554 | 0.9343 |
| 4 | 0.0404 | 0.9554 |
| 5 | 0.0255 | 0.9775 |
| 6 | 0.0186 | 0.9825 |
| 7 | 0.0197 | 0.9842 |
| 8 | 0.0125 | 0.9901 |
| 9 | 0.0062 | 0.9978 |
| 10 | 0.0043 | 0.9976 |
| **Average** | - | 0.9673 |

### 4.2.5 Text Correction and Normalization

To enhance the accuracy and readability of the recognized text, the system performs several correction procedures:

- **Spell Checking:** The assembled sentence is passed through the `pyspellchecker` library, which corrects common typographical errors based on dictionary matching.
- **Custom Casing Rules:** Confusions between uppercase and lowercase characters (e.g., 'S', 'O', 'X') are resolved using handcrafted rules, which enforce lowercase casing for such characters unless they appear at the start of a word.
- **Acronym Detection:** Using the `spaCy` NLP library, named entities of types `ORG` and `GPE` are detected and automatically converted to uppercase, correcting cases such as "usa" to "USA".

### 4.2.6 Digital Text Rendering and Canvas Replacement

After text recognition and correction, the final polished sentence is rendered visually and overlaid onto the original canvas, replacing the handwritten region.

- The corrected sentence is converted into an image using the `PIL.ImageDraw` module. Font size and spacing are adjusted for visual consistency.
- The rendered image is pasted over the original bounding box coordinates of the handwritten area, ensuring accurate spatial alignment.
- This final output provides a clean, digitally refined version of the user's input while maintaining its original layout and intent.

# Chapter 5

# Results and Discussion

## 5.1 Lecture Notes Generation

The Lecture Notes Generation feature demonstrated effective performance in generating concise, structured notes from slide content and lecture speech using a pipeline based on OCR, Whisper model, and a fine-tuned T5 transformer model for both explanation generation and summarization.

### Model Performance on Explanation Generation

The T5 model trained for generating explanatory text from key concepts was evaluated over 2 epochs.

Table 5.1: Explanation Generation Training and Validation Loss

| Epoch | Training Loss | Validation Loss |
|---|---|---|
| 1 | 1.5202 | 1.2902 |
| 2 | 1.1057 | 1.1560 |

The model was further evaluated using 5 manually annotated test samples, and the results are summarized in Table 5.2. The evaluation focused on commonly used natural language generation metrics such as BLEU and ROUGE.

Table 5.2: Explanation Generation Evaluation Metrics (Average)

| Metric | Score |
|---|---|
| BLEU | 0.1492 |
| ROUGE-1 F1 | 0.5608 |
| ROUGE-2 F1 | 0.2510 |
| ROUGE-L F1 | 0.3298 |

The BLEU score of 0.1492 indicates that while the model generates relevant phrases, it does not closely match the exact wording of human-generated explanations, which is expected in abstractive tasks. On the other hand, the ROUGE-1 F1 score of 0.5608 suggests that over half of the unigrams in the reference explanations are being captured. The ROUGE-2 and ROUGE-L scores show moderate performance in capturing

bi-gram structures and long-sequence overlaps, respectively. These results highlight that the model is capable of generating semantically relevant and partially structured explanations, although there is room for improving fluency and factual alignment through fine-tuning and data augmentation.

### Model Performance on Summarization

For summarization, the model was trained to condense lecture content while preserving meaning.

Table 5.3: Summarization Training and Validation Loss

| Epoch | Training Loss | Validation Loss |
|---|---|---|
| 1 | 0.5583 | 0.5746 |
| 2 | 0.3614 | 0.5139 |

The following table summarizes the average test performance on 5 validation samples using standard summarization evaluation metrics, shown in Table 5.4.

Table 5.4: Summarization Evaluation Metrics (Average)

| Metric | Score |
|---|---|
| BLEU | 0.4397 |
| ROUGE-1 F1 | 0.7215 |
| ROUGE-2 F1 | 0.5713 |
| ROUGE-L F1 | 0.6316 |

The BLEU score of 0.4397 indicates a relatively high overlap between the model-generated summaries and reference texts at the phrase level, suggesting strong lexical similarity. A ROUGE-1 F1 score of 0.7215 shows that the majority of important unigrams from the human-written summaries are preserved, while the ROUGE-2 F1 score of 0.5713 reflects that many bigram-level co-occurrences are also retained—indicating good local coherence. Moreover, the ROUGE-L score of 0.6316 suggests the model effectively captures the longest common subsequence, implying logical structuring and alignment with the reference summaries. Overall, these results suggest that the summarization model performs well in preserving both the content and structure of lecture material, making it a suitable candidate for educational summarization tasks.

## Qualitative Example Test Case

Two slides were used to evaluate the combined functionality. Below is the provided input:

### Slides Content:

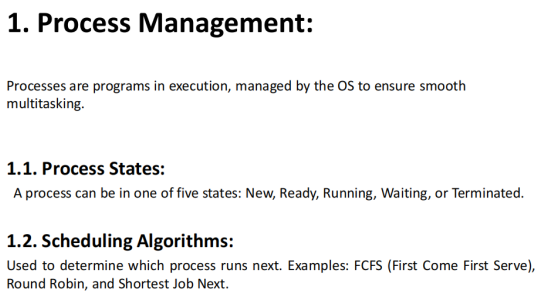


Figure 5.1: Slide 1

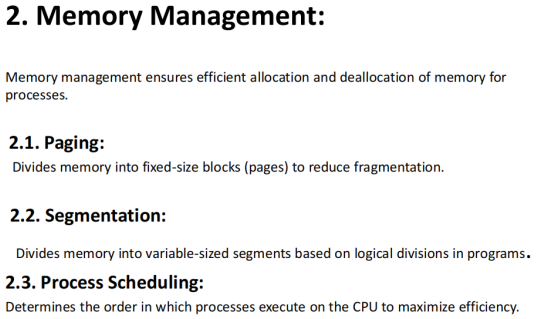


Figure 5.2: Slide 2

### Audio Transcript:

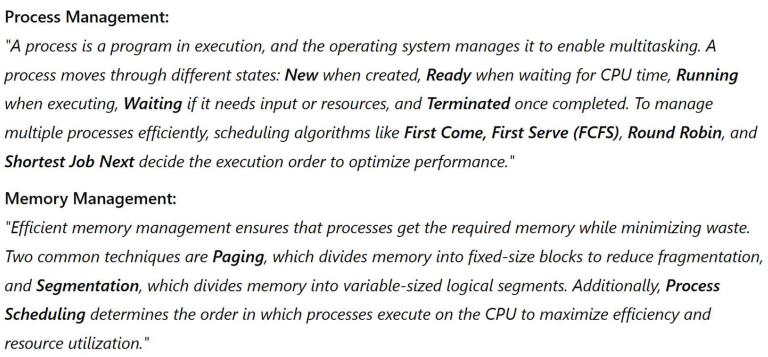


Figure 5.3: Audio Transcript

### Generated Notes:

**1. Process Management:**
Process management is a systematic approach to planning, organizing, and controlling the execution of tasks, projects, or activities to achieve a specific goal or objective. It involves the design, implementation, and monitoring of processes to ensure efficient and effective use of resources, minimize waste, and maximize productivity. The key principles of process management include process planning, process goals identification, and process areas for improvement.

1.1. Process States:
In conclusion, a process state is a fundamental concept in computer science that refers to the current status of a program or process in a system. It encompasses execution, memory, and input/output operations, and can be in one of five states: New, Ready, Running, Waiting, or Terminated. Understanding process states is essential for developing efficient, scalable, and reliable software systems.

1.2. Scheduling Algorithms:
In conclusion, scheduling algorithms like first-come-first-served, round-robin, and shortest job next are used to manage multiple processes efficiently. By understanding these concepts and algorithms, we can design and implement efficient scheduling algorithms that optimize performance and efficiency.

**2. Memory Management:**
Memory management is a critical concept in computer science that involves allocating and deallocating memory resources in a computer system. The key principles of memory management include memory fragmentation, efficient memory management, and memory waste minimization. Effective memory management ensures efficient use of memory resources, reduces memory waste, and improves overall system performance.

2.1. Paging:
Paging is a memory management technique that allocates and deallocates memory resources to programs or processes. It's a crucial concept in operating systems, allowing programs to use more memory than is physically available. By allocating a fixed-size block of memory to a program or process, paging helps reduce fragmentation and improve system performance.

2.2. Segmentation:
Divides memory into variable-sized segments based on logical divisions in programs. Segments are organized in a hierarchical structure, with each segment representing a different level of abstraction. Two common techniques are paging and segmentation, which divide memory into fixed-size blocks and reduce fragmentation. By understanding these concepts, developers can create more efficient and effective memory management systems.

2.3. Process Scheduling:
Process scheduling is a critical concept in computer science that involves allocating system resources to processes (programs or threads) to maximize efficiency and resource utilization. The goal of process scheduling is to ensure that each process receives a fair share of processing time, minimizing delays, and maximizing throughput. Additionally, process scheduling determines the order in which processes execute on the CPU to maximize performance and efficiency.

Figure 5.4: Generated Notes

## 5.2 Whiteboard Text Refinement

The Whiteboard Text Refinement module was designed to enhance handwritten or rough board content into clean, readable text using Optical Character Recognition (OCR) and deep learning-based classification and filtering mechanisms. The primary goal was to ensure correct word reconstruction and reduce noise in low-quality board images.

### Model Performance: CNN vs Deep Capsule Network

We evaluated two different architectures for refining and verifying OCR-extracted words: a baseline Convolutional Neural Network (CNN) and our custom Deep Capsule Network (CapsNet). Both were trained and tested using labeled word images.

### CNN Model Results

Table 5.5: CNN 10-Fold Cross-Validation Results

| Metric | Value |
|---|---|
| Average Accuracy | 86.99% |
| Average Precision | 0.8738 |
| Average Recall | 0.8700 |
| Average F1-Score | 0.8652 |
| Accuracy Std Dev | $\pm$0.73% |

### Deep Capsule Network (DCapsNet) Results

Table 5.6: Deep CapsNet 10-Fold Cross-Validation Results

| Metric | Value |
|---|---|
| Average Total Loss | 0.0321 |
| Average Accuracy | 96.73% |
| Average Precision | 0.9687 |
| Average Recall | 0.9673 |
| Average F1-Score | 0.9670 |

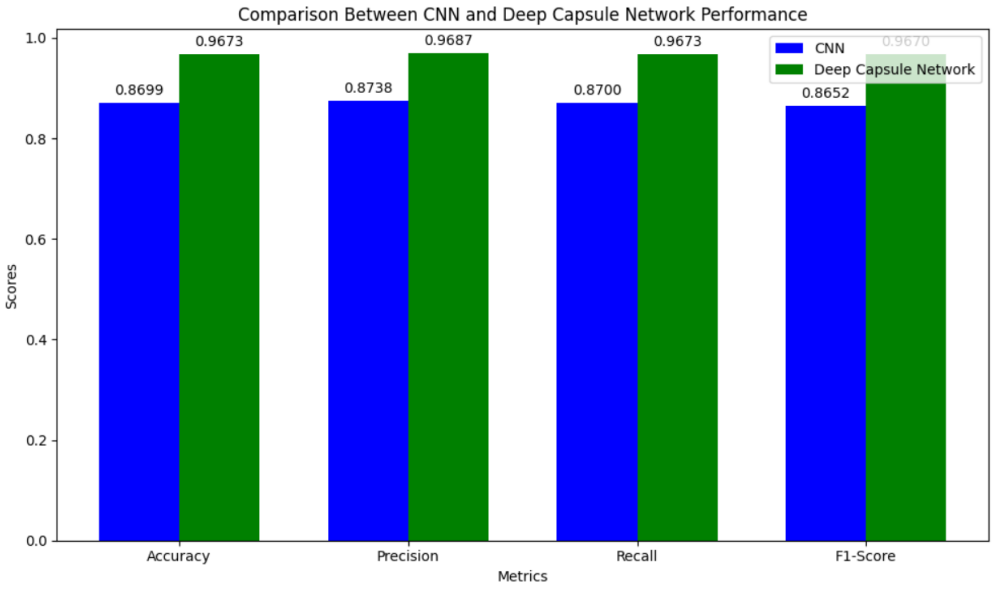


Figure 5.5: CNN and DCapsNet Comparison

### Test Performance on Unseen Samples

To further validate generalizability, we evaluated the models on 12,352 unseen test samples (selected as 200 random images from each class). The CapsNet achieved an overall accuracy of 86.77%, showing robustness even with noisy whiteboard data.

**Conclusion** Both modules—Lecture Notes Generation and Whiteboard Text Refinement—demonstrate practical AI-driven enhancements for modern classrooms. Together, they automate the traditionally manual process of note-taking and content digitization, offering scalable benefits for educators and students alike.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

This project explored two innovative AI-powered features designed to enhance the teaching and learning experience: **Lecture Notes Generation** and **Whiteboard Text Refinement**. Both systems were developed with a strong emphasis on automation, accuracy, and user-centric design.

The **Lecture Notes Generation** module demonstrated the effectiveness of transformer-based architectures in generating educational content. Leveraging pre-trained T5 models for both explanation generation and summarization, we were able to produce structured, coherent notes from raw lecture inputs. The system achieved a ROUGE-L F1 score of up to **0.6316** and a BLEU score of **0.4397** for summarization, indicating high-quality generation and linguistic overlap with reference texts. For explanation generation, while the BLEU score was comparatively lower (**0.1492**), the model still produced informative and accurate elaborations, showcasing potential for further tuning and dataset expansion.

The **Whiteboard Text Refinement** module successfully transformed handwritten inputs into polished digital text. By combining real-time canvas capture with image processing techniques and a Deep Capsule Network (DCaps) model, the system achieved an impressive classification accuracy of **96.73%** on the 10-fold evaluation. Additionally, text correction mechanisms such as spell-checking, acronym recognition, and context-aware casing significantly improved the final output's readability. Replacing handwritten regions with clean digital text enhanced clarity and ensured a professional presentation.

Together, these modules advance the goal of building an intelligent assistant for educators by automating mundane tasks and enhancing content delivery.

## 6.2 Future Work

While the current implementation delivers promising results, there are several directions in which the system can be improved and extended:

- **Model Optimization and Fine-Tuning:** Further fine-tuning of the T5 models with larger and more diverse educational datasets can improve explanation generation accuracy, especially for domain-specific lectures.
- **Multi-modal Input Handling:** Future iterations could integrate multimodal inputs, including diagrams, slide visuals, and handwritten annotations, for richer note generation and visual-text alignment.
- **Real-Time Lecture Processing:** Deploying the lecture notes generator as a live tool that processes real-time lecture audio and slide changes would enhance classroom interactivity and immediate feedback.
- **Whiteboard Object Detection:** Expand whiteboard refinement to detect and digitize shapes, arrows, and diagrams using additional image recognition models or transformer-based vision-language models.
- **On-Device Deployment:** Optimizing the models for lightweight edge deployment would allow educators to use these features without constant internet connectivity, enabling widespread usage in resource-limited environments.
- **Integration with LMS:** Embedding the system into Learning Management Systems (LMS) such as Moodle or Google Classroom can streamline content sharing.

In conclusion, the project lays a strong foundation for an AI-driven educational assistant. With additional development, it holds the potential to transform traditional classrooms into smart, efficient, and highly interactive learning spaces.

# References


[1] OpenAI. "Whisper: Robust Speech Recognition via Large-Scale Weak Supervision." *arXiv preprint arXiv:2212.04356*, 2022.

[2] Korem, N. "Efficient and Accurate Transcription in Mental Health Research." *Academia.edu*, 2023.

[3] Sardar, D., & Bhardwaj, V. "Using EasyOCR Library for Text Extraction." *Online Article*, 2023.

[4] He, J., et al. "Deep Learning-Based OCR Enhancements for Low-Contrast Images." *IEEE Transactions on Image Processing*, 2022.

[5] Raffel, C., et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." *Journal of Machine Learning Research*, 2020.

[6] Fan, A., et al. "ELI5: Long-Form Question Answering." *Proceedings of the 57th Annual Meeting of the ACL*, 2019.

[7] Reimers, N., & Gurevych, I. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." *arXiv preprint arXiv:1908.10084*, 2019.

[8] Zhang, Y., et al. "Hierarchical Document Summarization with Multi-level Attention." *Information Processing & Management*, 2021.

[9] Pilault, J., et al. "Extractive+ Abstractive Hybrid Summarization via Document Structure Learning." *arXiv preprint arXiv:2005.06500*, 2020.

[10] Jia, W., et al. "A CNN-Based Approach to Detecting Text from Handwritten Notes." *IEEE*, 2018.

[11] Mandal, S., & Nandi, S. "Handwritten Indic Character Recognition Using Capsule Network." *arXiv preprint arXiv:1901.00166*, 2019.

[12] Dong, S., et al. "Gesture-Based User Interface for Smart Classrooms." *International Journal of Human–Computer Interaction*, 2019.

[13] Jung, S. H., et al. "Touchless Smart Whiteboard Using Inactivity-Based Segmentation." *Sensors*, 2020.

[14] Zhang, Y., et al. "Enhancing OCR Performance with Adaptive Preprocessing Techniques." *Journal of Imaging*, 2021.

[15] Li, X., et al. "CNN-Based Preprocessing Framework for Handwriting Recognition." *Pattern Recognition Letters*, 2019.

[16] Stolcke, A., & Brants, T. "Advances in Spell Correction Models for Handwritten Text Recognition." *Computational Linguistics*, 2020.

[17] Skurzok, Ł., & Nowiński, A. "Adapting Traditional Whiteboarding for Remote Education." *Springer*, 2024.