SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY GAUTAM
BUDDHA UNIVERSITY, GREATER NOIDA, 201312, U. P., (INDIA)

Candidate's Declaration

We hereby certify that the work embodied in this **Internship Report**, submitted in partial fulfilment of the requirements for the award of the degree of M.Sc. Computer Science, to the School of Information and Communication Technology, Gautam Buddha University, Greater Noida, is an authentic record of our own work carried out during the internship. This report is based on the learning and experience gained under the supervision of **Prof. Prakash Kumar Saraswat**, School of ICT. The content presented in this report is original and has not been submitted to any other University or Institute for the award of any other degree or diploma.

NAME	ROLL NO	SIGNATURE
NISHANT GOEL	235/PMD/001	

Supervisor's Certification
This is to certify that the above statement made by the candidates is correct to the best of my knowledge
and belief. However, responsibility for any plagiarism-related issue solely stands with the students.

Signature of the Supervisor:
Name with Designation: **Prof. Prakash Kumar Saraswat** (Supervisor)
Date:
Place: Greater Noida

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to all those who supported and guided me throughout the course of my internship project.

First and foremost, I extend my heartfelt thanks to **Prof. Prakash Kumar Saraswat**, whose constant encouragement, valuable insights, and expert guidance have been instrumental in shaping this internship experience into a meaningful learning journey.

I am also profoundly grateful to Mr. Saurabh Gupta, my Team Manager at Unicloud, for his continuous support, mentorship, and for providing me with the opportunity to be a part of challenging real-world projects. His constructive feedback and technical guidance have greatly enriched my practical knowledge and skill set.

I would also like to thank all the team members and staff at Unicloud for their cooperation and assistance, which made my internship a productive and enjoyable experience.

Lastly, I acknowledge the support of my institution, Gautam Buddha University, for facilitating this internship program and helping me take this significant step in my professional development.

List Of Tables

List Of Figures

ABSTRACT

In the era of digital transformation, organizations and individuals increasingly rely on large volumes of unstructured data such as PDF documents, multimedia content, and real-time visual feeds. Extracting meaningful insights from such diverse formats presents a significant challenge, especially when speed, accuracy, and automation are essential. This project addresses these challenges by proposing a set of intelligent AI-powered tools for natural language interaction, information retrieval, and structured data generation from both text and video sources.

The major highlight of this project is a system called "IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics," which allows users to interact with static PDF documents using natural language queries. By leveraging advanced models like Google's Gemini 1.5 Pro and various open-source LLMs such as Mixtral-8x7B, Zephyr-7B, and Baichuan-13B, the system can accurately retrieve and respond to user queries from large document sets. In addition, another key module focuses on educational automation—extracting questions from academic PDFs and converting them into a structured Excel-based format, ideal for building question banks or mock papers

.

Complementary contributions include a dialogue summarization system using a PEFT-tuned Phi-2 model with LoRA adapters, a real-time fire detection system using YOLOv8, and a people detection interface for bounding box visualization in video feeds. The project also features a portfolio-style internship website to showcase the complete integration and usability of all tools. Collectively, these systems automate document understanding, enhance multimedia analysis, and demonstrate real-world application of AI in academic and industrial contexts.

CHAPTER - 1

INTRODUCTION

This chapter introduces the context of the project, defines its objectives and scope, and outlines the purpose and structure of the report.

1.1 BACKGROUND OF THE PROJECT

In the age of digital transformation, the volume of documents generated, stored, and processed has grown exponentially. Extracting relevant information from these documents, particularly in PDF format, is time-consuming and challenging without intelligent tools. Traditional keyword-based search methods lack contextual understanding, making them inefficient for detailed document exploration. To address this issue, the project titled **"IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics"** was developed to facilitate natural language interaction with PDF files using state-of-the-art AI and NLP technologies.

With the increasing demand for automated educational tools and intelligent search systems, the need for a robust solution that can summarize content, answer user queries, and extract structured information has become critical. In addition to the main system, the project integrates several complementary modules. These include a **question extraction tool** that converts academic PDFs into structured Excel formats, **real-time fire and people detection** using YOLOv8, and a **demo portfolio website** built as part of an internship experience. Another notable module involves **dialogue summarization using a LoRA-adapted Phi-2 model**, where PEFT techniques are applied to efficiently generate concise summaries from long conversations. This contributes to tasks like meeting summarization, customer service automation, and educational content processing.

These innovations collectively demonstrate the use of cutting-edge models such as Google Gemini Pro 1.5, Mixtral-8x7B, Zephyr-7B, Baichuan-13B, and Microsoft Phi-2 with LoRA. Together, they enable intelligent querying, automated summarization, and content generation across multiple formats. The complete solution reflects strong applicability in both academic and industrial domains, while significantly reducing manual effort and enhancing document interaction.

1.2 OBJECTIVES

The key goals of this project are outlined below:

- **To develop an AI-powered system** that allows users to interact with the content of PDF documents using natural language queries, eliminating the need for manual reading and searching.

- **To create a chatbot interface** using Streamlit that offers a user-friendly and interactive experience for querying uploaded PDF documents in real-time.

- **To extract structured data**, such as multiple-choice or descriptive questions from academic PDFs, and export them to an Excel file for easy review and question bank generation.

- **To build real-time object detection systems** for identifying people and detecting fire in video streams using the YOLOv8 model, contributing to safety and surveillance applications.

- **To develop a demo internship portfolio website** that showcases the various modules and contributions made during the internship period, enhancing the presentation and usability of the overall system.

- **To integrate and compare multiple AI models** such as Gemini Pro 1.5, Mixtral-8x7B, Zephyr-7B, Baichuan-13B, and MiniLM to evaluate their performance for embeddings and natural language generation.

- **To implement a dialogue summarization module** using the Phi-2 transformer model with LoRA adapters through PEFT, enabling the system to condense lengthy conversations into meaningful summaries.

1.3 PURPOSE, SCOPE AND APPLICABILITY

Purpose:

This project addresses the growing challenge of extracting meaningful information from unstructured digital documents, particularly PDFs. As digital content expands, users struggle with manual scanning or basic keyword searches that miss context. The solution is an intelligent system enabling users to query documents in natural language and receive accurate, context-aware answers—streamlining the process and enhancing usability.

It integrates advanced AI models like Google Gemini Pro 1.5, Mixtral-8x7B, Baichuan13B, Zephyr-7B, and Microsoft Phi-2 (with LoRA adapters) to deliver deep content understanding, summarization, question generation (exportable to Excel), and real-time vision analytics (fire/people detection using YOLOv8). This open-source, customizable system outperforms existing tools (e.g., ChatPDF) by offering local deployment, full data control, and added visual analytics.

A live demo site showcases its practical relevance across education, research, and safety applications.

Scope:

The project assumes that users—especially in academic/research settings—face difficulties retrieving specific info from lengthy PDFs. Existing tools often have paywalls, limits, or lack offline capability. This project provides a locally hosted, flexible solution with natural language querying and summarization.

Challenges include the need for high-performance hardware for large models and reliable internet for cloud-based ones. The system uses text extraction, embedding (MiniLM/Gecko), FAISS for retrieval, and various LLMs for responses. It also features Excel export of generated questions and YOLOv8-based video analysis.

Applicability:

Applicable across academic, professional, and research fields, the system helps users quickly locate relevant info, generate assessments, and summarize dialogues. Its visual modules (fire/people detection) add value in safety, surveillance, and smart city monitoring. By offering a customizable, affordable alternative to commercial platforms, the project enhances digital accessibility and AI usability across diverse sectors—positioning itself as a scalable, evolving tool for education, automation, and public safety.

CHAPTER -2

Details of the Internship

2.1 Internship Experience at UniCloud Technologies Pvt. Ltd.

As part of my MSc program, I completed a three-month internship at UniCloud Technologies Pvt. Ltd., a leading company in cloud computing and AI solutions. The experience provided practical exposure to machine learning, data science, and industrystandard workflows.

About UniCloud Technologies Pvt. Ltd.:

UniCloud specializes in scalable cloud infrastructure and AI-driven solutions, offering services in:

- **Cloud Infrastructure Management:** Hosting, virtualization, and container orchestration.

- **AI/ML Solutions:** NLP, computer vision, and AI model deployment.

- **DevOps & Automation:** CI/CD pipelines and infrastructure-as-code.

- **SaaS Development:** Robust platforms across domains.

The company fosters innovation, collaboration, and continuous learning, making it an ideal learning environment.

Internship Objectives and Activities:

I worked with the AI & ML Research Team on:

1. **Model Fine-Tuning & Deployment:**

   o Fine-tuned LLaMA 3 8B, Mistral 7B, and Gemma 2B models using PEFT (LoRA) and 4-bit quantization for efficient deployment.

2. **Inference Pipelines:**

   o Built scalable pipelines for dialogue summarization and developed interactive demos using LangChain, Streamlit, and Python APIs.

3. **Cloud & DevOps:**

   o Gained experience in cloud integration, containerization, and production-ready AI deployments.

Skills and Learning Outcomes:

- Practical skills in transformer models, NLP, and computer vision.

- Experience with Hugging Face, Kaggle, Google Colab, and VS Code.

- Knowledge of model fine-tuning, quantization, and cloud-based deployment.

Conclusion:

The internship was a transformative experience that enhanced my technical skills and professional readiness. It provided valuable exposure to real-world AI applications, preparing me for future roles in computer science and AI.

2.2 Existing System

Several AI platforms like ChatPDF, AskYourPDF, and Humata enable users to query PDF documents in natural language. These tools break content into chunks, embed them using models (e.g., OpenAI embeddings, sentence-transformers), and retrieve relevant information. While effective, they have notable limitations:

- **Usage Restrictions:** Free plans often limit file size, query count, and document uploads.

- **Privacy Concerns:** As cloud-based systems, they risk exposure of sensitive data.

- **Lack of Customization:** These platforms don't allow integration of custom models, embedding controls, or UI modifications, and are not open-source—hindering offline use and institutional control.

In object detection, models like YOLOv5, YOLOv7, and YOLOv8 excel in tasks such as people or fire detection. However, they are typically standalone modules, not integrated with document processing systems, making multi-purpose deployments inefficient.

For question generation, many educational tools rely on manual or semi-automated methods, with limited automation in converting academic content into structured formats like Excel. Few systems offer flexible formatting or tagging.

In text summarization, advanced models like BART, T5, GPT-4, and Pegasus perform well but require heavy hardware and are often not fine-tuned for domain-specific tasks (e.g., dialogue summarization). Lightweight PEFT methods like LoRA are rarely used with accessible models (e.g., Microsoft Phi-2) in practical settings.

Overall, while strong individual tools exist for document querying, object detection, question generation, and summarization, they remain fragmented and lack integration. No current solution offers a unified, customizable, and locally deployable framework combining these capabilities. This project addresses that gap by providing an all-in-one, accessible, and flexible system.

2.3 Limitations of the Existing System

Current AI tools like ChatPDF, AskYourPDF, and Humata, despite their usefulness, have significant limitations. Most impose strict usage caps—on queries, document size, and uploads—unlockable only via paid plans, which is a barrier for students and institutions with limited budgets. Data privacy is another concern, as documents are processed on cloud servers, posing risks when handling sensitive academic or legal content.

Customization is minimal; these closed-source platforms don't allow swapping AI models, embedding methods, or storage strategies, making them unsuitable for research requiring model experimentation or domain-specific tuning. Typically, they rely on a single model, lacking multi-model flexibility and local deployment options—critical for offline or low-connectivity environments.

Few existing systems can automatically extract structured academic elements like questions from PDFs, and most rely on basic keyword matching without converting output into formats like Excel—vital for building question banks. Moreover, while YOLOv8 and similar object detection systems excel in real-time analytics, they aren't integrated with document interaction platforms, leaving opportunities for unified AI workflows untapped.

Dialogue summarization tools like GPT-4 and BART are powerful but hardware-intensive and often inaccessible without paid APIs. Limited adoption of lightweight tuning methods like LoRA also hinders affordable fine-tuning for specific datasets.

In summary, existing systems lack openness, scalability, and integration across document querying, question generation, summarization, and real-time object detection. This project directly tackles these gaps with a unified, customizable, and deployable framework, built for flexibility and low-resource environments.

2.4 Proposed System

In today's data-driven environment, professionals and educators face challenges extracting contextual information from PDFs and videos. Traditional methods like keyword searches are inefficient, while commercial solutions (e.g., ChatPDF, AskYourPDF) are limited, lack transparency, and offer little customization—especially problematic for academic or offline needs. Additionally, real-time monitoring for safety and surveillance is often overlooked by these platforms.

To address these gaps, the proposed system offers an integrated, intelligent framework that combines document querying, academic content extraction, dialogue summarization, and real-time object detection in a single, user-friendly solution. It is locally hostable, modular, and model-flexible, enabling seamless interaction via a responsive web interface.

Key Modules:

1. **Natural Language Interaction with PDFs:**
   Users can chat with PDFs via a chatbot interface. Text chunks from PDFs are embedded (using Gecko or MiniLM) and stored in FAISS for similarity search. LLMs like Gemini Pro, Mixtral-8x7B, and Zephyr-7B generate accurate, contextual responses.

2. **Academic Content Structuring:**
   The system extracts questions from academic PDFs and formats them into Excel spreadsheets—automating question bank creation for educators and minimizing manual work.

3. **Dialogue Summarization with Phi-2 + LoRA (PEFT):**

Using the Phi-2 model fine-tuned with LoRA adapters, this module summarizes long dialogues efficiently, making it ideal for meetings, interviews, or classroom discussions—even on modest hardware.

4. **Real-Time Fire and People Detection (YOLOv8):**
   This module analyzes video feeds frame-by-frame, detecting fire and people, with bounding boxes and confidence scores—enhancing safety monitoring and smart surveillance.

5. **Unified Web Interface:**
   Built with Streamlit, the UI lets users upload documents, interact with AI, monitor video feeds, and download outputs (e.g., Excel sheets) easily—ensuring an intuitive experience for all users.

2.5 BENEFITS OF THE PROPOSED SYSTEM

The proposed system offers a unified, open-source solution that enhances document querying, educational automation, dialogue summarization, and real-time object detection. Its key benefit is enabling natural language interaction with PDF documents, making it faster and easier to extract information. It supports multiple advanced language models (Gemini Pro, Mixtral-8x7B, Zephyr-7B, Baichuan-13B), offering flexibility and scalability across cloud or local setups. Unlike commercial tools, it's customizable, offline-capable, and free of usage limits. The question extraction module automates conversion of questions into structured Excel files, saving educators time. Dialogue summarization is optimized with lightweight LoRA-based Phi-2 models, allowing efficient summarization without high-end hardware. The system also integrates YOLOv8 for fire and people detection, broadening its scope to surveillance and safety. A Streamlit web interface makes it easy to use for all, promoting research and innovation in AI and Computer Vision.

2.6 FEATURES OF THE PROPOSED SYSTEM

The system combines document intelligence, academic automation, conversational summarization, and real-time video analysis. Key features include:

- **Natural Language PDF Querying:** Chat-based interface powered by vector search and large language models.

- **Multi-Model Support:** Works with Gemini Pro, Mixtral, Baichuan, Zephyr, and multiple embedding models.

- **Question Extraction:** Detects questions from PDFs and exports them to Excel in a structured format.

- **Dialogue Summarization:** Uses Phi-2 with LoRA for efficient, domain-specific summarization.

- **Object Detection:** YOLOv8 detects fire and people in real-time video streams.

- **Streamlit Web App:** User-friendly, browser-based interface with no technical setup needed.

- **Local Deployment:** Fully offline-capable, no subscriptions, customizable, and opensource.

2.7 SYSTEM REQUIREMENTS SPECIFICATION

The system must:

- Allow natural language queries on PDFs with accurate, context-based answers.

- Enable question extraction from educational PDFs and export to Excel. • Summarize long dialogues using LoRA-tuned Phi-2 models efficiently.

- Detect fire and people in videos in real time with adjustable settings.

- Integrate all modules into a web app showcasing uploads, queries, summaries, and video analysis.

- Ensure smooth performance, intuitive UI, offline functionality, and flexibility for advanced users.

2.7.1 USER CHARACTERISTICS

- **Students:** Query academic PDFs, summarize transcripts.

- **Teachers:** Automate question bank creation.

- **Researchers:** Extract targeted info, compare model outputs.
- **Content Creators:** Build quizzes and learning modules.

- **Developers:** Customize and expand the system.

- **Admins:** Generate summaries and reports.

- **Security Staff:** Monitor videos for safety alerts.

- **Institutions:** Host shared access with role-based controls.

2.7.2 SOFTWARE AND HARDWARE REQUIREMENTS

Each module in the proposed system has distinct computational and software needs due to its use of different AI models, libraries, and user interaction features. The following subsections define the **software and hardware requirements separately for each major project component**.

2.7.2.1 IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics – Intelligent Document Querying

Software Requirements

- **Operating System:** Windows 10/11, Ubuntu 20.04+, or macOS

- **Python Version:** Python 3.10 or above **Libraries/Dependencies:**

- streamlit – for user interface

- PyPDF2 – for PDF text extraction

- langchain, langchain_community – for prompt chaining and processing

- sentence-transformers – for MiniLM-based embedding

- faiss-cpu – for similarity search

- google.generativeai – for integrating Gemini Pro via API

- dotenv – to load API keys securely

- torch – for model inference support

Tools: venv or Conda for environment setup

Hardware Requirements

- **Processor:** Minimum Intel Core i5 or AMD Ryzen 5

- **RAM:** At least 8 GB; 16 GB recommended

- **Storage:** 10–20 GB free space for documents and vector indices
- **GPU:** Not mandatory; can run fully on CPU

- **Display:** HD screen for Streamlit UI (1280x720 or higher)

2.7.2.2 Question Extraction and Excel Generation

Software Requirements

- **Python Version:** Python 3.10+ **Libraries/Dependencies:**

- PyPDF2 – for reading PDF documents

- pandas – for handling tabular data

- openpyxl – for Excel (.xlsx) file creation

- streamlit – to interact via browser

- langchain – to structure prompt-based extraction

- huggingface_hub, transformers – for model-based generation if required

Hardware Requirements

- **Processor:** Any dual-core processor

- **RAM:** Minimum 4 GB; 8 GB recommended

- **Storage:** ~5 GB for PDFs and Excel outputs

- **GPU:** Not required

- **Display:** Standard desktop or laptop resolution

2.7.2.3 Dialogue Summarization using LoRA-based Phi-2

Software Requirements

- **Python Version:** Python 3.10+ **Libraries/Dependencies:**

- datasets – for loading Hugging Face datasets

- transformers – to load base model (Phi-2)

- peft – for loading LoRA adapter on Phi-2

- torch – to support inference

- huggingface_hub – to connect to pre-trained models

- **Optional:** Jupyter Notebook for experimentation

Hardware Requirements

- **Processor:** Intel Core i5 or equivalent

- **RAM:** At least 8 GB; 12–16 GB recommended for longer summaries

- **Storage:** 5–10 GB for dataset, tokenizer, LoRA adapter files

- **GPU:** Not necessary but will improve generation speed (NVIDIA 4GB+)

- **Display:** Not UI-intensive; command-line or notebook sufficient

2.7.2.4 Real-Time People and Fire Detection using YOLOv8

Software Requirements

- **Python Version:** Python 3.10+ **Libraries/Dependencies:**

- ultralytics – official YOLOv8 wrapper

- opencv-python – for video frame extraction and display

- torch – for model inference (must be compatible with GPU)

- numpy – image processing support

- streamlit or custom HTML/JS UI – to visualize detections

- **Optional:** CUDA/cuDNN for GPU acceleration

Hardware Requirements

- **Processor:** Intel Core i7 or Ryzen 7

- **RAM:** Minimum 8 GB; 16 GB preferred

- **Storage:** 10–15 GB for model files and video processing

- **GPU:** Strongly recommended (NVIDIA GTX 1650 or better)

- **Display:** Full HD display for bounding box overlays

2.7.2.5 Internship Website Demonstration

Software Requirements

- **Frontend:** HTML, CSS, JavaScript (React optional) ● **Backend:** Streamlit / Flask or similar for Python integration ● **Python Dependencies:**
- All packages required by the four modules integrated into one environment
- os, pathlib, shutil – for backend file handling ● **Hosting Tools (optional):**
- Heroku, Streamlit Cloud, or local server deployment **Hardware Requirements**

- **Processor:** Any modern CPU (dual-core and above)
- **RAM:** 8 GB minimum to load modules concurrently
- **Storage:** ~10 GB total for website assets and processed data
- **GPU:** Not mandatory unless inference is live
- **Display:** Responsive layout to support laptops, tablets, and desktop monitors

Summary:

This modular breakdown ensures each component of the system has the right tools and platform to function effectively. While some modules are lightweight and work on general-purpose laptops, others — like YOLOv8 detection — benefit greatly from GPU acceleration. By designing flexible hardware/software requirements for each module, the system can be scaled and deployed in both **resource-constrained academic labs** and **production-grade environments**.

CHAPTER -3

System Design

The system design phase transforms the problem requirements into a blueprint for construction. It includes architectural design, data flow, relationships among entities, user interface layout, and output report structures. The system is modular, with each module addressing a specific aspect of document processing, AI-based summarization, detection, or web deployment.

Module 1: IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics (AIPowered PDF Question Answering)

User
↓
Upload PDF Files
↓
Extract Text with PyPDF2
↓
Split Text into Chunks
↓
Generate Embeddings
↓
Store in FAISS Vector DB
↓
User Asks Question?
→ Search FAISS for Relevant Chunks → Generate Prompt with Context → Get Answer from Gemini Pro
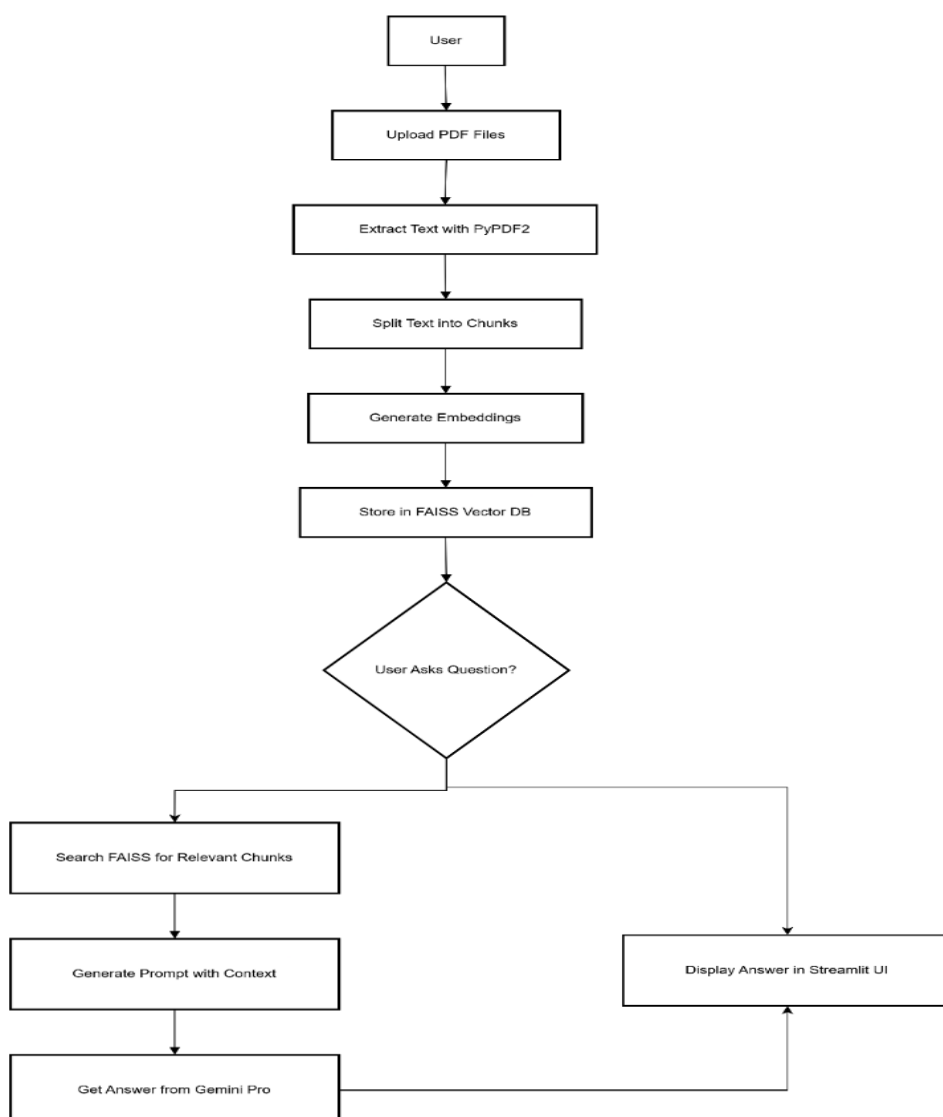→ Display Answer in Streamlit UI

Fig.3.1

Workflow of the PDF Query System

The PDF Query System lets users upload PDFs and ask questions via a web interface. The process involves these steps:

1. Upload PDF Files:
Users upload PDFs through a Streamlit interface.

2. Text Extraction:
   PyPDF2 extracts text. OCR can be added later for

scanned/image PDFs. 3. Text Chunking:

   LangChain's RecursiveCharacterTextSplitter splits text

into smaller, meaningful chunks for better context handling.

4. Embedding Generation:

   Chunks are converted into embeddings using Google

Gemini's embedding-001 model to capture their

meaning. 5. Storing Embeddings:

   Embeddings are stored in a FAISS vector database for

fast similarity searches. 6. User Query:

   Users enter a question, which is embedded using the

same Gemini model. 7. Context Retrieval:

   FAISS retrieves the top 3–5 most relevant chunks based

on the query. 8. LLM Response:

   Gemini 1.5 Pro uses the retrieved chunks to generate an

answer. If no answer is found, it replies "Not in context."

Temperature is set to 0.3 for accuracy. 9. Answer Display:

   The response is shown in the Streamlit interface for the user.
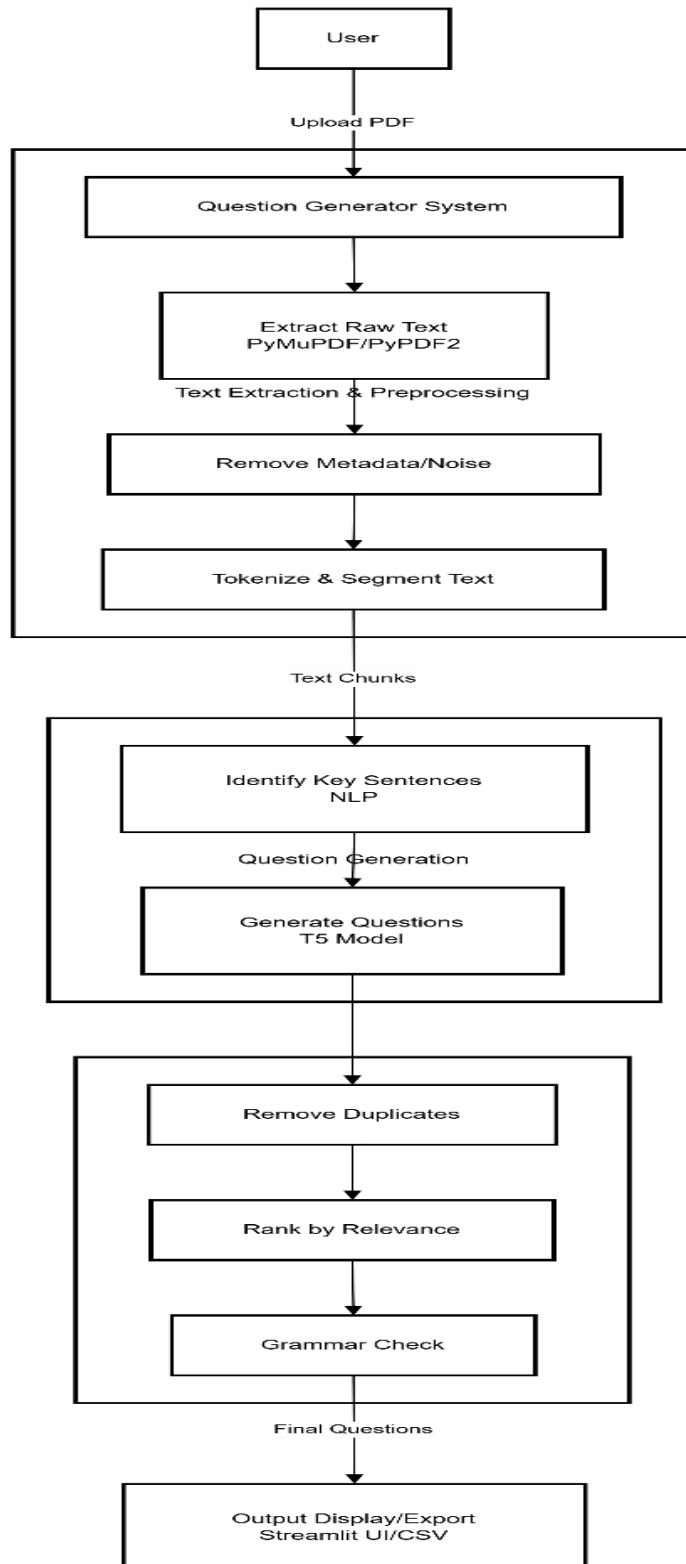
Module 2: Question Generator from PDF


```
                        ┌──────────────┐
                        │     User     │
                        └──────────────┘
                              │
                          Upload PDF
                              │
      ┌───────────────────────────────────────────┐
      │   ┌───────────────────────────────────┐   │
      │   │    Question Generator System      │   │
      │   └───────────────────────────────────┘   │
      │                  │                         │
      │   ┌───────────────────────────────────┐   │
      │   │       Extract Raw Text            │   │
      │   │       PyMuPDF/PyPDF2              │   │
      │   └───────────────────────────────────┘   │
      │        Text Extraction & Preprocessing     │
      │   ┌───────────────────────────────────┐   │
      │   │       Remove Metadata/Noise       │   │
      │   └───────────────────────────────────┘   │
      │                  │                         │
      │   ┌───────────────────────────────────┐   │
      │   │     Tokenize & Segment Text       │   │
      │   └───────────────────────────────────┘   │
      └───────────────────────────────────────────┘
                          │
                      Text Chunks
                          │
      ┌───────────────────────────────────────────┐
      │   ┌───────────────────────────────────┐   │
      │   │     Identify Key Sentences        │   │
      │   │             NLP                   │   │
      │   └───────────────────────────────────┘   │
      │           Question Generation              │
      │   ┌───────────────────────────────────┐   │
      │   │      Generate Questions           │   │
      │   │          T5 Model                 │   │
      │   └───────────────────────────────────┘   │
      └───────────────────────────────────────────┘
                          │
      ┌───────────────────────────────────────────┐
      │   ┌───────────────────────────────────┐   │
      │   │       Remove Duplicates           │   │
      │   └───────────────────────────────────┘   │
      │                  │                         │
      │   ┌───────────────────────────────────┐   │
      │   │       Rank by Relevance           │   │
      │   └───────────────────────────────────┘   │
      │                  │                         │
      │   ┌───────────────────────────────────┐   │
      │   │        Grammar Check              │   │
      │   └───────────────────────────────────┘   │
      └───────────────────────────────────────────┘
                          │
                    Final Questions
                          │
               ┌───────────────────────┐
               │   Output Display/Export│
               │    Streamlit UI/CSV   │
               └───────────────────────┘
```

Fig.3.2

Question Generator from PDF – Level 1 Workflow Explanation

The Level 1 workflow provides a deeper look into the internal steps involved in the automatic question generation system. This module transforms raw academic content from PDF files into structured questions through a multi-stage process involving text extraction, natural language processing, and generation models.

1) **Text Extraction & Preprocessing:** • Extract Raw Text:

   > Text is extracted using PyMuPDF or PyPDF2. OCR (e.g., Tesseract) can be added later for scanned PDFs.

   • Clean Text:

Removes metadata, extra whitespace, and artifacts.

   • Segment Text: Splits text into sentences/paragraphs using NLTK or spaCy.

2) **Question Generation:** • Identify Key Sentences:

Finds important sentences (definitions, concepts) using NLP heuristics.

   • Generate Questions:

   > Uses models like valhalla/t5-base-qg-hl to generate questions.

   > *Example:*Input: "Photosynthesis converts sunlight into chemical energy."Output: "What process converts sunlight into chemical energy?"

3) **Post-Processing:** • Remove Duplicates:

   > Eliminates repeated questions. •
   Rank by Relevance:

   > Scores questions by importance (keywords/position).

   • Grammar Check:

   > Uses language-tool-python to improve fluency.

4) **Output & Export:** • Display in Web UI:

   > Shows questions in the Streamlit app.

   • Export Options:

   > Download as CSV/Excel via pandas for reuse in quizzes or LMS platforms

Key Technologies Used

Step	Tools/Libraries
PDF Extraction	PyMuPDF, PyPDF2
NLP Processing	NLTK, spaCy
Question Generation	Hugging Face Transformers (T5 models)
Export/Display	Streamlit, pandas, language-toolpython

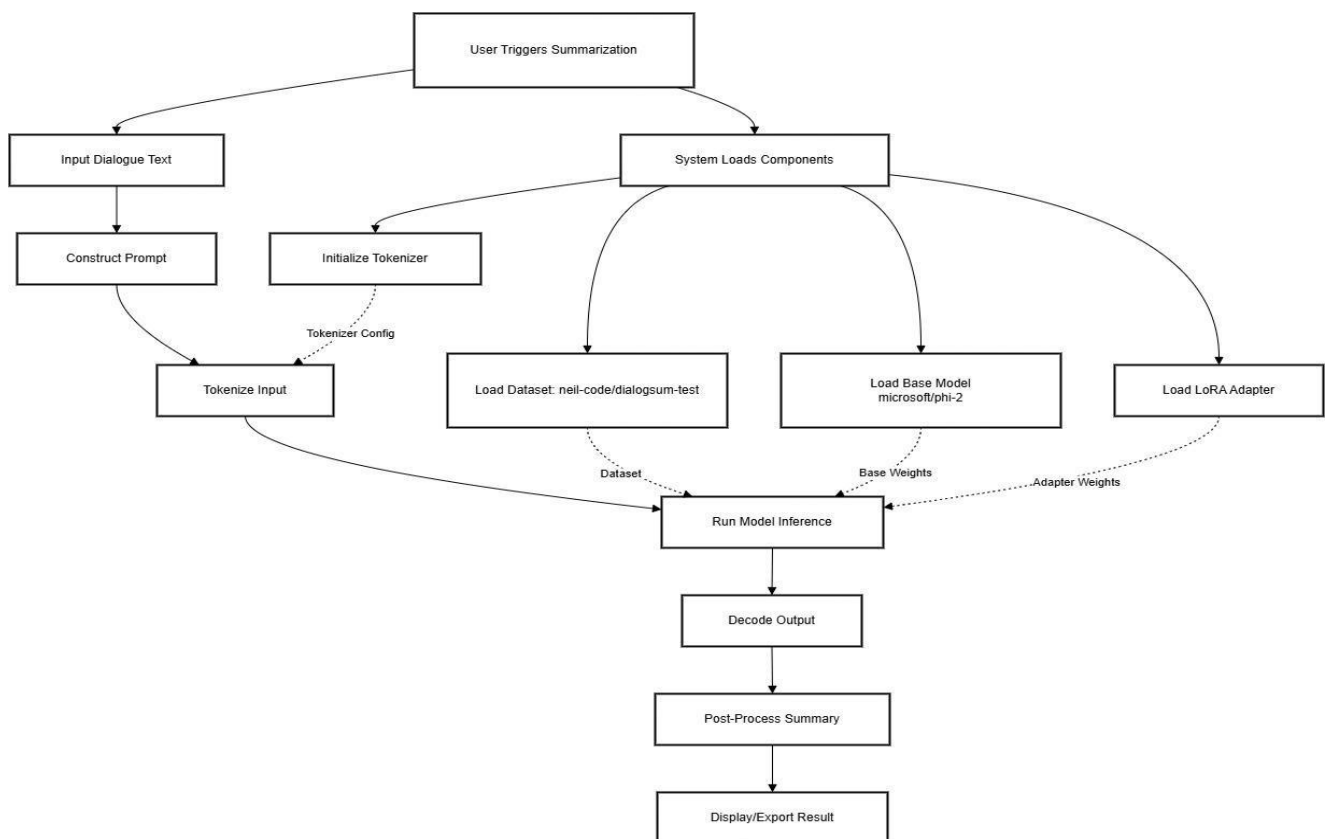**Module 3:   Dialogue Summarization System (with Phi-2 + LoRA Adapter)**

Fig.3.3

23

Dialogue Summarization System (Phi-2 + LoRA Adapter)

This module condenses long dialogues into brief summaries using Microsoft's lightweight Phi-2 model enhanced with a LoRA adapter for efficient fine-tuning.

Phase 1: Model Setup

- Load Dataset:
  Fetches neil-code/dialogsum-test from Hugging Face, containing dialogues + reference summaries.
- Initialize Tokenizer:
  Loads the Phi-2 tokenizer, setting EOS for padding & enabling truncation for proper formatting.
- Load Model + LoRA Adapter:
  Loads microsoft/phi-2 with AutoModelForCausalLM and integrates a LoRA adapter via PeftModel.from_pretrained() to add summarization without full retraining.

Phase 2: Summarization Execution

- Prompt Construction:
  Builds a structured prompt for each dialogue.
- Tokenization:
  Converts prompts into token IDs + attention masks (with padding/truncation).
- Model Inference:
  Runs model.generate() with: o
  max_new_tokens = 150 o
  pad_token_id from tokenizer o
  set_seed(42) for reproducibility.
- Output Decoding:
  Decodes token IDs into clean summaries (removing special tokens/prompt echoing).

Phase 3: Result Handling

- Display Summary:
  Shows the generated summary in the console or via Streamlit UI.
- Reference Comparison (Optional):

Displays the human-written summary for side-by-side evaluation.

Key Tools:

Step	Tools/Libraries
Dataset Loading	datasets (Hugging Face) – neil-code/dialogsum-test
Tokenization	AutoTokenizer (Hugging Face Transformers)
Base Model	microsoft/phi-2 via AutoModelForCausalLM
LoRA Adapter	peft – PeftModel.from_pretrained()
Inference Pipeline	torch, transformers
Decoding	tokenizer.decode(), prompt cleanup logic
Reproducibility	set_seed(42)
Output Display	Console, Streamlit (optional)

CHAPTER -4

ER DIAGRAM

This section outlines the relational structure of the system. The ER diagram represents the core entities involved in the **IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics** system, the relationships between them, and how data flows from document upload to user query resolution.

Module 1: IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics – Entity Relationship Diagram

The entity-relationship model for Module 1 captures the flow of user-uploaded PDFs through the document processing pipeline into searchable chunks and embeddings. It also maps how users interact with the system through queries and receive intelligent responses.

Entity Relationships Overview

- **USER** uploads multiple **PDFs**
- Each **PDF** contains many **TEXT_CHUNKS**
- Each **TEXT_CHUNK** has one corresponding **EMBEDDING**
- A **USER** can make many **QUERY** entries
- Each **QUERY** generates a **RESPONSE**

This logical structure supports scalable document analysis, search, and conversational interfaces via AI.
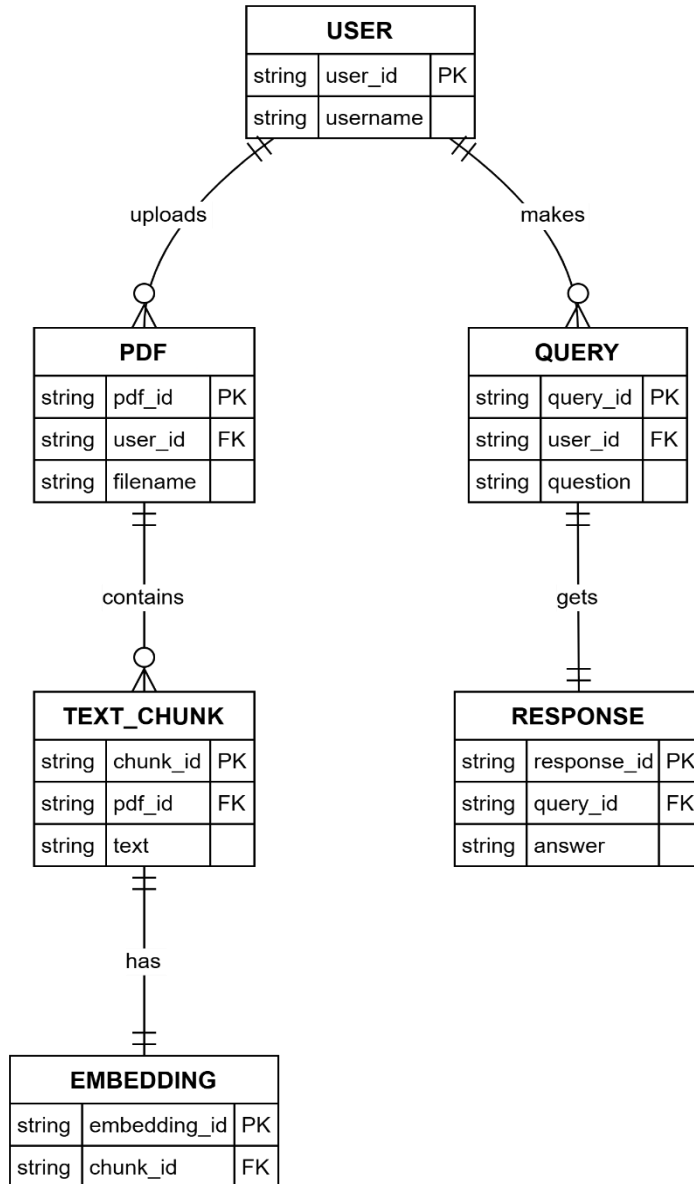
Fig.4.1

Module 2: Question Generator from PDF – Workflow Explanation

The **Question Generator from PDF** module is designed to automatically extract, segment, and transform educational PDF content into meaningful questions. This system supports both student and educator roles and enables export of questions in multiple formats such as CSV or UI-based test interfaces. The ER diagram highlights the flow of data and the interrelationship between key components.

Workflow Based on ER Diagram

1. User Interaction and PDF Upload

- A **USER** (either a teacher or student) logs into the system.
- The user uploads a PDF file containing educational material.
- The file is logged in the **PDF_UPLOAD** table, storing:
    - upload_id, user_id, upload_time, and the original filename. PDF content can be stored as binary data for backup/reference.

ER Link: USER ||--o{ PDF_UPLOAD : uploads

2. PDF Segmentation into Text Chunks

- The uploaded PDF is parsed using libraries like PyMuPDF or PyPDF2.
- It is split into manageable **TEXT_CHUNK** segments (e.g., per paragraph or sentence) for analysis.
- Each chunk is linked to the corresponding upload_id and contains:
    - chunk_id, chunk_text, and page_number from the PDF.

ER Link: PDF_UPLOAD ||--|{ TEXT_CHUNK : contains

3. AI-Powered Question Generation • Each **TEXT_CHUNK** is passed into a question generation model (e.g., T5 or GPTbased).

- The model analyzes context and generates one or more **GENERATED_QUESTION** records with:
    - question_id, question_text, question_type (factual/inferential), and confidence_score.
- Questions are directly mapped to their originating chunks for traceability.

ER Link: TEXT_CHUNK ||--|{ GENERATED_QUESTION : generates

4. Export and Output Handling

- Once generated, questions may optionally be:
    - Displayed in the web UI, o   Exported to Excel/CSV format, or o Integrated into LMS platforms.
- These are logged in the **QUESTION_OUTPUT** table, capturing:
    - output_format, export_time, and linkage to the question_id.

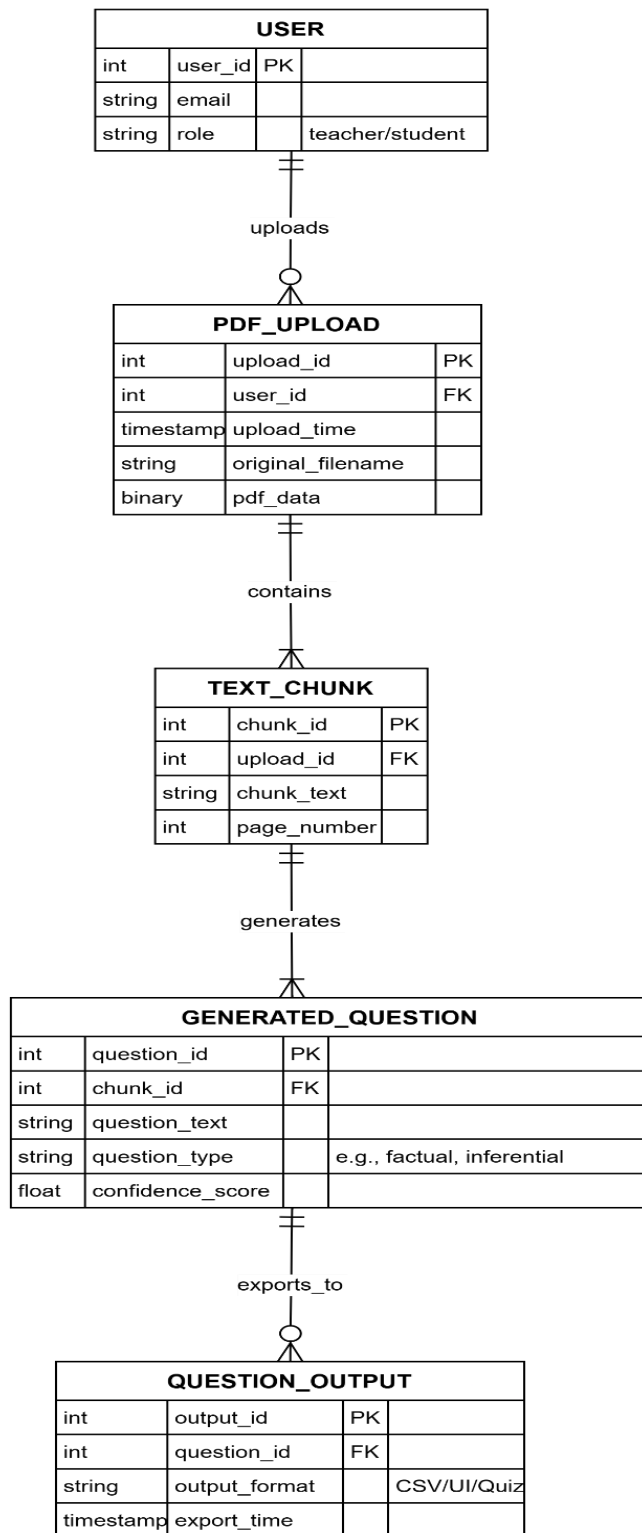*ER Link:* GENERATED_QUESTION ||--o{ QUESTION_OUTPUT : exports_to

USER

int	user_id	PK	
string	email		
string	role		teacher/student

uploads

PDF_UPLOAD

int	upload_id	PK
int	user_id	FK
timestamp	upload_time	
string	original_filename	
binary	pdf_data	

contains

TEXT_CHUNK

int	chunk_id	PK
int	upload_id	FK
string	chunk_text	
int	page_number	

generates

GENERATED_QUESTION

int	question_id	PK	
int	chunk_id	FK	
string	question_text		
string	question_type		e.g., factual, inferential
float	confidence_score		

exports_to

QUESTION_OUTPUT

int	output_id	PK	
int	question_id	FK	
string	output_format		CSV/UI/Quiz
timestamp	export_time		

Fig.4.2

Module 3: Dialogue Summarization – Workflow Explanation

This module enables intelligent summarization of conversational data using a fine-tuned transformer model (e.g., microsoft/phi-2 with LoRA adapters). The system captures dialogues,

29

processes them into structured prompts, tokenizes the input, and generates summaries through model inference. The workflow is deeply connected with the ERD structure that ensures traceability, reproducibility, and performance tracking.

Workflow Based on ER Diagram

1. Dialogue Submission by User

- A USER interacts with the system via CLI or a web interface and submits a new DIALOGUE.
- Each dialogue is linked to a user_id and stored with a timestamp and content. **ER Link:**

  USER ||--o{ DIALOGUE : submits

2. Prompt Creation

- The raw DIALOGUE content is combined with a fixed instruction template to form a PROMPT.
- Example Template:
  "Instruct: Summarize the following conversation.\n[dialogue]\nOutput:" **ER**

Link: DIALOGUE ||--|| PROMPT : converted_to

3. Tokenization of Input

- The PROMPT is tokenized using the tokenizer associated with the base model (phi-2) using Hugging Face.
- The token IDs are stored in TOKENIZED_INPUT along with a reference to the prompt.

  ER Link: PROMPT ||--|| TOKENIZED_INPUT : processed_into

4. Model Inference

- The TOKENIZED_INPUT is passed to the summarization model (e.g., phi-2 + LoRA) for generation.
- A fixed seed ensures deterministic outputs.
- Output tokens are stored in the MODEL_OUTPUT entity for traceability and decoding.

  ER Link: TOKENIZED_INPUT ||--|| MODEL_OUTPUT : generates

30

5. Summary Generation

- The MODEL_OUTPUT tokens are decoded into readable text to form the SUMMARY.
- The SUMMARY is linked back to the original USER and model OUTPUT.

ER Link:

- MODEL_OUTPUT ||--|| SUMMARY : decoded_to
- USER ||--o{ SUMMARY : receives

6. Model and Dataset Tracking

- Each MODEL_OUTPUT is influenced by a specific MODEL_CONFIGURATION:
    - Base model (e.g., microsoft/phi-2)
    - Adapter (e.g., LoRA) o   Dataset   used   (e.g.,   neil-code/dialogsum-test)
- This ensures reproducibility of results and proper audit logging for future comparisons.

ER Links:

- MODEL_CONFIGURATION ||--o{ MODEL_OUTPUT : influences
- DATASET ||--o{ MODEL_CONFIGURATION : trains

USER

string	user_id	PK	
string	role		admin/end-user

submits

DIALOGUE

string	dialogue_id	PK
text	content	
timestamp	created_at	
string	user_id	FK

converted_to

PROMPT

string	prompt_id	PK
text	template	
string	dialogue_id	FK

DATASET

string	dataset_id	PK	
string	name		neil-code/dialogsum-test
text	description		

processed_into

trains

TOKENIZED_INPUT

string	input_id	PK
int[]	token_ids	
string	prompt_id	FK

MODEL_CONFIGURATION

string	config_id	PK	
string	base_mode		microsoft/phi-2
string	adapter		LoRA
string	dataset_id	FK	

generates

influences

MODEL_OUTPUT

string	output_id	PK
int[]	output_tokens	
string	input_id	FK

receives

decoded_to

SUMMARY

string	summary_id	PK
text	content	
string	output_id	FK
string	user_id	FK

Fig.4.3

32

CHAPTER -5

ALGORITHM

Implementation

5.1 IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics – Natural Language Query System

The "IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics" module represents the foundational component of the entire project. It leverages recent advances in Natural Language Processing (NLP), vector embeddings, and large language models (LLMs) to enable intelligent querying of static PDF documents through a conversational interface. This module transforms traditionally passive content—academic notes, research articles, or reports—into interactive, searchable knowledge sources.

Objective of the Module

The key objective is to allow users to interact with unstructured PDF content using natural language queries. Users can upload one or more PDFs and ask questions in English. The system responds with contextually relevant, coherent answers using the Gemini Pro model, based on the most relevant sections of the uploaded PDFs. This system mimics the behavior of a human assistant who has read and understood the documents.

Algorithm: IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics Using LangChain and Gemini Pro

ModuleName: Natural Language PDF Question Answering
Objective: Enable semantic interaction with PDF documents using a conversational AI interface
Input: One or more PDF files uploaded by the user; a user-typed question
Output: A relevant, well-formed answer extracted and generated from document content

Step-by-Step Algorithm Description

Step 1: Load Environment and Configure API

- **1.1** Securely load environment variables using the dotenv library to access sensitive keys like the GOOGLE_API_KEY.

- **1.2** Authenticate the session by configuring the Gemini API using the loaded credentials. This prevents hardcoding keys in source code, ensuring best practices in security and maintainability.

Step 2: Upload and Extract Text from PDF Files

- **2.1** Using Streamlit's file uploader widget, allow users to select and upload one or multiple PDF documents.
- **2.2** For each uploaded file, extract text page-by-page using PdfReader from the PyPDF2 library.
- **2.3** Concatenate all extracted text into a single raw string. This consolidated text will form the document base for all future search and inference operations.
- **2.4** (Optional Extension): Future versions of the system can incorporate OCR (Optical Character Recognition) capabilities to handle scanned documents or image-based PDFs.

Step 3: Text Chunking and Preprocessing

- **3.1** Since language models have input size limitations (e.g., token limits), the raw text must be divided into smaller segments.
- **3.2** Use LangChain's RecursiveCharacterTextSplitter to split the text into meaningful "chunks." Each chunk is approximately 1000 characters with 200 characters of overlap to maintain context continuity.
- **3.3** This chunking strategy ensures semantic coherence while also allowing the system to localize relevant content during query matching.

Step 4: Generate Semantic Embeddings and Store in FAISS

- **4.1** Pass each text chunk into Google's embedding-001 model via the GoogleGenerativeAIEmbeddings class. This converts natural language into highdimensional vector representations.
- **4.2** Embeddings capture the semantic meaning of each chunk, enabling contextual retrieval instead of simple keyword matches.
- **4.3** Store all vector embeddings along with their associated text in a local FAISS index. This index supports high-speed nearest-neighbor searches based on cosine similarity.
- **4.4** Save the FAISS index to disk for persistent storage and reuse across sessions.

Step 5: Accept User Query and Convert to Vector Representation

- **5.1** Allow the user to type a question in plain English using Streamlit's text input box.
- **5.2** Convert the user's question into a semantic embedding using the same embedding001 model to ensure alignment with stored text vectors.
- **5.3** Use FAISS to perform a similarity search, retrieving the top-N most relevant text chunks (typically 3 to 5).
- **5.4** These chunks now represent the "context" that will guide the language model's response generation.

Step 6: Generate Answer Using Gemini Pro

- **6.1** Design a custom prompt template that explicitly instructs the model to answer only from the provided context.
- **6.2** Initialize Gemini Pro (gemini-1.5-pro) as the LLM for generating answers.
- **6.3** Use LangChain's load_qa_chain() with the stuff chain type to combine context and prompt.
- **6.4** Feed the retrieved chunks and user question into the QA chain to generate a final answer.
- **6.5** The model is instructed to return "answer is not available in the context" if relevant information is not found.

Step 7: Display Results and Handle Errors

- **7.1** Display the final answer directly in the Streamlit interface for the user.
- **7.2** Include exception handling for issues like missing API keys, corrupt PDF uploads, or embedding failures.
- **7.3** Provide real-time feedback via progress indicators (e.g., spinners) and success messages.
- **7.4** (Optional): Allow users to regenerate answers, switch LLMs, or export the Q&A session.

System Design Considerations

- **Modularity**: Each component (PDF reader, chunker, embedder, retriever, generator) is decoupled for easy replacement or upgrading.
- **Scalability**: The FAISS vector store allows future extension to thousands of documents with negligible performance degradation.
- **Privacy**: The system can operate entirely offline if embedding and LLM models are deployed locally.
- **Customizability**: Developers can swap Gemini for another model (e.g., Mixtral, Claude) by changing a few parameters.

Output
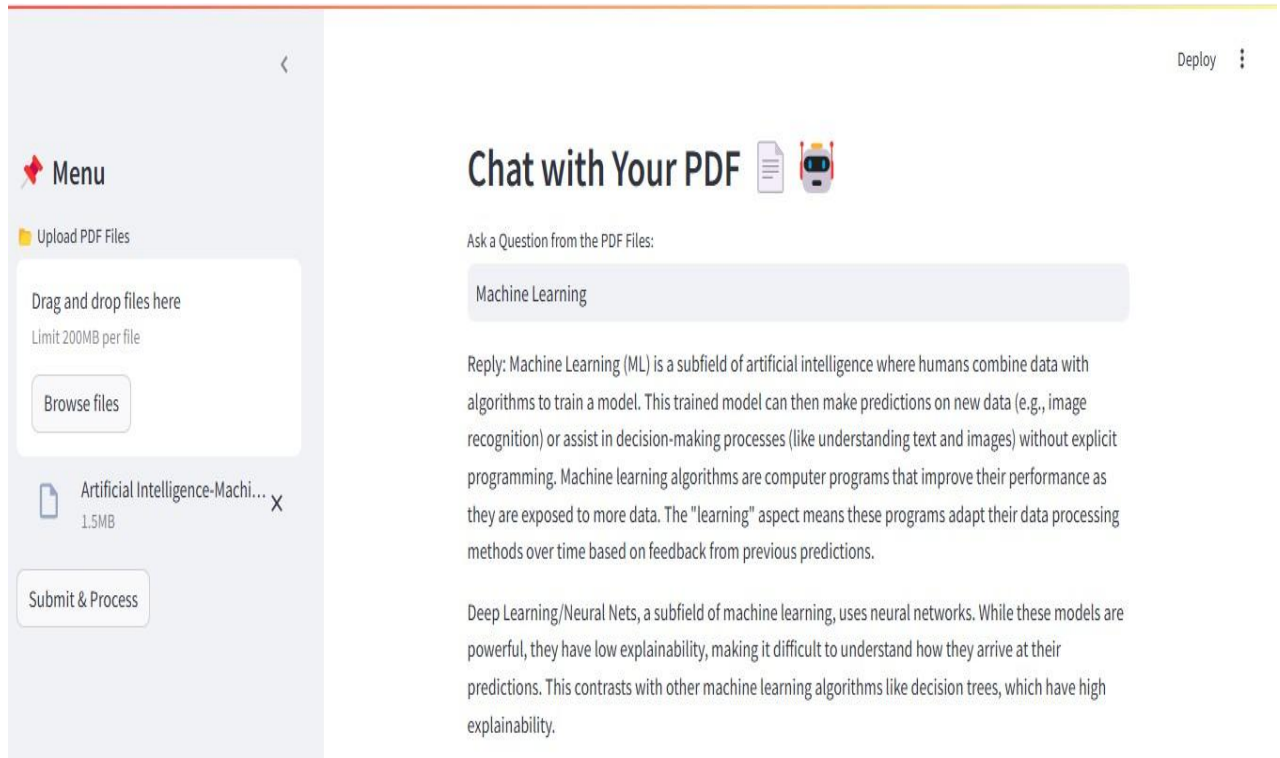
Fig.5.1

5.2 Automated Question Paper Generator System

The Automated Question Paper Generator System is a robust, modular pipeline designed to convert educational PDFs into fully formatted, examination-ready question papers. It automates the traditionally manual process of exam creation by integrating document analysis, question generation using large language models, and structured paper composition with customizable formatting. The system is capable of producing multiple-choice and descriptive questions from input materials, organizing them by section, allocating marks, and exporting in multiple formats including PDF, CSV, and HTML.

This module supports two modes of operation: a web interface for manual control and an APIbased system for automation and integration with other educational platforms. It is built with scalability in mind, allowing educational institutions to generate personalized, qualitycontrolled assessments with minimal human intervention.

Objective of the Module

To transform educational content from static PDF format into structured, semantically rich question papers with balanced mark distribution and formatting appropriate for institutional examination standards.

Input: Educational PDF documents, generation parameters (e.g., number of questions, types, marks per section)

Output: A structured question paper in formats such as PDF, CSV, and HTML

Algorithm: Intelligent Assessment Generator

Step 1: Document Ingestion

1.1 Accept PDF documents from the user through either a web interface or REST API endpoint.

1.2 Validate the file type and ensure it meets the maximum allowed file size criteria. 1.3 Extract raw textual content from the PDF using a text parser. If PyPDF2 fails due to layout or encoding issues, switch to an alternative loader as a fallback.

1.4 Clean the extracted text by removing headers, footers, page numbers, and special characters. Normalize whitespace and line breaks to prepare for chunking.

Step 2: Content Processing

2.1 Divide the cleaned text into semantic chunks of approximately 1500 characters each, with 300 characters of overlap to preserve cross-chunk context.

2.2 Apply text normalization techniques such as lowercasing and removal of unwanted symbols or encoding artifacts.

2.3 Structure chunks by preserving logical breaks like section headings, numbered lists, or bullet points if present in the original document.

Step 3: Question Generation

3.1 Initialize a local language model (e.g., Mistral-7B) with generation parameters including:

- Temperature: 0.5 to balance factual accuracy and variability
- Max tokens: 1024 to control length of generated outputs
- Top-p sampling: 0.9 for maintaining diversity

3.2 Generate multiple-choice questions using prompt templates that enforce a four-option format. Include validation logic to ensure each question has:

- A clear stem
- Four plausible options
- A marked correct answer
- An explanation field (optional)

3.3   Generate descriptive questions with emphasis on conceptual clarity and relevance. Limit the expected answer range to between 50 and 100 words for practicality.

3.4   Implement retry logic in case of model failures or context window errors. For each failed generation:

- Retry a maximum of three times
- Reduce context length progressively to manage memory
- Switch to batch generation mode for large text volumes

Step 4: Paper Assembly

4.1 Validate the final question pool to ensure it meets the input configuration:

- Total number of questions
- Question types (MCQ vs Descriptive)
- Sectional mark distribution
- Instruction headers

4.2   Organize selected questions into paper sections. Group MCQs and descriptive questions separately, if required.

4.3   Prioritize higher-quality questions (based on model confidence or structure score) and avoid duplication of content across sections. 4.4 In cases where the question pool is insufficient:

- Reduce the number of questions proportionally
- Notify the user of shortages
- Fill with the next best available questions based on scoring criteria

Step 5: Output Generation

5.1 Format the composed paper into a printable PDF layout with consistent styling and proper pagination. Include instructions, section titles, and spacing as per academic guidelines. 5.2 Generate a CSV version of the paper that contains each question along with its metadata (type, marks, options). This output can be used for import into databases or LMS systems. 5.3 Provide an HTML preview with responsive design for in-browser review. Include visual indicators for section marks, question types, and layout validation.

Step 6: Delivery System

6.1 Make available RESTful API endpoints for:
- Processing uploaded PDFs
- Generating question papers
- Downloading finalized outputs
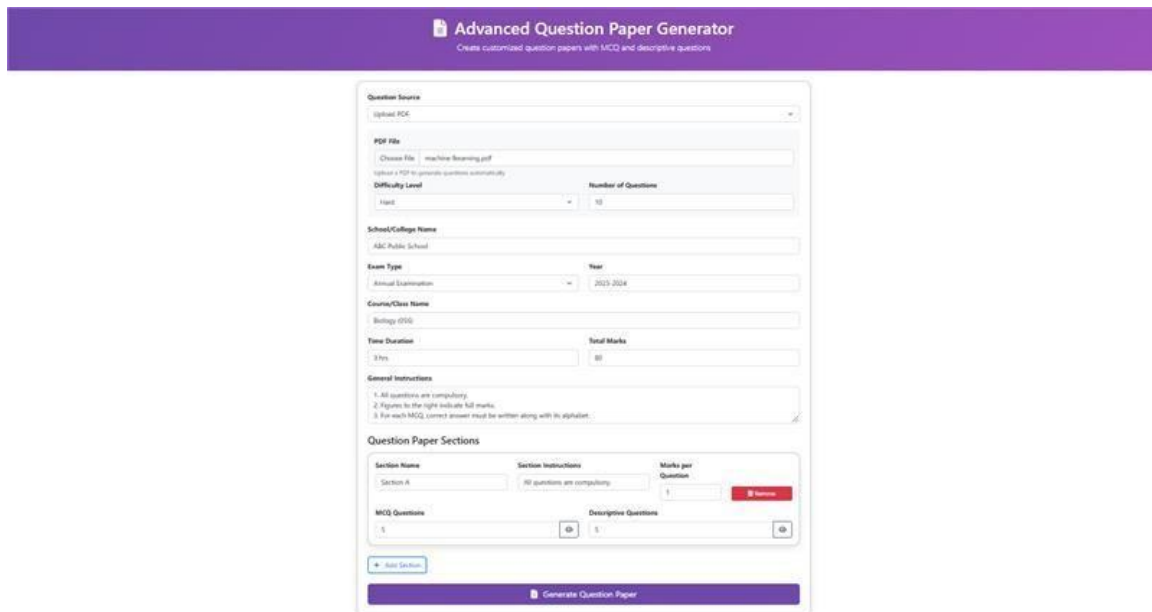
6.2 Offer a web interface for manual usage that allows users to:

- Upload documents
- Configure paper structure
- Preview results in real-time
- Export in desired formats

6.3 Display real-time feedback through loading indicators, error alerts, and completion confirmations to guide the user experience.

Output

Fig.5.2

5.3 Dialogue Summarization Using LoRA-Fine-Tuned Phi-2 Model

This module implements a dialogue summarization system powered by a fine-tuned large language model. It leverages a pre-trained foundation model (Phi-2 by Microsoft) enhanced with parameter-efficient fine-tuning via LoRA (Low-Rank Adaptation). The summarizer is designed to generate concise and context-aware summaries of human conversations, making it highly applicable to domains like customer service transcripts, academic interviews, or meeting recordings.

The user can select a specific dialogue from a pre-loaded test dataset, view the original conversation, and then generate a model-produced summary. The system provides both the ground truth (reference) summary and the one generated by the model, allowing real-time comparison of model performance.

Objective of the Module

To generate accurate and concise summaries from conversational text using a LoRA-finetuned large language model (LLM). The system demonstrates the effectiveness of adapterbased fine-tuning on small and efficient models like Phi-2.

Input: A dialogue transcript selected from a structured dataset
Output: A summary generated by the fine-tuned Phi-2 model

Algorithm: Dialogue Summarizer with LoRA-Fine-Tuned Phi-2

Step 1: Dataset Loading

1.1 Load a benchmark dialogue summarization dataset from HuggingFace (e.g., DialogSum).

1.2 Cache the dataset locally to improve performance and avoid redundant loading.

1.3 Select the "test" subset of the dataset for evaluation purposes, ensuring the integrity of training and testing separation.

Step 2: Model Initialization

2.1 Define the base pre-trained language model to be used (Phi-2 by Microsoft), which is optimized for small-device inference and efficiency.

2.2 Load the tokenizer corresponding to the Phi-2 model and ensure the padding token is set correctly to prevent misalignment during generation.

2.3 Initialize the Phi-2 model in inference mode, loading its weights to CPU memory.

2.4 Apply LoRA adapters to the model by loading low-rank fine-tuning weights from a local directory.

2.5 Set both base and adapted models to evaluation mode to disable gradient tracking and reduce resource usage.

2.6 Cache the model and tokenizer in memory to avoid repeated initialization.

Step 3: User Interaction and Input Selection

3.1 Create a graphical interface using Streamlit to enable interaction.

3.2 Display a user-friendly slider allowing the user to select a dialogue from the dataset by index.

3.3 Extract and display the full dialogue text from the selected entry.

3.4 Retrieve and display the reference summary associated with the same dialogue as the ground truth.

Step 4: Prompt Construction and Preprocessing

4.1 Upon user action (e.g., button click), construct a prompt that instructs the model to generate a summary.

4.2 Format the input prompt with clear delineation using markers such as "Instruct:" and "Output:" to guide the model.

4.3 Tokenize the prompt using the Phi-2 tokenizer with truncation and padding enabled to fit model input constraints.

4.4 Ensure that tokenized inputs include both input IDs and attention masks for proper model processing.

Step 5: Inference and Text Generation

5.1 Run the input through the LoRA-enhanced Phi-2 model using inference mode without gradient tracking.

5.2 Configure generation parameters:

- Set the maximum number of new tokens to limit output length
- Use the appropriate padding token to maintain clean output
- Use greedy or deterministic decoding for consistent results

5.3 Generate the model's output summary as a continuation of the prompt.

5.4 Decode the generated token sequence into a human-readable string and strip extraneous formatting or prompt echoes.

5.5 Extract the final summary segment by parsing the "Output:" section from the decoded text.

Step 6: Output Display and Evaluation

6.1 Display the model-generated summary in the interface using a visually distinct component (e.g., a success box or highlighted area).

6.2 Allow users to compare the reference (ground truth) summary with the generated summary for evaluation.

6.3 Optionally, allow export or copying of the result for further use in downstream tasks like training, feedback collection, or benchmarking.
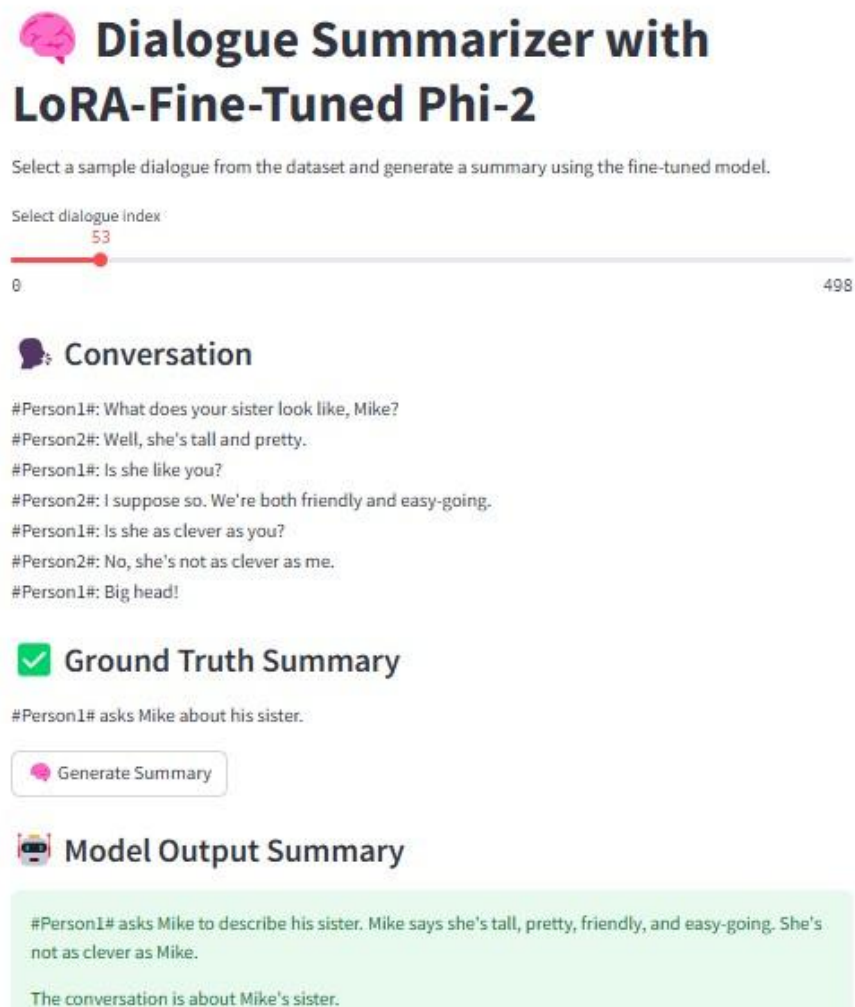
System Design Considerations

- **Adapter-Based Fine-Tuning**: LoRA reduces the need to fine-tune full model weights by introducing low-rank matrices. This allows for quick model updates without large compute or memory demands.
- **Model Efficiency**: Phi-2 is chosen for its balance between model capacity and resource usage, making it suitable for real-time applications on limited hardware.
- **Reproducibility**: A fixed random seed is used to ensure consistency in model output across multiple runs.
- **Performance**: Caching mechanisms for both data and model initialization improve user experience and reduce startup time.

- **Interactivity**: The Streamlit UI provides real-time control and transparent feedback, making the system accessible even to non-technical users.
- **Evaluation**: Presenting ground truth summaries alongside model predictions enables subjective and objective evaluation of summarization quality.

This module demonstrates the practical use of local language models with adapter-based finetuning for summarization tasks. It illustrates the end-to-end flow from input selection to summary generation and visual presentation, validating the effectiveness of parameterefficient methods like LoRA for targeted downstream tasks. The system is lightweight, portable, and highly adaptable for integration into broader NLP pipelines or educational platforms.

Output

Fig.5.3

CHAPTER -6

CONCLUSION

6.1 Key Takeaways and Skills Learnt

During the course of this project, I acquired a comprehensive understanding of how artificial intelligence, machine learning, and modern web technologies can be integrated to develop intelligent systems that are useful, scalable, and user-friendly. The key learning outcomes span across various domains of software development, including **natural language processing (NLP)**, **computer vision**, **user interface design**, and **system architecture**.

One of the primary takeaways was understanding how to extract, chunk, and vectorize text from unstructured formats such as PDFs and query this text using Large Language Models (LLMs) like **Google Gemini Pro**, **Mixtral-8x7B**, **Zephyr-7B**, and **Baichuan-13B**. This required exploring how embedding models like **Google Gecko (models/embedding-001)** and **MiniLM** work in vector space and how to use FAISS for similarity search and retrieval.

The **implementation of Parameter-Efficient Fine-Tuning (PEFT)** using **LoRA adapters** was a significant technical skill I developed. I explored loading PEFT adapters on top of base models like **Microsoft's Phi-2**, performing inference on dialogue summarization tasks with high accuracy and reproducibility.

On the frontend side, I enhanced my skills in building interactive and user-focused web applications using **HTML**, **CSS**, and **JavaScript**. I created a **video visualization system** where bounding boxes and confidence scores are rendered in real-time from preprocessed JSON files, which simulated real-world detection overlays.

I also learned how to design proper **system design diagrams**, such as **Data Flow Diagrams (DFD)**, **Entity Relationship Diagrams (ERD)**, and **Modular architecture breakdowns**, and implement these concepts in tools like **draw.io**, which helped visualize the system's functionality and data flow for both academic and practical clarity.

Lastly, I explored model evaluation, experimentation with multiple models, backend inference pipelines using Hugging Face's transformers, and frontend deployment using **Streamlit**, which deepened my understanding of end-to-end ML systems.

6.2 Key Contributions

The following were the main contributions I made during the course of this project:

- **IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics System**: A complete AI-powered solution to query PDF documents using natural language. It included modules for text extraction, chunking, embedding, FAISS vector store creation, and response generation using models like Gemini 1.5 Pro and Mixtral.

- **Question Generator from PDF**: A dedicated module that reads educational documents and generates **well-structured questions** using models such as **valhalla/t5-base-qghl**. It also offers **CSV/Excel export**, simulating an automated exam paper creation system for educators.
- **Dialogue Summarization using Phi-2 + LoRA Adapter**: A fine-tuned LLM system for generating concise summaries from conversational data. It used the Hugging Face datasets library for loading test data and peft for adapter integration.

6.3 Limitations

Module 1: IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics

- Performance depends heavily on system RAM and embedding model size.
- Inconsistent results for non-English or poorly structured PDFs.
- Requires internet for Gemini API; fallback models may reduce accuracy.

Module 2: Question Extraction

- Accuracy drops for scanned PDFs or documents with mixed formatting.
- No manual correction or classification before Excel export is currently supported.
- Limited to English and grammatically structured content.

Module 3: Dialogue Summarization

- Model may not perform well with multilingual or domain-specific jargon.
- Requires moderate hardware even with LoRA optimization.
- No feedback loop for summary refinement within the UI.

6.4 Future Scope

Module 1: IntelliPDF: AI-Powered Document Interaction and Real-Time Vision Analytics

- Integrate OCR to support image-based or scanned PDFs.
- Expand language support to include multilingual querying.
- Introduce feedback-based learning to improve answer relevance over time.

Module 2: Question Extraction

- Add support for scanned images using Tesseract OCR.
- Allow user tagging, categorization, and manual correction of extracted questions.
- Introduce filters for question type and difficulty levels.

Module 3: Dialogue Summarization

- Extend LoRA adapters to handle multilingual and domain-specific data.

- Integrate into customer support or CRM systems for automatic log summarization.
- Add interactive summary refinement via feedback prompts.

6.5 My Testimonials About the Internship

My internship experience was one of the most enriching phases of my academic journey. It provided me with the opportunity to apply theoretical concepts in a practical setting, learn about real-world problems, and work on solutions that go beyond basic implementation. The internship allowed me to independently explore and innovate with state-of-the-art AI technologies, especially in NLP and computer vision.

What stood out most was the freedom I was given to explore a wide variety of models, compare their outputs, and integrate them meaningfully across multiple project modules. This helped me gain clarity on model selection, API usage, and the impact of prompt engineering on model accuracy. Working with Hugging Face's ecosystem and Gemini's tools enhanced my comfort with industry-standard libraries.

Beyond technical skills, I also gained hands-on experience with project documentation, report formatting (especially in LaTeX and Overleaf), and the importance of structured communication when building user-oriented systems. I learned how to break down a problem into **design**, **development**, **testing**, and **deployment** stages—an essential mindset for professional software engineering.

This internship also helped me build a portfolio of tangible projects that I can showcase in future roles. It made me more confident in contributing to larger AI and data science teams in both research and industry. I am deeply grateful to my guide and mentors for their consistent support and constructive feedback throughout the journey.

CHAPTER-7
REFERENCES

[1] P.E. Agre and D. Chapman, *Unpublished memo*, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1986.

[2] R.J. Bobrow and J.S. Brown, "Systematic understanding: synthesis, analysis, and contingent knowledge in specialized understanding systems," in *Representation and Understanding*, R.J. Bobrow and A.M. Collins, Eds. New York: Academic Press, 1975, pp. 103–129.

[3] R.A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.

[4] R.A. Brooks, "A hardware retargetable distributed layered architecture for mobile robot control," in *Proc. IEEE Int. Conf. Robotics and Automation*, Raleigh, NC, 1987, pp. 106–110.

[5] R.A. Brooks, "Intelligence without representation," *Artificial Intelligence*, vol. 47, no. 1–3, pp. 139–159, 1991.

[6] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[7] L. Steels, "Cooperation between distributed agents through self-organization," in *Decentralized AI*, Y. Demazeau and J.P. Müller, Eds. Amsterdam: North-Holland, 1990, pp. 175–196.

[8] R.C. Arkin, *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1998.

[9] M.J. Mataric, "Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior," *Trends in Cognitive Sciences*, vol. 2, no. 3, pp. 82–87, 1998.

[10] L.P. Kaelbling, M.L. Littman, and A.W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[11] Google AI, *Google Generative AI – Gemini API*, accessed March 2025. [Online]. Available: https://ai.google.dev/

[12] Hugging Face, *HuggingFace Model Hub – Transformers Library*, accessed February 2025. [Online]. Available: https://huggingface.co/docs/transformers

[13] Unsloth Team, *Unsloth: Efficient LoRA and PEFT for LLMs*, accessed February 2025. [Online]. Available: https://unsloth.ai

[14] PyPDF2 Library, *PDF Extraction in Python*, accessed January 2025. [Online]. Available: https://pypi.org/project/PyPDF2/

[15] Facebook AI Research, *FAISS: A Library for Efficient Similarity Search*, accessed January 2025. [Online]. Available: https://github.com/facebookresearch/faiss

[16] OpenCV, *Open Source Computer Vision Library*, accessed January 2025. [Online]. Available: https://opencv.org/

[17] Ultralytics, *YOLOv8: Real-Time Object Detection Models*, accessed February 2025. [Online]. Available: https://docs.ultralytics.com/

[18] Roboflow, *Labeling and Training YOLOv8 Models*, accessed January 2025. [Online]. Available: https://roboflow.com/

[19] LangChain Documentation. *LangChain: Building Applications with LLMs*. Accessed March 2025. https://docs.langchain.com

[20] Microsoft. *Phi-2 Model*. Hugging Face, Accessed March 2025. https://huggingface.co/microsoft/phi-2