## 5. Technical Architecture

This section describes the end-to-end architecture of **ReadStudyAssist AI**, including system components, data flow, layer separation, and integration strategy. The architecture is designed to support three core capabilities: **Text Summarisation**, **Document Q&A**, and **Content Generation for quizzes**.

### 5.1 Architecture Overview

ReadStudyAssist AI uses a modular four-layer architecture:

1. **User Interface Layer** – Web application where learners upload documents, ask questions, and review generated summaries/quizzes.
2. **Service Layer** – Backend API responsible for requesting routing, document-processing pipelines, and communication with AI components.
3. **AI/LLM Layer** – Handles summarization, Q&A and quiz generation through prompt-based interactions and function calling.
4. **Data Layer** – Responsible for storing raw files, processed text chunks and metadata.

### 5.2 System Architecture Diagram (Description)

A clean diagram should show the following elements:

**User → UI → Backend Service → LLM Provider → Backend → Storage**

- **Frontend (e.g., Streamlit/React)**
  - Document upload
  - Query interface
  - Display views for summaries, extracted text, and quizzes
- **Backend (FastAPI/Flask)**
  - Document ingestion module
  - Text cleaning and chunking
  - Embedding & vector store operations (if RAG implemented)
  - LLM orchestration module
  - Quiz generation module

    - Logging/Observability (e.g., Langfuse)
- **LLM Provider (Gemini)**
    - Summarisation API
    - Q&A API
    - Content generation API
- **Database/Storage**
    - Document storage (local or cloud)
    - Vector DB (Chroma/FAISS) for semantic search (if used)

### 5.3 Layer Structure

#### (1) User Interface Layer

**Responsibilities:**

- Collect user inputs (upload PDFs, ask questions)
- Present summarised content and generated quizzes
- Communicate with backend through REST endpoints

#### (2) Service Layer

This layer orchestrates all business logic.

**Modules:**

##### *a. Document Ingestion Module*

- Accepts PDF/Docx uploads
- Converts files into plain text
- Splits into semantic chunks for processing and retrieval

##### *b. AI Request Orchestrator*

- Builds structured prompts for summarisation, Q&A, and quiz generation
- Handles function calling (e.g., to trigger quiz generation)
- Manages retries/errors

##### *c. Output Formatting Module*

- Formats summaries, Q&A responses, and quizzes into UI-friendly formats

#### (3) AI / LLM Layer

The system relies on a single LLM provider - Gemini, accessed through:

- **Summarisation prompt templates** (for full or chapter-level summaries)
- **Context-aware Q&A prompts** (LLM + retrieved chunks)
- **Quiz generation prompts** (MCQs, fill-in-the-blank, short answer)

### (4) Data Layer

Includes:

#### *a. Document Storage*

- Stores raw PDFs and extracted text
- Metadata: upload timestamps, source type, processed embeddings

#### *b. Logs and Observability*

- Captures LLM call metadata
- Stores user interactions for debugging, evaluation, and usage patterns

### 5.4 Data Flow Diagram (Description)

Your diagram should display the following flow:

1. **User Uploads a Document**
   ↓
2. **Backend Extracts Text & Splits into Chunks**
   ↓
3. **User Requests a Summary / Q&A / Quiz**
   ↓
4. **Backend Retrieves Relevant Chunks** (if Q&A)
   ↓
5. **Backend Sends Structured Prompt to LLM**
   ↓
6. **LLM Generates Response**
   ↓
7. **Backend Formats Output**
   ↓
8. **UI Displays Results to User**

This flow ensures each component is clearly represented in your final architecture diagram.

### 5.5 Component Explanations

#### Frontend - Streamlit

- Handles user interactions with minimal logic
- Designed this way to ensure responsiveness and low coupling

#### Backend API

- Central hub for all operations
- Ensures consistent logic across summarisation, Q&A and quizzes

#### LLM Provider - Gemini

- Abstracts summarization, question-answering and content generation
- Chosen due to high-quality responses and low implementation complexity

#### Storage

- Stores raw and transformed data for reuse
- Prevents redundant computation