AI Policy Explainer & Study Assistant

Creating Intelligent Document Understanding with Groq & LangChain

Project Overview

The AI Policy Explainer & Study Assistant is a comprehensive web application powered by Groq's high-performance LLM models that provides instant document analysis, summarization, and interactive learning features. It allows professionals and students to upload policy documents or study materials and get clear explanations, structured summaries, Q&A support, and assessment tools.

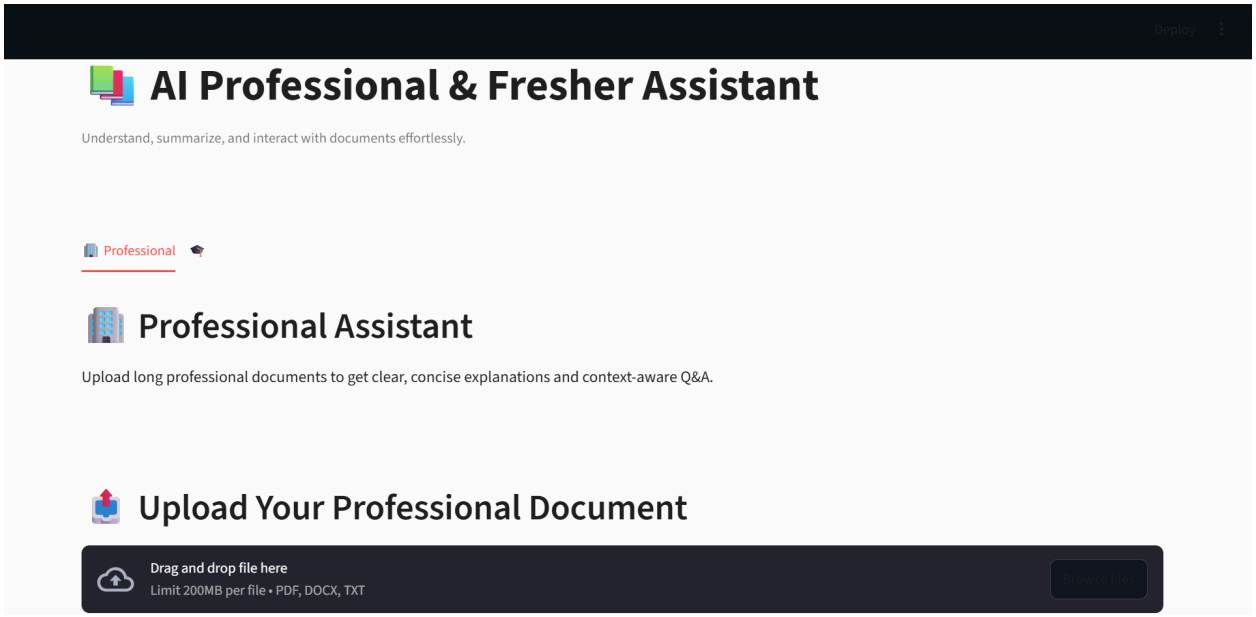


Figure 1 - Main application interface showing both Professional and Fresher tabs with document upload functionality

This system bridges the gap between complex documentation and user understanding by leveraging advanced retrieval-augmented generation (RAG) to provide accurate, context-aware responses based on the uploaded content.

Scenario 1: Professional Policy Analysis

Corporate professionals, legal experts, and compliance officers can use the AI Policy Explainer to quickly understand complex policy documents, regulations, and procedural manuals without spending hours reading through dense text.

By uploading policy documents in PDF, DOCX, or TXT formats, users enable the system to analyze the content, extract key provisions, and generate easy-to-understand summaries. The Q&A feature allows professionals to ask specific questions about policy implications, exceptions, or implementation guidelines.

For instance, a compliance manager can upload a new data protection policy and instantly get a structured summary highlighting key requirements, compliance deadlines, and enforcement mechanisms. They can then ask specific questions like "What are the reporting requirements for data breaches?" and receive accurate, context-based answers.

This helps organizations save time, ensure regulatory compliance, and maintain consistent policy understanding across teams.

Scenario 2: Academic Study Assistance

Students and educators can use this system as an intelligent study companion for processing textbooks, research papers, and course materials. The application generates concise summaries, creates revision-friendly quick notes, and generates assessment quizzes.

By analyzing study materials, the AI Study Assistant can extract key concepts, generate flashcard-style notes for quick revision, and create multiple-choice quizzes to test understanding. This supports different learning styles and helps students focus on core concepts.

For example, a student uploading a chapter on machine learning can get a simplified summary, generate quick notes with key definitions and formulas, and then take an interactive quiz to assess their understanding before exams.

This use case demonstrates how the system can help students learn more efficiently, reduce study time, and improve academic performance through personalized, AI-driven educational support.

Architecture

The AI Policy Explainer & Study Assistant uses a combined architecture that integrates Groq's high-speed LLM inference, LangChain for document processing, FAISS for vector search, and Streamlit for an interactive web interface.

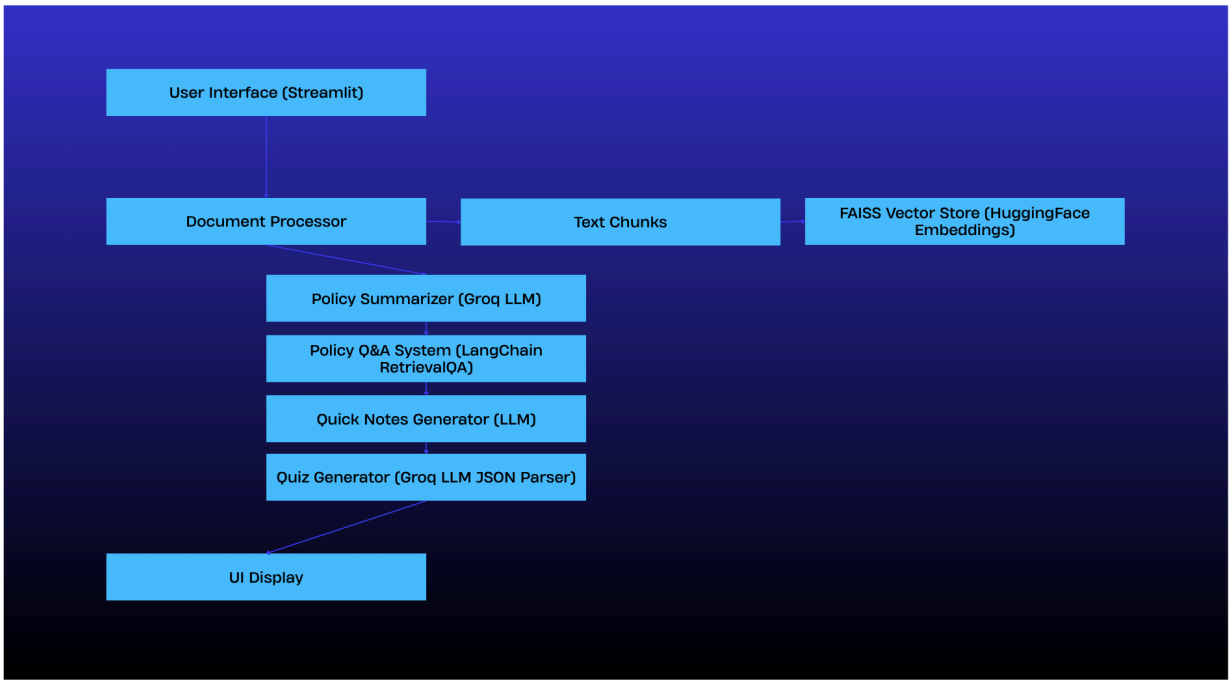


Figure 2 - System architecture diagram showing document processing pipeline and component interactions

Core Components

a. DocumentProcessor

- **Functionality:** Handles the initial data ingestion, loading and extracting text from PDFs, DOCX, and TXT files. It cleans the text and uses **LangChain's** `RecursiveCharacterTextSplitter` to divide the material into appropriately sized, overlapping chunks.

b. VectorStoreManager

- **Functionality:** Responsible for creating the searchable semantic index. It converts the document chunks into embeddings using **HuggingFace's MiniLM model** (specifically `sentence-transformers/all-MiniLM-L6-v2`) and persists them in the **FAISS vector database** for efficient, semantic search.

c. PolicySummarizer

- **Functionality:** Utilizes **Groq's Llama 3.1** via a LangChain integration to produce high-quality summaries. The prompts are engineered to ensure the output is structured, easy-to-understand, and includes key bullet points or highlights.

d. PolicyQASystem

- **Functionality:** The core of the interactive chat feature. It employs **LangChain's** `RetrievalQA` **chain**. When a query is received, it first queries the **FAISS store** for the most semantically relevant text chunks, which are then passed to the LLM to generate a precise, contextually grounded answer.

e. StudyFlashcardGenerator (Quick Notes)

- **Functionality:** Designed for rapid knowledge transfer, this module distills the summarized content into a highly focused list of **8–12 concise bullet points**.
- **Format:** The standardized format is `- **Topic:** Short explanation`, optimizing the content for rapid revision and concept recall.

f. QuizGenerator

- **Functionality:** Automatically generates a set of **12 multiple-choice questions** based on the content summary.
- **Output & Integration:** The output is strictly controlled to a **structured JSON format** containing the question, four options, and the correct answer key. This structure enables a seamless integration with the quiz UI, which provides radio buttons, instant scoring, and performance feedback.

Project Flow

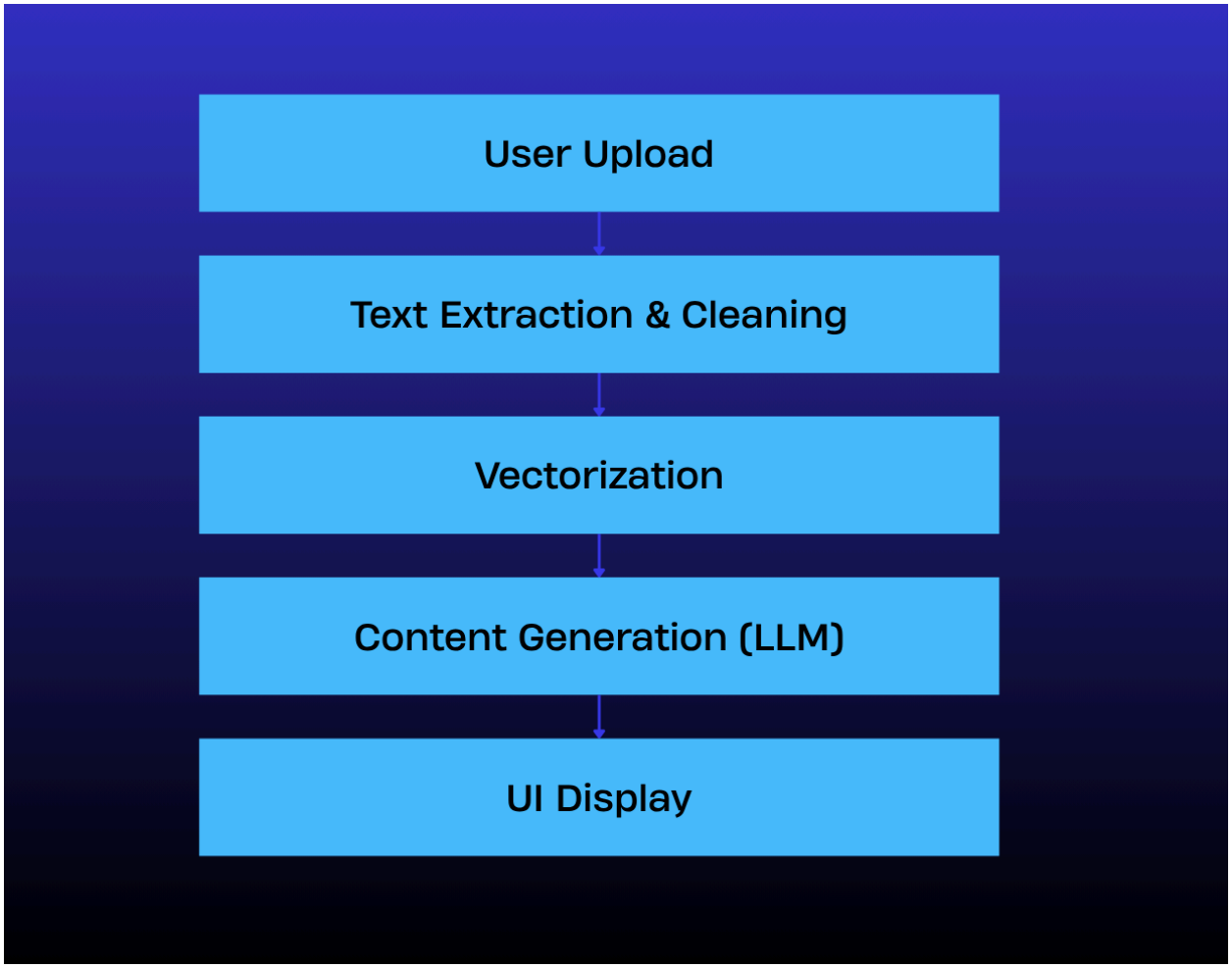


Figure 3 - Project workflow

- **User Upload:** The process begins with the user uploading a document file (**PDF, DOCX, or TXT**) through the application's interface.
- **Text Extraction & Cleaning:** The system's **Document Processor** extracts the raw textual content from the file and performs initial cleaning to remove noise and artifacts.
- **Vectorization:** The cleaned text is chunked, converted into high-dimensional vectors (embeddings) using a **HuggingFace model**, and stored in the **FAISS Vector Store** for semantic search.
- **Content Generation (LLM):** The **Groq LLM** uses the vectorized data to generate all key outputs, including the comprehensive **Summary**, contextual **Q&A responses**, **Quick Notes**, and the structured **MCQ Quiz**.
- **UI Display:** The **Streamlit UI** renders all generated content in the appropriate tabs and interactive modules for the user to consume and engage with.

To Accomplish This Project, You Must Complete the Following Activities:

- Install and configure all required libraries and APIs
- Initialize the Groq LLM model and embedding systems
- Develop document processing and vector storage functions
- Build the multi-tab Streamlit user interface with professional and study modes
- Implement Q&A, quick notes, and quiz generation features
- Deploy the complete application using Streamlit

Prior Knowledge

You should have an understanding of:

1. Large Language Models (LLMs) and Groq API

Groq provides powerful generative AI models such as LLaMA 3.1 for language understanding and reasoning.
Knowledge of configuring Groq API keys, setting temperature parameters, and sending structured prompts is beneficial.

Official Documentation:
https://console.groq.com

2. Streamlit

Streamlit is a Python framework for building interactive web applications.
Understanding its widgets, layout management, and session states is essential for frontend development.

Official Tutorials:
https://docs.streamlit.io/develop/tutorials

3. LangChain

LangChain allows chaining multiple AI components such as summarization, retrieval, and question answering.
Understanding how to work with vector stores, document loaders, and LLM chains will help extend this app.

Official Documentation:
https://python.langchain.com/docs/introduction/

4. Environment Management

The **python-dotenv** library is used for securely managing environment variables like API keys, ensuring sensitive information is not exposed.

Official Documentation:
https://pypi.org/project/python-dotenv/

5. Prompt Engineering

Prompt engineering involves designing input prompts to guide AI models toward accurate, relevant, and well-structured responses.

Learning Resources:
https://platform.openai.com/docs/guides/prompt-engineering?lang=python

6. Python Programming

A good grasp of Python programming concepts is essential, including file handling, class-based architecture, and data serialization.

Official Documentation:
https://docs.python.org/3/

Project Structure

This project is developed using Python 3.11+ and requires the following structure:

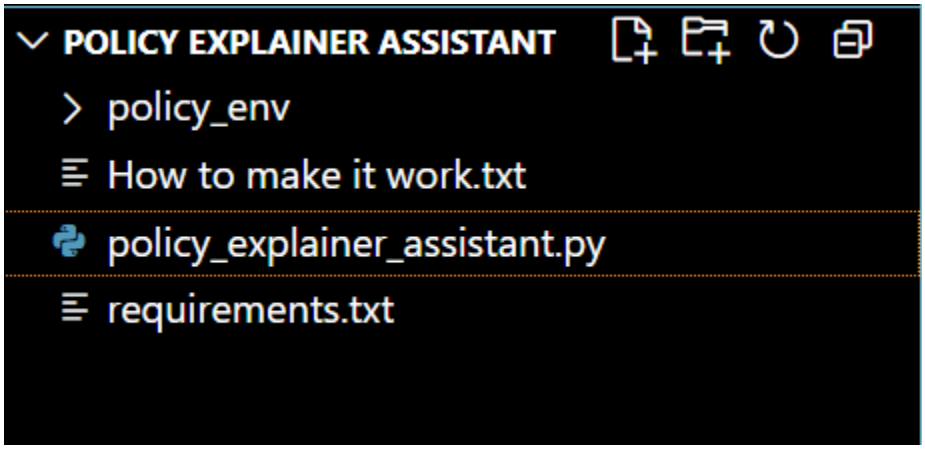


Figure 4 - Project file structure in VS Code Explorer

- **Policy_env:** It is the virtual environment which we will create to run the main streamlit application using the 'How to make it work.txt'
- **How to make it work.txt:** Contains small crisp, short points on how to create the virtual environment, activate it, install dependencies and run the main application
- **Requirements.txt:** Contains the required dependencies to make the main application work

Milestone 1: Requirements Specification

Milestone: Libraries Installation and Setup

This milestone ensures all necessary libraries are installed and configured to run the AI Policy Explainer & Study Assistant app. It prepares the environment for secure API access, interactive web interfaces, and efficient content handling.

Using requirements.txt for Library Installation

This approach ensures reproducibility and easy setup for all users by installing all required libraries at once from a requirements.txt file.

Steps

1. **Ensure requirements.txt Exists**

The project folder contains a comprehensive requirements.txt file with the following content:

```
# === Core Frameworks ===
streamlit>=1.38.0
python-dotenv>=1.0.1

# === LangChain Ecosystem ===
langchain>=0.2.14
langchain-community>=0.2.12
langchain-groq>=0.1.3

# === Vector Store & Embeddings ===
faiss-cpu>=1.8.0
sentence-transformers>=2.7.0
huggingface-hub>=0.23.4

# === Document Processing ===
pypdf>=4.2.0
PyPDF2>=3.0.1
docx2txt>=0.8
tiktoken>=0.7.0

# === Text & NLP Utilities ===
nltk>=3.8.1
regex>=2024.5.15

# === Data Handling ===
pandas>=2.2.2
numpy>=1.26.4

# === Development & Logging ===
requests>=2.32.3
tqdm>=4.66.3
openai>=1.42.0

# === Optional UI/UX Enhancements ===
rich>=13.7.1
```

Figure 5 - requirements.txt file content showing all dependencies

2. **Open the project folder directory**

Use the following command for windows: cd “your_project_directory”, like

```
C:\Users\rithi>cd "C:\Users\rithi\Downloads\Policy explainer assistant"
```

3. **Create a virtual environment**

Create a virtual environment using this command, python -m venv policy_env, as shown in the picture below:

```
C:\Users\rithi\Downloads\Policy explainer assistant>python -m venv policy_env
```

4. **Activate the environment**

Activate the created virtual environment by using this command, policy_env\Scripts\activate, like shown in the picture below:

```
C:\Users\rithi\Downloads\Policy explainer assistant>policy_env\Scripts\activate
```

5. **Install All Libraries**

Run the following command in your project directory to install all dependencies at once by using this command, pip install -r requirements.txt, like shown in the picture below:

```
(policy_env) C:\Users\rithi\Downloads\Policy explainer assistant>pip install -r requirements.txt
```

Make sure (policy_env) appears before your directory for this command

Explanation of Each Library

- **Core Frameworks**
  - **streamlit>=1.38.0 -** Web application framework that creates interactive UIs for data science and AI projects without frontend coding
  - **python-dotenv>=1.0.1 -** Manages environment variables and API keys securely by loading them from the python script
- **LangChain Ecosystem**
  - **langchain>=0.2.14 -** Main framework for building LLM-powered applications with chains, agents, and memory
  - **langchain-community>=0.2.12 -** Community-contributed integrations, tools, and components for LangChain
  - **langchain-groq>=0.1.3 -** Official integration between LangChain and Groq's high-speed LLM API
- **Vector Store & Embeddings**
  - **faiss-cpu>=1.8.0 -** Facebook's vector similarity search library for efficient document retrieval and matching
  - **sentence-transformers>=2.7.0 -** Generates sentence embeddings for semantic search and text similarity
  - **huggingface-hub>=0.23.4 -** Interface to Hugging Face's model hub for downloading and managing AI models
- **Document Processing**
  - **pypdf>=4.2.0 -** Modern PDF processing library for extracting text and metadata from PDF files

    - **PyPDF2>=3.0.1 -** Alternative PDF library for reading, splitting, and merging PDF documents
    - **docx2txt>=0.8 -** Extracts text content from Microsoft Word (.docx) documents
    - **tiktoken>=0.7.0 -** OpenAI's tokenizer for counting tokens in text, useful for managing context windows
- **Text & NLP Utilities**
    - **nltk>=3.8.1 -** Natural Language Toolkit for text processing, tokenization, and linguistic analysis
    - **regex>=2024.5.15 -** Advanced regular expressions for complex text pattern matching and manipulation
- **Data Handling**
    - **pandas>=2.2.2 -** Data manipulation and analysis library for handling structured data
    - **numpy>=1.26.4 -** Fundamental package for numerical computing and array operations
- **Development & Logging**
    - **requests>=2.32.3 -** HTTP library for making API calls to external services
    - **tqdm>=4.66.3 -** Progress bar utility for tracking long-running operations
    - **openai>=1.42.0 -** OpenAI API client library (included for compatibility with some LangChain components)
- **Optional UI/UX Enhancements**
    - **rich>=13.7.1 -** Library for rich text and beautiful formatting in terminal applications

Milestone 2: Initializing the Models

The Groq LLM model and embedding systems are configured and initialized in the main application file.

Activity 1: Configuration Setup

This activity implements a centralized configuration system that manages all API keys and application settings.

Explanation

- **Direct API configuration:** API keys are configured directly in the Config class
- **Centralized settings:** All model parameters and application settings managed in one place
- **Parameter optimization:** Pre-configured chunk sizes, overlap, and search parameters for optimal performance

```python
# -------------------- CONFIGURATION --------------------
class Config:
    """
    Central configuration class with your exact settings
    """

    def __init__(self):
        # Your exact API key and settings
        self.GROQ_API_KEY = "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"
        self.MODEL_NAME = "llama-3.1-8b-instant"
        self.CHUNK_SIZE = 1000
        self.CHUNK_OVERLAP = 200
        self.SIMILARITY_SEARCH_K = 4
        self.MAX_CHAT_HISTORY = 20
        self.TEMPERATURE = 0.1
        self.APP_MODE = "policy"
```

Figure 6 - Configuration class implementation in the main application

Activity 2: Configuring and Initializing the Model

This activity initializes the Groq LLM model and embedding system for document processing and content generation.

Explanation

- **Initializes Groq LLM:** Configures the high-speed Llama-3.1-8b-instant model
- **Sets up embeddings:** Initializes HuggingFace sentence transformers for document vectorization
- **Configures parameters:** Sets appropriate temperature, chunk sizes, and search parameters

```python
# -------------------- CONFIGURATION --------------------
class Config:
    """
    Central configuration class with your exact settings
    """

    def __init__(self):
        # Your exact API key and settings
        self.GROQ_API_KEY = "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"
        self.MODEL_NAME = "llama-3.1-8b-instant"
        self.CHUNK_SIZE = 1000
        self.CHUNK_OVERLAP = 200
        self.SIMILARITY_SEARCH_K = 4
        self.MAX_CHAT_HISTORY = 20
        self.TEMPERATURE = 0.1
        self.APP_MODE = "policy"

        # Vector store settings
        self.VECTOR_STORE_PATH = "./vector_cache"

        # App settings
        self.APP_NAME = "Policy Explainer & Study Assistant"

    def validate_config(self):
        """Validate that all required configurations are set"""
        if not self.GROQ_API_KEY:
            raise ValueError("GROQ_API_KEY not found")
        return True


config = Config()
```

Figure 7 - Model initialization code showing Groq and embedding setup

Milestone 3: Building the Core Functions

This milestone covers the key functional components including document processing, summarization, Q&A system, and study features.

Activity 1: Creating Document Processing System

This activity builds a comprehensive document processing pipeline that handles multiple file formats, text cleaning, and intelligent chunking.

Explanation

- **Multi-format support:** Processes PDF, DOCX, and TXT files using appropriate loaders
- **Text cleaning:** Removes redundant spaces, page numbers, and formatting artifacts
- **Intelligent chunking:** Uses recursive text splitting with overlap to preserve context
- **Metadata handling:** Adds filename and processing information to document chunks

```python
# -------------------- DOCUMENT PROCESSOR --------------------
class DocumentProcessor:
    """
    Handles document loading, text cleaning, and chunking for retrieval.
    Works for both Policy and Student document modes.
    """

    def __init__(self):
        self.chunk_size = config.CHUNK_SIZE
        self.chunk_overlap = config.CHUNK_OVERLAP

    def load_document(self, file_path: str) -> str:
        """Loads a document (PDF, DOCX, TXT) and returns cleaned text."""
        ext = os.path.splitext(file_path)[1].lower()

        try:
            if ext == ".pdf":
                loader = PyPDFLoader(file_path)
                docs = loader.load()
                text = "\n".join([d.page_content for d in docs])
            elif ext == ".docx":
                text = docx2txt.process(file_path)
            elif ext == ".txt":
                with open(file_path, "r", encoding="utf-8") as file:
                    text = file.read()
            else:
                raise ValueError(f"Unsupported file type: {ext}")

            return self._clean_text(text)

        except Exception as e:
            raise Exception(f"Error loading document {file_path}: {str(e)}")
```

```python
        except Exception as e:
            raise Exception(f"Error loading document {file_path}: {str(e)}")

    def _clean_text(self, text: str) -> str:
        """Cleans text by removing redundant spaces, page numbers, and formatting issues."""
        import re

        # Remove multiple newlines, page numbers, and unnecessary spacing
        text = re.sub(r"\n{2,}", "\n", text)
        text = re.sub(r"Page\s*\d+\s*(of\s*\d+)?", "", text, flags=re.IGNORECASE)
        text = re.sub(r"\s{2,}", " ", text)
        text = text.strip()

        return text

    def chunk_document(self, text: str, metadata: dict = None):
        """
        Splits long text into manageable chunks for embedding and retrieval.
        Returns list of LangChain Document objects.
        """
        if not text.strip():
            raise ValueError("Empty document text. Cannot create chunks.")

        text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=self.chunk_size,
            chunk_overlap=self.chunk_overlap,
            separators=["\n\n", "\n", ".", " "],
        )

        chunks = text_splitter.split_text(text)
```

```python
        chunks = text_splitter.split_text(text)
        documents = [
            Document(page_content=chunk, metadata=metadata or {}) for chunk in chunks
        ]

        return documents
```

Figure 8 - Document processing class showing multi-format support

Activity 2: Creating Vector Store Management

This activity implements the vector storage and retrieval system using FAISS and HuggingFace embeddings.

Explanation

- **Vector embedding:** Converts text chunks into numerical vectors using sentence transformers

- **FAISS integration:** Creates efficient vector store for similarity search
- **Retrieval setup:** Configures retriever with appropriate search parameters

```python
# -------------------- VECTOR STORE MANAGER --------------------
class VectorStoreManager:
    """
    Manages the creation and handling of vector stores using FAISS and HuggingFace embeddings.
    """

    def __init__(self):
        self.embedding_model_name = "sentence-transformers/all-MiniLM-L6-v2"

        try:
            self.embeddings = HuggingFaceEmbeddings(
                model_name=self.embedding_model_name
            )
            # Removed print statement to avoid console output in Streamlit
        except Exception as e:
            raise RuntimeError(f"Error initializing embedding model: {str(e)}")

    def create_vector_store(self, documents):
        """
        Converts processed document chunks into FAISS vector embeddings.
        """
        if not documents or len(documents) == 0:
            raise ValueError("Cannot create vector store from an empty document list.")

        try:
            vector_store = FAISS.from_documents(documents, self.embeddings)
            # Removed print statement - success will be shown in the UI
            return vector_store

        except Exception as e:
            raise RuntimeError(f"Error creating vector store: {str(e)}")
```

Figure 9 - Vector Store Management

Activity 3: Building Summarization System

This activity creates an intelligent summarization system that generates structured, easy-to-understand summaries of complex documents.

Explanation

- **Structured prompts:** Uses carefully designed prompts to extract key information
- **Multi-aspect analysis:** Identifies main purpose, key rules, important takeaways, and exceptions

- **Study-friendly format:** Adapts output for educational materials with bullet points and simplified explanations

```python
# -------------------- SUMMARIZER --------------------
class PolicySummarizer:
    """
    Handles summarization of policy or study documents.
    Uses Groq's Llama-3 model to generate clear, structured summaries.
    """

    def __init__(self):
        self.llm = ChatGroq(
            model=config.MODEL_NAME,
            groq_api_key=config.GROQ_API_KEY,
            temperature=config.TEMPERATURE,
        )

    def summarize_document(self, document_text: str) -> str:
        """
        Generates a simple and structured summary of the given document text.
        """
        if not document_text or not document_text.strip():
            return "⚠️ No document text available to summarize."

        # Limit input length to prevent model overload
        text = document_text[:12000]

        template = """
        You are an AI assistant that simplifies complex text into clear, concise summaries.

        Summarize the following document in easy-to-understand language.
        Highlight the key ideas, important rules, and any critical details.

        Make sure your summary includes:
        - ✅ The main purpose or topic
```

```
        Make sure your summary includes:
        - ✅ The main purpose or topic
        - 📘 Key rules, points, or sections
        - 🧠 Important takeaways or principles
        - 📄 Any exceptions or special cases
        - 💡 Overall summary in simple language

        If this is a study or textbook material, also:
        - Provide concise notes or bullet points for each concept
        - Simplify explanations to help students understand easily

        Use bullet points and short paragraphs for readability.

        --- Document ---
        {document_text}
        """

        prompt = PromptTemplate(
            input_variables=["document_text"], template=template.strip()
        )
        chain = LLMChain(llm=self.llm, prompt=prompt)

        try:
            summary = chain.run(document_text=text)
            return summary.strip()
        except Exception as e:
            return f"⚠️ Error while summarizing: {str(e)}"
```

Figure 10 - Summarization system with structured prompt template

Activity 4: Implementing Q&A System

This activity builds a retrieval-augmented Q&A system that provides accurate, context-based answers to user questions.

Explanation

- **Retrieval-augmented generation:** Combines vector search with LLM for accurate answers
- **Context-aware responses:** Uses document chunks as context for relevant answers
- **User-friendly format:** Provides clear, concise, and beginner-friendly explanations

```python
# -------------------- Q&A SYSTEM --------------------
class PolicyQASystem:
    """
    Handles question answering using the Groq LLM + vector retrieval.
    Supports both policy document queries and study material clarifications.
    """

    def __init__(self, vector_store):
        if vector_store is None:
            raise ValueError("Vector store is not initialized.")

        self.llm = ChatGroq(
            model=config.MODEL_NAME,
            groq_api_key=config.GROQ_API_KEY,
            temperature=config.TEMPERATURE,
        )

        self.retriever = vector_store.as_retriever(
            search_kwargs={"k": config.SIMILARITY_SEARCH_K}
        )

        self.qa_chain = self._create_qa_chain()

    def _create_qa_chain(self):
        template = """
        You are a helpful assistant that answers questions based on provided context.
        Keep your answer clear, concise, and beginner-friendly.

        Context:
        {context}
```

```
        Question:
        {question}

        Provide your answer in a way that's:
        - Simple and direct
        - Step-by-step if needed
        - Avoids jargon
        - Helpful for understanding the main point
        """

        prompt = PromptTemplate(
            input_variables=["context", "question"],
            template=template.strip(),
        )

        qa_chain = RetrievalQA.from_chain_type(
            llm=self.llm,
            chain_type="stuff",
            retriever=self.retriever,
            chain_type_kwargs={"prompt": prompt},
            return_source_documents=True,
        )

        return qa_chain

    def ask_question(self, question: str):
        """Handles user query and returns LLM-generated answer."""
        if not question.strip():
            return {"answer": "Please ask a valid question.", "source_documents": []}
```

```
        try:
            result = self.qa_chain.invoke({"query": question})
            return {
                "answer": result["result"],
                "source_documents": result.get("source_documents", []),
            }
        except Exception as e:
            return {
                "answer": f"⚠️ Error while answering: {str(e)}",
                "source_documents": [],
            }
```

Figure 11 - Q&A system implementation with retrieval-augmented generation

Activity 5: Creating Study Features

This activity implements specialized study features including quick notes generation and interactive quiz creation.

Explanation

- **Quick notes generation:** Creates flashcard-style revision points from study materials
- **Quiz generation:** Automatically creates multiple-choice questions from content
- **Interactive assessment:** Provides immediate feedback and scoring for quizzes

```python
# -------------------- FLASHCARD GENERATOR --------------------
class StudyFlashcardGenerator:
    """
    Generates concise flashcard-style cues for revision based on a study summary.
    """

    def __init__(self):
        self.llm = ChatGroq(
            model=config.MODEL_NAME,
            api_key=config.GROQ_API_KEY,
            temperature=config.TEMPERATURE,
        )

        template = """
You are a concise flashcard generator.

From the given study summary, extract 8-12 key concepts or cues for quick revision.

Follow these rules strictly:
- Output each item as a separate Markdown bullet point on a new line.
- Each bullet must follow this format: - **Topic Name:** short explanation.
- Keep explanations brief (maximum 12-15 words).
- Do not include any introductory or closing sentences.
- Do not include numbering, section titles, or labels like "Flashcards" or "Concepts".
- Focus on clarity and usefulness for quick recall.

Example output:
- **Photosynthesis:** Process where plants convert sunlight into energy.
- **Newton's Laws:** Describe the relationship between motion and forces.
- **Supply and Demand:** Explain balance between market needs and product availability.
```

```
Now, create the cues based on this summary:

{summary_text}
"""
        self.prompt = PromptTemplate(
            input_variables=["summary_text"], template=template.strip()
        )
        self.chain = LLMChain(llm=self.llm, prompt=self.prompt)

    def generate_flashcards(self, summary_text: str) -> str:
        """
        Generate flashcards (markdown bullets) from the provided summary_text.
        """
        if not summary_text or not summary_text.strip():
            return "⚠️ No summary available to generate flashcards."

        limited_text = summary_text[:10000]

        try:
            output = self.chain.run(summary_text=limited_text)
            return output.strip()
        except Exception as e:
            return f"⚠️ Error generating flashcards: {str(e)}"

```

Figure 12 - Study features implementation for quick notes and quizzes

Milestone 4: Creating Streamlit Frontend

This contains the Streamlit-based user interface.

Activity 1: UI Initialization

This activity sets up the Streamlit interface for the AI Policy Explainer & Study Assistant app, defining its layout, title, and appearance.

Explanation

- **Configures page settings:** Sets the app's title, icon, and layout for a clean, wide-screen view
- **Displays main heading:** Shows the project title on the interface
- **Introduces the app:** Provides users with a clear entry point and context before interacting with features

```python
# -------------------- MAIN APP CLASS --------------------
class PolicyExplainerApp:
    """
    Streamlit-based GenAI assistant for:
    - Policy document explanation (Professional Mode)
    - Study material summarization, Q&A, and Quick Notes (Fresher Mode)
    """

    def __init__(self):
        self.setup_page()
        self.document_processor = DocumentProcessor()
        self.vector_manager = VectorStoreManager()
        self.initialize_session_state()

    # -------------------- PAGE SETUP --------------------
    def setup_page(self):
        st.set_page_config(
            page_title="AI Professional & Fresher Assistant",
            page_icon="📚",
            layout="wide",
        )
        st.markdown(CUSTOM_CSS, unsafe_allow_html=True)
        st.title("📚 AI Professional & Fresher Assistant")
        st.caption("Understand, summarize, and interact with documents effortlessly.")
        st.markdown("---")
```

Figure 13 - Streamlit UI initialization and main page setup

Activity 2: Dual-Mode Interface (Professional & Fresher Tabs)

This activity implements the dual-mode interface with separate functionality for professional and educational use cases.

Explanation

- **Professional tab:** Focused on policy analysis and corporate document understanding
- **Fresher tab:** Designed for study materials with educational features
- **Unified architecture:** Shared backend with mode-specific frontend components

```python
    # -------------------- PROFESSIONAL TAB --------------------
    def render_professional_tab(self):
        st.header("🏢 Professional Assistant")
        st.markdown(
            "Upload long professional documents to get clear, concise explanations and context-aw
        )
        st.markdown("---")

        if not st.session_state.policy_processed:
            self.render_upload_section("Professional")
        else:
            col1, col2 = st.columns([3, 1])
            with col1:
                st.success(
                    f"📄 **Active Document:** {st.session_state.policy_filename}"
                )
            with col2:
                if st.button("🔄 Upload New Document", key="new_policy"):
                    st.session_state.policy_processed = False
                    st.session_state.policy_summary = ""
                    st.session_state.policy_vector_store = None
                    st.session_state.policy_chat_history = []
                    st.session_state.policy_filename = ""
                    st.rerun()
            self.render_summary_and_questions("Professional")
```

```python
    # -------------------- FRESHER TAB --------------------
    def render_fresher_tab(self):
        st.header("🎓 Fresher Assistant")
        st.markdown(
            "Upload study materials to generate summaries, ask supplementary questions, and view
        )
        st.markdown("---")

        if not st.session_state.study_processed:
            self.render_upload_section("Fresher")
        else:
            col1, col2 = st.columns([3, 1])
            with col1:
                st.success(
                    f"📄 **Active Study Material:** {st.session_state.study_filename}"
                )
            with col2:
                if st.button("🔄 Upload New Material", key="new_study"):
                    st.session_state.study_processed = False
                    st.session_state.study_summary = ""
                    st.session_state.study_vector_store = None
                    st.session_state.study_chat_history = []
                    st.session_state.study_filename = ""
                    st.session_state.flashcard_data = None
                    st.session_state.quiz_data = None
                    st.rerun()
            self.render_summary_and_questions("Fresher")
```

Figure 14 - Dual-mode interface showing Professional and Fresher tabs

Activity 3: Document Upload and Processing Interface

This activity creates the document upload system with progress indicators and error handling.

Explanation

- **File upload widget:** Supports multiple document formats
- **Processing feedback:** Shows progress spinners and success/error messages
- **Session management:** Maintains document state across user interactions

```python
    # -------------------- UPLOAD SECTION --------------------
    def render_upload_section(self, tab_name):
        st.header(f"📤 Upload Your {tab_name} Document")
        uploaded_file = st.file_uploader(
            f"Choose a {tab_name.lower()} document (.pdf, .docx, .txt)",
            type=["pdf", "docx", "txt"],
            key=f"uploader_{tab_name.lower()}",
            label_visibility="collapsed",
        )
        if uploaded_file is not None:
            if st.button(
                f"🚀 Process {tab_name} Document", key=f"process_{tab_name.lower()}"
            ):
                if self.process_uploaded_document(uploaded_file, tab_name):
                    st.success(f"✅ {tab_name} document processed successfully!")
                    st.rerun()

```

Figure 15 - Document upload interface with file type support

Milestone 5: Application Deployment

This milestone involves running and hosting the Streamlit app.

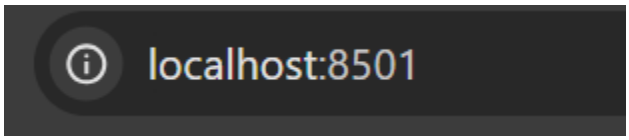
Activity 1: Running the Application Locally

To start the Streamlit app locally:

Use “streamlit run policy_explainer_assistant.py” in your already being used command prompt, like:

```
(policy_env) C:\Users\rithi\Downloads\Policy explainer assistant>streamlit run policy_explainer_assistant.py
```

This will launch the web interface in the default browser at:



Activity 2: Application Execution

You can now view your Streamlit app in your browser with details:

Local URL: http://localhost:8501

Network URL: http://192.168.29.158:8501

```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.29.158:8501
```

Figure 17 - Streamlit application running successfully in terminal

Output Preview

Output 1: Streamlit UI

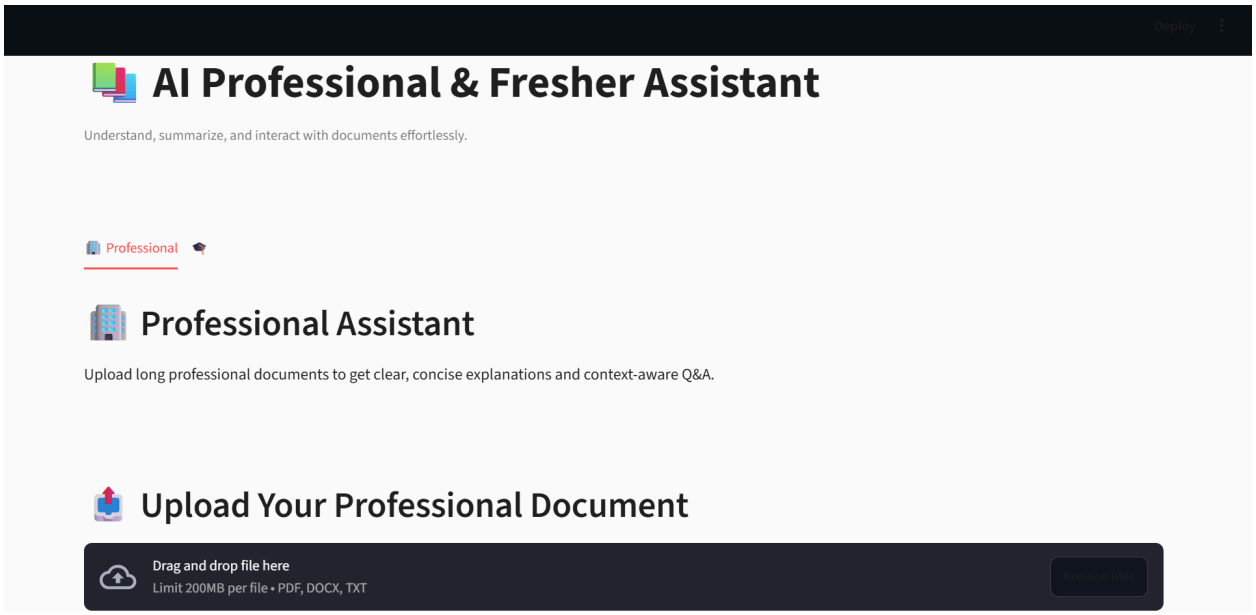


Figure 18 - Complete application interface showing main area

Main Area Features:

- Document upload status and processing indicators
- Generated summary display with structured formatting
- Interactive Q&A section for follow-up questions
- Quick notes generation for study materials
- Quiz creation and navigation buttons after viewing the Quick notes

Output 2: Generated Content Examples

Professional Mode Output Example:

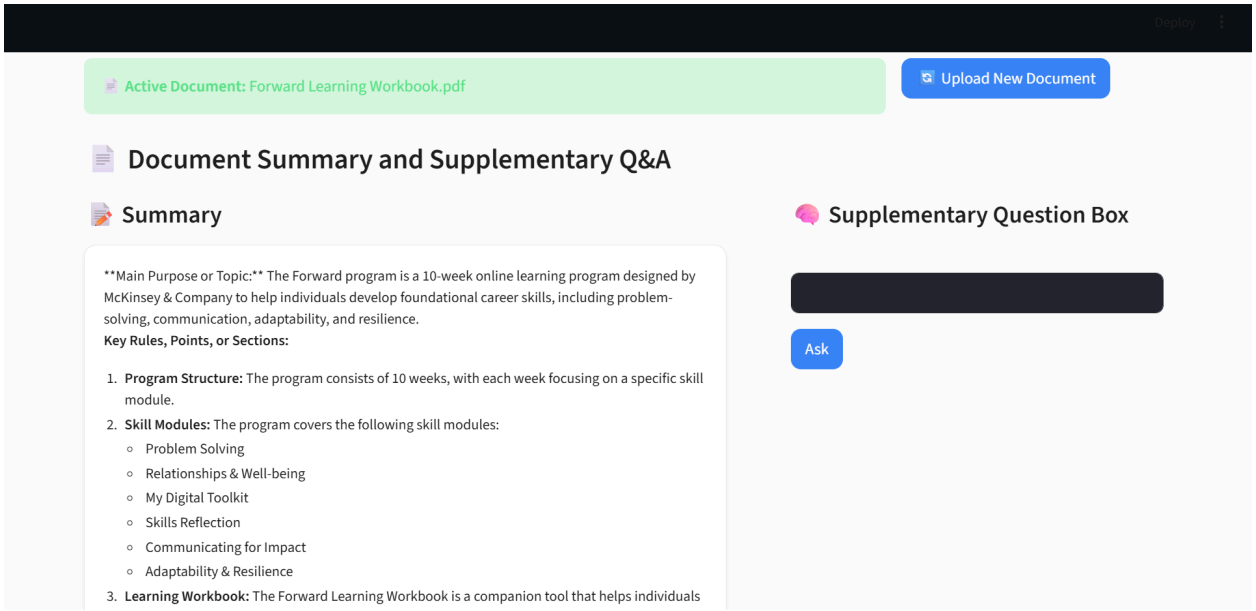


Figure 19 - Example of generated policy summary with structured format in Professional mode

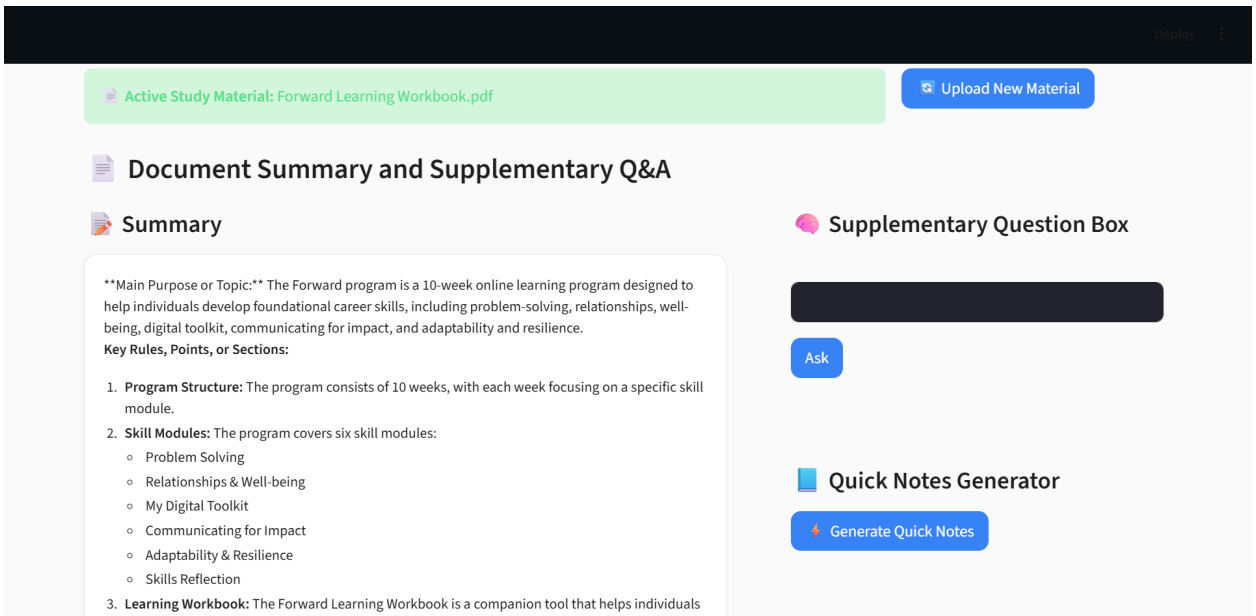
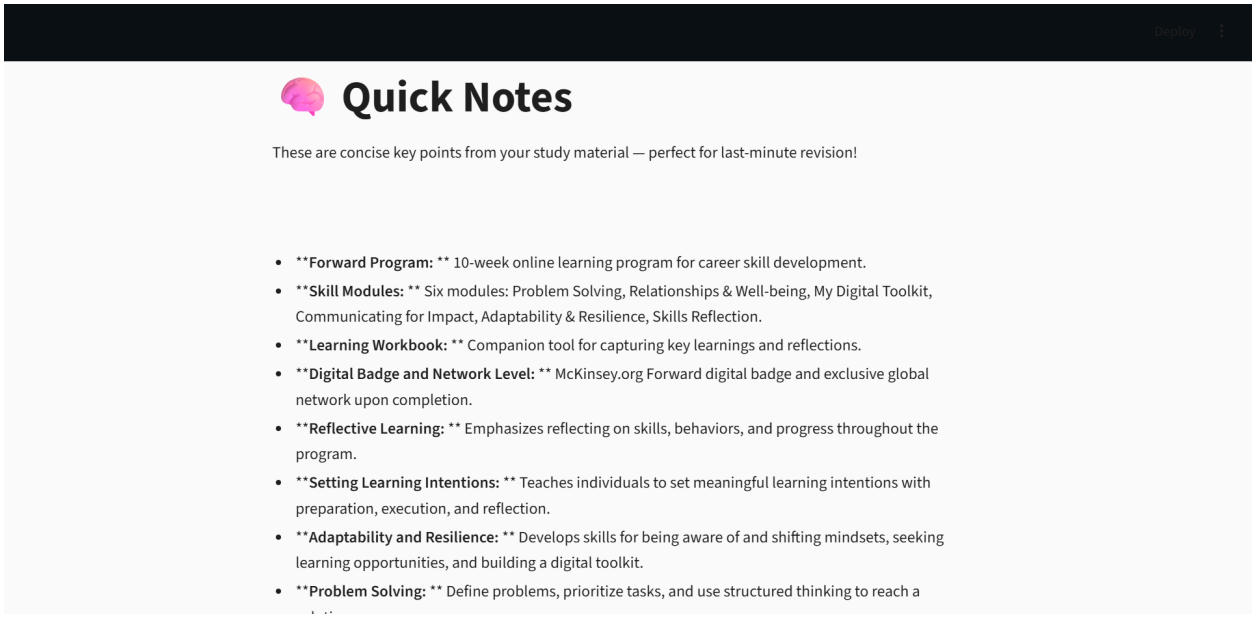


Figure 20 - Example of generated policy summary with structured format in Fresher mode

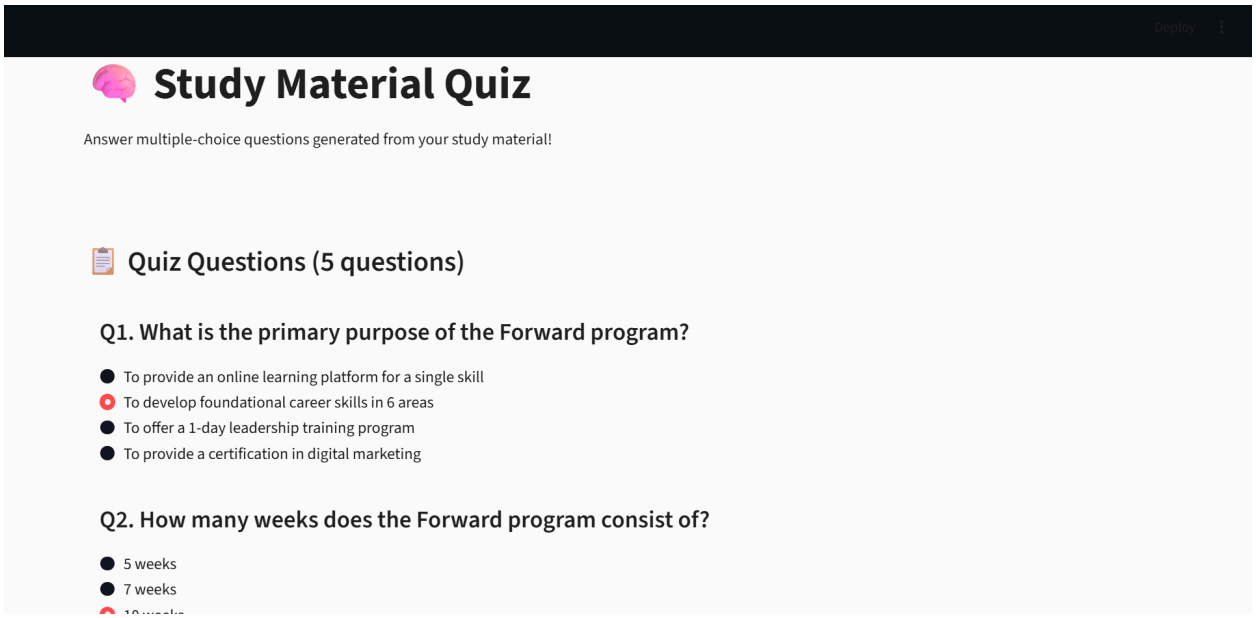


🧠 Quick Notes

These are concise key points from your study material — perfect for last-minute revision!

- **Forward Program:** ** 10-week online learning program for career skill development.
- **Skill Modules:** ** Six modules: Problem Solving, Relationships & Well-being, My Digital Toolkit, Communicating for Impact, Adaptability & Resilience, Skills Reflection.
- **Learning Workbook:** ** Companion tool for capturing key learnings and reflections.
- **Digital Badge and Network Level:** ** McKinsey.org Forward digital badge and exclusive global network upon completion.
- **Reflective Learning:** ** Emphasizes reflecting on skills, behaviors, and progress throughout the program.
- **Setting Learning Intentions:** ** Teaches individuals to set meaningful learning intentions with preparation, execution, and reflection.
- **Adaptability and Resilience:** ** Develops skills for being aware of and shifting mindsets, seeking learning opportunities, and building a digital toolkit.
- **Problem Solving:** ** Define problems, prioritize tasks, and use structured thinking to reach a

Figure 21 - Generated quick notes for study materials



🧠 Study Material Quiz

Answer multiple-choice questions generated from your study material!

📋 Quiz Questions (5 questions)

Q1. What is the primary purpose of the Forward program?

- To provide an online learning platform for a single skill
- To develop foundational career skills in 6 areas
- To offer a 1-day leadership training program
- To provide a certification in digital marketing

Q2. How many weeks does the Forward program consist of?

- 5 weeks
- 7 weeks

Q3. What is one of the key skills developed in the Forward program?

- Leadership skills
- Adaptability and resilience skills
- Digital marketing skills
- Time management skills

Q4. What is the name of the companion tool used in the Forward program?

- Forward Learning Workbook
- McKinsey.org Forward Digital Badge
- Network Level
- Learning Journal

Q5. What is the benefit of completing the Forward program?

- Receiving a certification in digital marketing
- Joining the Network Level, an exclusive global network of lifelong learners
- Gaining a single skill
- Receiving a 1-day leadership training program

🎯 You scored **4/5** (80.0%)

📊 Quiz Results Review

Q1. ✅ What is the primary purpose of the Forward program?

Your answer: To develop foundational career skills in 6 areas

Correct answer: B) To develop foundational career skills in 6 areas

Q2. ✅ How many weeks does the Forward program consist of?

Your answer: 10 weeks

Correct answer: C) 10 weeks

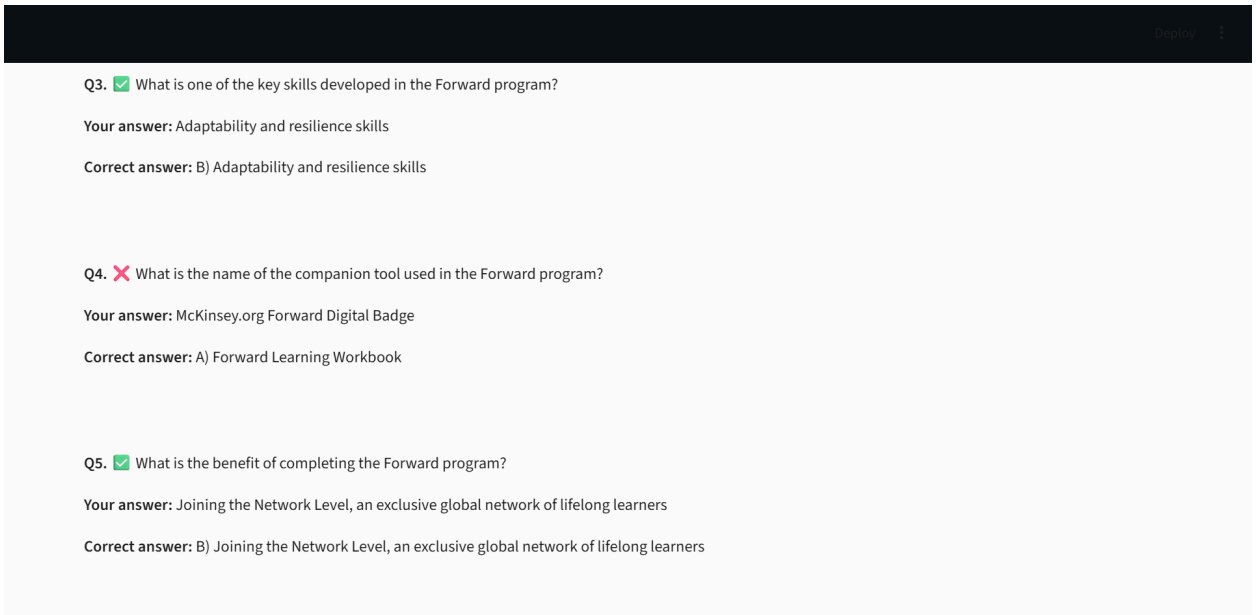


Figure 22 - Interactive quiz interface with questions and scoring

Conclusion

The AI Policy Explainer & Study Assistant revolutionizes how professionals and students interact with complex documents. By combining advanced AI understanding with user-friendly interfaces, it enables efficient document analysis and accelerated learning.

This project demonstrates the powerful synergy between Groq's high-speed LLM inference, LangChain's document processing capabilities, and Streamlit's interactive UI framework. The result is a practical, deployable solution for real-world document understanding and educational support.

With further enhancements such as multilingual support, campaign analytics, and integration with cloud storage services, this application can evolve into a complete AI-powered knowledge management system for organizations and educational institutions of any size.