

NLP Tutor

*NLP Mini-Project report submitted to Visvesvaraya
National Institute of Technology, Nagpur in
fulfillment of requirement for the award of the degree
of*

**Masters of Technology in
Applied AI**

by

SAURABH SHARMA (MT23AAI018)

&

SADASHIV NAYAK A (MT23AAI039)

under the guidance of

Dr. Saugata Sinha



**Department of Electronics and Communication
Engineering Visvesvaraya National Institute of
Technology, Nagpur Nagpur 440010 (India)**

22th June, 2025

Mini Project Report: NLP TutorBot

Introduction

This report provides an in-depth analysis of the NLP TutorBot, rebranded as ChatPDF, an AI-powered tool designed to simplify and optimize interaction with PDF documents. It facilitates querying, extraction, summarization, and contextual responses, enhanced with a user-friendly Streamlit interface and YouTube video suggestions.

Objective

The primary goals of ChatPDF are:

- Enable querying of user-uploaded PDF content.
- Simplify extraction and summarization of document information.
- Offer an accessible, intuitive interface for enhanced usability.

System Overview

The architecture includes:

- **Input:** PDF uploads processed via Streamlit.
- **Processing:** Text extraction, chunking, embedding, summarization, and RAG-based querying.
- **Output:** Contextual answers and YouTube links displayed in the UI.

Code Details and Structure

The project is structured with the following Python files, each contributing to its functionality:

- **functions.py:** Contains core logic for PDF processing and conversation.
 - **get_pdf_content:** Extracts text from PDFs using PyPDF2, iterating over pages to build raw text.
 - **get_chunks:** Splits text into 1000-character chunks with 200-character overlap using CharacterTextSplitter.
 - **get_vectorstore:** Generates embeddings with HuggingFaceEmbeddings (all-MiniLM-L6-v2) and stores them in FAISS, including error handling for invalid chunks.
 - **conversation_chain:** Implements LangChain's RAG pipeline with a history-aware retriever, using LLaMA3-70b-8192 via Groq API to generate responses, yielding chunks for streaming output.

- **app.py:** Manages the Streamlit UI and backend processing.
 - Initializes Streamlit with custom CSS from `html_template.py` and handles file uploads with `hash_files` for unique identification.
 - `async_process_files`: Processes PDFs asynchronously using threading, updating progress and ETA in session state.
 - `get_youtube_videos`: Fetches top 3 videos using `yt-dlp` based on user queries.
 - `save_vectorstore_to_disk` and `load_cached_files`: Manages caching with `pickle` in the `cache/` directory.
 - Includes UI elements like chat input, progress bars, and app management buttons (e.g., CLEAR CACHE, RESTART APP).
- **models.py:** Configures the language model.
 - Loads environment variables (e.g., API key) from `config/.env` using `dotenv`.
 - Initializes ChatGroq with LLaMA3-70b-8192, setting parameters like temperature (0.7) and `max_tokens` (8192) for optimal performance.
- **html_template.py:** Defines UI styling and templates.
 - `css`: Styles chat messages with Tailwind-inspired classes, setting background colors and layouts.
 - `ai_template` and `human_template`: HTML templates for bot and user messages, embedding avatar images (e.g., `ai_profile_photo.png` at 120x120 pixels).
 - `hide_st_style`: Hides Streamlit's default menu and footer for a cleaner look.

Architecture Explanation

The ChatPDF architecture is a sophisticated, RAG-driven system optimized for document interaction:

- **Input Layer:** Streamlit accepts PDF uploads, hashing files with `hash_files` in `app.py` for caching in `cache/`. The UI provides options to load cached sessions or process new files.
- **Preprocessing Layer:** `get_pdf_content` in `functions.py` uses `PyPDF2` to extract text, while `get_chunks` splits it into manageable chunks. This layer ensures scalability for large documents.

- **Embedding Layer:** `get_vectorstore` employs `HuggingFaceEmbeddings` to create vector representations, stored in FAISS for efficient retrieval. Caching with `pickle` allows merging of multiple vectorstores.
- **Summarization Layer:** Although not fully implemented, the design reserves space for `facebook/bart-large-cnn` to summarize sections, enhancing content overview.
- **Retrieval and QA Layer:** `conversation_chain` leverages `LangChain`'s history-aware retriever to contextualize queries, using RAG to fetch relevant chunks. The `LLaMA3-70b-8192` model (via Groq API) generates responses, with `deepset/roberta-base-squad2` as a fallback. Optional `LangSmith` integration tracks quality.
- **Output Layer:** `app.py` renders responses in the Streamlit chat interface using templates from `html_template.py`. `get_youtube_videos` integrates `yt-dlp` to suggest videos, enhancing educational value.
- **Async Processing:** `async_process_files` uses threading to handle background tasks, updating `st.session_state` with progress and ETA for a responsive user experience.

Workflow

The process flows as follows:

1. Users upload PDFs via Streamlit, triggering `async_process_files`.
2. Text is extracted and chunked, then embedded and stored in FAISS.
3. Queries are processed by `conversation_chain`, retrieving relevant content.
4. Responses are generated and displayed, with YouTube links fetched concurrently.

Features

Key functionalities include:

- **PDF Interaction:** Guides users in querying content.
- **Dynamic Data Querying:** RAG ensures contextual accuracy.
- **User-Friendly Interface:** Streamlit with custom styling offers seamless navigation.

Challenges and Solutions

Development challenges addressed:

- **Challenge:** Long processing times for large PDFs.
- **Solution:** Asynchronous processing with progress tracking.

- **Challenge:** Context accuracy in queries.
- **Solution:** History-aware retriever with RAG.

Technologies Used

The project utilizes:

- **Python:** Core language.
- **LangChain:** Drives RAG and conversational logic.
- **Hugging Face Embeddings and FAISS:** Enables efficient storage and retrieval.
- **Streamlit:** Powers the UI.
- **Groq API and LLaMA3-70b-8192:** Handles high-performance NLP.
- **LangSmith (Optional):** Evaluates conversation quality.
- **yt-dlp:** Fetches YouTube videos.

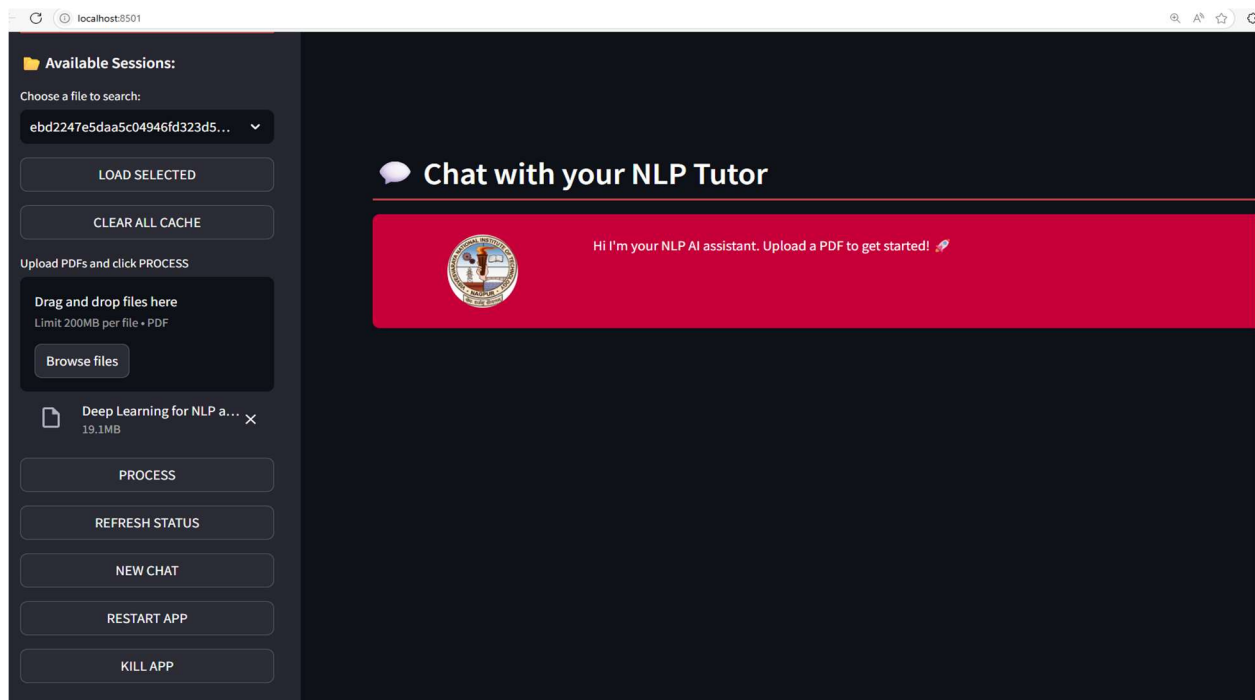
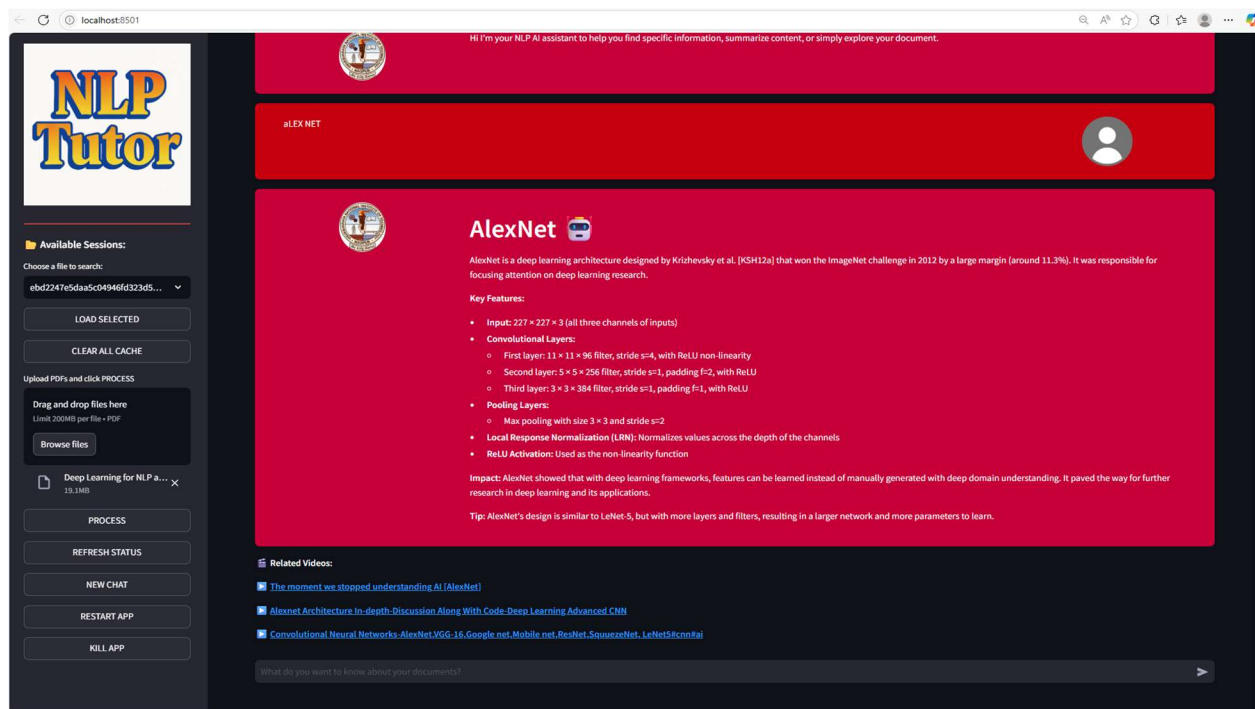
Project Structure

Directory layout:

- `cache/`: Stores vectorstores and processed PDFs.
- `config/`: Holds `.env` for API keys.
- `docs/`: Test PDFs.
- `prints/`: Screenshots.
- `src/`: Source code.
- `README.txt`: Project overview.
- `requirements.txt`: Dependencies.

App Snippet for Output

Sample UI interaction:



Future Enhancements

Proposed upgrades:

- Multi-language support with Noto Serif fonts.

- Advanced summarization with google/flan-t5-large.
- Cloud deployment for scalability.

Conclusion

NLP TutorBot integrates advanced NLP and RAG(Retrieval Augmented Generation) with a responsive UI, offering a robust platform for document interaction with significant potential for future growth.