DAY – 97

19 December 2025

Chatbot Logic File (chatbot_logic.py)

1. Purpose of chatbot_logic.py

This file contains the **core intelligence of the chatbot**.
It is responsible for:

- Loading data from **dataset.txt** and **PDF files**
- Cleaning and normalizing user input
- Understanding user questions (including follow-up questions)
- Finding the **best matching answer**
- Supporting **English and Punjabi**
- Returning short or detailed answers based on user request

This file is connected to `app.py`, which calls the `get_response()` function.

2. Libraries Used

This file uses multiple libraries for different purposes:

- **NLTK** → Text processing (stopwords, tokenization)
- **Scikit-learn** → TF-IDF, Logistic Regression, cosine similarity
- **PyPDF2** → Reading and extracting text from PDF
- **LangChain + Groq (optional)** → Advanced RAG-based answers
- **Regex & Unicode** → Text cleaning and language handling
- **dotenv** → Load API keys securely
- **requests** → Translation support

3. Environment & Configuration

- API keys (Gemini, HuggingFace, Groq) are loaded using .env
- Dataset paths:
  - dataset.txt → Manual Q&A
  - PRSC.pdf → Official document
- Debug flag is used for error tracking

4. Text Preprocessing Functions

These functions prepare text for matching:

- **normalize_text()**
  - Converts text to lowercase
  - Removes symbols, stopwords
  - Handles Punjabi Unicode text
- **tokenize_and_stem()**
  - Breaks sentence into keywords
- **remove_stopwords()**
  - Removes common English words

This improves accuracy of matching.

5. PDF Processing Logic

- robust_extract_pdf_text()
  - Extracts clean text from PDF
  - Removes headers, footers, page numbers, figure references
- parse_pdf_qa_strict()
  - Detects questions and answers inside PDF
  - Converts them into usable Q&A format

6. Dataset & Training

The function **load_and_train()**:

- Loads Q&A from dataset.txt
- Extracts Q&A from PDF
- Stores questions and answers
- Builds:
  - TF-IDF vectorizer
  - Logistic Regression classifier
  - Token-based index

This prepares the chatbot for answering questions.

7. Matching Techniques Used

The chatbot uses a **hybrid approach**:

1. TF-IDF similarity
2. Machine Learning classifier
3. Rule-based keyword matching
4. Embedding similarity (optional)
5. RAG (PDF + LLM fallback)

The best answer is selected based on highest confidence.

8. Follow-Up Question Handling

The function **handle_follow_up()**:

- Detects words like *this, above, brief, explain again*
- Finds the last meaningful user question
- Rewrites the query to make it complete

This allows **context-aware conversation**.

9. Short Answer Feature

If user asks:

- "short"
- "brief"
- "in one line"
- "definition"

Then **force_short_answer()**:

- Converts long answers into bullet points
- Removes explanations
- Keeps only main keywords

10. Language Translation

- Punjabi input is translated to English for processing
- Final answer is translated back to Punjabi if required
- Uses Google Translate API

11. Main Function – get_response()

This is the **main function used by app.py**.

It performs the following steps:

1. Loads data if not already loaded
2. Handles follow-up questions
3. Detects language
4. Normalizes input
5. Finds best answer using hybrid search
6. Applies short-answer logic if requested
7. Translates answer if needed
8. Updates chat history
9. Returns final formatted answer

12. Chat History Management

- Stores last 50 user-bot interactions
- Used for follow-ups and RAG context
- Prevents unlimited memory usage

13. Conclusion

This file acts as the **brain of the chatbot system**.
It ensures:

- Intelligent question understanding
- Accurate answer matching
- Multilingual support
- Context-aware conversation
- Short and long answer handling

Together with app.py, this logic makes the chatbot **smart, flexible, and user-friendly**.