Urdu LLM v01

1. Abstract

Urdu Notebook Assistant v0.2 is an academic prototype built using **Streamlit** for experimenting with Urdu text summarization and question-answering. The application provides a **ChatGPT-style interface** where users can:

- Start new chat sessions,
- Save and manage chat histories,
- Paste Urdu text for analysis (summarization, Q&A),
- Interact with a minimal AI-powered assistant.

The system uses **local rule-based text processing functions** (`simple_summarize` and `simple_qa`) as placeholders for real AI models. It also demonstrates chat persistence via a local **JSON-based database** (`chats_v02.json`).

2. Program Overview

- **Framework**: Streamlit (UI + session management)
- **Storage**: JSON file (`chats_v02.json`) to persist chat history across sessions
- **Core Features**:
    - Multi-chat support (switch, save, delete, create new chats)
    - Urdu text summarization (rule-based, first 2 sentences)
    - Simple Q&A (template-based answer)
    - Modern chat UI (ChatGPT-style messages)
    - Sidebar for chat management

When we talk about the "program," we mean how everything works together. The program runs as a **web app** in your browser thanks to Streamlit. Instead of you writing commands in the terminal, you open a webpage where you see buttons, text boxes, and chat bubbles. This makes the experience much closer to modern AI chat applications like ChatGPT.

The program has two major sides: the **user interface (UI)** and the **data storage system**. The UI handles what you see, like the chat bubbles, sidebar, and buttons. The data storage system is in charge of remembering what you said earlier, saving chats, and loading them when you come back later. To store this memory, the program uses a file called `chats_v02.json`. Every time you write a message or the assistant replies, this file gets updated with the conversation history.

If you think of the program as a small café: Streamlit is the café itself where customers sit (the interface), the assistant is the waiter who responds, and the JSON file is the notebook where orders and conversations are written down so nothing is forgotten.

3. Data Types & Structures

3.1 Data Structures

- **Chat Storage (`chats_v02.json`)** :

```
{
  "chat_id": {
    "title": "Chat Title",
    "timestamp": "2025-08-29T02:55:32.755587",
    "messages": [
      {
        "role": "user" | "assistant",
        "content": "Message text",
        "timestamp": "ISO_DATETIME"
      }
    ]
  }
}
```

- **In-memory session state (`st.session_state`)**:
  - `current_chat_id: str` – Active chat session UUID
  - `messages: list[dict]` – Messages for active chat
  - `chats: dict` – Loaded chats

3.2 Data Types

- **Message Object**:

```
{
    "role": "user" | "assistant",
    "content": str,
    "timestamp": str  # ISO format
}
```

- **Chat Object**:

```
{
    "title": str,
    "timestamp": str,
    "messages": list[Message]
}
```

4. Functions

4.1 Persistence Functions

The program is divided into many **functions**, and each function has a clear job. This is important because instead of writing one long piece of code, we break it into small reusable chunks.

There are functions for saving and loading chats. `load_chats()` reads the JSON file and gives you all your past conversations. If the file doesn't exist or something goes wrong, it just returns an empty dictionary. `save_chats()` does the opposite: it takes the updated dictionary of chats and writes it back into the file. Together, these two functions act as the "memory librarian."

- `load_chats() -> dict`
  Loads saved chats from `chats_v02.json`.
- `save_chats(chats_data: dict) -> bool`
  Saves chats back to JSON file.
- `save_current_chat()`
  Saves current chat session with auto-generated title (from first user message).

4.2 Chat Management Functions

Then there are chat management functions. `create_new_chat()` starts a fresh chat by giving it a new ID and clearing old messages. `switch_chat()` allows you to jump between old and new chats, just like switching between tabs. `delete_chat()` removes a chat, which is like tearing a page out of your notebook. There's also `add_message()` which is the actual act of writing something into the conversation: when you or the assistant speak, this function is called.

- `create_new_chat()`
  Creates a new session with fresh UUID.
- `switch_chat(chat_id: str)`
  Loads messages for an existing chat.
- `delete_chat(chat_id: str)`
  Deletes a specific chat, reinitializing session if it was active.
- `add_message(role: str, content: str)`
  Appends a message to session and persists chat.

4.3 Text Analysis Functions

Finally, there are text analysis functions. `simple_summarize()` takes some Urdu text, cuts it into sentences using the ۔ character (which acts like a full stop in Urdu), and then returns just the first two sentences as a "summary." It doesn't understand meaning—it just shortens text. `simple_qa()` takes a text and a question but doesn't actually answer it. Instead, it gives a canned reply saying "based on the text, here is your answer." This is a placeholder that later could be replaced by a real AI model.

- `simple_summarize(text: str) -> str`
  - Splits Urdu text by ۔ (full stop).
  - Returns first 2 sentences as summary.
  - Prefixes result with "📄 خلاصہ".
- `simple_qa(text: str, question: str) -> str`
  - Returns a **templated response** acknowledging the question.
  - Does not extract actual answers (placeholder).

5. User Interface (Streamlit Components)

The interface is what makes this program enjoyable instead of scary. On the left side (the sidebar), you see all your past chats, buttons to start a new one, and even a button to delete everything. On the right side (the main chat area), you see the messages displayed in colored bubbles: dark grey for the user and black-green for the assistant. This visual styling is done with **CSS**, which is like paint for the web.

At the bottom, you can type a new message or paste a whole text for summarization. If you choose chat mode, it feels like talking to a chatbot. If you choose analysis mode, you paste longer text and then either summarize it or ask a question. This flexibility shows how the same system can support both conversation and document analysis.

- **Sidebar**:
  - New Chat button
  - List of saved chats (sortable by timestamp, deletable individually or all)
  - Version info
- **Main Panel**:
  - Chat history display (styled as ChatGPT bubbles)
  - Input options:
    - **Chat Mode**: free-form conversation
    - **Text Analysis Mode**: paste Urdu text → summarize or ask questions

6. Limitations

1. **Summarization & QA are simplistic**
   - Summarization only extracts first 2 sentences.
   - Q&A does not perform retrieval, just returns a canned message.
2. **Persistence Issues**
   - JSON file storage is not scalable (race conditions, corruption if file edited manually).
   - No database or cloud sync support.
3. **Single-user system**
   - Session state is not multi-user aware.
   - If deployed, all users would share the same `chats_v02.json`.
4. **No real AI/NLP**
   - No semantic understanding or context-based answers.
   - Limited to Urdu text hardcoded splitting rules.

Since this is only my second version of the prototype, I can clearly see its current weaknesses. The biggest one is that the summarization and Q&A functions are **very basic placeholders**. For example, my summarizer just takes the first two sentences of the text instead of generating a true summary, and my Q&A function doesn't really "answer" based on meaning, rather it just returns a template response. I know this isn't real AI, but I included it to show the *workflow* of how the system would behave if a proper model were connected.

Another limitation comes from the fact that I am storing all chats in a **single JSON file**. This is simple and good for learning, but I understand that it is not safe if multiple people use the app at once. The file could get overwritten or corrupted. Also, since there is no concept of users or logins, all chats are shared. If this were a bigger system, people would expect their own private chat histories. I know the solution would be to use a database like SQLite or PostgreSQL, and to add authentication, but I also recognize that this is **too complex for my level right now**. That's why I kept JSON, it let me move forward without getting stuck in database design.

I also see that my prototype cannot handle **large texts or complex analysis**. If someone pastes an entire novel, my summarizer will just chop it up instead of really condensing the meaning. Similarly, my assistant cannot hold deep conversations because it doesn't have a real language model behind it yet. For now, I accept this because the purpose was to design a working chat interface in Urdu, not to solve natural language processing completely.

7. Future Improvements

1. **AI Integration**

   The most obvious improvement is to replace the fake AI with real AI. Instead of `simple_summarize`, you could send the text to OpenAI's GPT API or DeepSeek's R1 model, which would return a true natural language summary in Urdu. Similarly, `simple_qa` could be upgraded to use embeddings and semantic search, making it possible to answer questions based on context.

2. **Storage Enhancements**

   Another improvement would be to switch from JSON to a proper database like SQLite or PostgreSQL. This would allow multiple users, faster queries, and less chance of losing data. Along with that, adding authentication (logins) would make each user have their own private chats. However, while this is a great direction for future growth, it's also more complex than what a beginner needs to handle right now. Working with databases means learning about SQL queries, schemas, and sometimes even server setups. For the purpose of this prototype, sticking with JSON is perfectly fine because it keeps things simple and helps you focus on learning the basics of chat handling, text processing, and UI building. You can always upgrade later once you're comfortable.

3. **Advanced Features**
   - File/document uploads (PDF, Word, TXT) for summarization
   - Export chats (PDF, DOCX)
   - Search across chat history
4. **UI/UX Improvements**
   - Add streaming responses (like ChatGPT typing effect)
   - Mobile-friendly view
   - Pin/rename chats

8. Prototype Intent

It's very important to understand that this is a **prototype**. This project is a minimal prototype intended for:

- Demonstrating UI/UX concepts in Urdu language,
- Testing chat-based text summarization flows,
- Exploring integration possibilities with large language models (LLMs).

It is **not a production-ready system** but a **foundation for future research and development**.

9. Learning Outcome

The first thing I learned was how to use **Streamlit** to create an actual web interface. Before this, I only thought of Python as something that runs in the terminal, but now I can see how Python can power an interactive app that feels like a modern AI tool. I learned how Streamlit's session state works, which was challenging at first, but it helped me understand how to keep track of multiple conversations and switch between them.

I also learned the basics of **working with data storage**. Using a JSON file to save chat history showed me how data can persist even when the app restarts. Although I now realize that JSON has its limits, it was still a valuable introduction to the idea of storing structured data in a way that is both human-readable and machine-friendly. It gave me my first taste of thinking about "where does the data go?" in an app.

Another big lesson was around **natural language processing**, even if my functions were simple. Writing the `simple_summarize` and `simple_qa` functions made me think about how text can be broken down, manipulated, and reformatted. Even though I know they are not true AI, they helped me sketch the *flow* of how a more advanced model (like OpenAI's GPT or DeepSeek) could be inserted later.

I also learned how to **think critically about limitations**. I now understand and clearly explain their weaknesses and propose how to improve them later. In future iterations, I hope to integrate API for smarter answers, but I am aware that the scope of this project is vast and will be time-consuming.