MANIPAL INSTITUTE OF TECHNOLOGY
BENGALURU
(A constituent unit of MAHE, Manipal)

School of Computer Engineering

Project Report: Smart Inventory Management with Natural Language Processing and LLMs

Submitted by:

Preethana- 235890064
Manas Waykole - 235890176
Mariya - 235890028
Adhish V elavan Selvakumar Shanthi - 235890220
Sreepallavi Manikonda - 235890080
K Soveet Kumar Prusty - 235890160

Project: AI Inventory Manager Date: October 27, 2025 Core Technologies: Streamlit, Python, SQLite, OpenAI Whisper, Ollama (LLaMA 3.2)

1. Executive Summary

This project demonstrates the creation of a smart inventory management program that bridges the space between natural language and database operations. The program allows users to manage a product inventory with natural spoken or typable commands (e.g., "add 10 potatoes" or "delete 5 apples").

It employs a modern, locally-hosted stack of AI: OpenAI Whisper for high-accuracy speech-to-text transcription and a LLaMA 3.2 3B model (run through Ollama) to translate these natural language instructions into executable SQL queries. The entire app is wrapped in an intuitive, real-time web UI built with Streamlit, serving as a powerful proof-of-concept for next-gen usable business software.

2. Problem Statement

Legacy inventory management software, while functional, relies on rigid interfaces:

- They must navigate complex forms and menus.
- Direct database interaction requires SQL knowledge.
- This is a deterrent to non-technical users, slowing operations and increasing the likelihood of human error.

The underlying problem is one of efficiency and access. How do we allow any user to look at a database directly, securely, and naturally, just by requesting what they need?

3. System Architecture & Data Flow

The program operates on a simple but efficient data flow:

1. **Input (User):** The user submits a command in one of two ways:
    - Audio Input: Posting of an audio file (WAV, MP3, etc.).
    - Text Input: Entering text in a text field.
2. **Transcription (Whisper):** When there is audio provision, the whisper model translates the speech into a string of text.

3. **NL-to-SQL (LLM):** The transcribed text is entered into the llama3.2 model via Ollama. A well-crafted prompt provides the LLM with the database schema (Inventory table) and stringent instructions for how to format the output.

4. **Generation (LLM):** The LLM generates a raw SQL query (e.g., INSERT INTO Inventory. or UPDATE Inventory.).

5. **Post-Processing & Validation:** The Python script intercepts raw SQL. It intelligently converts simple INSERT statements into a strong UPSERT command (INSERT. ON CONFLICT. DO UPDATE). This is a critical step to prevent mistakes and correctly process new records and quantity updates for existing records.

6. **Execution (SQLite):** The SQL script that has been validated is executed against the in-memory SQLite database.

7. **Feedback (Streamlit):** Streamlit interface automatically refreshes and displays to the user the updated inventory table in real-time.

Technology Stack:

- Web Framework: Streamlit
- Database: SQLite (in-memory)
- Speech-to-Text: whisper (OpenAI)
- LLM Engine: Ollama
- LLM Model: llama3.2:latest (3B parameters)
- Core Language: Python (with pandas for data display)

4. Main Features & Implementation Details

- **Dual-Modal Input:** Both text and audio input are natively supported, providing users with maximum freedom.
- **Local-First AI:** It was a choice of design to utilize Ollama in combination with a local llama3.2 model. This ensures:
    - **Privacy:** Nothing ever escapes the user's machine.
    - **Cost-Effectiveness:** No API call or resulting cost.
    - **Speed:** Low latency for a small, lightweight model like 3B LLaMA 3.2.

- **Intelligent Prompt Engineering:** LLM accuracy is driven by a strong system prompt that defines its role, the exact table schema ("item no", item_name, quantity), and comes with concise rules to elicit tidy, light SQL generation.

- **Robust UPSERT Logic:** The code doesn't trust the LLM. It contains a re.match and post-processing module to transform a plain INSERT into a solid UPSERT statement. This logic (ON CONFLICT(item_name) DO UPDATE SET quantity = quantity + {qty}) forms the basis of stability in the system's database, so that the same natural language can be used to execute "add 5 Potato" (a known item) and "add 10 Apple" (an unknown item) and one safe query.

- **Session State Management:** Streamlit's st.session_state is used to maintain a persistent SQLite connection (conn) to prevent resetting the in-memory database upon each user action.

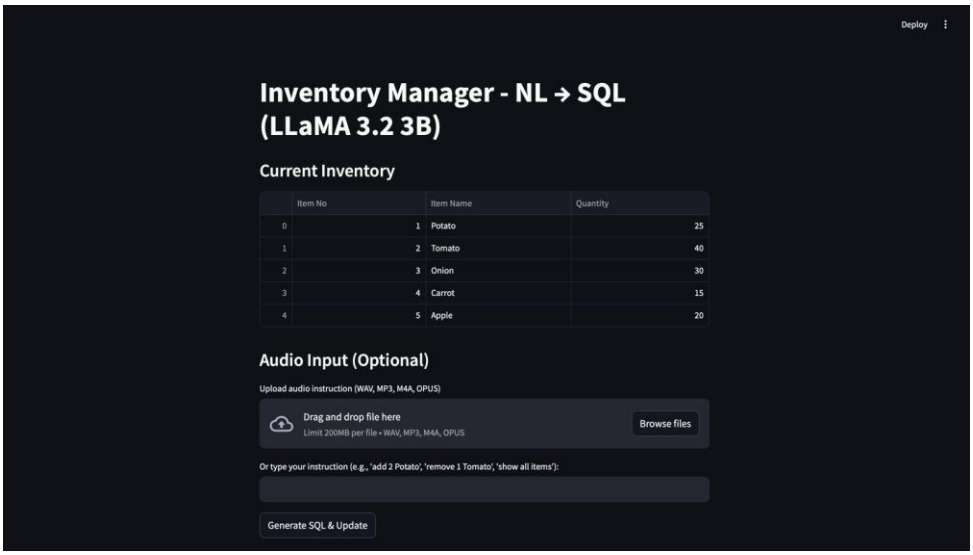

Fig 1: Initial view of the Dashboard with the view of Database.

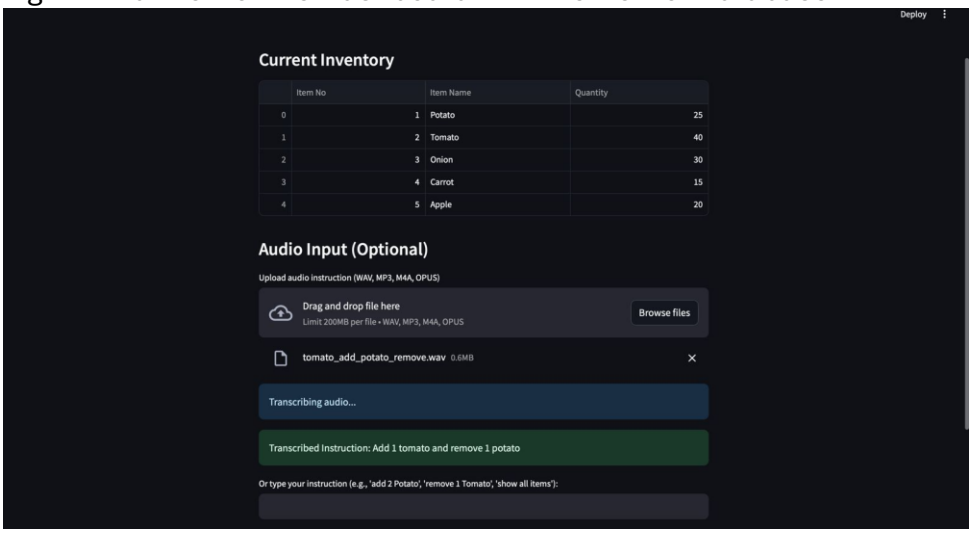

Fig 2: View of the Dashboard while the audio gets transcribed.

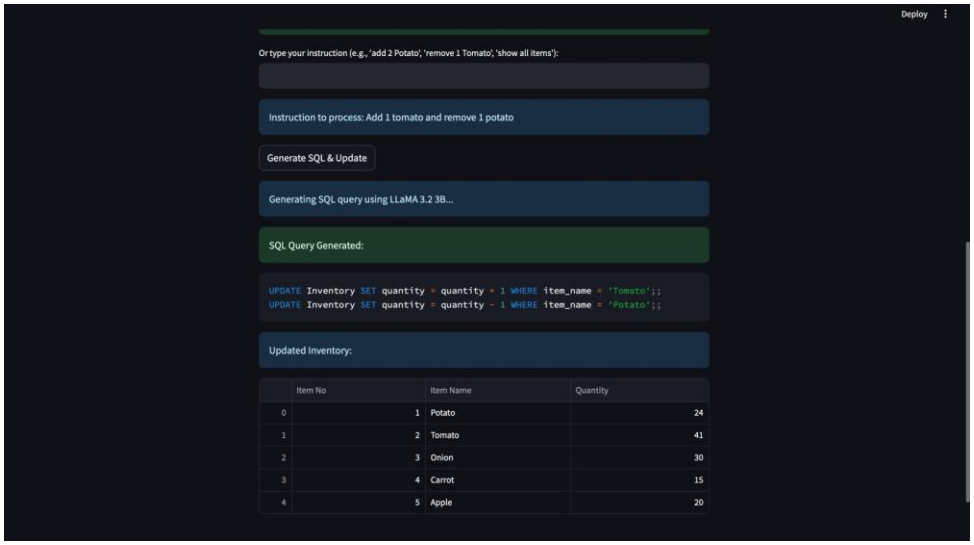

Fig 3: View of the Dashboard after SQL query generation and Database updation.

5. Challenges & Solutions

1. **Challenge: LLM Hallucination/Incorrect SQL.** An 3B model might misinterpret an unclear instruction or generate incorrect SQL.

    - Solution: This was solved in two ways:

        a. **Strict Prompting:** The prompt strictly constrains the LLM to producing SQL for the given schema.

        b. **Post-Processing:** The UPSERT logic is a validation, catching the most common LLM output (INSERT) and hardening it.

2. **Challenge: State of Database.** What if adding new item vs modifying current one? Single user utterance "add 5 apples" can be interpreted in two ways.

    - **Solution:** UPSERT (INSERT. ON CONFLICT) logic does this nicely. We can make item_name UNIQUE and attempt an INSERT and, if that fails (because the item exists already), DO UPDATE the quantity instead. This is more efficient than asking the LLM to SELECT first and then UPDATE or INSERT.

3. **Challenge:** Security (SQL Injection). In theory, a bad user could say "drop table Inventory."

    - **Solution (Current):** This is a standard risk with any NL-to-SQL system. The existing proof-of-concept relies on the LLM's adherence to instructions in order to avoid this.

- **Solution (Future):** A production-grade system would require an "allow-list" parser to guarantee that the LLM generated only INSERT, UPDATE, or SELECT queries in safe, pre-defined templates.

6. Future Scope & Improvements

- **Storage:** Switch from storing in an in-memory database to a file-based SQLite database (sqlite3.connect('inventory.db')) so data will persist application restarts.

- **Enhanced Security:** Impose a strict validation layer that translates the LLM-generated SQL and allows only safe, whitelisted query syntax.

- **Advanced NLP:** Refine the prompt to allow more advanced queries, e.g.:

    - "Show all with less than 20 in stock." (SELECT. WHERE quantity < 20)

    - "How much tomato do I have?" (SELECT quantity FROM Inventory WHERE item_name = 'Tomato')

- **UI Improvements:** Add data visualization (e.g., an inventory level bar chart) and user login for a multi-user environment.

- **Batch Operations:** Support commands like "add 5 potato, 10 onion, and 3 carrot" and allow the LLM to generate several SQL statements.

7. Conclusion and Methodology Diagram

The project is a good demonstration of an extremely useful application of modern, local AI models. With the combination of a simple frontend, a reliable speech-to-text engine, and a fast LLM, we have created an inventory app that is powerful, simple to use, and affordable. The ability of the system to convert natural language into robust SQL queries, particularly its "UPSERT" function, is a good foundation for the development of more intelligent and user-friendly data apps in the future.

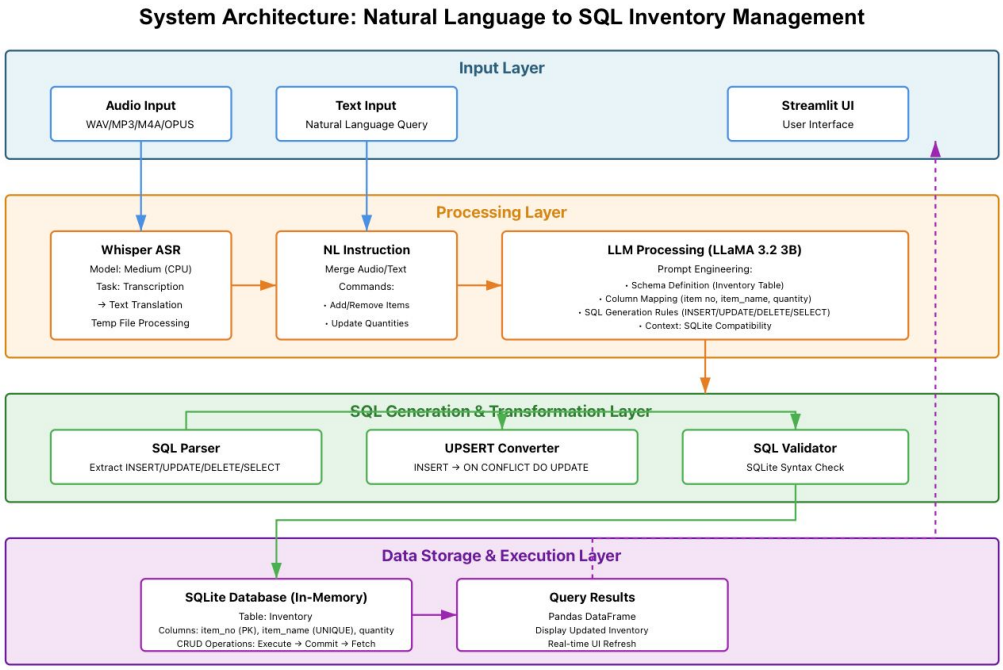

Fig 4: Methodology Diagram