International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Warehouse Chatbot

Ramya B N[1], Mayur Simha M[2], Vivek Patel Y[3], Shravan CU[4], Vrushank Skanda B[5]

[1]Assistant Professor, Artificial Intelligence and Machine Learning, Jyothy Institute of Technology, Bengaluru, Karnataka, India
[2,3,4,5]Student, Artificial Intelligence and Machine Learning, Jyothy Institute of Technology, Bengaluru, Karnataka, India
DOI : https://doi.org/10.5281/zenodo.15541801

A B S T R A C T

This paper presents a chatbot system designed to simplify warehouse inventory checks using a combination of rule-based item search and natural language fallback responses. The system is built using the Streamlit framework for the user interface, Pandas for stock data processing, and OpenAI's GPT-3.5 model to handle queries beyond item matching. Stock quantities are loaded from a CSV file, and queries such as "How many laptops are in stock?" are matched against known inventory items. If no direct match is found, the chatbot leverages GPT-3.5 to provide a relevant, conversational fallback response. A dark-themed, responsive UI ensures ease of interaction. The project emphasizes real-time usability, scalability, and minimal system resource usage.

Keywords: Chatbot, Streamlit, GPT-3.5, Inventory Management, Stock Assistant, Natural Language Processing, CSV

1. INTRODUCTION

1.1. Motivation and Background

Warehouse inventory systems often face operational inefficiencies due to human error or lack of automation. Manually checking stock is time consuming and can interrupt workflows. This paper introduces an assistant chatbot that automates such queries through a simple, web-based system.

The chatbot utilizes a lightweight frontend built in Streamlit and a backend connected to a stock CSV file. For cases where the user query does not directly match the item list, it uses OpenAI's GPT-3.5 model to generate intelligent fallback responses. This hybrid approach ensures fast and meaningful responses for both structured and unstructured user inputs.

2. Methodology

2.1. System Architecture Diagram

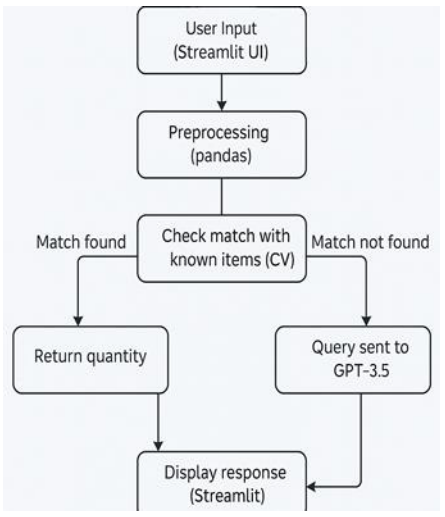

User Input (Streamlit UI)
Preprocessing (pandas)
Match found
Check match with known items (CV)
Match not found
Return quantity
Query sent to GPT-3.5
Display response (Streamlit)

Figure 1: Flowchart

The system architecture consists of several sequential components that collaborate to process user queries and return stock-related responses. The stages are as follows:

1. **User Input Stage**:

   Users interact with the chatbot through a web-based Streamlit interface, where they enter queries related to stock availability (e.g., "Do we have laptops in stock?").

2. **Preprocessing Stage**:

   The query is converted to lowercase, and unnecessary characters are removed to standardize the input. The system then extracts potential item names from the query.

3. **Item Matching Stage**:

   The system compares the extracted item against a dataset loaded from a CSV file using Pandas. If a match is found, it retrieves the corresponding stock quantity.

4. **Fallback Processing Stage**

   If the item is not found in the dataset, the query is forwarded to the GPT-3.5 language model via the OpenAI API. The model generates a context-aware response.

5. **Response Generation Stage**:

   The result—either a stock quantity message or a GPT-generated fallback—is displayed on the Streamlit interface using st.success() or st.info() components.

6. **User Interface Rendering**:

   The dark-themed UI is designed to be intuitive, ensuring users can clearly view both their input and the assistant's response in real time.

2.2 Data Input and Preprocessing

The chatbot uses a CSV file (`stock_data.csv`) with two columns: `item` and `quantity`. This file is loaded using pandas. The user query is normalized to lowercase and then searched against the list of known items. If a match is found, the corresponding quantity is returned.

2.3 Chatbot logic and NLP handling

When the query does not match any known item, it is passed to the GPT-3.5 model using OpenAI's API. The assistant provides a fallback response based on the given system prompt.

2.4 Output Generation

The response, whether a quantity result or GPT-generated message, is rendered in the UI using Streamlit widgets like st.success() or st.info().

2.5 UI Development

A dark-themed UI was created using Streamlit's built-in styling and formatting..

3. Results and Analysis

The chatbot was tested with a sample CSV containing items like laptops, routers, and monitors. Queries like "How many laptops are there?" or "Check router stock" returned accurate results instantly. When queried with unrelated or abstract questions, GPT handled the fallback elegantly.

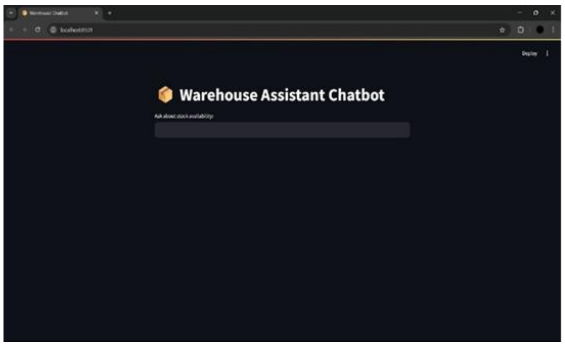
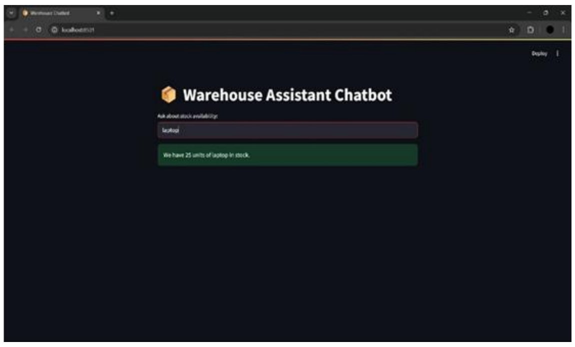


Figure 1 : Input



We have 25 units of laptop in stock.

Figure 2 : Output

4. Conclusion

This project demonstrates a functional and user-friendly chatbot interface for warehouse inventory queries. It effectively combines deterministic item lookup with intelligent fallback using natural language understanding.

5. Limitations

Despite the chatbot's effectiveness in handling stock-related queries, it exhibits several limitations that need to be addressed:

- **Limited Vocabulary Matching**:

  The system performs direct string matching for item lookup. If the query uses synonyms or misspellings (e.g., "notebooks" instead of "laptops"), the system may not recognize the item unless it falls back to GPT-3.5.

- **Dependence on Static CSV Data**:
- The inventory data is loaded from a static CSV file. Any updates to stock require manual editing of the CSV file or restarting the application to reflect changes.
- **Lack of Role-based Access or Authentication**:

  The system does not support authentication or user roles (e.g., admin vs viewer), which limits its use in environments requiring data access control.

- **Latency with Fallback (GPT-3.5)**:

  When a query is passed to the GPT-3.5 model, there may be a noticeable delay in response due to API communication, especially with slower internet connections or API rate limits.

- **No Error Handling for Invalid Inputs or API Failures**:

  If the CSV file is missing or the OpenAI API fails (e.g., network error, quota exceeded), the system does not currently have robust error recovery or fallback messages.

- **Scalability Constraints**:

  For larger inventory datasets (e.g., 1000+ items), performance might degrade as the system uses linear search and basic file loading instead of optimized data structures or a database.

6. Future Work

To overcome the current limitations and improve scalability, several enhancements are proposed:

- **Fuzzy Matching and Synonym Support**:

  Integrate NLP tools like fuzzywuzzy or spaCy to recognize close matches, typos, and synonyms in user queries for more accurate item detection.

- **Database Integration**:

  Replace the static CSV file with a lightweight database like SQLite or a cloud-based database (e.g., Firebase, PostgreSQL) to support live inventory updates, persistence, and scalability.

- **Role-based Access Control**:

  Implement login features with user roles (admin, employee, viewer) to restrict access to sensitive stock data or editing rights.

- **Offline Mode and GPT Alternatives**:

  Provide offline functionality by training a local intent classifier for basic queries and use GPT only when needed, reducing API dependency and cost.

- **Multilingual Query Support**:

  Extend the chatbot's understanding to multiple languages by integrating translation APIs or multilingual NLP models for better inclusivity.

- **Mobile and Progressive Web App (PWA)**:

  Wrap the Streamlit app into a mobile-friendly interface or build it as a PWA for easier access on phones and tablets.

- **Analytics and Usage Logging**:

  Add a dashboard to monitor frequently asked items, user activity logs, and system performance metrics for continuous improvement.

- **Voice Input Integration**:

  Integrate voice-to-text APIs (like Google Speech or Whisper) to allow users to speak their queries instead of typing

REFERENCES

- **OpenAI GPT-3.5 Documentation**

  OpenAI. (2024). https://platform.openai.com/docs Used for fallback natural language response generation.

- **Streamlit Documentation**

  Streamlit Inc. (2024). https://docs.streamlit.io Used for building the user interface and input forms.

- **Pandas Library**

  McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. Core library for loading and processing inventory data from CSV files.

- **FuzzyWuzzy Library (for future matching)**

  Seatgeek Inc. https://github.com/seatgeek/fuzzywuzzy (Planned for implementation in future improvements).

- **Natural Language Processing with spaCy**

  Explosion AI. https://spacy.io Referenced for potential enhancements in query understanding and entity extraction.

- **Secure Deployment Resources**

  PythonAnywhere, Heroku, and Streamlit Community Cloud documentation for secure web deployment guidelines.