Advanced Traditional NLP-Based Chatbot Implementation

Ravi Kiran Raju Kanumuri

University of the Cumberlands

MSAI631 - M90 Artificial Intelligence for Human-Computer Interaction

Dr. Alan Dennis

Feb 15, 2026

Abstract

This project presents the design and implementation of a traditional, non–large language model (non-LLM) chatbot using classical Natural Language Processing (NLP) techniques. The chatbot was developed using Python, Flask, TF-IDF vectorization, and cosine similarity to classify intents and generate predefined responses. Unlike modern deep learning-based conversational agents, this system demonstrates how interpretable and computationally efficient traditional methods can still provide meaningful human–computer interaction. The chatbot supports multiple prompts, handles malformed input, provides a list of capabilities, and includes a confidence scoring mechanism. This report outlines the chatbot development lifecycle, including requirements analysis, system design, implementation, testing, and future improvements. It also reflects on challenges encountered and the knowledge gained throughout the process.

Introduction

Chatbots have become an integral part of modern human–computer interaction. From customer service automation to intelligent tutoring systems, conversational agents play a vital role in enhancing user engagement and operational efficiency. While contemporary chatbot systems often rely on deep learning architectures and large language models (LLMs), traditional NLP approaches remain relevant for controlled environments where interpretability, resource efficiency, and deterministic responses are prioritized.

The purpose of this project was to design and implement a simple chatbot using traditional, non-LLM methods. The system was required to respond to multiple prompts, provide

a list of capabilities, handle malformed or unexpected input, and demonstrate extendability for future integration with AI-as-a-service platforms such as Azure Cognitive Services. Rather than relying on neural networks, this implementation uses TF-IDF vectorization and cosine similarity for intent classification, illustrating how classical NLP techniques can effectively support conversational interaction.

Background

Traditional chatbots are typically based on rule-based logic or vector-space models. In this project, the chatbot uses a vector-space representation of text combined with similarity-based matching to determine the most appropriate response.

TF-IDF (Term Frequency–Inverse Document Frequency) is a statistical measure that evaluates the importance of a word in a document relative to a corpus. Words that appear frequently in a specific sentence but not across all sentences receive higher importance weights. This representation allows textual patterns to be transformed into numerical vectors.

Cosine similarity is then used to measure the similarity between the user's input vector and each stored intent pattern vector. The cosine of the angle between two vectors indicates how similar they are. A value closer to 1 indicates substantial similarity, while a value closer to 0 indicates weak similarity. By selecting the pattern with the highest similarity score, the chatbot determines the most relevant intent.

This method provides several advantages. It is computationally efficient, interpretable, and deterministic. Unlike neural networks, which operate as black-box systems, TF-IDF and cosine similarity allow developers to analyze and adjust how text similarity is computed directly.

Chatbot Development Lifecycle

Figure 1: *Chatbot Development Lifecycle*

Requirement Analysis

The first stage of development involved analyzing the assignment requirements. The chatbot needed to satisfy several conditions. It had to respond to multiple prompts, provide users with a description of its capabilities, handle malformed input gracefully, and demonstrate potential for future extension. Additionally, the chatbot was required to use traditional NLP methods rather than LLM-based approaches.

Based on these requirements, I defined key system objectives: implement intent classification using vector-space modeling, incorporate a fallback mechanism for low-confidence predictions, and design a web interface that enables intuitive interaction.

System Design

The system architecture was designed with modularity and extensibility in mind. The knowledge base was stored in a JSON file containing predefined intents. Each intent included a tag, multiple linguistic pattern variations, and corresponding responses. Expanding each intent with several variations improved classification accuracy and reduced misclassification.

A preprocessing pipeline was implemented to normalize input text. Tokenization and lemmatization were performed using the Natural Language Toolkit (NLTK). Converting words to their base forms improved matching consistency across variations of the same term.

The core classification pipeline followed these steps:

1. User input collection.
2. Text preprocessing (lowercasing, tokenization, lemmatization).

3. TF-IDF vector transformation.
4. Cosine similarity computation.
5. Intent selection based on maximum similarity.
6. Confidence threshold evaluation.
7. Response generation.

A **threshold value of 0.35** was selected after experimentation. Lower thresholds produced incorrect matches, while higher thresholds rejected valid queries. The chosen threshold balanced precision and recall effectively.

Implementation

The chatbot was implemented using Python and Flask to create a lightweight web server. Scikit-learn was used to implement TF-IDF vectorization and cosine similarity. The system loads the JSON intent file during initialization and constructs the TF-IDF matrix from stored patterns.

To improve transparency, a confidence scoring mechanism was added. Instead of returning only a textual response, the chatbot also returns the cosine similarity score. This addition enhances interpretability and demonstrates an understanding of classification reliability.

The web interface was modernized using HTML, CSS, and JavaScript. AJAX-based requests were implemented to avoid page reloads and simulate real-time chat interaction. The interface includes chat bubbles, a dark theme design, and a confidence score display, creating a professional and user-friendly experience.

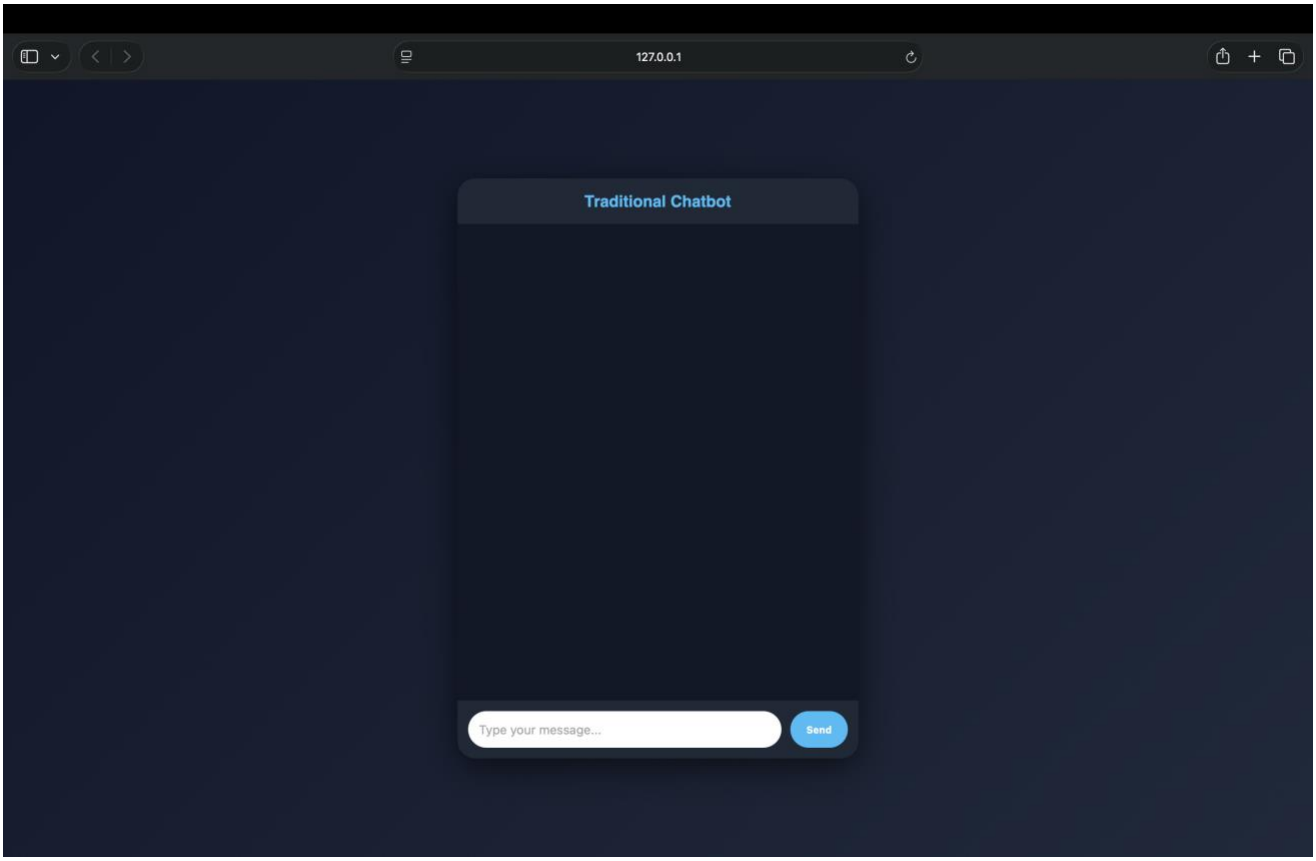

Figure 2: *Illustrates the modern web-based interface of the chatbot*

Testing

Testing was conducted through multiple scenarios. Greeting variations were tested to verify that linguistic diversity did not affect classification accuracy. Reworded technical questions were evaluated to assess the robustness of vector similarity. Malformed input, such as random characters, was tested to ensure fallback functionality operated correctly.

The confidence score feature also allowed inspection of the reliability of the predictions. High-confidence responses aligned closely with known patterns, while ambiguous inputs produced lower scores and triggered fallback behavior.

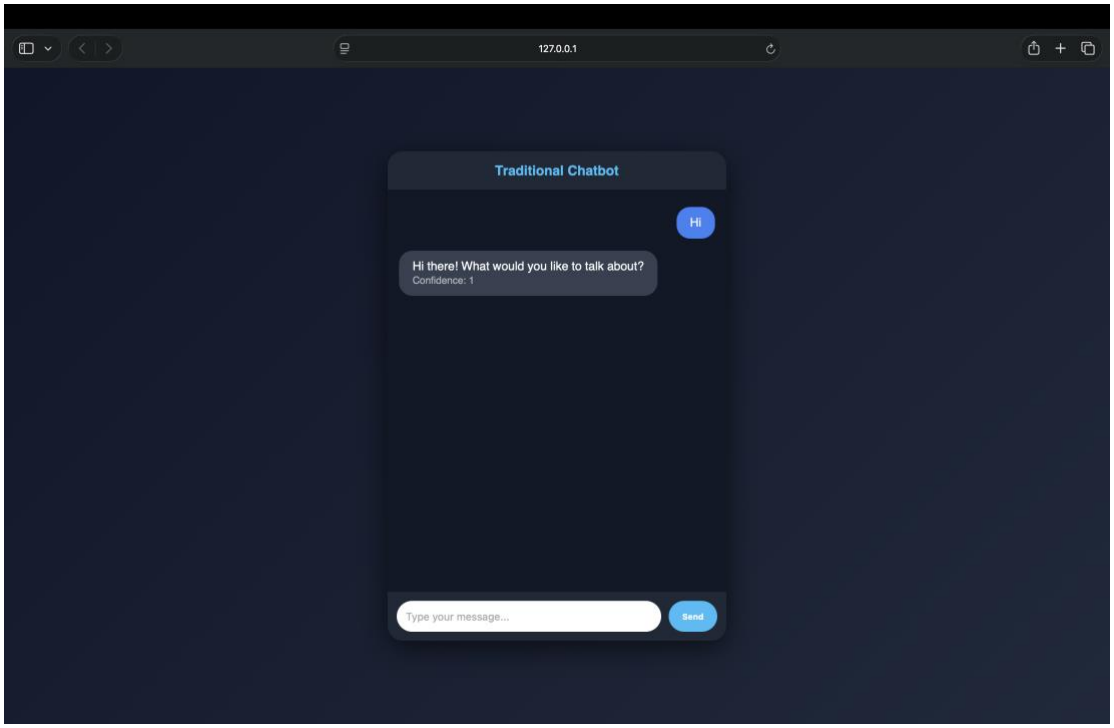

Figure 3: *Demonstrates successful intent recognition for greeting patterns.*

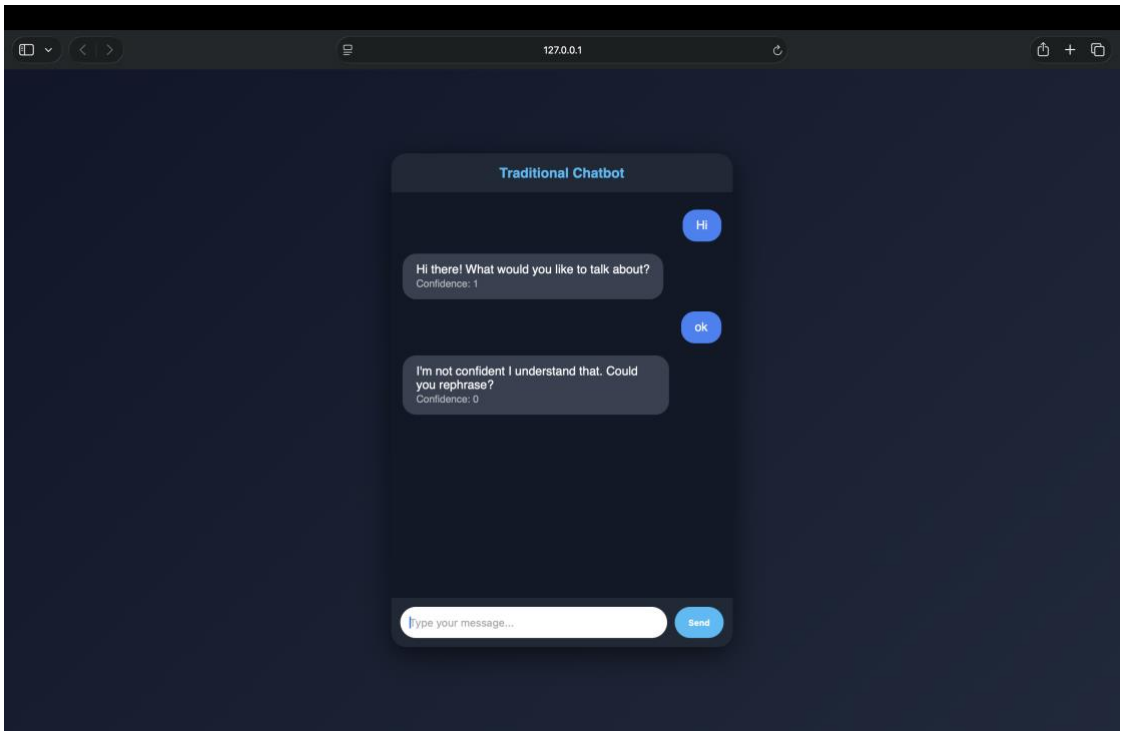

Figure 4: Shows the fallback mechanism triggered by low similarity scores – *When the user input does not match known intents, the chatbot prompts the user to rephrase*.

Deployment Considerations

Although the chatbot was run locally using Flask, it was designed to be cloud-ready. Because the core logic is modular, future integration with Azure Cognitive Services or other APIs would require minimal changes. The architecture supports extensibility, fulfilling the assignment requirement for potential AI-as-a-service integration.

Challenges and Mitigation Strategies

Several challenges emerged during development. The first issue involved NLTK resource errors. When executing the program, tokenizer packages such as “punkt” were missing. This resulted in runtime exceptions. The issue was resolved by explicitly downloading required NLTK resources during initialization. This experience reinforced the importance of dependency management in NLP projects.

Another challenge involved Python version compatibility. Specific packages behaved differently across Python versions. To mitigate this issue, a requirements.txt file was created to specify exact library versions, ensuring reproducibility across environments.

Intent misclassification was also observed during early testing. When only one or two patterns were provided per intent, cosine similarity sometimes matched incorrect categories. This problem was addressed by expanding each intent to include four to six linguistic variations. Increasing training density significantly improved classification reliability.

Finally, the initial user interface design was too simplistic and relied on page reloads, which reduced the user experience. The issue was mitigated by implementing asynchronous AJAX calls and redesigning the interface using modern styling techniques.

What I Learned

This project significantly strengthened my understanding of classical NLP techniques. I gained practical experience implementing TF-IDF vectorization beyond theoretical study. I also developed a deeper appreciation for preprocessing steps such as lemmatization, which greatly influence classification performance.

The addition of confidence scoring enhanced my understanding of model interpretability. Observing similarity values in real-time provided insight into how classification thresholds affect system behavior.

From a software engineering perspective, I learned the importance of modular design, dependency control, and structured project organization. Creating a GitHub repository, documentation, and reproducible environment reinforced professional development practices.

Most importantly, this project demonstrated that traditional NLP methods remain effective for controlled conversational systems. While LLMs dominate modern AI discussions, classical techniques still provide efficient, interpretable, and resource-conscious solutions.

Future Improvements

Future enhancements include integrating sentiment analysis, persistent storage of chat history, and deployment to a cloud platform. A hybrid classifier combining TF-IDF with

machine learning algorithms, such as logistic regression, could further improve accuracy. Integration with external APIs would also enhance functionality.

Use of Artificial Intelligence Tools

This paper was developed with the assistance of an AI-based language model (ChatGPT) to support content organization, clarify technical concepts, and improve academic writing. All ideas were reviewed, edited, and contextualized by the author to ensure originality, accuracy, and compliance with academic integrity standards.

Conclusion

This project successfully demonstrates the design and implementation of a traditional NLP-based chatbot using TF-IDF vectorization and cosine similarity. The system satisfies all assignment requirements by responding to multiple prompts, providing capability descriptions, handling malformed input, and supporting extensibility. Through this development process, I strengthened both technical and analytical skills while gaining practical insight into classical NLP methodologies. The final implementation highlights the continued relevance of traditional approaches in modern conversational AI systems.

References

Microsoft. (n.d.). *Microsoft Bot Framework*. https://dev.botframework.com/

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. O'Reilly Media.

Jurafsky, D., & Martin, J. H. (2021). *Speech and language processing* (3rd ed.).