Probabilistic Finite-State Chatbot: Combining Stochastic Automata and Language Processing

Authors:

Dean Worrels

Timon Stacy

Roy Ibarra

April 24, 2025

Many modern chatbots are built using finite-state machines, in which each user input triggers a specific, predetermined response. While this method ensures consistency, it often results in rigid and repetitive interactions. This project uses the application of probabilistic automata to create a more flexible and natural user experience. Unlike FSMs, PAs incorporate stochastic transitions, allowing the system to select from a set of potential responses based on a probability, thereby introducing variety and unpredictability. We implemented a chatbot in C++ that processes user input, detects keywords, and navigates through a tree of conversational states. Depending on the matched state, the bot outputs one of several possible responses selected using randomized probabilities. This approach allows the chatbot to mimic a more human-like variability while maintaining a structured response model.

Background and Development

The idea for our chatbot originated from group discussions about the limitations of deterministic automata. We envisioned a chatbot that could offer more dynamic and less predictable responses by incorporating randomness into its state transitions. We began by focusing on the overall structure of the chatbot, deciding how user input would be processed, how states would be connected, and how probabilistic transitions would operate. From there, we moved to implementation in C++, leveraging the language's support for standard libraries and random number generation. A key focus for our chatbot was simulating a typical online customer support service, specifically, in the context of shipping and order tracking. We took inspiration from real-world systems such as Amazon’s automated support tools, which rely on rule-based or AI-driven responses. However, unlike Amazon’s complex natural language models, our chatbot uses a simpler keyword-driven approach that can still provide varied and dynamic responses using probabilistic logic. It mimics the functionality of assisting users with shipping-related questions by responding to phrases like “update my order” or “What’s the status of my delivery?” This gives the chatbot practical value and relevance.

State Structure

This chatbot is written around a custom stateNode structure that defines each conversational state. Each node contains a unique integer key, a state type label such as START0 or QUESTION1, a list of associated keywords, possible responses, and pointers to other nodes to enable tree traversal. This architecture supports both binary tree navigation and flexible child connections. The chatbot initializes its conversation structure using a recursive addToTree function, which inserts nodes into a binary search tree based on their integer keys. Initially, two key states are defined: a “start” state and a “begin” state for handling customer service-related queries. These nodes are configured at runtime through the startProcess function. When a user enters input, the Extract_word function tokenizes the input into individual words. The word_bank function then compares these words against a list of known keywords. If a match is found, the findKeywordState function is used to locate the corresponding node in the tree. This enables the chatbot to determine the appropriate conversational state in response to the user’s query. To ensure the conversations are varied, the chatbot uses a probabilistic response. The output function generates random values to choose among three pre-defined responses within the matched node. This randomness prevents repetitive replies and introduces subtle unpredictability into the conversation.

User Interaction and Functionality

The chatbot responds to user input by recognizing keywords such as “start,” “order,” “update,” and “order number”. For instance, if a user inputs “I want to update my order”, the system transitions to the order-related node and selects a response, such as “What is your order number?” or “Do you need help knowing when your order is going to arrive?”. It accepts full-sentence inputs through getline. The chatbot runs in a loop, ensuring multiple queries can be handled. The chatbot also incorporates a fallback mechanism, when no keywords are detected in the user’s input, it outputs the message "Please re-enter your statement. I got lost trying to understand it.... my creators haven't developed that far.", encouraging the user to rephrase or try again. This will prevent halting and guide the user back to supported keywords.

Monte Carlo Simulations

Monte Carlo simulations were considered to evaluate the chatbot’s stochastic behavior. By simulating numerous interaction paths, we can assess how often each response is used, how varied the conversations are, and whether the state transitions align with the intended probabilities. This approach helps verify that the probabilistic model is functioning as expected and provides randomized outputs.

Deterministic vs Probabilistic Chatbots

Traditional deterministic chatbots always return the same response for a given input, which simplifies control but often leads to repetitive interactions. In contrast, our model leverages stochastic transitions to introduce variability, making conversations feel more dynamic.

Testing and Analysis

We conducted manual testing sessions using various keyword combinations and inputs to evaluate the chatbot's response accuracy and consistency. These tests allowed us to verify that keyword matching was functioning correctly and that fallback messages were triggered appropriately when inputs did not match any defined state. We also evaluated the performance of output generation by logging selected responses over repeated trials. This helped us identify if any particular response was disproportionately favored and provided insight into the fairness of the random selection process.

Conclusion

Through this project, we demonstrated how probabilistic automata can enhance traditional finite-state chatbot models by introducing randomness into state transitions and response generation. By combining structured keyword detection with randomized output, the chatbot achieves more natural responses while maintaining a logical flow. Although limited in scope and currently keyword-based, the chatbot provides a functional foundation for more advanced features. In the future, implementing natural language processing tools, refining transition probabilities, and expanding the states could significantly improve the chatbot's functionality and scalability. These enhancements would allow the bot to handle more complex and natural conversations while retaining its probabilistic structure.

References

“What Is the Monte Carlo Simulation? - The Monte Carlo Simulation Explained - AWS.” Amazon Web Services, Inc., aws.amazon.com/what-is/monte-carlo-simulation.