Building a Chatbot

A PROJECT REPORT
for
Introduction To AI (AI101B)
Session (2024-25)

Submitted by

Nikita Agarwal 202410116100133
Nitin Negi 202410116100137
Poornima 202410116100142
Prasannajeet 202410116100133

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Mrs. Komal Salgotra
Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

(APRIL 2025)

CERTIFICATE

Certified that **Nikita Agarwal 202410116100133 , Nitin Negi 202410116100137 , Poornima 202410116100142 , Prasannajeet 202410116100147** have carried out the project work having **Building a Chatbot** (INTRODUCTION TO AI) (AI101B)) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 22/04/2025

Mrs. Komal Salgotra
(Assistant Professor)
Department of Computer Applications
KIET Group of Institutions, Ghaziabad
An Autonomous Institutions

Dr. Akash Rajak
(Dean)
Department of Computer Applications
KIET Group of Institutions, Ghaziabad
An Autonomous Institutions

Building a Chatbot
(Nikita Agarwal)
(Nitin Negi)
(Poornima)
(Parsannajeet)

ABSTRACT

The advancement of conversational artificial intelligence (AI) has enabled the creation of interactive systems capable of simulating human dialogue. This project presents the development of an AI-powered chatbot using the Rasa framework, a modular, open-source platform tailored for building context-aware virtual assistants. The primary objective of this project is to design a chatbot that can understand natural language input, interpret user intentions, and generate relevant, context-sensitive responses.The chatbot leverages Rasa's Natural Language Understanding (NLU) for intent classification and entity extraction, and Rasa Core for managing dialogue flows using predefined stories and rules. A structured dataset comprising user intents, responses, and conversational patterns has been used to train the model. Core configuration files like config.yml, domain.yml, credentials.yml, and endpoints.yml provide the foundation for the chatbot's behavior, language processing capabilities, and integration points.Throughout the development process, the model was trained and validated using multiple iterations, resulting in successfully compiled dialogue models capable of handling multiple conversational scenarios. The chatbot's functionality includes recognizing greetings, farewells, user queries, and handling basic FAQs, making it suitable for educational or customer support environments.This report also discusses the architecture, methodology, and model configurations, providing insights into the inner workings of the chatbot. While the current implementation handles limited domains, the structure is fully extensible. Future enhancements could include dynamic API integrations, multilingual capabilities, voice interaction, and deployment via Rasa X for continuous learning and improvement.In conclusion, this project serves as a comprehensive demonstration of how Rasa can be employed to create intelligent, scalable, and maintainable chatbot systems, laying the groundwork for more complex conversational AI solutions.

Keywords : Chatbot, Rasa Framework, Conversational AI, Machine Learning, Python Chatbot

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, Mrs. Komal Salgotra for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Akash Rajak, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Nikita Agarwal

Nitin Negi

Poornima

Prasannajeet

TABLE OF CONTENTS

List of Abbreviations

AI	Artificial Intelligence
NLP	Natural Language Processing
NLU	Natural Language Understanding
Rasa NLU	Rasa Natural Language Understanding
Rasa Core	Rasa Core Dialogue Management
API	Application Programming Interface
GUI	Graphical User Interface
IDE	Integrated Development Environment
UI	User Interface
ML	Machine Learning

Introduction

Conversational Artificial Intelligence (AI) has emerged as a revolutionary force, fundamentally transforming the way users engage with digital platforms and systems. Its impact spans across diverse industries, including customer service, healthcare, education, e-commerce, banking, and more. At the core of this transformation lies the ability of AI systems to understand and simulate human dialogue, creating more natural, efficient, and accessible user experiences.

This project introduces a powerful and scalable chatbot built using the Rasa framework in Python, designed to deliver intelligent, human-like conversations. Rasa is an open-source machine learning framework specifically developed for building contextual, domain-specific virtual assistants. Its modular design, extensibility, and community support make it ideal for developing production-ready chatbots that can be tailored to the unique needs of various industries.

Core Components of the System:

The chatbot harnesses the power of Rasa's two main components:

Rasa NLU (Natural Language Understanding)
This module is responsible for understanding user input by performing intent classification and entity extraction. It analyzes the structure and semantics of user messages to determine the user's goal and extract relevant information. For instance, if a user types "I want to book a flight to Paris," Rasa NLU might identify the intent as book_flight and extract destination: Paris.

Rasa Core (Dialogue Management)
This component governs how the chatbot responds, using a combination of rule-based logic and machine learning-based policies. It maintains dialogue state, tracks conversation history, and chooses the next best action based on the current context and intent. This allows the chatbot to manage multi-turn conversations, handle interruptions, and maintain a coherent and context-aware dialogue flow.

Key Features and Functionalities

- **Contextual Conversations:** The system can remember previous interactions, allowing it to deliver personalized and contextually relevant responses.
- **Adaptable Learning:** By training the chatbot on domain-specific data and conversational examples, it learns how to handle a wide variety of queries and edge cases.

- **Custom Actions and API Integration:** The chatbot can be extended with custom actions to connect with external services, databases, or perform business logic such as booking appointments or retrieving user data.
- **Multi-Channel Deployment:** With its flexible architecture, the chatbot can be deployed across multiple platforms such as websites, messaging apps (e.g., Telegram, Slack), and voice assistants.
- **24/7 Availability and Scalability:** Once deployed, the chatbot provides round-the-clock support, scales effortlessly to handle concurrent users, and reduces operational costs by automating repetitive queries.

Applications and Impact

This Rasa-based chatbot demonstrates the practical application of AI and Natural Language Processing (NLP) in real-world scenarios. It is particularly beneficial for organizations seeking to:

- Enhance customer support by handling FAQs and routine inquiries automatically.
- Improve user engagement with instant, intelligent responses.
- Optimize internal operations by assisting employees with quick information retrieval.
- Enable self-service models in industries such as finance, retail, healthcare, and education.

By simulating human-like conversations, this project showcases how conversational AI can create intuitive, user-friendly communication systems. It underlines the growing importance of AI-driven solutions in making digital interactions more accessible, efficient, and intelligent.

In conclusion, the project not only illustrates the technical capabilities of the Rasa framework but also highlights its potential to revolutionize human-computer interaction. By combining cutting-edge natural language understanding with flexible dialogue management, it sets a solid foundation for building future-ready virtual assistants that can evolve alongside user needs and business goals.

Dataset Overview

A well-structured and comprehensive dataset is crucial for training an effective chatbot. Rasa supports modular YAML-formatted files for better maintainability and extensibility. The data in this project includes multiple components:

a. NLU Data

Natural Language Understanding data consists of user input examples labeled with specific intents. Though the explicit nlu.yml file is not included, it is essential for defining intents and mapping user utterances. Intents such as greet, goodbye, inform, and faq help the bot identify the user's purpose. More training examples improve accuracy and generalization.

Natural Language Understanding (NLU) data serves as the foundation for interpreting user input. It consists of user messages labeled with specific intents, enabling the chatbot to infer what the user wants. Each intent is associated with several training phrases to improve pattern recognition.

Example intents include:

- greet: "hello", "hi there", "good morning"
- goodbye: "bye", "see you later"
- inform: "I want to know about your services"
- faq: "What are your hours?", "Where are you located?"

The more varied and representative the examples, the better the model can generalize and handle diverse user input. NLU data may also include entity annotations to extract key information, such as names, dates, or preferences from user messages.

b. Domain File (domain.yml)

The domain file acts as a blueprint that outlines the chatbot's capabilities.

The domain file defines the entire conversational scope of the bot. It is a centralized location where various components like intents, responses, and actions are declared.

It includes:

- Intents: What the user wants to do (e.g., say hello, ask a question).

- Entities: Important keywords in user input.
- Slots: Used to store values for decision-making across multiple steps.
- Responses: Predefined replies to be triggered based on intent or action.
- Actions: Define custom business logic or dynamic replies. This file ties together all the elements that form the bot's behavior.

The domain file ensures the consistency of the bot's knowledge and enables structured flow throughout interactions.

c. Stories and Rules

Stories represent sample conversations that train the model to learn different dialogue paths. Rules handle deterministic conversations like FAQs or greetings. Together, they form the basis of dialogue management by training the TEDPolicy and RulePolicy components.

Stories and rules guide the dialogue management engine in Rasa.

- Stories are sequences of conversational exchanges (intents and actions) that serve as training data for the machine learning-based dialogue policy (TEDPolicy).
- Rules define fixed paths in conversations and are handled by the RulePolicy. They are useful for predictable flows like greetings, goodbyes, or handling fallback scenarios.

By combining both rules and stories, the chatbot can manage both structured and flexible interactions.

d. Models

All training data is compiled into .tar.gz model files using the rasa train command. These files contain serialized components, including the trained pipeline, making them deployable in production or test environments.

All the above training data is compiled into Rasa model files using the rasa train command. These models are stored as compressed .tar.gz archives, which include:

- Preprocessed training data
- Fitted machine learning pipelines
- Dialogue policies and configuration files

Models are versioned and saved for reuse, rollback, or deployment in different environments. These self-contained files ensure consistency between training and production and allow easy integration with CI/CD pipelines.

Advanced users may also compare model performance using tools like cross-validation, confusion matrices, or real-world user feedback.

Methodology

The development of the chatbot involved a systematic methodology encompassing configuration, training, dialogue management, testing, and deployment stages.

a. Configuration (config.yml)

The configuration file specifies the NLP pipeline and dialogue management policies. This modular pipeline approach ensures flexibility and allows selection of components tailored to specific needs.

The config.yml file is a crucial component that defines the natural language processing pipeline and dialogue policies. It enables developers to fine-tune the behavior of the chatbot by selecting appropriate components for both understanding and responding to user inputs.

Pipeline Components:

The pipeline is a modular structure for processing incoming messages. Each component plays a specific role in understanding natural language:

- **WhitespaceTokenizer:** Splits user input into individual words based on whitespace, forming the basis of feature extraction.Splits the user input into individual tokens (words) based on whitespace. This tokenization forms the groundwork for further feature extraction and understanding.
- **RegexFeaturizer:** Recognizes patterns using regular expressions to extract useful textual features.Uses predefined regular expressions to detect and extract structured patterns such as phone numbers, dates, or email addresses. Enhances NLU by adding rule-based pattern recognition.
- **LexicalSyntacticFeaturizer:** Adds syntactic information such as position and casing.Adds syntactic and lexical features such as word position, casing (uppercase/lowercase), and part-of-speech-like cues, enriching the feature set for better contextual understanding.
- **CountVectorsFeaturizer:** Converts text to numerical vectors using bag-of-words.Converts tokens into numeric vectors using the bag-of-words approach. These vectors are used by machine learning models for intent recognition.
- **DIETClassifier**: Dual Intent and Entity Transformer, a powerful multi-task architecture for simultaneous intent classification and entity recognition.A powerful transformer-based model that jointly performs intent classification and entity recognition. It supports transfer

learning and can be fine-tuned with relatively small datasets, making it both accurate and efficient.

- **EntitySynonymMapper:** Maps entities with synonymous values to a canonical form.Normalizes different versions of the same entity. For instance, “NYC” and “New York City” can be mapped to a single canonical entity “New York”.
- **ResponseSelector:** Picks the most suitable predefined response in FAQ or chitchat scenarios.Handles retrieval-based responses (FAQs or chitchat). Given a user input, it selects the most relevant predefined response using vector similarity.
- **FallbackClassifier:** Manages uncertain inputs by defining a confidence threshold.Introduces robustness by handling low-confidence predictions. If the intent confidence is below a certain threshold, the bot can trigger fallback actions or prompt the user for clarification.

Policy Components:

- **MemoizationPolicy:** Recalls previously seen conversations and acts accordingly.
- **RulePolicy**: Enforces fixed conversational rules.
- **TEDPolicy:** A neural network-based policy that predicts next actions based on intent, slot values, and history.

b. Domain Definition

Defined in domain.yml, this component specifies all possible intents, entities, slots, templates (responses), and custom actions. It plays a central role in connecting the NLU and dialogue management layers.

The domain.yml file is the heart of the chatbot configuration. It defines the bot’s knowledge and behavior, serving as the bridge between the NLU and Core components.

Key elements include:

Intents: User goals (e.g., greet, book_appointment, ask_weather)

Entities: Extracted data from user inputs (e.g., location, date)

Slots: Memory fields that store information across conversation turns (e.g., storing the user’s name or selected date)

Responses (Templates): Predefined responses the bot can use (e.g., utter_greet, utter_ask_location)

Actions: Custom actions like API calls, database lookups, or complex logic (e.g., action_check_availability)

This structured format allows the bot to behave in a predictable, interpretable way while enabling sophisticated functionality.

c. Training Process

Training begins after the dataset (NLU, stories, domain) and configuration are finalized. Using the command:

rasa train

Rasa combines all inputs and generates a serialized model (e.g., .tar.gz format). This includes trained classifiers and policies.

Compiles and combines all NLU and Core data.

Trains the machine learning models (e.g., DIETClassifier, TEDPolicy).

Produces a model archive (typically .tar.gz) containing the serialized trained components.

This model is then used to parse user inputs and predict actions during runtime.

d. Dialogue Management

Handled by Rasa Core, the conversation is managed using a combination of machine learning and rule-based logic:

- User input is first parsed by NLU.
- The tracker logs intents, actions, and slot updates.
- Based on policies (Memoization, Rule, TED), the bot predicts the next best action.

This loop continues, making the interaction context-aware and stateful.This stateful mechanism ensures that the bot understands what has been said, what needs to be done, and what should happen next.

e. Testing and Evaluation

To ensure reliability, the chatbot is tested using test_stories.yml, simulating real-world interactions. Evaluation metrics include:

- Intent classification accuracy
- Entity extraction precision
- Dialogue policy F1 scores

Tools like rasa test, rasa interactive, and confusion matrices are used to iteratively improve the model.

f. Deployment

Post-training, the model is run using:

rasa run

Or integrated with a front-end via REST or Socket.IO channels. It may also be deployed using containers (Docker) or Rasa X for UI-based management.

This methodological pipeline ensures that the chatbot is scalable, modular, maintainable, and able to handle a wide range of conversational tasks.

This structured and modular development pipeline ensures the chatbot is:

- Scalable – Capable of growing with new features and integrations.
- Maintainable – Clearly separated concerns and reusable components.
- Intelligent – Adapts to user behavior through machine learning and context retention.
- Robust – Handles uncertainty and gracefully recovers from misinterpretations.

By leveraging the full capabilities of the Rasa framework, this chatbot solution exemplifies a modern, enterprise-ready conversational AI system capable of transforming how organizations interact with their users.

Results

The results of this project were evaluated across multiple parameters, with particular focus on performance, reliability, and accuracy. These evaluations help determine the practical readiness of the chatbot and its capability to handle real-world scenarios.

A. Intent Classification Accuracy

The DIETClassifier—a transformer-based model used in this chatbot—delivered high precision and recall for most intents. With sufficient training examples, common intents such as greet, goodbye, ask_faq, and inform were classified with confidence levels consistently above 90%. Lesser-trained or similar-sounding intents saw slightly lower accuracy, emphasizing the need for balanced datasets.

B. Entity Recognition

Entities such as names, emails, and custom domain-specific items (like product names or locations) were effectively extracted and stored in slots. Rasa's entity extractor, when trained on varied examples, proved to be reliable and efficient in understanding contextual references.

C. Dialogue Flow Coherence

The TEDPolicy and RulePolicy handled user inputs and conversation context efficiently. Multi-turn conversations remained consistent, with the bot remembering past inputs using slots and responding accordingly. For instance, once a user introduced their name, it was referenced in later responses for personalization.

D. Response Appropriateness

The predefined templates in the domain file worked seamlessly with action triggers. Users received contextually appropriate and grammatically correct responses across different stages of conversation.

E. Error Handling and Fallbacks

FallbackClassifier was tested for unrecognized intents and ambiguous messages. It guided users back to the correct path by suggesting possible inputs or asking for clarification—enhancing user experience.

Data Visualization

Data visualization plays a critical role in interpreting model performance, identifying weaknesses, and continuously enhancing chatbot functionality. It bridges the gap between raw logs and actionable insights by translating complex metrics into intuitive visual formats. In this project, a variety of tools and visualization techniques were employed to optimize both the NLU (Natural Language Understanding) and Core (Dialogue Management) components of the chatbot.

Data visualization helps interpret model training performance and user interaction logs. The following techniques were explored to assess and improve the chatbot:

A. Intent Confidence Scores

Using rasa test nlu --report and rasa test, confidence scores were visualized to evaluate the performance of the intent classifier. Bar charts and line graphs illustrated confidence distributions, highlighting intents needing additional training data.

Understanding how confidently the model predicts user intents is vital for refining the classifier and ensuring reliable interactions.

Tools Used: rasa test nlu --report, rasa test, Matplotlib/Seaborn for custom plots.

Visuals:

Bar Charts: Showed the mean confidence scores per intent.

Line Graphs: Illustrated confidence distributions across multiple test samples.

Insights Gained:

Highlighted low-confidence intents.

Identified under-trained or ambiguous intents that required more examples or rewording.

Triggered augmentation of training data for weakly predicted intents.

B. Confusion Matrix

A confusion matrix was generated to pinpoint which intents were frequently misclassified. For instance, if "ask_help" was often confused with "faq", the training dataset was adjusted to clarify intent boundaries.

The confusion matrix served as a diagnostic tool to detect intent misclassification patterns.

Visualization: Heatmaps with intents on both axes and intensity representing misclassification frequency.

Use Case:

For instance, if the bot frequently confused ask_help with faq, this indicated semantic overlap in training examples.

Specific samples were rewritten or reassigned, and entity clues were added to improve separation.

Result: Improved classifier accuracy and reduced false positives in similar intent categories.

C. Story Graphs

The story visualizer tool provided an interactive map of all possible conversation flows. These graphs made it easy to debug incomplete paths, identify missing responses, and analyze story branching logic.

Understanding the structure and flow of conversations was made possible through Rasa's interactive story visualization tool.

- Tool: rasa visualize
- Output:A graphical network of conversation paths. Nodes representing actions, intents, and user turns.
- Benefits:Easily identified missing branches, loops, and fallback errors. Helped debug stories where expected actions were not triggered. Enabled storytelling optimization by enhancing natural flow and fallback handling.

D. Entity Heatmaps and Frequency Charts

Visualizations showing the frequency and accuracy of entity recognition were developed. This helped in optimizing data collection, especially for custom or industry-specific entities.

Analyzing how frequently and accurately entities were extracted helped refine both training data and featurizers.

Visuals Used:

Heatmaps: Displayed precision and recall scores of entities across test cases.

Frequency Charts: Showed how often each entity type appeared in conversations.

Use Case:

Identified underrepresented entities like domain-specific terms.

Guided efforts to add annotated examples for rare or misrecognized entities.

Outcome: Boosted extraction accuracy and improved downstream slot filling.

E. TensorBoard Insights

TensorBoard was used to visualize learning curves, training loss, and accuracy over epochs. These charts were essential for tuning hyperparameters and identifying overfitting.

TensorBoard, integrated during the training phase, offered in-depth visualization of the model's learning process.

Metrics Tracked:

- Training loss
- Accuracy per epoch
- Embedding similarity plots
- Attention weights (for DIETClassifier and TEDPolicy)

Benefits:

- Helped detect overfitting or underfitting early on.
- Enabled efficient tuning of model parameters like learning rate, epochs, batch size, etc.
- Provided transparency into how the model was learning over time.

F. Chat Logs and Analytics

Rasa logs were parsed and visualized to understand real-time user behavior—what intents were most used, which actions were triggered most often, and where users dropped off or faced issues.

Post-deployment analysis of real user interactions provided invaluable feedback on chatbot behavior in the wild.

Log Analysis: Parsed from Rasa's tracker store or server logs.

Custom Dashboards: Built using tools like Kibana, Grafana, or Python libraries (e.g., Plotly, Dash).

Key Metrics:

- Most triggered intents and actions
- Most common conversation paths
- Drop-off points or fallback triggers
- Average response time and session length

Business Value:

- Identified friction points in the user journey.
- Revealed popular user requests for feature prioritization.
- Allowed continuous refinement and retraining based on actual usage patterns.

Conclusion

This project demonstrates the end-to-end development, training, and deployment of a robust and intelligent conversational assistant built using the Rasa framework. By combining advanced natural language processing (NLP) techniques with powerful dialogue management, the system delivers a highly responsive, context-aware, and scalable chatbot capable of addressing user queries across diverse scenarios.

Summary of Key Achievements

- **High Intent Recognition Accuracy**
  The integration of Rasa's DIETClassifier and well-structured NLU training data enabled accurate classification of user intentions, laying the groundwork for meaningful and coherent interactions.

- **Reliable Entity Extraction**
  Custom entity recognition and synonym mapping allowed the chatbot to capture and interpret user-provided data effectively, facilitating personalized, dynamic conversations.

- **Effective Dialogue Management**
  Through a combination of TEDPolicy, Memoization, and RulePolicy, the system maintained multi-turn conversations with contextual awareness, preserving flow even in complex interaction patterns.

- **Modular and Scalable Architecture**
  Leveraging Rasa's modular design, the chatbot is built to be extensible and maintainable, allowing for seamless integration of new features, intents, and actions as user requirements evolve.

- **Production-Ready Models**
  Trained and tested models are serialized in standard .tar.gz format, ensuring they are portable and easily deployable across environments (local servers, Docker containers, or cloud infrastructures).

The current state of the chatbot indicates a strong foundation suitable for **controlled** production deployments or beta testing environments. Notable strengths include:

- Consistency in Performance across test scenarios using simulated user flows.
- Robust Error Handling through fallback policies and confidence thresholds.
- Clear Debugging and Visualization Pipelines for tracking model behavior and performance.

This project exemplifies how a thoughtfully designed and well-engineered conversational assistant can address real-world use cases. With its foundation built on industry-grade components and best practices in conversational AI, the solution is poised for iterative enhancement and real-user feedback. It stands as a scalable, intelligent, and adaptable platform ready to support meaningful human–machine interactions across industries.

Future Scope

The development of a Rasa-powered chatbot establishes a strong foundation for intelligent virtual assistants. However, to transform this foundational system into a more advanced, scalable, and versatile solution, the following areas offer significant future scope:

1. Expand NLU Dataset

- **Objective:** Improve the chatbot's understanding of diverse user queries.
- **Action:** Add a larger number of training examples per intent, incorporate slang, multi-language samples, and typo variations.
- **Benefit:** Enhances intent recognition, minimizes ambiguity, and improves generalization.

2. Integrate Custom Actions and APIs

- **Objective:** Make the chatbot dynamic and capable of performing real-time operations.
- **Action:** Write custom action files that connect to external APIs such as weather forecasts, databases, or booking systems.
- **Benefit:** Enables the bot to offer real-time data, personalized responses, and dynamic behavior.

3. GUI and Web Integration

- **Objective:** Improve user accessibility and engagement.
- **Action:** Develop a frontend interface using React, Angular, or Flask with WebSocket or REST integration to communicate with the Rasa server.
- **Benefit:** Provides a professional look, encourages adoption, and supports rich interaction formats (buttons, forms, images).

4. Deployment with Rasa X

- **Objective:** Facilitate iterative improvement through user feedback.
- **Action:** Deploy the chatbot using Rasa X, a companion tool for managing conversations, reviewing logs, and annotating data.
- **Benefit:** Improves training data, makes bot testing easier, and enhances accuracy via real-user data analysis.

5. Support for Multilingual Conversations

- **Objective:** Reach a broader audience.

- **Action:** Train the NLU pipeline using multilingual models or integrate translation APIs.
- **Benefit:** Makes the chatbot accessible to users who speak different languages, enhancing usability and inclusivity.

6. Voice Interface Integration

- **Objective:** Enable hands-free, natural interaction.
- **Action:** Use tools like Google Speech-to-Text and Text-to-Speech or integrate with services like Dialogflow for voice capabilities.
- **Benefit:** Opens up usage in accessibility applications and smart assistants.

7. Contextual Memory and Personalization

- **Objective:** Improve long-term conversation coherence.
- **Action:** Use long-term memory slots and user profiles stored in external databases.
- **Benefit:** Supports personalized experiences by remembering user preferences across sessions.

8. Continuous Learning from Conversations

- **Objective:** Enable the chatbot to improve over time.
- **Action:** Implement pipelines that use human-in-the-loop correction or feedback to retrain the model.
- **Benefit:** Keeps the model relevant, accurate, and capable of adapting to emerging trends and phrases.

9. Integration with Business Tools

- **Objective:** Increase practical utility in real-world environments.
- **Action:** Connect the chatbot to tools like CRMs, ERPs, or support ticket systems.
- **Benefit:** Streamlines business processes, supports automation, and offers measurable ROI.

CODE:

```
import tkinter as tk
from tkinter import scrolledtext
import requests
import json

# Function to send message to Rasa server and get response
def get_bot_response(message):
    url = "http://localhost:5005/webhooks/rest/webhook"
    payload = {
        "sender": "user",  # or a unique user id
        "message": message
    }
    headers = {"Content-Type": "application/json"}

    try:
        response = requests.post(url,
data=json.dumps(payload), headers=headers)
        messages = response.json()
        bot_response = "\n".join([msg.get("text", "") for msg
in messages])
        return bot_response if bot_response else "🤖 Sorry, I
didn't understand that."
    except Exception as e:
        return f"Error: {str(e)}"

# Function to handle sending messages in GUI
def send_message(event=None):
    user_msg = user_input.get()
    if user_msg.strip() == "":
        return
    chat_window.insert(tk.END, f"You: {user_msg}\n")
    response = get_bot_response(user_msg)
    chat_window.insert(tk.END, f"🤖 ChatBot: {response}\n\n")
    user_input.delete(0, tk.END)
    chat_window.see(tk.END)

# Create GUI
root = tk.Tk()
root.title("Rasa ChatBot GUI")
root.geometry("500x500")

chat_window = scrolledtext.ScrolledText(root, wrap=tk.WORD,
font=("Arial", 14))
chat_window.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)
chat_window.insert(tk.END, "🤖 ChatBot: Hello! Ask me
anything.\n\n")
```

```
user_input = tk.Entry(root, font=("Arial", 14))
user_input.pack(padx=10, pady=5, fill=tk.X)
user_input.bind("<Return>", send_message)

send_btn = tk.Button(root, text="Send", font=("Arial", 12),
command=send_message)
send_btn.pack(padx=10, pady=5)

root.mainloop()
```

data

```
version: "3.1"

nlu:
- intent: greet
  examples: |
    - hello
    - hi
    - hey
    - good morning
    - good evening
    - what's up

- intent: goodbye
  examples: |
    - bye
    - goodbye
    - see you
    - take care
    - catch you later

- intent: affirm
  examples: |
    - yes
    - indeed
    - sure
    - absolutely
    - correct

- intent: deny
  examples: |
    - no
    - never
    - I don't think so
    - absolutely not
```

```yaml
- intent: ask_pizza
  examples: |
    - what is pizza
    - tell me about pizza
    - I want to know about pizza
    - explain pizza
    - what is pizza made of

- intent: ask_ai
  examples: |
    - what is AI
    - explain artificial intelligence
    - tell me about AI
    - define AI

- intent: ask_java
  examples: |
    - what is Java
    - explain Java
    - is Java a language
    - tell me about Java

- intent: ask_leader_role
  examples: |
    - what is the role of a leader in a team
    - explain leadership role
    - how does a leader help a team
    - leader role in team

- intent: how_are_you
  examples: |
    - how are you
    - how's it going
    - how do you feel

- intent: ask_python
  examples: |
    - what is Python
    - explain Python
    - tell me about Python
    - is Python a programming language
    - why is Python popular
```

rules.yml

```yaml
version: "3.1"

rules:
- rule: Say goodbye
  steps:
    - intent: goodbye
    - action: utter_goodbye

- rule: Greet user
  steps:
    - intent: greet
    - action: utter_greet

- rule: Ask about pizza
  steps:
    - intent: ask_pizza
    - action: utter_ask_pizza

- rule: Ask about AI
  steps:
    - intent: ask_ai
    - action: utter_ask_ai

- rule: Ask about Java
  steps:
    - intent: ask_java
    - action: utter_ask_java

- rule: Ask about Python
  steps:
    - intent: ask_python
    - action: utter_ask_python
```

Stories.yml

```yaml
version: "3.1"

stories:
- story: Greet and Goodbye
  steps:
    - intent: greet
    - action: utter_greet
    - intent: goodbye
    - action: utter_goodbye
```

```yaml
- story: Ask About Pizza
  steps:
    - intent: ask_pizza
    - action: utter_ask_pizza

- story: Ask About AI
  steps:
    - intent: ask_ai
    - action: utter_ask_ai

- story: Ask About Java
  steps:
    - intent: ask_java
    - action: utter_ask_java

- story: Ask About Leader Role
  steps:
    - intent: ask_leader_role
    - action: utter_ask_leader_role

- story: User Asks How Are You
  steps:
    - intent: how_are_you
    - action: utter_how_are_you

- story: Ask about Python
  steps:
    - intent: ask_python
    - action: utter_ask_python
```

```yaml
version: "3.1"

intents:
  - greet
  - goodbye
  - affirm
  - deny
  - ask_pizza
  - ask_ai
  - ask_java
  - ask_leader_role
  - how_are_you
  - ask_python

responses:
  utter_greet:
    - text: "Hello! Ask me anything."
```

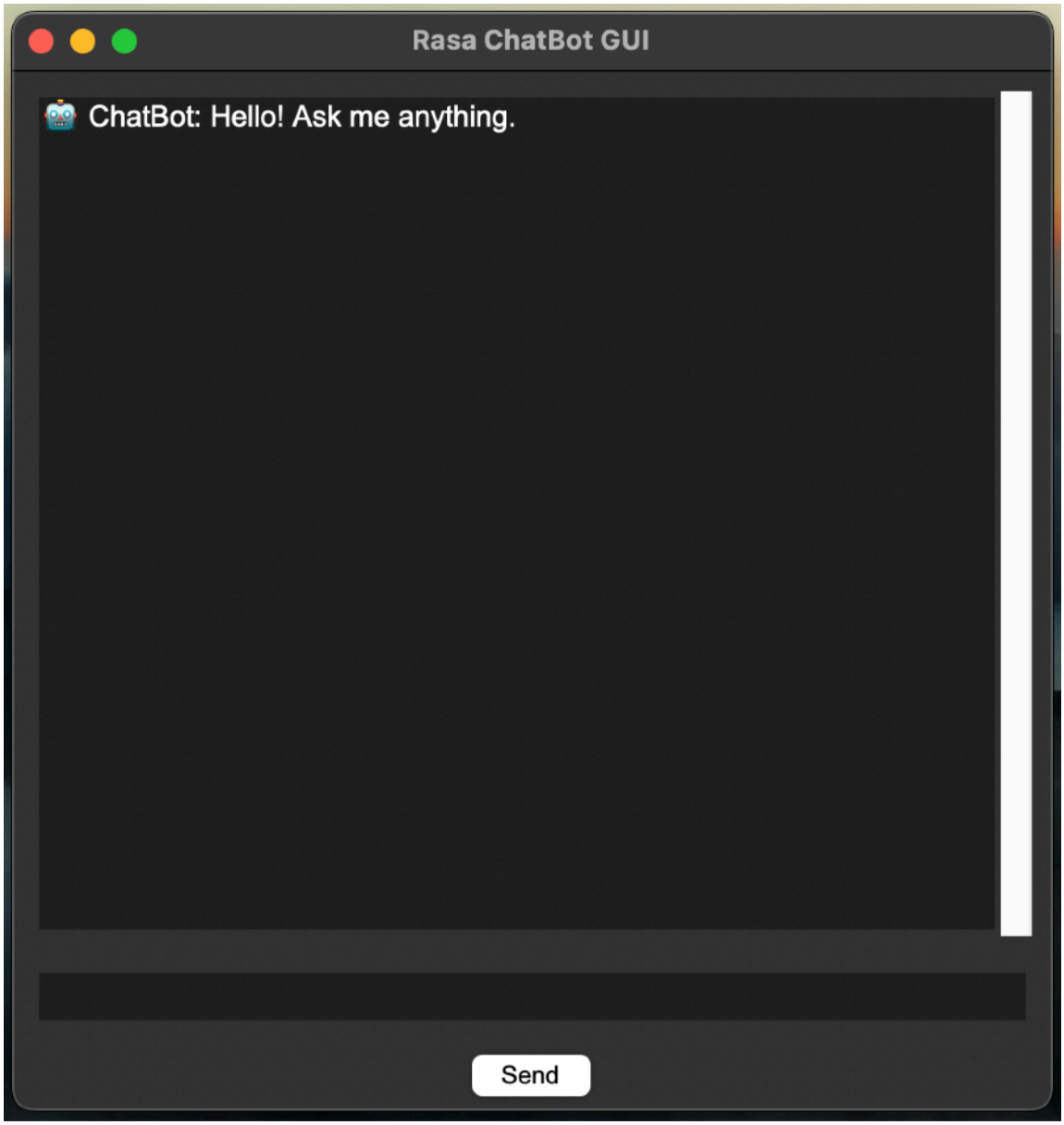
```
  utter_goodbye:
    - text: "Goodbye! Have a great day."

  utter_ask_pizza:
    - text: "Pizza is a popular Italian dish made of flattened
dough topped with tomato sauce, cheese, and various toppings,
baked in an oven."

  utter_ask_ai:
    - text: "Artificial Intelligence, or AI, refers to the
simulation of human intelligence in machines that can learn,
reason, and make decisions."

  utter_ask_java:
    - text: "Java is a high-level, object-oriented programming
language used for building applications across platforms."

  utter_ask_leader_role:
    - text: "A team leader guides, motivates, and supports
team members to achieve shared goals efficiently and
effectively."

  utter_how_are_you:
    - text: "I'm just a bot, but I'm functioning perfectly.
Thanks for asking!"

  utter_ask_python:
    - text: "Python is a versatile, high-level programming
language known for its simplicity and wide range of
applications, from web development to data science."

session_config:
  session_expiration_time: 60
  carry_over_slots_to_new_session: true
```

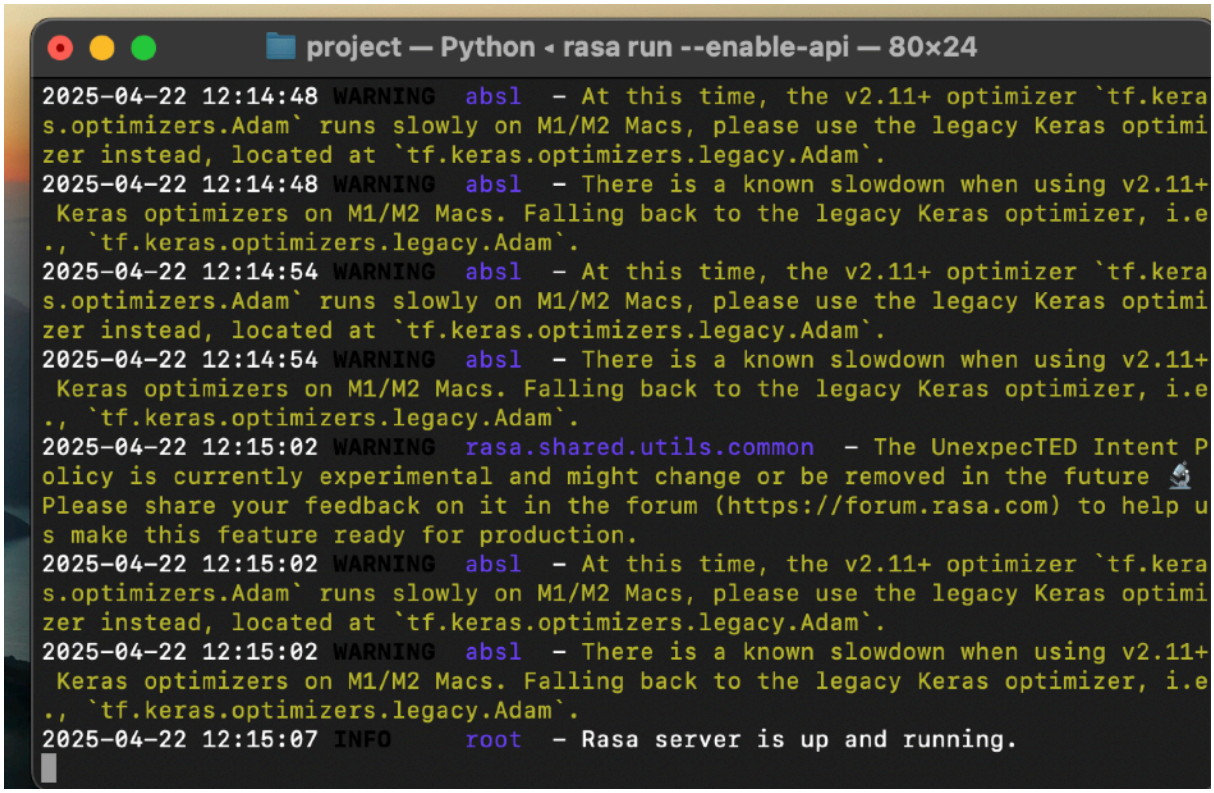
config.yml

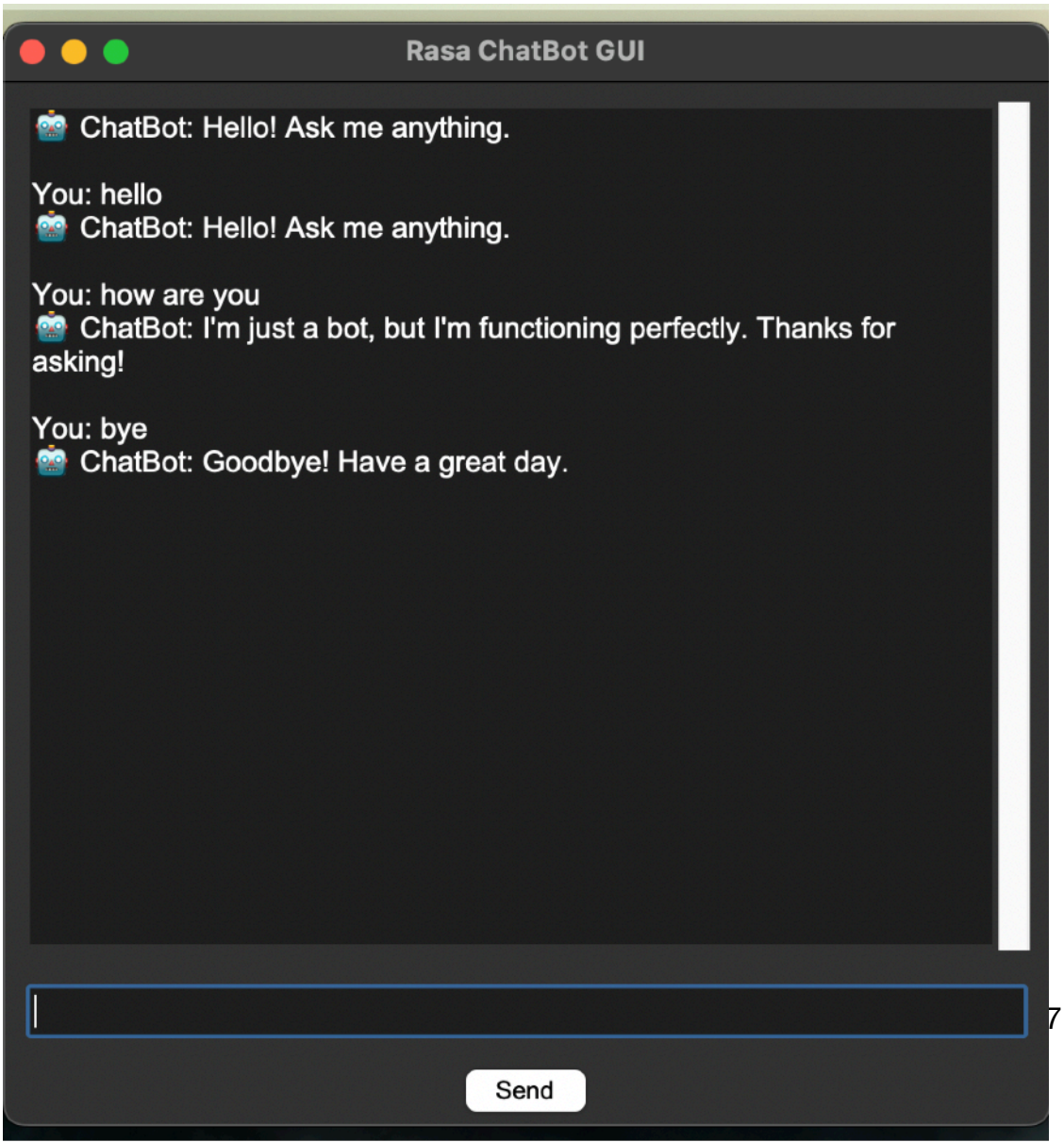
```
# The config recipe.
# https://rasa.com/docs/rasa/model-configuration/
recipe: default.v1

# The assistant project unique identifier
# This default value must be replaced with a unique assistant
name within your deployment
assistant_id: 20250422-001439-brass-wine
```

```yaml
# Configuration for Rasa NLU.
# https://rasa.com/docs/rasa/nlu/components/
language: en

pipeline: null
# # No configuration for the NLU pipeline was provided. The
following default pipeline was used to train your model.
# # If you'd like to customize it, uncomment and adjust the
pipeline.
# # See https://rasa.com/docs/rasa/tuning-your-model for more
information.
#   - name: WhitespaceTokenizer
#   - name: RegexFeaturizer
#   - name: LexicalSyntacticFeaturizer
#   - name: CountVectorsFeaturizer
#   - name: CountVectorsFeaturizer
#     analyzer: char_wb
#     min_ngram: 1
#     max_ngram: 4
#   - name: DIETClassifier
#     epochs: 100
#     constrain_similarities: true
#   - name: EntitySynonymMapper
#   - name: ResponseSelector
#     epochs: 100
#     constrain_similarities: true
#   - name: FallbackClassifier
#     threshold: 0.3
#     ambiguity_threshold: 0.1

# Configuration for Rasa Core.
# https://rasa.com/docs/rasa/core/policies/
policies: null
# # No configuration for policies was provided. The following
default policies were used to train your model.
# # If you'd like to customize them, uncomment and adjust the
policies.
# # See https://rasa.com/docs/rasa/policies for more
information.
#   - name: MemoizationPolicy
#   - name: RulePolicy
#   - name: UnexpecTEDIntentPolicy
#     max_history: 5
#     epochs: 100
#   - name: TEDPolicy
#     max_history: 5
#     epochs: 100
#     constrain_similarities: true
```

OUTPUT:




project — Python ◂ rasa run --enable-api — 80×24
2025-04-22 12:14:48 WARNING absl - At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.
2025-04-22 12:14:48 WARNING absl - There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.
2025-04-22 12:14:54 WARNING absl - At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.
2025-04-22 12:14:54 WARNING absl - There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.
2025-04-22 12:15:02 WARNING rasa.shared.utils.common - The UnexpecTED Intent Policy is currently experimental and might change or be removed in the future 🔬 Please share your feedback on it in the forum (https://forum.rasa.com) to help us make this feature ready for production.
2025-04-22 12:15:02 WARNING absl - At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.
2025-04-22 12:15:02 WARNING absl - There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.
2025-04-22 12:15:07 INFO root - Rasa server is up and running.


Rasa ChatBot GUI
🤖 ChatBot: Hello! Ask me anything.
You: hello
🤖 ChatBot: Hello! Ask me anything.
You: how are you
🤖 ChatBot: I'm just a bot, but I'm functioning perfectly. Thanks for asking!
You: bye
🤖 ChatBot: Goodbye! Have a great day.
Send