Summer Internship Report
on
"Helpdesk Chatbot in Rasa using various Artificial Intelligence Language Models"

Submitted in partial fulfillment of the Requirement for the award of the degree of

Bachelor of Engineering

in

Computer Science and Engineering

by

MOHAMMED MISBAHUDDIN 1608-20-733-095

Under the guidance of

Mrs. B. J. Praveena
Assistant Professor

Department of Computer Science and Engineering

MATRUSRI ENGINEERING COLLEGE

FEBRUARY 2024

MATRUSRI ENGINEERING COLLEGE

SAIDABAD - 500059

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the internship report entitled **"Helpdesk Chatbot in Rasa using various Artificial Intelligence Language Models"** submitted by **Mr. Mohammed Misbahuddin** bearing **H.T. No: 1608-20-733-095,** in the partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Science and Engineering** is a bonafide work carried by him/her**.**

The results of the investigations enclosed in this report have been verified and found satisfactory.

Internal Supervisor
Dept. of CSE
MECS.

HOD
Dept. of CSE
MECS.

दूरभाष/Telephone:040-24188039/8062
फैक्स/Fax :040-24344457/2488062
ई-मेल /E-mail: hrdg.asl@gov.in

भारत सरकार, रक्षा मंत्रालय
Government of India, Ministry of Defence
रक्षा अनुसंधान एवं विकास संगठन
Defence Research & Development Organisation
उन्नत प्रणाली प्रयोगशाला
ADVANCED SYSTEMS LABORATORY
डॉ. ए.पी.जे. अब्दुल कलाम प्रक्षेपास्त्र समष्टि
Dr. A.P.J. Abdul Kalam Missile Complex
कंचनबाग डाकघर, हैदराबाद - 500 058
PO Kanchanbagh, Hyderabad - 500 058

CERTIFICATE

This is to certify that **Mr. Mohammed Misbahuddin** (Roll No. 1608-20-733-095), B.E III year (CSE) from **Matrusri Engineering College**, Saidabad, Hyderabad has successfully completed internship on "**Helpdesk Chatbot in Rasa using various Artificial Intelligence Language Models**" at Advanced Systems Laboratory, DRDO, Hyderabad under the guidance of **Sri. Biplab Mandal, Sc 'D'** from **5th May to 6th June 2023**

This Certificate is issued by this Laboratory after completion of internship as a part of their Educational Curriculum. No stipend / Salary is entitled by the candidate during the course of the internship. The Certificate cannot be submitted as a documentary proof for claiming the internship duration as service experience for any of their future employment / recruitment.

His performance and conduct during the period was good.

(HEMANT KUMAR)
SCIENTIST 'E'
HRDG, ASL
FOR DIRECTOR, ASL

हेमंत कुमार / HEMANT KUMAR
प्रधान, एच.आर.डी.जी. / Head, HRDG
ए.एस.एल., हैदराबाद-58. / ASL, Hyderabad-58.

Biplab Mandal

(BIPLAB MANDAL)
SCIENTIST 'D'
CSeG / ASL
EXTERNAL GUIDE

ACKNOWLEDGEMENT

This internship consumed huge amount of work, research and dedication. Still implementation would not have been possible if I did not had support of my Internship Guide, Internship Coordinator, Head of the Department and Principal. Therefore, I like to extend my sincere gratitude to all of them.

I wish to express my gratitude to internal internship supervisor **Mrs. B. J. Praveena**, Assistant Professor and internship coordinator for their indefatigable inspiration, constructive criticisms, advices and encouragement throughout this dissertation work.

I am grateful to my external internship supervisor **Mr. Biplab Mandal**, Sc 'D', ASL Lab, DRDO for provision of expertise, technical support and guidance in the implementation of the internship project.

I would like to express my sincere thanks to the Professor and Head of the Department, **Dr. P. Vijaya Pal Reddy**, for permitting me to do this internship.

I would like to express my gratitude to **Dr. D. Hanumantha Rao**, principal of Matrusri Engineering College who permitted to carry out this project as per the academics.

I would like to thank CSE Department for providing me this opportunity to share and contribute my part of work to accomplish the internship in time and all the teaching and support staff for their steadfast support and encouragement.

Nevertheless, I express my gratitude towards my family and colleagues for their kind cooperation and encouragement which helped me in completion of this internship.

Mohammed Misbahuddin (1608-20-733-095)

ABSTRACT

The project aims to develop an AI helpdesk system (chatbot) which can answer to the user’s questions or queries with the available knowledge. The chatbot will be designed to provide quick and accurate responses to common questions and issues faced by users, such as technical support, customer support, HR support, and more.

The system will be developed using natural language understanding (NLU) framework called RASA which can understand and respond to user queries in a conversational manner using natural language processing (NLP) and advance machine learning techniques. Enhancing the chatbot's semantic accuracy will be achieved through the utilization of various language models provided by Rasa. The chatbot can be integrated into an existing platform or website and will be able to access a knowledge base containing information relevant to the user's queries.

The project will involve the design and development of the chatbot's user interface, the development of knowledge for the chatbot and integration of all components using RASA. The end result will be an AI helpdesk system or chatbot that provides users with efficient and effective support, improving user satisfaction and reducing the workload on support staff.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In today's digital era, the demand for efficient and user-centric support services has propelled the development of innovative solutions leveraging artificial intelligence (AI) technologies. As businesses strive to meet the diverse needs of their customers, employees, and stakeholders, the significance of providing real-time assistance and streamlining customer interactions responsive and effective helpdesk solutions cannot be overstated. Traditional support channels, while valuable, often struggle to keep pace with the growing volume and complexity of user inquiries, leading to prolonged response times, frustration, and diminished user satisfaction.

To address these challenges and streamline the support process, the development of AI-powered helpdesk systems has emerged as a promising solution. Leveraging advancements in natural language processing (NLP), machine learning (ML), and conversational AI, these systems offer a transformative approach to customer service, enabling organizations to deliver real-time assistance and personalized support experiences at scale. In particular, the RASA framework has garnered attention for its robust natural language understanding (NLU) capabilities and flexibility in building conversational AI systems.

This project aims to harness the power of RASA framework along with diverse language models to develop an advanced AI-driven helpdesk solution. By integrating various artificial intelligence language models within the RASA framework, the chatbot will be equipped to understand and respond to user inquiries with unparalleled accuracy and efficiency.

The RASA framework, renowned for its open-source nature and customizable architecture, serves as the cornerstone of this project. With its comprehensive suite of tools for natural language processing (NLP) and machine learning (ML), RASA empowers developers to build sophisticated conversational AI systems tailored to specific use cases.

At the core of the RASA framework lies its NLU component, which enables the chatbot to comprehend the nuances of human language and extract key information from user queries. Through the integration of diverse language models, such as BERT, GPT, and Transformer-based models, the chatbot gains access to a vast repository of linguistic knowledge, enabling it to interpret user intent with exceptional accuracy and provide contextually relevant responses.

By leveraging these AI language models within the RASA framework, the chatbot transcends traditional rule-based approaches, evolving into a dynamic and adaptive

conversational agent capable of engaging users in natural, human-like interactions. Moreover, the modular architecture of RASA facilitates seamless integration of these language models, allowing developers to experiment with different models and select the most suitable ones based on performance and use case requirements.

Moreover, by integrating seamlessly into existing platforms or websites, the chatbot will serve as a readily accessible resource for users, empowering them to obtain assistance and resolve issues with minimal effort and friction. Through continuous learning and refinement, the system will evolve to anticipate user needs, adapt to changing circumstances, and deliver increasingly sophisticated levels of assistance over time.

In essence, this project represents a convergence of cutting-edge technologies—RASA framework and AI language models—to create a sophisticated helpdesk chatbot that redefines the boundaries of customer support. By harnessing the power of AI-driven conversation, the chatbot aims to enhance user experience, streamline support operations, and drive tangible business value for organizations across diverse industries.

2. BACKGROUND KNOWLEDGE

2.1 Evolution of Chatbots:

Chatbots have traversed a fascinating journey, reflecting the advancements in AI and natural language processing (NLP) technologies. Initially, chatbots relied on rule-based systems, where they followed a predefined set of rules to generate responses based on keywords detected in user queries. These early chatbots were limited in their ability to understand the nuances of human language and often provided rigid and impersonal responses.

However, with the advent of AI and machine learning, chatbots evolved to incorporate more sophisticated techniques such as pattern recognition and semantic analysis. These advancements enabled chatbots to discern user intent more accurately, leading to more meaningful interactions and improved user satisfaction.

In recent years, chatbots have leveraged neural networks and deep learning algorithms to achieve even greater levels of sophistication. By training on vast amounts of conversational data, modern chatbots can generate responses that are contextually relevant and exhibit a more natural conversational flow, akin to human interaction.

2.2 RASA Framework:

RASA stands out as a leading open-source framework for building conversational AI applications. Its popularity stems from its versatility, allowing developers to create custom chatbots tailored to specific use cases. Here's a closer look at some key aspects of the RASA framework:

- **Ease of Use:** RASA offers a user-friendly interface and comprehensive documentation, making it accessible to developers with varying levels of experience. Its modular architecture facilitates easy customization and integration with existing systems, reducing development time and complexity.
- **Scalability:** RASA is designed to scale seamlessly, enabling developers to deploy chatbots across multiple channels and handle large volumes of user interactions without compromising performance. This scalability ensures that chatbots can meet the demands of growing user bases and evolving business needs.
- **Performance:** RASA's advanced NLU capabilities enable chatbots to understand user intent with remarkable accuracy. By leveraging machine learning algorithms, RASA

continuously learns from user interactions, refining its understanding of language patterns and improving response quality over time.

- **Core Features:** RASA offers a rich set of features for building conversational AI applications, including intent classification, entity extraction, dialogue management, and integration with external knowledge bases and APIs. These features empower developers to create chatbots that can engage users in meaningful conversations and provide valuable assistance across various domains.

2.3 Language Models in RASA:

RASA supports integration with a range of language models, including state-of-the-art models like BERT and GPT. These language models enhance the chatbot's ability to understand and generate natural language responses, contributing to a more seamless and human-like conversational experience.

- **BERT (Bidirectional Encoder Representations from Transformers):** BERT is a pre-trained language model developed by Google, known for its ability to capture bidirectional context in text data. By integrating BERT into RASA, chatbots can analyze the context of user queries more effectively, leading to more accurate intent classification and entity extraction.
- **GPT (Generative Pre-trained Transformer):** GPT, developed by OpenAI, is renowned for its ability to generate human-like text based on the provided input. Integrating GPT into RASA enables chatbots to produce responses that are not only contextually relevant but also exhibit a more natural and conversational tone, enhancing the overall user experience.

2.4 Helpdesk System via Chatbots and Knowledge Bases:

A helpdesk system via chatbots harnesses AI technologies to provide users with timely and personalized support for their inquiries and issues. These chatbots are equipped with access to knowledge bases containing a wealth of information, including FAQs, troubleshooting guides, and company policies. By leveraging this knowledge, chatbots can deliver relevant solutions to user queries, reducing the need for human intervention and improving overall efficiency.

2.5 Integration of Helpdesk Systems:

Integrating helpdesk systems into existing platforms, websites, and applications is essential for providing seamless access to support services for users. By embedding chatbots within these environments, organizations can ensure that users can easily access assistance whenever they need it, without having to navigate to a separate support portal. This integration enhances user experience, streamlines support operations, and reduces the burden on human agents, ultimately leading to higher levels of customer satisfaction and retention.

3. INTERNSHIP OBJECTIVES AND GOALS

The objective and goals of the internship includes the activities that are required to develop the project as per project requirements, specifications and functionalities.

1. **Understanding Chatbots and Helpdesk Systems:**
   - Gain comprehensive knowledge about the principles and functioning of chatbots, including their evolution and different types.
   - Explore the architecture and components of helpdesk systems built via chatbots, understanding their role in enhancing customer support.
2. **Designing User-Friendly Interface:**
   - Design an intuitive and user-friendly interface for interacting with the chatbot, ensuring seamless communication and ease of use for users.
   - Implement features that facilitate smooth navigation and interaction, allowing users to access support services efficiently.
3. **Developing REST APIs:**
   - Create RESTful APIs to establish communication between the frontend interface and the backend system, enabling data exchange and request handling.
   - Ensure robustness and security of the APIs, adhering to best practices for API development and integration.
4. **Building Backend System with RASA:**
   - Utilize the RASA framework to develop a backend system capable of processing user queries and generating appropriate responses.
   - Implement NLU pipelines within RASA to extract intent and entities from user messages, enabling accurate interpretation of user queries.
5. **Creating Knowledge Base:**
   - Curate a comprehensive knowledge base containing relevant information and resources required by the chatbot to provide informative responses.
   - Organize and structure the knowledge base to facilitate efficient retrieval and utilization of information during interactions with users.
6. **Tuning NLU Pipeline for Accuracy:**
   - Fine-tune the NLU pipeline of RASA to enhance the chatbot's ability to understand user queries and extract relevant information.

- Configure the pipeline parameters and algorithms based on the specific use case and requirements, optimizing performance and accuracy.

7. **Integrating Language Models:**
   - Integrate advanced language models such as BERT and GPT into the RASA framework to augment the chatbot's language understanding capabilities.
   - Fine-tune and customize the language models to adapt to the nuances and complexities of the support domain, maximizing accuracy and relevance of responses.
8. **Continuous Knowledge Base Update:**
   - Establish processes for continuous monitoring and updating of the knowledge base to ensure that the chatbot remains relevant and up-to-date.
   - Implement mechanisms for gathering feedback from users and incorporating new information and insights into the knowledge base on a regular basis.

4. TECHNOLOGY STACK / SKILLS ACQUIRED

4.1 Frontend:

- Proficiency in **HTML, CSS, and JavaScript** for designing and developing user interfaces.
- Knowledge of **Bootstrap** framework for responsive and mobile-first web development.
- Implementation of **REST APIs** to enable communication between the frontend interface and the backend system.

4.2 RASA Framework:

- Understanding of machine learning concepts and libraries for **natural language understanding (NLU)** and dialogue management.
- Familiarity with **TensorFlow, spaCy, and NLTK** for building and training machine learning models within the RASA framework.
- Integration of advanced **language models (LLMs)** such as **BERT and GPT** to enhance the chatbot's language understanding capabilities and response generation.

4.3 Backend:

- Expertise in **Python** programming language for backend development.
- Utilization of Flask, a lightweight web framework, for building **RESTful APIs** and backend services.
- Implementation of **data handling and processing logic** to manage user requests and responses effectively.

5. ACHIEVEMENTS AND CONTRIBUTIONS

5.1 Achievements:

Successful Development of an AI-Powered Helpdesk Chatbot:

- Led the design and implementation of an AI-driven helpdesk chatbot using the RASA framework, effectively addressing user queries and streamlining support processes.
- Achieved project milestones on time and within budget, demonstrating effective project management skills and commitment to delivering high-quality solutions.

Enhanced User Experience and Support Efficiency:

- Implemented user-friendly interfaces utilizing HTML, CSS, JavaScript, and Bootstrap, resulting in an intuitive and visually appealing user experience.
- Developed REST APIs with Flask to enable seamless communication between the frontend interface and the backend RASA system, improving support efficiency and reducing response times.

RASA Framework Expertise:

- Demonstrated proficiency in the RASA framework, utilizing advanced machine learning techniques and libraries such as TensorFlow, spaCy, and NLTK to develop and train NLU models.
- Conducted extensive experimentation and optimization to fine-tune the NLU pipeline and improve the chatbot's language understanding capabilities, resulting in more accurate and contextually relevant responses.

Continuous Knowledge Base Maintenance:

- Established automated processes for monitoring and updating the knowledge base, ensuring that the chatbot remains equipped with the latest information and insights.
- Implemented feedback mechanisms to gather user input and incorporate new knowledge into the system, contributing to ongoing improvement and relevance of the chatbot's responses.

5.2 Contributions:

Project Planning and Design:

- Collaborated with project stakeholders to define project requirements, scope, and objectives, ensuring alignment with organizational goals and user needs.
- Provided valuable insights and recommendations during the planning and design phases, contributing to the development of a robust and scalable solution architecture.

Frontend and Backend Development:

- Led the frontend and backend development efforts, leveraging expertise in frontend technologies such as HTML, CSS, JavaScript, and backend technologies including Python and Flask.
- Designed and implemented responsive user interfaces, RESTful APIs, and backend services, facilitating seamless communication and interaction with the chatbot across various platforms and devices.

Integration of Advanced Language Models:

- Successfully integrated advanced language models such as BERT and GPT into the RASA framework, enhancing the chatbot's ability to understand user queries and generate contextually relevant responses.
- Conducted rigorous testing and validation to ensure the accuracy and performance of the integrated language models, resulting in improved conversational capabilities and user satisfaction.

Knowledge Sharing and Collaboration:

- Actively participated in knowledge sharing sessions, code reviews, and collaborative discussions with team members, sharing insights, best practices, and lessons learned throughout the project.
- Mentored junior team members and provided guidance on technical challenges, fostering a culture of continuous learning and professional development within the team.

6. METHODOLOGY OF THE PROJECT

6.1 Setting Up Rasa Framework

Step 1: Installing Rasa (for Linux)

a. Install Python

Rasa currently works only with python 3.8, 3.9, 3.10 versions.

```
$ sudo apt update
$ apt-get install build-essential
$ sudo apt-get install libssl-dev openssl
$ wget https://www.python.org/ftp/python/3.8.0/Python-3.8.0.tgz
$ tar xzvf Python-3.8.0.tgz
$ cd Python-3.8.0
$ ./configure
$ make
$ sudo make install
```

check if python is installed

```
$ python3 --version
```

b. Install pip

```
$ sudo apt install python3-pip
```

check if pip is installed

```
$ pip3 –version
```

c. Create a virtual environment using virtual environment

create and enter folder

```
$ mkdir rasaproject
$ cd rasaproject
```

next we install python virtual environment

```
$ sudo apt install python3-venv
```

we can now create a virtualenv

```
$ python3 -m venv ./venv
```

activate virtualenv

```
$ source ./venv/bin/activate
```

d. Install Rasa

```
$ pip install rasa
```

check if rasa is installed

```
$ rasa --version
```

Step 2: Project Initialization

```
$ python -m rasa init
```

This will create a basic project structure with necessary files and directories.

Step 3: Define Training Data

a. Define the Intents with their corresponding training examples under the **nlu** section in data/**nlu.yml**
b. Define different conversational flows using user intents and responses. Map each user intent with an action using the nomenclature 'utter_<intent-name>', which will be invoked when that particular intent is triggered. This is done under **stories** section in data/**stories.yml**
c. Now to define forms or fallback intent (if any), define them in **rules** section of data/**rules.yml**
d. Now in the **intents** section of **domain.yml** mention all the intents present in nlu.yml / stories.yml. Then define all the action in **responses** section of **domain.yml**
e. Now configure the **pipeline** and **policies** in **config.yml**

We have used a pre-trained BERT language model for training the model and the intent classification is done using DIET Classifier which is a algorithm built by Rasa using Transformer architecture same as BERT.

Step 4: Train the NLU Model

As we are using BERT architecture, so we need to install rasa transformers

```
$ pip install rasa[transformers]
```

Now to train the model use

```
$ python -m rasa train
```

This will train the Rasa model using your defined training data.

Step 5: Run the Chatbot:

a. To interact with chatbot in command prompt use (this needs internet)

```
$ python -m rasa shell
```

b. To parse the user query and see its intent classification with confidence score use

```
$ python -m rasa run nlu
```

c. To interact with chatbot using Rest channel or through rasa api over a frontend use

```
$ python -m rasa run -m models --enable-api --cors "*"
```

Note: To interact with chatbot using Rest channel or through rasa api uncomment the **rest** section in **credentials.yml** and uncomment the **action_endpoint** section in **endpoints.yml**

6.2 What is RASA ?

Rasa is an open-source framework for building conversational AI chatbots, assistants, and contextual assistants. It provides a complete development environment to build, train, and deploy chatbots that can understand natural language and carry out complex tasks.

The Rasa framework consists of two main components:

1. **Rasa NLU (Natural Language Understanding):** The Rasa NLU component uses machine learning models to recognize intents and extract entities from user inputs. These models can be trained on annotated training data to improve their accuracy. Rasa NLU also allows developers to define custom actions, which can be used to perform tasks based on the user's input.
2. **Rasa Core:** It is responsible for managing the conversation flow and deciding how the chatbot should respond to user inputs. It uses a machine learning-based approach to learn from previous conversations and predict the most appropriate action to take based on the user's intent and the current context.

Rasa also provides tools for testing and evaluating chatbots, as well as deploying them on various platforms such as Facebook Messenger, Slack, and Telegram. It also has a growing community of developers and users who contribute to the development of the framework and provide support to fellow users.

6.2.1 Different files in Rasa:

nlu.yml: This file contains the NLU (natural language understanding) data for your chatbot. It includes the training examples for each intent, as well as any entity labels and synonyms.

```yaml
- intent: network_password_reset
  examples: |
    - How do I reset my password?
    - Can you guide me through the process of resetting my password?
    - I forgot my password. What should I do?

- intent: network_connectivity_troubleshoot
  examples: |
    - How can I troubleshoot network connectivity issues?
    - My network is not working
    - What steps should I follow to fix network connection problems?

- intent: software_updates_installation
  examples: |
    - How do I install software updates on my computer?
    - What is the process for installing software updates?
```

Figure 6.1 nlu.yml

stories.yml: This file contains the stories for your chatbot, which are the examples of conversations between the user and the chatbot. Each story is a sequence of actions that the chatbot takes in response to user input.

```yaml
- story: User asks about resetting network password
  steps:
  - intent: network_password_reset
  - action: utter_network_password_reset

- story: User asks about troubleshooting network connectivity issues
  steps:
  - intent: network_connectivity_troubleshoot
  - action: utter_network_connectivity_troubleshoot

- story: User asks about installing software updates
  steps:
  - intent: software_updates_installation
  - action: utter_software_updates_installation
```

Figure 6.2 stories.yml

domain.yml: This file contains the domain of your chatbot, including the intents, entities, actions, slots, and responses. It defines what your chatbot can do and how it should respond to user input.

```yaml
intents:
  - network_password_reset
  - network_connectivity_troubleshoot
  - software_updates_installation

responses:
  utter_network_password_reset:
  - text: "To reset your network password, you can follow these steps:\n
1. Go to the password reset portal or contact the IT service desk.\n
2. Provide the necessary identification or authentication details.\n
3. Follow the prompts to reset your password.\n
4. Create a new password that meets the specified complexity requirements.\n
5. Confirm the new password.\n "

  utter_network_connectivity_troubleshoot:
  - text: "Here are some steps to troubleshoot network connectivity issues:\n
1. Check physical connections: Ensure all cables are plugged in and devices are powered on.\n
2. Restart networking devices: Power cycle your modem, router, and switches.\n
3. Check IP settings: Verify that your device is configured with the correct IP address\n
4. Disable/enable network adapter: Temporarily disable and re-enable your network adapter.\n "

  utter_software_updates_installation:
  - text: "The process of installing software updates can vary:\n
1. Check for updates: Open the software or go to the system settings to check for updates.\n
2. Download updates: If updates are available, initiate the download process.\n
3. Install updates: Once the download is complete, follow the prompts to install the updates."
```

Figure 6.3 domain.yml

config.yml: This file contains the configuration for your chatbot, including the language, pipeline, policies for processing user input and the hyperparameters for training the model.

```yaml
pipeline:
- name: WhitespaceTokenizer
- name: LanguageModelFeaturizer
  model_name: "bert"
  model_weights: "rasa/LaBSE"
  cache_dir: null
- name: DIETClassifier
  epochs: 200
- name: FallbackClassifier
  threshold: 0.3

policies:
 - name: MemoizationPolicy
 - name: TEDPolicy
   max_history: 5
   epochs: 170
 - name: RulePolicy
```

Figure 6.4 config.yml

endpoints.yml: This file contains the endpoints for your chatbot, including the URL of the NLU server and the URL of the action server.

actions.py: This file contains the code for the custom actions that your chatbot can take in response to user input. It defines the logic for processing user input and generating responses.

models: This directory contains the trained models for your chatbot. Each model is a snapshot of your chatbot's state at a particular point in time, and can be used to run your chatbot in production.

6.3 Rasa Architecture

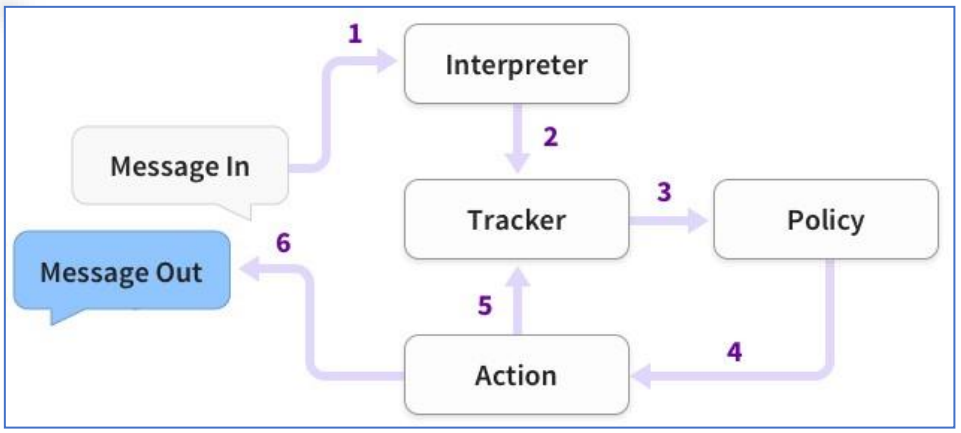

Figure 6.5 Rasa Architecture

The steps are:

1. The message is received and passed to an Interpreter, which converts it into a dictionary including the original text, the intent, and any entities that were found. This part is handled by NLU.
2. The message is passed from the Interpreter to the Tracker. The Tracker is the object which keeps track of conversation state and receives the structured output from interpreter.
3. The current state of the tracker is sent to each policy.
4. The policy decides which appropriate next action is to be performed.
5. The log of selected action is maintained by tracker.
6. The appropriate response is provided to the user, using intents defined in nlu.md, like utter_response.

6.4 Why Rasa Over Other Frameworks

Rasa framework is best suitable for retrieval-based chatbots for several reasons:

1. **Open source:** Rasa is an open-source framework, which means that developers can use and modify it according to their specific needs without incurring licensing fees or other costs.
2. **Customization:** Rasa provides a high degree of customization, which allows developers to create chatbots that are tailored to their specific business needs. This is particularly important for businesses that have unique requirements for their chatbot functionality.
3. **Scalability:** Rasa is highly scalable and can be used to build chatbots that can handle large volumes of traffic. This makes it a great option for businesses that need to build chatbots that can handle a high volume of customer inquiries.
4. **Natural language processing:** Rasa provides a powerful natural language processing (NLP) engine that can understand user input and respond appropriately. This allows businesses to build chatbots that can engage in human-like conversations with their customers.
5. **Integration:** Rasa integrates easily with other systems and platforms, such as messaging apps, CRMs, and customer support systems. This makes it a flexible solution that can be integrated into existing workflows and systems.

While other frameworks like opendialog, chatterbot, and Dialogflow may also have some of these features, Rasa is specifically designed for building conversational AI applications, which makes it the best choice for building sophisticated, high-performance chatbots.

6.5 Frontend of the Chatbot

The frontend development of the chatbot entails crafting a dynamic web interface using a combination of HTML, CSS, and JavaScript. **HTML** serves as the foundation for structuring the content of the web page, allowing for the creation of various components essential for the chatbot interface, including the chat window, input field, and interactive buttons. By leveraging HTML's semantic markup, we ensure proper organization and accessibility of the chatbot's elements.

CSS plays a pivotal role in enhancing the visual aesthetics and user experience of the chatbot interface. Through CSS styling, we can customize the appearance of each component, ensuring consistency with the branding guidelines of the website or platform. This includes defining colors, fonts, spacing, and other visual properties to create a cohesive and visually appealing user interface.

JavaScript serves as the backbone of interactivity within the chatbot interface. It enables seamless communication between the frontend and backend systems, facilitating dynamic interactions and real-time responses to user inputs. JavaScript functionalities include handling user queries, transmitting data to the backend via APIs, processing responses from the backend, and updating the interface accordingly.

REST API (Representational State Transfer Application Programming Interface) allows the frontend interface, built using HTML, CSS, and JavaScript, to communicate with the backend system, typically developed using Python and Flask in this project. Through RESTful endpoints exposed by the backend, the frontend can send user queries, receive responses, and perform various operations required for the chatbot's functionality.

The communication flow typically involves the following steps:

1. **Sending User Queries:** When a user interacts with the chatbot interface, JavaScript functions are triggered to capture and process the user's input. This input, such as a message or command, is then packaged into a request and sent to the backend server via a REST API call.
2. **Processing Requests on the Backend:** Upon receiving the user query, the backend server processes the request by applying natural language understanding (NLU) techniques to extract the user's intent and relevant entities. This involves parsing and

analyzing the incoming message to understand its context and determine the appropriate response.

3. **Generating Responses:** Based on the analysis of the user query, the backend system generates a response containing the information or action requested by the user. This response may involve accessing external APIs or databases to retrieve relevant data, performing computations, or executing predefined actions.
4. **Sending Responses to the Frontend:** Once the response is generated, the backend server packages it into a structured format, typically JSON (JavaScript Object Notation), and sends it back to the frontend interface via another REST API call. The frontend interface then processes the response and displays it to the user within the chatbot interface.
5. **Updating User Interface:** Upon receiving the response from the backend, JavaScript functions on the frontend side handle the presentation and display of the response within the chat window. This may involve dynamically updating the chat history, rendering interactive elements, or displaying multimedia content as per the received response.

6.6 Backend of the Chatbot

The backend of the chatbot includes implementation of RASA framework with our customized dataset.

6.6.1 Project Architecture:

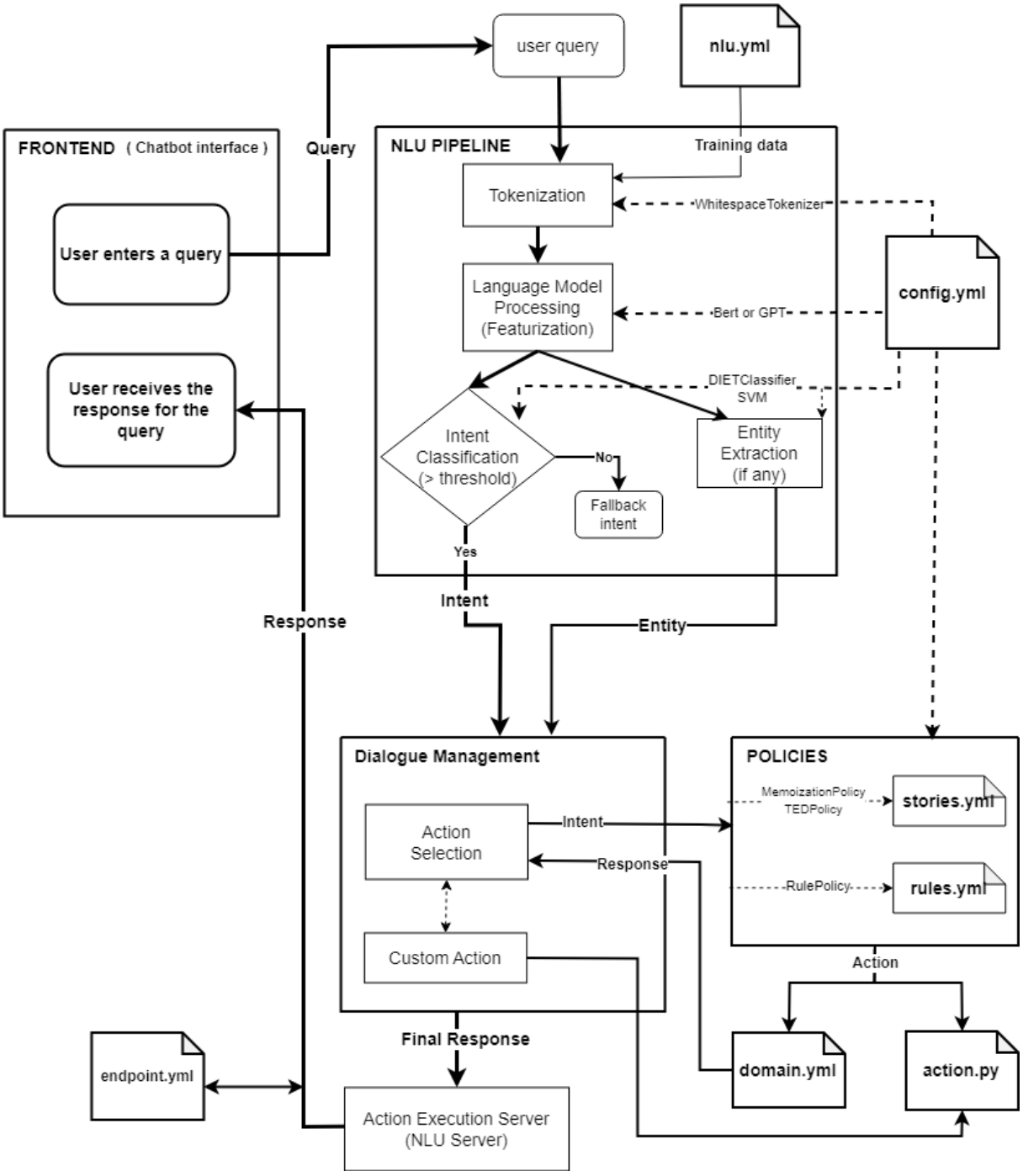

Figure 6.6 Project Architecture

6.6.2 Procedure for Finding the Response to a User Query In RASA:

1. **User Query**: The process starts with the user entering a query or message to the chatbot.
2. **NLU Pipeline**: The message is then sent through the NLU pipeline, which decides which approaches or methods are to be used for components such as:
    - Tokenization
    - Vectorization
    - entity extraction.
    - Number of epochs
    - Fallback threshold etc

    The NLU pipeline can be customized based on use cases and better performance. The methods or approaches used in NLU pipeline are saved in **config.yml**.
3. **Intent Classification**: In this step, the intent of the user message is determined. RASA uses machine learning algorithms such as SVM (Support Vector Machines).

    The intent classification algorithm tries to match the user's message to one of the predefined intents with the help of training examples in **nlu.yml** file. The intent with the highest score is selected as the predicted intent.

    Intent classification using support vector machine (SVM):

    Data preparation: The first step is to prepare the training data and the user query for the intent classification. The dataset should have enough examples for each intent to enable the model to learn the patterns in the data.

    Tokenize the user query and training data, which means breaking it down into individual words and creating a vocabulary of all the unique words in the dataset.

    Also removing of stopwords, punctuation and part-of-speech tagging on the data are performed.

    Vectorization and Feature Extraction: Once the text data has been preprocessed Vectorization is the process of converting text into numerical vectors that can be used in machine learning models. In RASA, the CountVectorizer and TfidfVectorizer classes from the scikit-learn library are used for vectorization. The CountVectorizer converts the text into a matrix of token counts, while the **TfidfVectorizer** converts the text into a matrix of term frequencies multiplied by inverse document frequencies (TF-IDF).

    After the data is vectorized, feature extraction is the process of selecting and transforming the most relevant information from the vectorized data to be used for

model training. In the context of intent classification in Rasa, feature extraction typically involves transforming the vectorized text data into a more compact representation that captures the most important information for distinguishing between different intents.

The **CountVectorsFeaturizer** component is a feature extractor that uses the bag of words (BoW) model to represent each input message as a vector of word frequencies. In BoW representation, each word in the text data is treated as a separate feature, and the feature vector is a count of how many times each word appears in the text.

The SpacyFeaturizer component uses the spaCy library to extract linguistic features from the input message, such as part-of-speech tags and dependency parse trees. These vector representations capture the semantic meaning of each word in the text and allow RASA to compare the similarity between different words.

This vector representation is then used as input to the support vector machine, to predict the intent of the query.

Featurization using Language Models

Featurization using advanced language models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) in RASA involves leveraging these models to extract rich, context-aware representations of text data. Here's an explanation of how this process works:

1. BERT (Bidirectional Encoder Representations from Transformers):

BERT is a state-of-the-art language model developed by Google that excels in capturing bidirectional context in text data. In the context of RASA, BERT can be used for featurization to encode user utterances and extract meaningful representations that capture both syntactic and semantic information.

Featurization Process with BERT:

1. **Tokenization:** Before feeding text data into BERT, it needs to be tokenized into subwords or word pieces. BERT utilizes a WordPiece tokenizer to break down input text into a vocabulary of subword tokens.
2. **Encoding**: The tokenized text is then fed into the BERT model, which consists of multiple layers of transformer encoders. These encoders process the input tokens in a bidirectional manner, capturing contextual information from both preceding and succeeding tokens.

3. **Feature Extraction:** BERT generates contextualized embeddings for each token in the input sequence. These embeddings represent rich, contextual features that capture the semantics of the text at various levels of granularity.
4. **Pooling:** To obtain a fixed-size representation for the entire input sequence, a pooling mechanism is applied to aggregate the token-level embeddings. Common pooling strategies include mean pooling or max pooling across token embeddings.

2. GPT (Generative Pre-trained Transformer):

GPT, developed by OpenAI, is another powerful language model known for its ability to generate human-like text based on the provided input. While BERT focuses on bidirectional context, GPT is trained to generate text in a left-to-right autoregressive manner.

Featurization Process with GPT:

1. **Tokenization:** Similar to BERT, text data is tokenized into subwords or word pieces using a tokenizer specific to GPT.
2. **Encoding:** The tokenized text is then fed into the GPT model, which consists of multiple layers of transformer decoders. These decoders generate contextualized embeddings for each token based on the left-to-right context.
3. **Feature Extraction**: GPT generates contextualized embeddings for each token, capturing the semantics and context of the text in a left-to-right fashion.
4. **Pooling:** Similar to BERT, a pooling mechanism can be applied to obtain a fixed-size representation for the entire input sequence.

Training the model: Once the data is pre-processed, the next step is to train the model. DIETClassifier (a version of SVM) is being used. SVM is a machine learning algorithm that works by finding the optimal hyperplane in a high-dimensional space that separates the different classes. In the context of intent classification, the SVM algorithm tries to find the best decision boundary between different intents.

Support Vector Machines (SVM) finds the best hyperplane by maximizing the margin between the two classes in the feature space. The margin is defined as the distance between the hyperplane and the closest data points of each class, also known as the support vectors. The goal of SVM is to find the hyperplane that maximizes this margin.

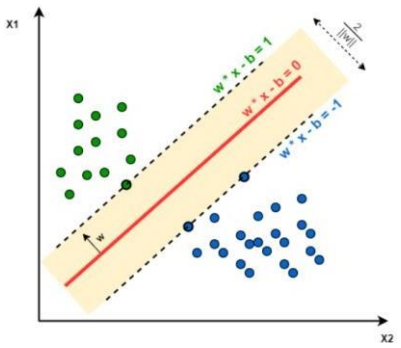

Figure 6.7 SVM Hyperplane

a hyperplane in a d-dimensional space can be represented as: $\boldsymbol{w^T x + b = 0}$
where w is the normal vector to the hyperplane, x is the input data, b is the bias term, and T represents the transpose operation.

In binary classification, we have two classes (positive and negative), and we aim to find a hyperplane that maximizes the margin between the two classes. The margin is defined as the distance between the hyperplane and the closest data points from each class.
Mathematically, the margin can be expressed as: $\textbf{margin} = \frac{2}{||w||}$
where ||w|| is the Euclidean norm of the weight vector w. The goal of SVM is to maximize this margin subject to the constraint that all data points are correctly classified.

For **multi-class classification** RASA uses **"one-vs-all"** approach
In this approach, SVM trains one binary classifier for each class, treating the data of that class as positive examples and the data of all other classes as negative examples. For class i, SVM constructs a hyperplane that separates the data points of class i from the data points of all other classes. The hyperplane is defined as: $\boldsymbol{w_i^T x + b_i = 0}$
where w_i is the weight vector and b_i is the bias term for the binary classifier that separates class i from all other classes.

To classify a new data point x, SVM runs each of the binary classifiers and computes a score for each class. The score for class i is defined as: $\boldsymbol{score_i = w_i^T x + b_i}$

The class with the highest score is chosen as the predicted class.

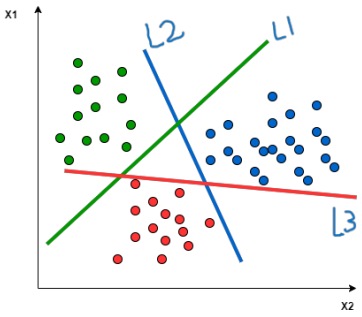

Figure 6.8 Predicted Classes in SVM

here, the line L1 tries to maximize the gap between green points and all other points at once.

Suppose that if the confidence score is very low (close to 0), that means the user query has not matched with any pre-defined intents. We can set a threshold so that any user query which has the score less than that threshold then a default response will be triggered saying “Sorry, I didn’t understand. It is out of my knowledge boundary.”

We can handle the fallbacks in multiple ways, the above one is a standard approach. Fallbacks can also be handled by redirecting a less score user query to a human response.

4. **Entities extraction:** In this step, any entities present in the user message are extracted. Entities are specific pieces of information that the user is trying to convey.

   For example, if the user message is "Book a flight from New York to London on 10th August", then the entities extracted would be "New York", "London", and "10th August". Entities are defined in the **nlu.md** file in RASA.

   To extract entities, RASA uses a combination of rule-based and machine learning-based approaches.

   Rule-based entity extraction:

   In rule-based entity extraction, we define patterns for each entity we want to extract. For example, if we want to extract a date, we can define a pattern that matches dates in various formats like "today", "tomorrow", "next week", "December 1st", etc. RASA provides built-in entity extractors like DIETClassifier and CRFEntityExtractor which

uses rule-based approach to extract entities. RASA by default uses a rule-based approach for entity recognition.

Machine learning-based entity extraction:

In machine learning-based entity extraction, we train a model to predict entities from text. RASA provides a pre-trained entity extractor called DIETClassifier which uses a neural network to predict entities. The neural network takes in the tokenized text and context of the conversation as input and outputs the entity label and its start and end positions in the text.

Once the entities are extracted, they are stored in slots. These slots can be filled with user inputs.

5. **Dialogue management:** Once the intent and entities of the user message have been determined, the dialogue management component of RASA decides what the chatbot should do next. It uses a dialogue policy to decide which action to take next based on the current conversation history. Dialogue policies are defined in the **config.yml** and **domain.yml** file in RASA.

6. **Action selection:** Once the dialogue management component has decided which action to take, the action server is called to execute the action. Actions are defined in the **actions.py** file in RASA. The action can be customized as per requirement and sent to the user. An action can be anything from sending a message to calling an API.

7. **Action execution:** The action server executes the chosen action and returns a response. The response is sent back to the user as a message through the Rasa NLU server API. The responses are defined in the **domain.yml** file in RASA.

8. **User receives response:** The user receives the response from the chatbot.

7. RESULT SCREENS

7.1 Screens of the helpdesk chatbot

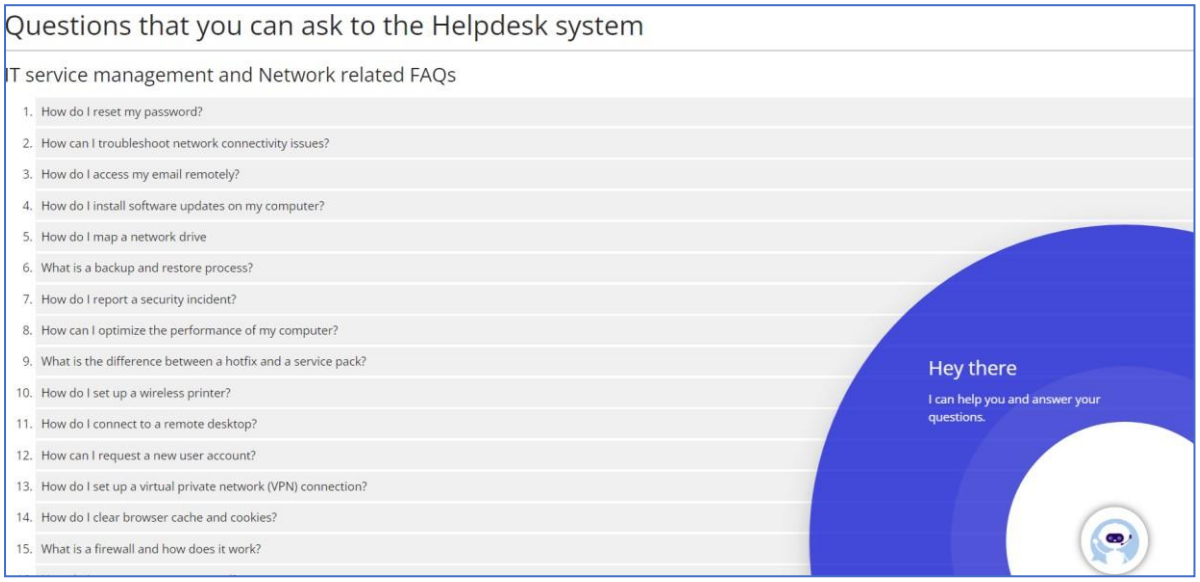

Figure 7.1 Helpdesk Chatbot Screen 1

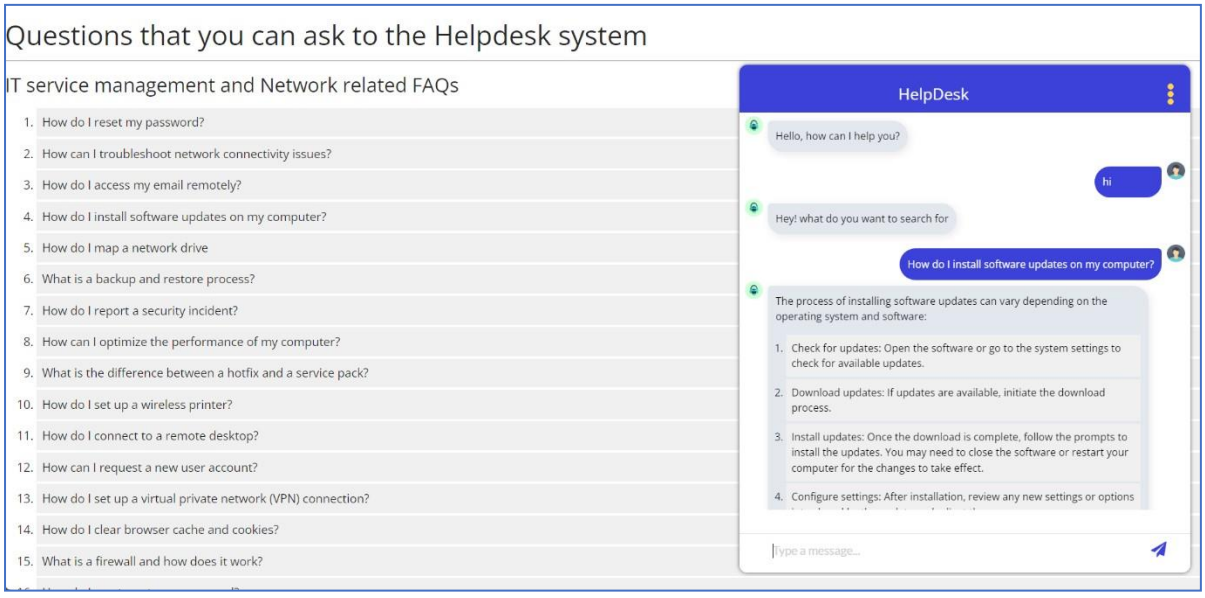

Figure 7.2 Helpdesk Chatbot Screen 2

7.2 Performance metrics with spaCy (default)


```
"accuracy": 0.9889705882352942,
  "macro avg": {
    "precision": 0.9680851063829787,
    "recall": 0.9787234042553191,
    "f1-score": 0.9716312056737588,
    "support": 272
  },
  "weighted avg": {
    "precision": 0.9834558823529411,
    "recall": 0.9889705882352942,
    "f1-score": 0.9852941176470589,
    "support": 272
  },
```


Figure 7.3 Performance metrics for spaCy

The below histogram allows you to visualize the confidence for all predictions, with the correct and incorrect predictions being displayed by blue and red bars respectively. Improving the quality of training data will move the blue histogram bars up the plot and the red histogram bars down the plot. It also helps in reducing the number of red histogram bars itself.

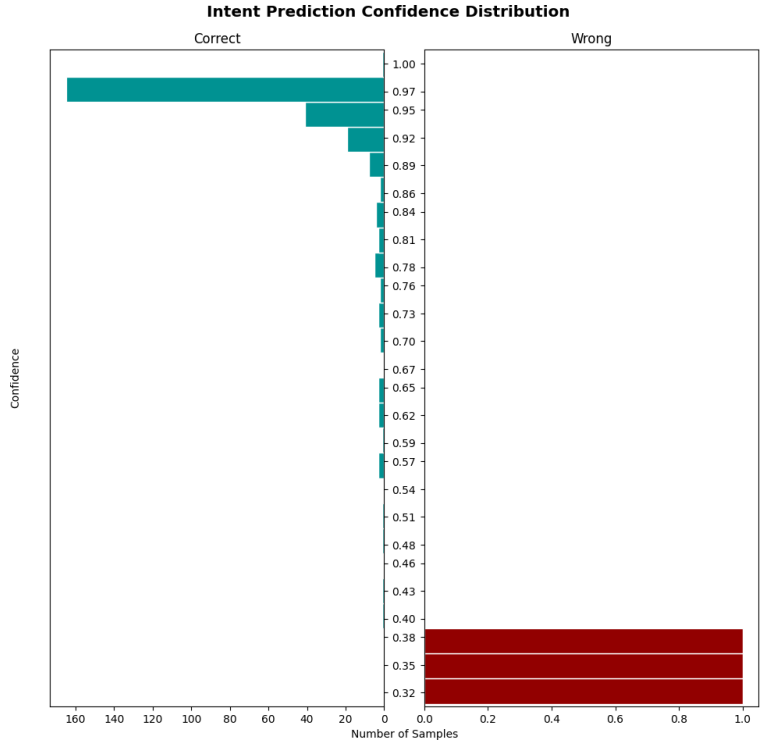

Figure 7.4 Histogram for spaCy

The default spaCy Rasa pipeline is less accurate, less precise than the pipeline with BERT/GPT.

7.3 Performance metrics with BERT/GPT


```
"accuracy": 1.0,
"macro avg": {
  "precision": 1.0,
  "recall": 1.0,
  "f1-score": 1.0,
  "support": 229
},
"weighted avg": {
  "precision": 1.0,
  "recall": 1.0,
  "f1-score": 1.0,
  "support": 229
```


Figure 7.5 Performance metrics for BERT/GPT

The below histogram allows you to visualize the confidence for all predictions, with the correct and incorrect predictions being displayed by blue and red bars respectively. Improving the quality of training data will move the blue histogram bars up the plot.

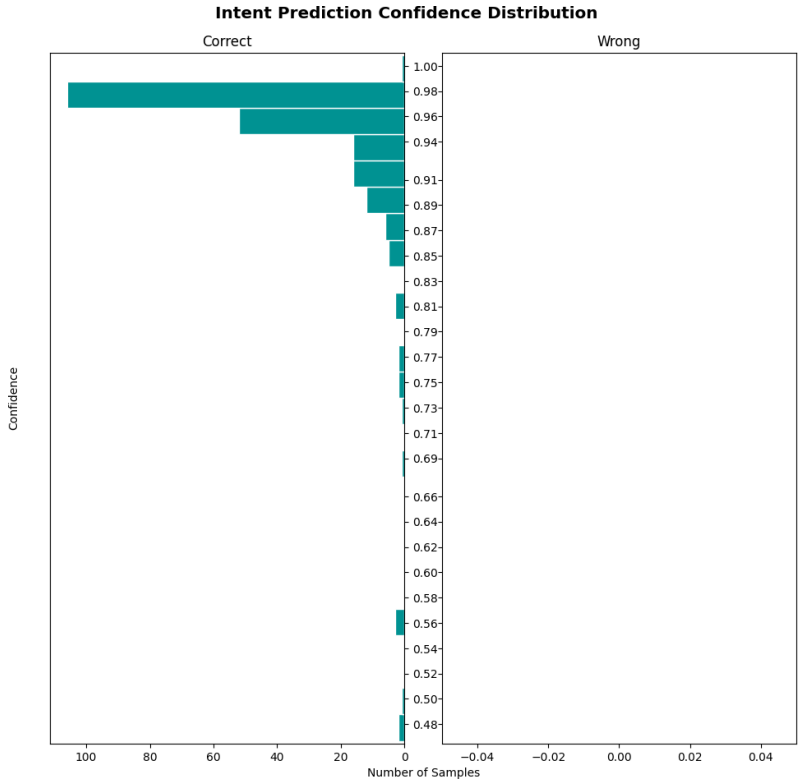

Figure 7.6 Histogram for BERT/GPT

The BERT/GPT Rasa pipeline more accurate, more precise than the default spaCy pipeline.

8. FUTURE APPLICATIONS

1. **Internal Employee Support and Troubleshooting:**
   - Within companies and organizations, employees can utilize the chatbot for a wide range of troubleshooting and support needs.
   - Employees can seek assistance for technical issues, such as troubleshooting software problems, resolving network connectivity issues, or configuring hardware devices.
2. **Customer Support in E-commerce:**
   - The chatbot can assist customers with product inquiries, order tracking, and returns, enhancing the overall shopping experience.
   - Integration with product databases allows the chatbot to provide personalized product recommendations based on customer preferences and browsing history.
3. **Healthcare Assistance:**
   - In healthcare settings, the chatbot can help patients schedule appointments, access medical records, and receive general health information.
   - Integration with electronic health record (EHR) systems enables the chatbot to provide personalized health recommendations and reminders for medication refills or appointments.
4. **Legal Services:**
   - Law firms can utilize the chatbot to provide legal assistance to clients, such as answering basic legal questions, providing information on case statuses, and scheduling consultations with attorneys.
   - Integration with legal databases and case management systems allows the chatbot to retrieve relevant legal documents and precedents to support client inquiries.
5. **Human Resources Management:**
   - The chatbot can assist HR departments with tasks such as employee leave management, performance reviews, and policy inquiries.
   - Integration with HR management systems allows the chatbot to automate routine HR processes, such as employee onboarding and offboarding.
6. **Real Estate Services:**
   - Real estate agencies can deploy the chatbot to assist clients with property searches, mortgage inquiries, and appointment scheduling for property viewings.

- Integration with property databases and MLS listings enables the chatbot to provide up-to-date information on available properties and market trends.

7. **Educational Support:**
    - Educational institutions can utilize the chatbot to provide academic support to students, such as answering course-related questions, providing study resources, and facilitating communication with professors.
    - Integration with learning management systems (LMS) allows the chatbot to access course materials and assignment deadlines, helping students stay organized and on track with their studies.
8. **Public Sector Services:**
    - Government agencies can deploy the chatbot to provide citizens with information on public services, government programs, and regulatory requirements.
    - Integration with government databases enables the chatbot to assist citizens with tasks such as applying for permits, paying taxes, and accessing public records.
9. **Automotive Assistance:**
    - Automotive companies can leverage the chatbot to provide customer support for vehicle maintenance, recalls, and warranty inquiries.
    - Integration with vehicle diagnostic systems allows the chatbot to troubleshoot common car problems and provide recommendations for service appointments or repairs.

9. CONCLUSION

The development of an AI-powered helpdesk chatbot using the RASA framework, alongside integration with advanced language models such as BERT and GPT, marks a significant milestone in enhancing user support experiences across various domains. Throughout the project, a comprehensive understanding of chatbot evolution, featurization techniques, and application scenarios has been acquired, laying the groundwork for impactful implementations.

The frontend interface, crafted using HTML, CSS, and JavaScript, ensures an intuitive user experience, while REST API communication enables seamless interaction with the backend system. Leveraging the capabilities of the RASA framework, the backend system processes user queries with precision, utilizing DIETClassifier (SVM) for intent classification and entity recognition tasks. Integration with BERT and GPT further enhances featurization, enabling the chatbot to capture nuanced semantic context and deliver accurate responses.

The applications of the developed helpdesk chatbot extend beyond organizational support to encompass diverse sectors such as healthcare, legal services, and e-commerce. By providing personalized assistance, automating routine tasks, and streamlining processes, the chatbot empowers users to access information and services efficiently, driving productivity and satisfaction.

The project underscores the potential of AI-driven chatbots to revolutionize customer support and service delivery, offering scalable solutions that adapt to evolving user needs. With continuous improvement and innovation, the developed chatbot stands poised to make significant contributions across industries, enhancing user experiences and organizational efficiency in the digital age.