Building a Simple Chatbot with Rasa

Zaynab Awofeso
Student at University of Lagos

Table of contents

About Me

Zaynab Awofeso is a third-year Computer Science student at the University of Lagos.

She is a Data Scientist currently transitioning to ML research.

As a Data Scientist, she has previously worked at Iqube Labs, KPMG Nigeria, and currently at ARM Careers.

Aside from tech, she enjoys writing and volunteering and is an active volunteer in many organizations at Unilag.

“Learning never exhausts the mind”

–Leonardo da Vinci,
Renaissance Polymath

01

Introduction

What is a chatbot?

A chatbot is an artificial intelligence (AI) software that can simulate a conversation with a user in natural language through messaging applications or websites or mobile apps.

They are widely used in various domains such as customer service, personal assistants, and more.

Rasa can be used to create chatbots that understand natural language and handle complex dialogues.

02

What is Rasa?

Rasa

RASA is an open-source chatbot framework based on machine learning.

With the help of it, we can easily create highly accurate and sophisticated chatbots and can easily integrate these chatbots with our website, Telegram, Facebook, etc.

Basic Concepts

Before we get into it, let’s look into 2 simple concepts that we should know before creating a chatbot.

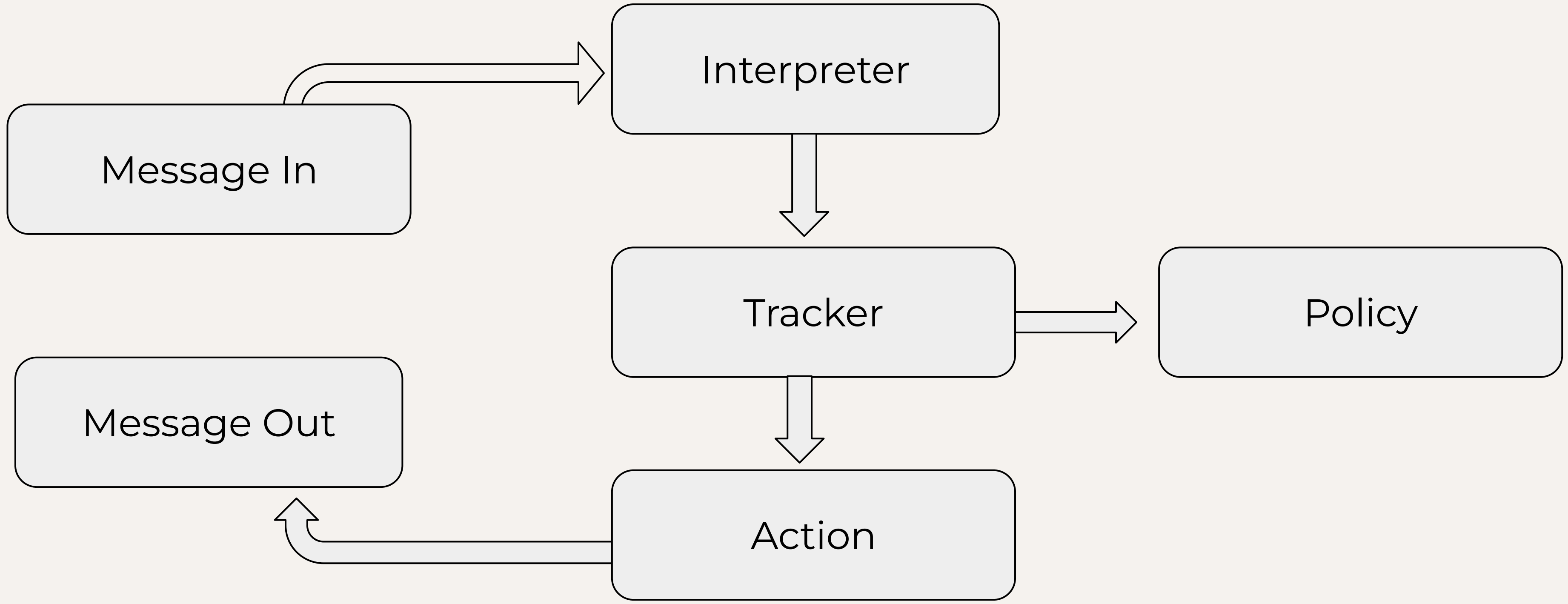
Query

A query is the user message for the chatbot in order to get details of something.

Response/Action

Action is the response from a chatbot based on the query.

Architecture of Rasa



Benefits of Rasa

- ❖ Flexibility in customization - allows developers to tailor chatbots to specific needs.
- ❖ Scalability - ensures chatbots can handle a growing number of users and interactions.
- ❖ Open-source nature - provides transparency and control over chatbot functionality.

03

Key Components of Rasa

The tasks of Chatbot can be divided into two main groups:

- ❖ Understand the User's Question.
- ❖ Maintain Conversation Flow.

Rasa provides two main components to handle these tasks separately:

- ❖ Rasa NLU
- ❖ Rasa Core

Rasa NLU (Natural Language Understanding)

Rasa NLU processes and interprets user messages to understand *intents* and extract relevant *entities*.

Intents: represent the purpose of the user's message (e.g., asking for weather info).

Entities: are specific pieces of information from user messages (e.g., location, date).

Example: *User message: "What's the weather like in New York tomorrow?"*

- Intent: ask_weather

- Entities: location (New York), date (tomorrow)

Rasa Core

Rasa Core manages the flow of conversation by predicting the next action based on the conversation context.
It uses *stories* to define possible conversation paths.

Stories are sequences of user inputs and bot responses that guide the chatbot’s behavior.

Example:

- *User: "I want to book a flight."*
- *Bot: "Sure, where would you like to go?"*
- *User: "To New York."*
- *Bot: "When do you want to travel?"*

04

Setting Up Rasa

System Requirements

❖ Ensure you have Python 3.6-3.8 installed.

❖ Install pip, the Python package installer.

❖ Set up a virtual environment for project isolation.

Installation Steps

- ❖ Activate your Virtual Environment

```
python -m venv ./venv
.\venv\Scripts\activate
```

- ❖ Install Rasa using pip

```
pip install rasa
```

- ❖ Verify the installation by checking the Rasa version

```
rasa -–version
```

05

Building Your First Chatbot

Creating a Rasa Project

There are two commands that you can use to create a Rasa project:

- ❖ “rasa init” command
- ❖ “rasa init — — no-prompt” command

“rasa init” command

“rasa init” initializes a new Rasa project by setting up the directory structure and creating the necessary files for your chatbot.

By default, it will prompt you with a series of questions to configure your project.

These questions include the project name, whether to include example training data, which NLU pipeline to use, and other configuration options.

This prompt allows you to customize the setup of your project based on your specific requirements.

“rasa init – – no-prompt” command

This command skips the interactive prompt and uses default settings for the project initialization.

It sets up the project with predefined configurations, including example training data and a pre-selected NLU pipeline.

This option is useful if you want a quick way to get started without going through the interactive configuration process.

However, if you choose this option, you will have to modify the generated files and configurations manually if you want to customize them later on.

Configuration Files

Rasa projects consist of several key configuration files that define the chatbot’s behavior and structure. These files include config.yml, domain.yml, nlu.yml, stories.yml, actions.py, rules.yml etc.

Each of these files plays a crucial role in configuring the natural language understanding (NLU), dialogue management, and custom actions of your chatbot.

nlu.yml

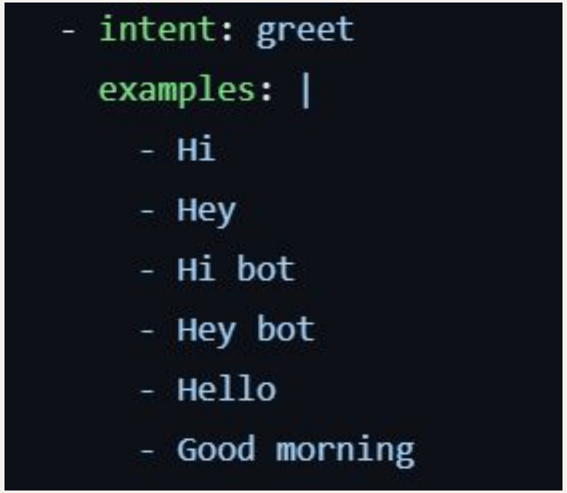
This file provides the training data necessary for your chatbot to understand user inputs accurately.

- ❖ For a bot to recognize what a user is saying, regardless of how the user phrases their message, we need to provide example messages from which the assistant can learn. These examples are grouped according to the idea or goal the message is expressing, also known as the intent.

- ❖ Intents are paired with example phrases that users might say to help the NLU model learn to identify the intent behind different messages.

- ❖ These intents and their examples are used as training data for the assistant's Natural Language Understanding (NLU) model.

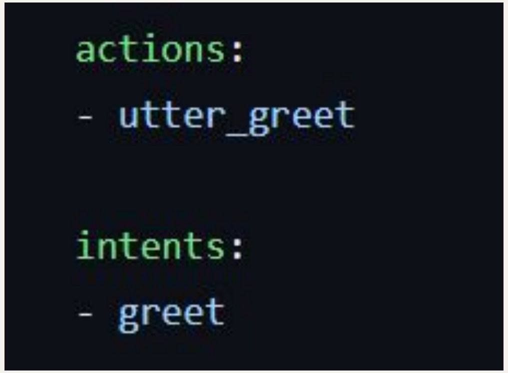
nlu.yml



```yaml
- intent: greet
  examples: |
    - Hi
    - Hey
    - Hi bot
    - Hey bot
    - Hello
    - Good morning
```

domain.yml

The domain.yml is the configuration file of everything that your assistant "knows". It contains Responses, Intents, Slots, Entities, Forms and actions.



```
actions:
- utter_greet

intents:
- greet
```

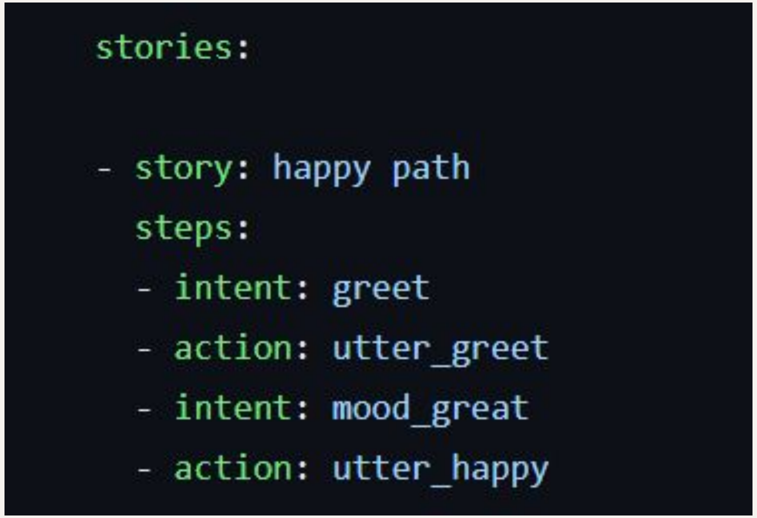
stories.yml

Stories are example conversations that train the bot to respond correctly depending on what the user has said previously in the conversation.

The story format shows the intent of user inputs (intents) followed by the bot's action or responses.

This file helps ensure that your chatbot can manage dialogues effectively by following predefined paths.

stories.yml



```
stories:

- story: happy path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: mood_great
  - action: utter_happy
```

actions.py

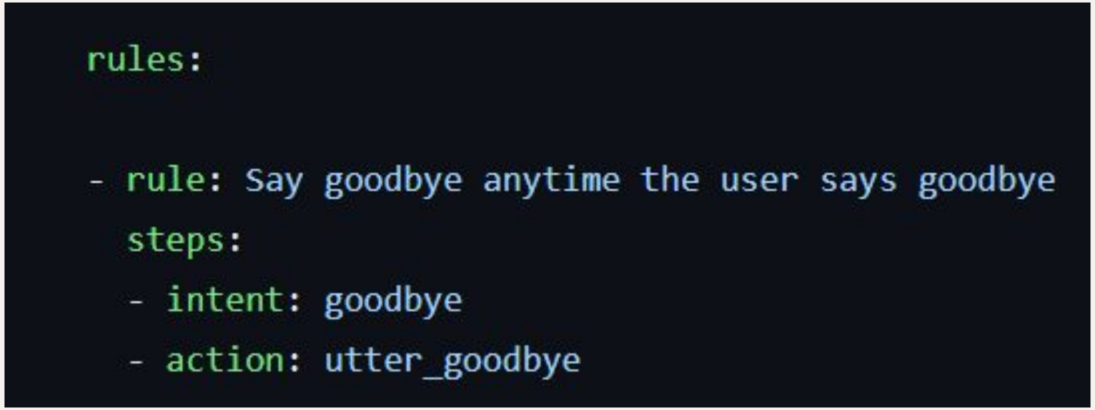
This is a python file to run custom actions. Custom actions allow the chatbot to perform tasks beyond predefined responses like performing specific tasks based on user inputs.

By defining these actions, you can extend the functionality of your chatbot, enabling it to handle more sophisticated interactions and operations.

rules.yml

Rules describe parts of conversations that should always follow the same path no matter what has been said previously in the conversation.

They help ensure consistent responses for common user inputs or specific actions.



```
rules:


- rule: Say goodbye anytime the user says goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye
```

Training the Model

We can train our model based on the data we provided in the above files.

```
rasa train
```

When you run the above code in the terminal, rasa will start training both the nlu model and the core model and save the trained model in the model folder.

Interacting with the Trained Model

You can interact with your bot in the command shell.

```
rasa shell
```

This command allows Rasa to load the trained model and interact with the bot in the shell.

We can stop this rasa shell either by *ctrl+c* or by typing *“/stop”*.

06

Use Cases and Examples

Use Cases and Examples

- ❖ Customer Support:
  - ***Example:*** *A telecom company can use a Rasa chatbot to handle customer inquiries about billing, troubleshooting common issues, and providing account information.*

- ❖ Healthcare:
  - ***Example:*** *A healthcare provider can use a Rasa chatbot to offer symptom checking, schedule appointments, and provide information about medical conditions.*

- ❖ Education:
  - ***Example:*** *An educational institution implements a Rasa chatbot to answer questions about courses, schedules, and campus facilities.*

Thank You!