

ABSTRACT

TITLE: AARUHI_BOT

The aim of this project is to develop a simple chatbot using the Rasa framework integrated with a NASA API. This chatbot is designed to provide users with quick and accurate information about various NASA missions, space events, and other space-related data. By leveraging the capabilities of the NASA API, the chatbot will be able to fetch and deliver up-to-date information in response to user queries. The project focuses on creating an intuitive and user-friendly interface that enables seamless interaction with the chatbot, enhancing the user's experience and knowledge about space exploration.

The procedure involves setting up the Rasa framework, integrating the NASA API, and designing the conversational flow. The chatbot will be trained with relevant intents and entities to understand user queries accurately. The NASA API will be used to fetch real-time data, which the chatbot will present to users in a coherent and informative manner. Key steps include configuring the Rasa environment, developing custom actions to interact with the API, and testing the chatbot to ensure reliability and accuracy. Keywords for this project include Rasa framework, NASA API, chatbot development, conversational AI, space information, and real-time data integration.

Introduction About Rasa Framework

Rasa is a second alternative framework for constructing conversational applications using open-sourced code. This allows you to build chatbots and virtual assistants which can understand Natural Language input.

Key Components of Rasa - The framework

1. ****Rasa NLU(Natural Language Understanding) ****

Understand user messages It identifies and processes the user messages that are being sent to extract some structured data or entities.

- Leverage machine learning models (SVM, TensorFlow or spacy) to categorize intents, we extract entities in order to get specific details such as dates and numbers.

2. **Rasa Core (Dialogue Management)**

- Controls the dialogue for handling how an AS responds to a message based on the conversation it previous one.

Uses machine learning to predict the next action

regular production for each specific conversations, and test of this model.

3. **Rasa SDK:**

- Allows you to customize the behaviour of your bot by writing custom Python code for actions.

- Integrates external systems or APIs to fetch data, perform actions, or provide responses.

4. Rasa X :

- A toolset that provides a user interface to improve and manage your Rasa AI assistant.
- Allows for interactive learning from real conversations, testing, and fine-tuning your model.

Rasa Training: Start here

1. Installation:

Install Rasa using pip:- `pip install rasa`

2. Creating a new Rasa project

Start a New Rasa Project:- `rasa init`

3. Define Intents and Entities:

Intents based on what users may want (e.g., greet, goodbye, inform_car_model)

Specify entities for particular values you wish to pull from the user messages (e.g., car_model, location)

4. Write Stories:

Define sample conversations (stories) that your bot should be able to manage. A story is a way of defining conversations by mapping an input to respond from your bot.

5. Train Your Model:

Train both - your NLU model and Dialogue management models that are based on data (NLU Data as well Stories) with the Rasa training commands.

```
rasa train
```

6. Run Your Assistant:

Running Standalone and Interfacing with your assistant either via command line or as a part of another Text/Messaging based interface:

```
rasa shell
```

7. Iterate and Improve:

Run the assistant on different inputs to test and accordingly change its design, training data or responses based on user feedback.

Example Code:

A simple example for the domain and stories format in which you will define your intents, entities, and response as below:

domain.yml:

intents:

- greet
- goodbye

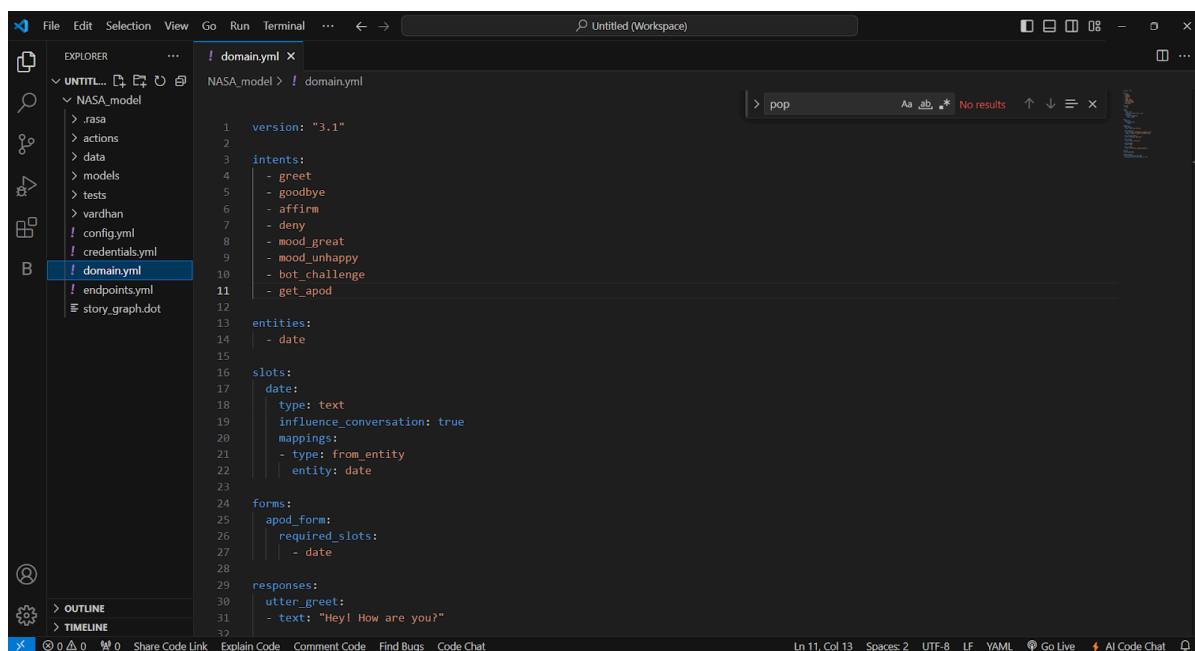
responses:

utter_greet:

- text: "Hello! How can I assist you today?"

utter_goodbye:

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



stories.md:

markdown

happy path

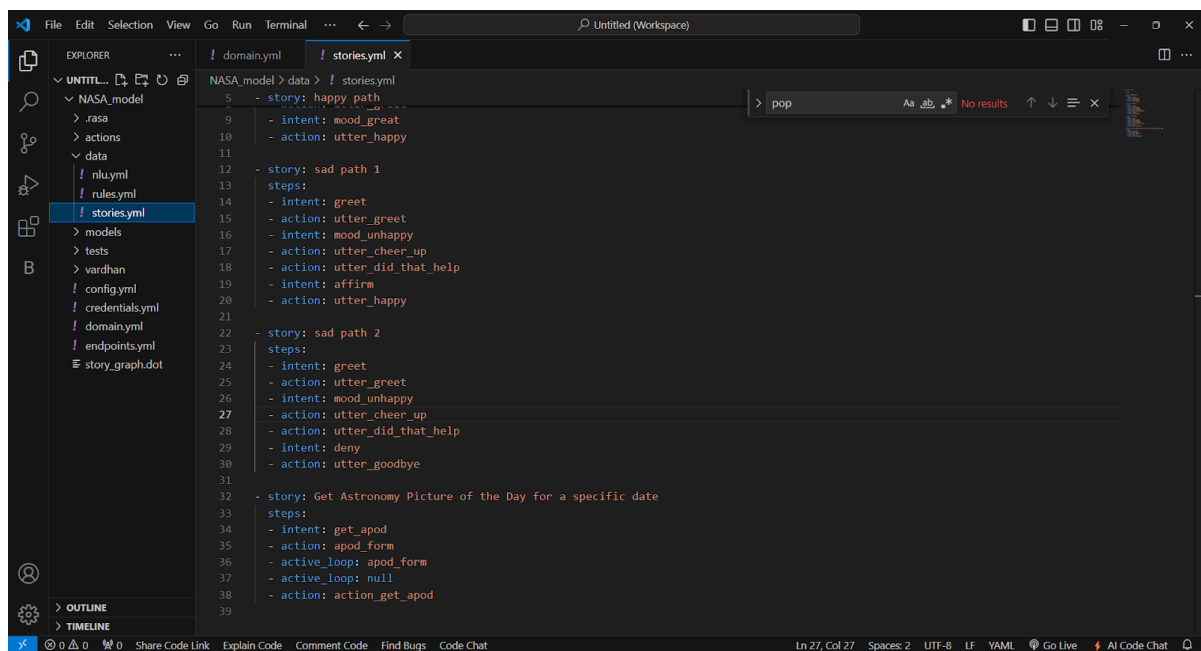
* greet

- utter_greet

* inform_car_model{"car_model": "Tesla Model S"}

- utter_thanks

- utter_goodbye`



Usage Of Rasa Framework:

Rasa is an open-source conversational AI library (MigrationBuilder). It enables the developers to develop machine learning based NLU (Natural Language Understanding) and dialogue management systems. Now we know some important features and use cases of the Rasa framework

NLU (Natural Language Understanding):

Rasa NLU to Understand user messages As the user is making a query, it processes this input in order to extract structured data using

machine-learning models like SVMs (Support Vector Machines) or transformers such as BERT.

Creates intent definitions (what the user is trying to do) and entities (useful information in what he just said), drawn from training data provided by developers. Rasa uses this data to learn how a human can write an utterance.

Dialogue Management:

Dialogue management is handled by Rasa Core (now known as RASA Open Source) It decides how an AI should reply in a conversation context with previous interactions.

Rasa uses policies to decide how the chatbot should act next. developers provide Rasa with a set of examples and replies in configuration files, auxilaing it on understanding what kind of answer is expected from where at which time!

Integration and Customization

Rasa can be integrated with some of the messaging platforms (Slack, Facebook Messenger) and even backend systems(via APIs), in order to build interactive chatbots.

Rasa developers can add custom actions in python to interact with other systems/databases or perform heavy computation.

Training and Evaluation:

Rasa provides utilities for training and testing models. This means that developers can iterate on their models Provide predictionsbDevelopers to refine the accuracy and performance.

AARUHI_BOT

The AARUHI_BOT specially designed to provide users with easy access to a wealth of information about NASA's activities, missions, and discoveries.

1. **Daily Inspiration:** NASA's Image of the Day features stunning and inspiring images related to space and astronomy.
2. **Educational Value:** Each image is accompanied by detailed descriptions that provide educational insights into various space phenomena, missions, and discoveries.
3. **Diverse Subjects:** The images cover a wide range of topics, including galaxies, planets, stars, space missions, and Earth as seen from space.
4. **High-Quality Visuals:** The images are high-resolution and often captured using advanced technology, providing a clear and breathtaking view of the universe.
5. **Historical Significance:** Some images highlight significant events in space exploration history, such as rocket launches, spacewalks, and landmark discoveries.

6. **Accessibility:** The Image of the Day is easily accessible on NASA's website and can be a great resource for educators, students, and space enthusiasts.
7. **Inspiration for Exploration:** These images can inspire interest in science, technology, engineering, and mathematics (STEM) fields and encourage curiosity about the universe.
8. **Cultural Impact:** NASA's images often become iconic, influencing art, literature, and popular culture.
9. **Scientific Discoveries:** Many images showcase new scientific discoveries and advancements in our understanding of the cosmos.
10. **Community Engagement:** NASA encourages public engagement by allowing users to share their favorite images on social media, fostering a global community of space enthusiasts.

1. Create a Virtual Environment (Optional but recommended)

- It's good practice to work in a virtual environment to keep your Python environment isolated.

```
PS D:\NASA_model> python -m venv v
PS D:\NASA_model> v\scripts\activate
(v) PS D:\NASA_model>
```

2. Install Rasa Open Source

- You can install Rasa Open Source and its dependencies using pip.

bash

Copy code

pip install rasa

This command installs the latest stable version of Rasa Open Source.

```
PS D:\NASA_model> .\vardhan\scripts\activate
(vardhan) PS D:\NASA_model> pip install rasa
```

3. Install Additional Dependencies (Optional)

- Depending on your project needs, you may want to install additional dependencies, such as spaCy for NLP processing or Rasa X for a GUI interface.

bash

Copy code

pip install rasa[spacy]

```
pip install rasa[spacy]
```

Replace spacy with other components you may need, such as rasa[transformers] for using transformer-based models.

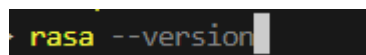
4. Validate Installation

- After installation, you can verify if Rasa has been installed correctly by checking the version.

bash

Copy code

```
rasa --version
```

A terminal window with a dark background. The text 'rasa --version' is written in a light blue font, followed by a white cursor bar.

5. Initialize a Rasa Project

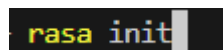
- You can create a new Rasa project using the rasa init command.

bash

Copy code

```
rasa init
```

This command initializes a new Rasa project in the current directory and sets up the basic project structure.

A terminal window with a dark background. The text 'rasa init' is written in a light blue font, followed by a white cursor bar.

6. Start Building Your Assistant

- Once your project is initialized, you can start defining your NLU (Natural Language Understanding) data, stories, and responses.

Additional Resources:

Certainly! Once you have Rasa installed and have initialized your project, here are further processes you might undertake when working with Rasa:

1. Define NLU Data

- ****NLU Training Data****: Create training data for Natural Language Understanding (NLU) using Markdown or YAML format. This data includes examples of user messages and their corresponding intents and entities.

```
! nlu.yml x
NASA_model > data > ! nlu.yml
1  version: "3.1"
2
3  nlu:
4  > - intent: greet...
19
20 > - intent: goodbye...
32
33 > - intent: affirm...
41
42 > - intent: deny...
51
52 > - intent: mood_great...
68
69 > - intent: mood_unhappy...
85
86 > - intent: bot_challenge...
92
93 - intent: get_apod
94   examples: |
95     - Show me the Astronomy Picture of the Day for [2021-06-20](date)
96     - What was the Astronomy Picture of the Day on [2022-01-15](date)?
97     - I want to see the picture for [2020-12-01](date)
98     - Give me the APOD for [2023-05-10](date)
99     - Show me the picture of the day for [2019-07-16](date)
100    - I want to see the APOD for [2018-11-01](date)
101    - Can you fetch the picture for [2022-02-20](date)?
102    - Get me the Astronomy Picture of the Day for [2021-03-25](date)
103    - I would like to see the APOD on [2020-04-13](date)
104    - Provide the APOD for [2017-08-21](date)
105    - Show me APOD for [2019-12-25](date)
106    - What was the APOD on [2020-01-01](date)?
107    - Fetch the APOD for [2021-05-05](date)
108    - Can you show the APOD for [2022-09-30](date)?
109    - I want to know the picture of the day on [2023-06-14](date)
110
111  regex:
112    - name: date
113      pattern: '\d{4}-\d{2}-\d{2}'
114
```

2. Stories

- Definition: A story in Rasa is a representation of a conversation between a user and the AI assistant. It consists of a sequence of steps (dialogue turns) where each step represents an action taken by either the user or the assistant.

```
! stories.yml X
NASA_model > data > ! stories.yml
1  version: "3.1"
2
3  stories:
4
5  > - story: happy path...
11
12 > - story: sad path 1...
21
22 > - story: sad path 2...
31
32 - story: Get Astronomy Picture of the Day for a specific date
33   steps:
34   - intent: get_apod
35   - action: apod_form
36   - active_loop: apod_form
37   - active_loop: null
38   - action: action_get_apod
39
```

3.Domain

General Definition: In a general sense, a domain refers to a specific area or field of knowledge, activity, or interest. For example, in the context of academic studies, "the domain of physics" refers to the entire field of physics.

```
! domain.yml X
NASA_model > ! domain.yml
1  version: "3.1"
2
3  intents:
4    - greet
5    - goodbye
6    - affirm
7    - deny
8    - mood_great
9    - mood_unhappy
10   - bot_challenge
11   - get_apod
12
13  entities:
14    - date
15
16  slots:
17    date:
18      type: text
19      influence_conversation: true
20      mappings:
21        - type: from_entity
22          entity: date
23
24  forms:
25    apod_form:
26      required_slots:
27        - date
28
29 > responses: ...
48
49  actions:
50    - action_get_apod
51
52  session_config:
53    session_expiration_time: 60
54    carry_over_slots_to_new_session: true
55
```

4.Actions:

One common use of actions.py is to define functions that encapsulate specific tasks or actions that your program needs to perform. For example:

```
actions.py X
NASA_model > actions > actions.py > ...
29 # actions.py
30 from typing import Any, Text, Dict, List, Union
31 from rasa_sdk import Action, Tracker
32 from rasa_sdk.executor import CollectingDispatcher
33 from rasa_sdk.events import SlotSet, EventType
34 import requests
35
36 class ActionGetApod(Action):
37
38     def name(self) -> Text:
39         return "action_get_apod"
40
41     def run(self, dispatcher: CollectingDispatcher,
42             tracker: Tracker,
43             domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
44
45         api_key = 'Ndjn8d1vUB1VYaNHcsAK4j9U8htObMUXV2AR3m5s'
46         date = tracker.get_slot('date')
47
48         if not date:
49             dispatcher.utter_message(text="Please provide a date in the format YYYY-MM-DD.")
50             return []
51
52         url = f'https://api.nasa.gov/planetary/apod?api_key={api_key}&date={date}'
53         response = requests.get(url)
54         data = response.json()
55
56         if data:
57             title = data.get('title')
58             explanation = data.get('explanation')
59             image_url = data.get('url')
60             dispatcher.utter_message(text=f"Title: \n{title}\n\nExplanation: \n{explanation}\n\n", image=f"\n{image_url}")
61         else:
62             dispatcher.utter_message(text="Sorry, I couldn't fetch the data from NASA at this time.")
63
64         return [SlotSet("date", None)]
65
```

5. Train Your Assistant

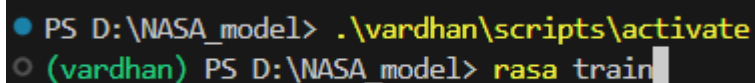
- Use the `rasa train` command to train your assistant based on the defined NLU and Core data.

```
```bash  

rasa train

``
```

This command trains both the NLU model and the dialogue management model.

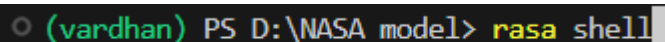


```
● PS D:\NASA_model> .\vardhan\scripts\activate
○ (vardhan) PS D:\NASA_model> rasa train
```

## 6. Test Your Assistant

- After training, you can interact with your assistant in the command line using `rasa shell`.

```
```bash  
  
rasa shell  
  
``
```



```
○ (vardhan) PS D:\NASA_model> rasa shell
```

This allows you to test how your assistant responds to user inputs based on the trained models.

7. Improve and Iterate

- Iterate on your assistant's performance by analyzing conversations, refining training data, adding more examples, adjusting policies, and evaluating performance metrics.

8. Deploy Your Assistant

- Once satisfied with your assistant's performance, you can deploy it on various platforms like your website, messaging platforms, or integrate it into your applications using Rasa's deployment guides.

Additional Tools and Features

- Custom Actions*: Implement custom actions using Python to perform actions like calling APIs, database queries, or any other custom logic.

- Forms: Use forms to handle slot filling in conversations where multiple pieces of information are needed from the user.

- ****Interactive Learning****: Use ``rasa interactive`` to interactively annotate new examples and improve your training data.

Documentation and Community

- Rasa offers comprehensive documentation and an active community where you can find tutorials, guides, and support from other developers working with By following these processes and utilizing the resources available through Rasa's documentation and community, you can build and deploy sophisticated conversational AI assistants tailored to your specific needs.

Reason of the Project

The NASA Bot project was conceived with the vision of making space exploration and astronomy more accessible, engaging, and educational for people of all ages. Here are the key reasons driving this initiative:

Introduction:

The NASA Bot project is designed to provide users with an interactive, informative, and engaging way to access information about NASA's activities, missions, and discoveries. This chatbot aims to enhance public engagement, support educational outreach, and inspire interest in space exploration and STEM fields. Leveraging advanced technologies, the NASA Bot offers real-time updates, multimedia content, and educational resources, making space exploration more accessible and exciting for everyone.

Objective:

The primary objectives of the NASA Bot project are:

- To foster public interest and engagement in NASA's missions and space exploration.
- To provide educators and students with easy access to NASA's educational resources.
- To deliver real-time updates on space missions, discoveries, and events.
- To inspire interest in STEM fields through interactive and engaging content.
- To make NASA's vast repository of information more accessible to a global audience.

Scope and Methodology

Scope:

- Provide real-time updates and notifications about NASA missions and events.
- Offer educational content, including articles, videos, and interactive tools.
- Answer user queries related to space, astronomy, and NASA's activities.
- Share multimedia content, such as images and videos, from NASA's archives.
- Engage users with space trivia, facts, and interactive Q&A.

Methodology:

- **Data Collection:** Aggregating data from NASA's APIs, including the Image of the Day, mission updates, and educational resources.
- **Development:** Using chatbot development frameworks such as Rasa to build and deploy the NASA Bot.
- **Integration:** Integrating the bot with various platforms (e.g., social media, websites) to maximize accessibility.
- **User Interaction:** Designing interactive and user-friendly interfaces for engaging conversations.
- **Testing:** Conducting extensive testing to ensure the bot provides accurate and relevant information.
- **Feedback Loop:** Implementing a feedback mechanism to continuously improve the bot based on user input.

Expected Outputs:

- A fully functional chatbot capable of providing detailed and accurate information about NASA's missions and space exploration.
- Enhanced public understanding and interest in space science and technology.
- Increased engagement with NASA's educational resources.
- Greater accessibility to real-time updates and multimedia content related to space exploration.

Implementation:

- **PlanPlanning and Design:** Define the bot's functionalities, user interface, and integration points.
- **Development:** Build the bot using Rasa framework, integrating NASA's APIs for data.
- **Testing:** Conduct rigorous testing to ensure reliability and accuracy.
- **Deployment:** Deploy the bot on chosen platforms (social media, websites, etc.).
- **Monitoring and Maintenance:** Continuously monitor the bot's performance and make necessary updates based on user feedback.

Conclusion:

The NASA Bot project aims to bridge the gap between NASA's wealth of knowledge and the public, making space exploration more engaging and accessible. By leveraging cutting-edge technology, the bot not only supports educational initiatives but also inspires curiosity and interest in the wonders of the universe. Through real-time updates, interactive content, and extensive educational resources, the NASA Bot will serve as a valuable tool for anyone interested in space and science.

References

NASA Open APIs: [NASA API](#)

Rasa Framework: [Rasa](#)

NASA's Educational Resources: [NASA Education](#)