Development Of AI Chatbot To Learn Programming

[1]Arthika Maria Roy G, [2]Nikithaa S, [3]Jayanth Akash V E, [4]Dinesh Kumar S, [5]Koushik Karan G N, [6]Subashini G

[1]Dept. of Robotics and Automation
PSG College of Technology
Coimbatore, India
arthikamaria@gmail.com

[2]Dept. of Robotics and Automation
PSG College of Technology
Coimbatore, India
nikithaa951@gmail.com

[3]Dept. of Robotics and Automation
PSG College of Technology
Coimbatore, India
jayanthakash.ve @gmail.com

[4]Dept. of Robotics and Automation
PSG College of Technology
Coimbatore, India
s.dinesh2613@gmail.com

[5]Dept. of Robotics and Automation
PSG College of Technology
Coimbatore, India
koushikkaran6@gmail.com

[6]Dept. of Robotics and Automation
PSG College of Technology
Coimbatore, India
suba.rae@ psgtech.ac.in

Abstract

With advancements made in the field of Artificial Intelligence, development of AI- powered chatbots that possess communication capabilities just like humans have been developed. Chatbots are based on techniques like Natural Language Processing that can understand the queries put forth by users and respond with them appropriately via a User Interface. They possess the ability to analyze data more accurately, thereby providing the right information to the user. This paper deals with the development of AI Chatbot using RASA which helps users learn about programming. RASA is a tool to develop and build custom AI chatbots with the help of Natural Language Understanding (NLU).

Keywords—*NLU, RASA, AI chatbot, Programming.*

I. INTRODUCTION

Chatbot is a kind of software application used to communicate or interact with humans either through text or by speech, thereby providing direct communication with a human agent. AI chatbots serve as online communication tools through which humans can communicate effectively instead of waiting for human responses. The chatbot classifications are based on various criteria, such as the area of expertise they refer to or service they provide [1]. They are widely used for various purposes like request processing, information gathering, customer service, educational assistance and so on. Chatbots can be embedded and used through any messaging application like Facebook messenger, Telegram, WhatsApp, Line, WeChat or can be used as virtual assistants like Alexa and Siri. They have the ability to interact with humans at all times of the day and not limited to any constraints like time or location. It reduces manpower in various businesses and help humans work effectively. Chatbots are used to recommend sights, hotels, hospitals, fun activities, and even travel plans [2]. Chatbots are an intelligent system being developed using artificial intelligence and natural language processing algorithms. AI-powered chatbots mimic human conversation and can identify the intent behind a person’s query with the help of Natural Language Processing [3] Chatbots have an effective User Interface which makes it easy for humans to communicate and interact effectively. These AI agents have increasingly occupied several roles in Human Machine Communication and have become more socially compatible. They aid in effective online interaction with humans either through voice, text or sometimes even both. Design of the chatbot’s functional framework introduces the theories of RASA NLU and also integrates RASA NLU along with neural network (NN) methods to implement it with the help of entity extraction and recognition [4]. Natural Language Processing is an advanced technique used for understanding human queries automatically by dividing the text into smaller parts. Besides many applications, chatbots are functionally helping in multiple horizons like educational information, schedule information for any lecture courses and grade information [5]. Chatbots are split into different categories based on knowledge, goal, service and response generated. Goal based chatbots are generally given a primary task to achieve like customer interactions. Knowledge based chatbots access the underlying data resources and respond to users appropriately. The developed chatbot is to solve programming interrogations put forth by users in an effective manner without human intervention. Study on both RASA and Botkit indicate that RASA would be more suitable for chatbot development [6]. Thus, RASA was used in the development of AI chatbot to help users learn programming.

II. RASA

RASA is an open-source ML structure to automate conversations with humans. With RASA, anyone can construct and develop provisional assistants on platforms such as Facebook Messenger, Telegram, and also vvoice assistants like Alexa Skills. RASA enables us to build assistants that can provide meaningful contextual conversations with humans. RASA has two modules: One is RASA NLU and the other one is RASA Core. RASA NLU is used for understanding user messages. It is also a part of RASA Open Source that executes entity extraction, intent classification, and response retrieval that helps in building intuitive chatbots. RASA NLU is utilized to comprehend language for chatbots and Artificial Intelligence assistants, and it majorly focuses on intent classification and entity extraction. RASA Core is used for holding discussions and determining what has to be done next. RASA X is a provision in RASA that assists one construct, develop, and establish Artificial intelligence Assistants from the RASA framework. RASA X comprises of a user interface and a REST API and is the latest release from RASA. RASA NLU is where RASA tries to find user texts and tries to identify intent and entity in that text or message. RASA Core helps with contingent message flow. On the basis of the user message, it can trigger RASA Action Server and can predict dialogue as a reply.

With RASA, personalized, automated interactions can be created. It dispenses supple conversational Artificial intelligence for constructing text-based and voice-based assistants and it is widely used by developers, conversational teams, and enterprises. RASA virtual assistant is a flexible architecture that allows to control access to data and deploys on our own infrastructure. It is used and believed by companies in healthcare, banking, and other organizations that are working under strict protocols in order to warrant compliance and support privacy standards. The training data, as well as models in RASA is completely controlled by the developer and it is not shared. Besides, it provides multi-channel customer occurrences that consists of ten built-in messaging channels, and endpoints for custom channels. The other features include the high-performance architecture and versatile, reusable infrastructure. RASA’s strong architecture meets the excess traffic command, without pressurizing on human assistive agents. Moreover, the technology in RASA is transferable across use cases. reply.

A. *RASA X and NLU*

The free toolset RASA X is utilized to develop virtual assistants constructed with the help of RASA Open Source. Combinedly, they comprise of all the attributes to generate robust text-based and voice-based chatbots and assistants. The major features of RASA include extracting synonyms from messages, holding complex conversations, providing interactive learning and integrating API calls. It provides the facility of turning free text in any language into organized data and also assists numerous intents and both pre-trained and custom entities. Besides, it is a completely custom NLU for any industry. Moreover, it retains crucial content and detains front-and-back conversations using Machine learning based dialogue management and handles topic changes in a smooth manner, integrates logic of business into discussion flows. In addition to it, with RASA, it is possible to create training data by conversing with the assistant, and dispense response or comment when a mistake or error is made. Custom actions in RASA can be used to interact with APIs, databases, and other systems, and also bridge with apprehension bases, context management systems, and CRMs.

B. *RASA File System*

The domain.yml file contains all the chatbot intents, entities, slots, and responses predefined which is used to respond to users. This file represents the chatbot's domain of information. The RASA CORE "policy" and RASA NLU "pipeline" are defined in the config.yml file. The pipeline consists of a list of NLU components.

Policies are a combination of rule-based guidelines and machine learning. TED policies are used to identify entities and predict suboptimal actions. Memoization policy attempts to match the stories described in the training data, and when it finds that story, which predicts the next action from the matching story.

The actions.py contains ’action’ classes, each of the action classes defines two methods which are name and run, name method is to return the name of that action and the run method is to execute what the developer wants the chatbot to execute. Anything that the chatbot should do will be defined in the actions class.

The data folder contains three .yml files: nlu, stories, and rules. These three files contain the data necessary for the training of chatbot. The NLU file contains sample text for each entity. The rules file contains a combination of intent and action. The order in which the intents and actions are listed in this file is the order in which these responses are executed. The story is like a rule. However, stories are primarily used to represent start and end goals. Stories are used to identify related conversations and predict appropriate actions at run time.

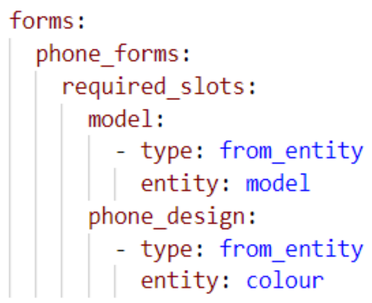
C. *Custom Action*

A custom action allows the user to run any code of user’s choice, including Application Programming Interface calls, queries of database etc. They have the ability of turning on lights, adding an event to a calendar, checking of a user's bank balance, or anything else that can be imagined. Any custom action that is to be used in stories should be added into the actions section of the domain. When a custom action to be executed is forecasted by the dialogue engine, it calls the action server and it must respond with a record of events and responses.

D. *Forms*

A form is defined by adding it to the forms section in domain. The name of the form is same as the name of the action which can be used in stories to manage form executions. Slot mappings for each slot in which the form should fill should be defined as well. One or more slot mappings for each and every slot can be specified accordingly.

The corresponding example in fig 1 phone_form describes the forms provided in domain.yml file along with various entities.



```
forms:
  phone_forms:
    required_slots:
      model:
        - type: from_entity
          entity: model
      phone_design:
        - type: from_entity
          entity: colour
```

Fig. 1. Forms provided in domain.yml file

E. *Slots*

As soon as the form action gets called for the first time, the form gets activated instantly and the user will be enquired for the next required slot value. It is done by searching for a response named utter_ask_<form_name><slot_name> or if the former is not found, utter_ask<slot_name>. These responses must be defined in the domain file for each required slot. A story or rule should be added to activate a form, which indicates when the form should be run by the assistant. On the other hand, once all needed slots are filled, the form gets deactivated automatically.

III. METHODOLOGY

The chatbot which is developed with RASA has a set of .yml files and python files to define the working process of the chatbot. The methodology of creating the necessary data and training the data is shown in fig. 2.

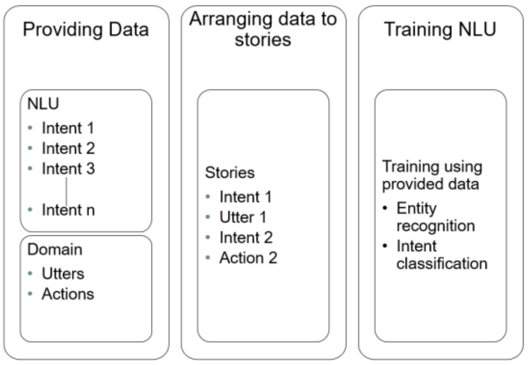
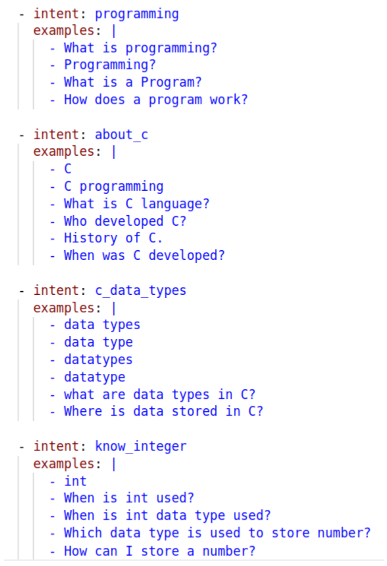


Fig. 2. Methodology of creation and training the data

A. *Providing Data*

1. nlu.yml

NLU training data consists of sample user utterances categorized by intent. Training samples can also include entities. An entity is a structured unit which is responsible for extracting the information from the user's message. The developer can also add additional information to the training data, such as regular expressions and lookup tables, so that the model recognizes intents and entities correctly. Typically, the examples are listed one per line as in the fig. 3. In this project the basic topics of C programming is added to the nlu.yml file with appropriate intents and few examples for training the model.

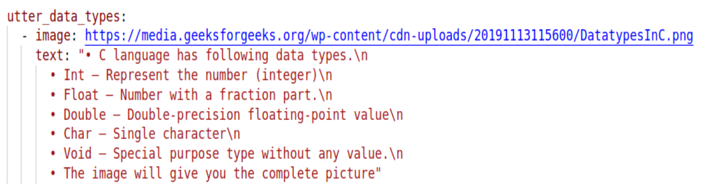


```
- intent: programming
  examples: |
    - What is programming?
    - Programming?
    - What is a Program?
    - How does a program work?

- intent: about_c
  examples: |
    - C
    - C programming
    - What is C language?
    - Who developed C?
    - History of C.
    - When was C developed?

- intent: c_data_types
  examples: |
    - data types
    - data type
    - datatypes
    - datatype
    - what are data types in C?
    - Where is data stored in C?

- intent: know_integer
  examples: |
    - int
    - When is int used?
    - When is int data type used?
    - Which data type is used to store number?
    - How can I store a number?
```

Fig. 3. nlu.yml containing various intents and its examples

2. domain.yml

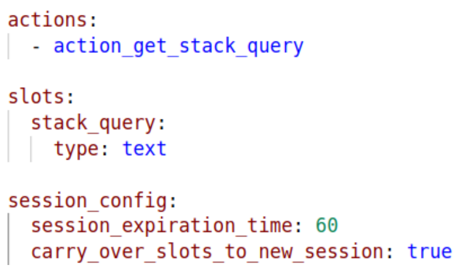
The domain.yml is a file which contains the intents and the entities provided in nlu.yml files. It also contains all the necessary responses which the chatbot refers to as utters as shown in fig. 4.



```
utter_data_types:
  - image: https://media.geeksforgeeks.org/wp-content/cdn-uploads/20191113115600/DatatypesInC.png
    text: "• C language has following data types.\n
      • Int – Represent the number (integer)\n
      • Float – Number with a fraction part.\n
      • Double – Double-precision floating-point value\n
      • Char – Single character\n
      • Void – Special purpose type without any value.\n
      • The image will give you the complete picture"
```

Fig. 4. The response to an intent along with an image URL

It will also contain any python file which is used to access anything out of RASA for example performing operations on a database or in this case to fetch the search results from stack overflow. It can also contain any images which needs to be sent to the user under utters. domain.yml also contains any forms, slots or action that is used in the chatbot as shown in fig. 5.

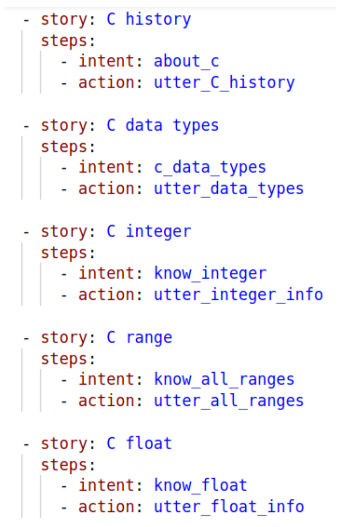


```
actions:
  - action_get_stack_query

slots:
  stack_query:
    type: text

session_config:
  session_expiration_time: 60
  carry_over_slots_to_new_session: true
```

Fig. 5. Actions and slots in domain.yml

B. *Arranging data into stories*

Stories are also a type of training data which is used to train the dialog management model of the chatbot. A story is a representation of a conversation between an AI assistant and a user, in which user input is represented as an intent, and assistant responses and actions are translated into a specific format that is represented as an action name. It is not necessary to deal with the specific content of the message your users are sending as shown in fig 6. Instead, the developer can use the output of the NLU pipeline.



```
- story: C history
  steps:
    - intent: about_c
    - action: utter_C_history

- story: C data types
  steps:
    - intent: c_data_types
    - action: utter_data_types

- story: C integer
  steps:
    - intent: know_integer
    - action: utter_integer_info

- story: C range
  steps:
    - intent: know_all_ranges
    - action: utter_all_ranges

- story: C float
  steps:
    - intent: know_float
    - action: utter_float_info
```

Fig. 6. Stories file with different paths

C. *Policies and Pipeline*

IV. The configuration file defines the policies and elements that the model uses to make predictions primarily based totally on the user's input. The language and pipeline keys specify the components used by the model to make the predictions of the NLU. Policy keys define the policies used by the model to predict next action. The NLU pipeline has components that work in a particular order to exercise input of the user and a systematic output. Specific pipelines used in this project are listed below.

- WhitespaceTokenizer –For every whitespace set apart character sequence a token is created. That is any character not in a-z,A-Z, 0-9_# @ & will be replaced with whitespace.
- RegexFeaturizer – For intent classification and object extraction this makes functionality. The RegexFeaturizer in the format of training data brings about a catalogue of regular expressions. To specify if the expression was established in the user’s message for individual regex a feature is allotted and defined. To clarify classification all the corresponding functionality will be set down into an entity extractor or an intent classifier.
- LexicalSyntacticFeaturizer – Create a function to extract entities. Move by sliding window on each token in user's message and create function based on configuration. Since there is a default configuration, you do not need to mention a profile.
- DIETClassifier –Used for entity recognition and intent classification DIET expanded as (Dual Intent and Entity Transformer) is an architecture based on multitasking. For the dual tasks this architecture is established on a transformer that is shared. With regard to the token input sequence a series of entity tokens is forecasted through a CRF (conditional random field) marker layer over the transformer output sequence. A vector space that is single semantic is formed by the combination of label intents and the output of transformer for the overall statement.
- EntitySynonymMapper –The same value will be mapped to that of the noticed entity values if the synonyms are defined from the training data.
- ResponseSelector – Derived straight from a set of responses from the candidate the response selector part is to create a response accession model that forecasts the bot’s response. The forecasted response which are marked by the dialog manager is the cause of the predictions in this certain model. Similar to the DIET classifier this model implants the response labels and user input in the same area and maintains the exact optimization and the neural network architecture.

There are certain policies and set of rules that the chatbot implements during each step of a conversation.In the respective tandem that the chatbot can utilize there are certain machine learning used. Given below are the policies which are implemented in this project.

- TED Policy – For forecasting next actions and acknowledging entities the Transformer Embedding Dialogue (TED) policies a multitasking architecture are used. For tasks that involve both, an architecture that constitute multiple transmitter encoders are utilized. In accordance to the token input sequence the series of entity tags is forecasted by the CRF (Conditional Random Field) that is a layer, tagging over the encoder output of the user sequence transformer. The dialog transformer encoder output and the action label of the system are implanted in a vector space that is single semantic as forecasted by the next action.
- MemoizationPolicy – The story that corresponds to the training data is recalled by the MemoizationPolicy. The Stories.yml file and the respective story must be in relation to the existing conversation. A confidence of 1.0 is derived from the story matched in the training data as forecasted by the next action. A value of 0.0 for confidence along with "none" is obtained if the match to the conversation is void .

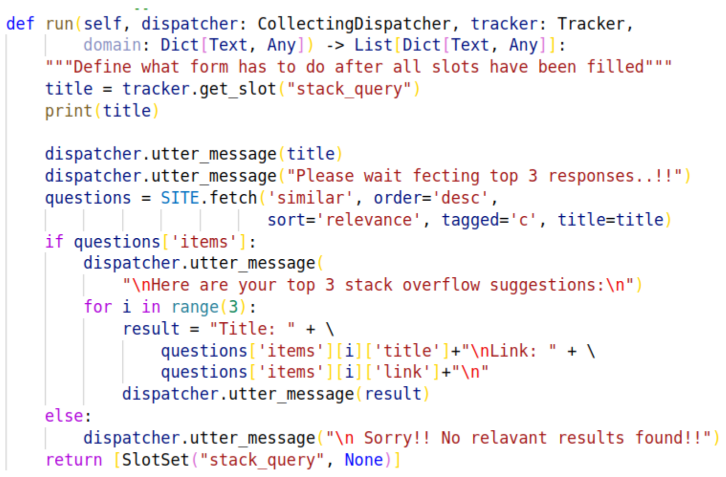
A. *Training the model*

The next step is to train the model using the already created nlu.yml, domain.yml and stories.yml using the command rasa train. The nlu.yml file is to train the NLU of the chatbot. Similarly, the stories.yml file is to train the dialog model. This command will create a model folder and places the trained model in that folder. The rasa train is the command which trains both the NLU and the Dialog model of the chatbot.

If the directory already contains the model, only the modified parts of the model will be retrained. The NLU or the dialog model can be trained individually with the commands run rasa train nlu or rasa train core if there is any change the nlu.yml part or stories.yml alone.

B. *Creating the custom actions*

A custom action is an action that can run any code that the developer wants. This can be used to make a query to a database, or an API call for this project. This project makes use of the API of stack overflow to obtain the search results from stack overflow. The API is called stackapi which is used in the python file actions.py. This file receives the data from the user through the entities and with the help of the run function in the python file as shown in the fig. 7., the program will fetch the top three search results from stack overflow. This action server is started using the command rasa run actions.

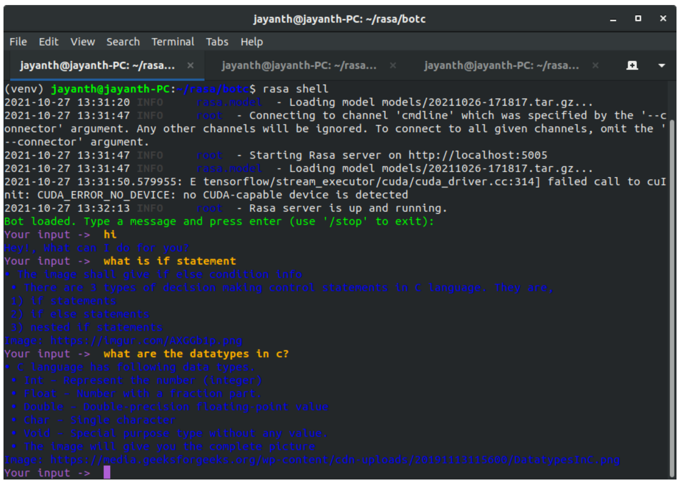


```python
def run(self, dispatcher: CollectingDispatcher, tracker: Tracker,
        domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
    """Define what form has to do after all slots have been filled"""
    title = tracker.get_slot("stack_query")
    print(title)

    dispatcher.utter_message(title)
    dispatcher.utter_message("Please wait fecting top 3 responses..!!")
    questions = SITE.fetch('similar', order='desc',
                           sort='relevance', tagged='c', title=title)
    if questions['items']:
        dispatcher.utter_message(
            "\nHere are your top 3 stack overflow suggestions:\n")
        for i in range(3):
            result = "Title: " + \
                questions['items'][i]['title']+"\nLink: " + \
                questions['items'][i]['link']+"\n"
            dispatcher.utter_message(result)
    else:
        dispatcher.utter_message("\n Sorry!! No relavant results found!!")
    return [SlotSet("stack_query", None)]
```

Fig. 7. actions.py

V. IMPLEMENTATION OF THE CHATBOT

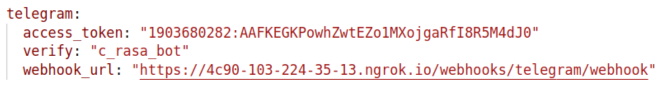
In order to test the chatbot for its correction in delivering the proper message the chatbot can be started in the command line interface using the command rasa shell as shown in the fig. 8**.**



```
jayanth@jayanth-PC: ~/rasa/botc
(venv) jayanth@jayanth-PC:~/rasa/botc$ rasa shell
2021-10-27 13:31:20 INFO     rasa.model  - Loading model models/20211026-171817.tar.gz...
2021-10-27 13:31:47 INFO     root  - Connecting to channel 'cmdline' which was specified by the '--c
onnector' argument. Any other channels will be ignored. To connect to all given channels, omit the '
--connector' argument.
2021-10-27 13:31:47 INFO     root  - Starting Rasa server on http://localhost:5005
2021-10-27 13:31:47 INFO     rasa.model  - Loading model models/20211026-171817.tar.gz...
2021-10-27 13:31:50.579955: E tensorflow/stream_executor/cuda/cuda_driver.cc:314] failed call to cuI
nit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
2021-10-27 13:32:13 INFO     root  - Rasa server is up and running.
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input ->  hi
Hey!, What can I do for you?
Your input ->  what is if statement
• The image shall give if else condition info
 • There are 3 types of decision making control statements in C language. They are,
 1) if statements
 2) if else statements
 3) nested if statements
Image: https://imgur.com/AXGGb1p.png
Your input ->  what are the datatypes in c?
• C language has following data types.
 • Int - Represent the number (integer)
 • Float - Number with a fraction part.
 • Double - Double-precision floating-point value
 • Char - Single character
 • Void - Special purpose type without any value.
 • The image will give you the complete picture
Image: https://media.geeksforgeeks.org/wp-content/cdn-uploads/20191113115600/DatatypesInC.png
Your input ->
```

Fig. 8. rasa shell

In order to implement the chatbot in telegram, a telegram bot is created using botfather and its access code is generated and also it is necessary to tunnel the local host. This is done using ngrok application which will tunnel the local host to another server which can be captured by the telegram api. This access code and the webhook URL needs to be updated in the credentials.yml file as shown in fig. 9.

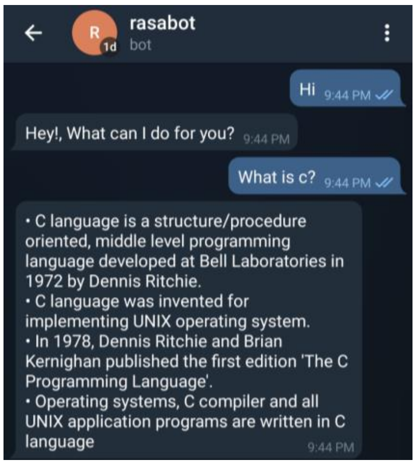


```
telegram:
  access_token: "1903680282:AAFKEGKPowhZwtEZo1MXojgaRfI8R5M4dJ0"
  verify: "c_rasa_bot"
  webhook_url: "https://4c90-103-224-35-13.ngrok.io/webhooks/telegram/webhook"
```

Fig. 9. Credentials.yml

After these changes in the credentials.yml file, it is required to start the RASA server from the local machine using the command rasa run. This command will host the server in localhost:5005/webhook which will be tunneled trough ngrok server to reach the telegram server.

While the RASA server, RASA action server and ngrok are running, the chatbot can be accessed from the telegram bot which was created earlier as shown in fig. 10.



rasabot
bot

Hi 9:44 PM

Hey!, What can I do for you? 9:44 PM

What is c? 9:44 PM

• C language is a structure/procedure oriented, middle level programming language developed at Bell Laboratories in 1972 by Dennis Ritchie.
• C language was invented for implementing UNIX operating system.
• In 1978, Dennis Ritchie and Brian Kernighan published the first edition 'The C Programming Language'.
• Operating systems, C compiler and all UNIX application programs are written in C language 9:44 PM

Fig. 10. Chatbot implementation in telegram

VI. Conclusion

The AI-powered chatbot has been implemented in telegram and it can answer all queries related to C programming language that are put forth by the users. The developed bot can understand basic questions on C language such as data types and respond via an image. For other complex queries it tries to fetch top three results from stack overflow which is a website for professional and enthusiastic beginners in programming. With the help of the top three search links provided by the chatbot, the user can extract the necessary information and clear their doubts. The chatbot serves as an effective learning tool for beginners and also provides the sufficient information by understanding the language of humans. This is achieved by simply extracting keywords from the query and understanding it with the help of Natural Language Processing.

VII. Further work

Future work is to host the RASA server in a docker container which is an open-source software development platform which permits packaging of application in containers that can be ported to any system. Further, the chatbot is now capable of providing queries related to C programming. It can be extended to other languages like Python, Java, C++ and so on. The developed Chatbot is capable of interacting only through text which can be extended to make speech as an option to ease user experience. Future work is to incorporate more programming languages and also make it more user friendly and intuitive.

References

1. Eleni Adamopoulou, Lefteris Moussiades, “An Overview of Chatbot Technology”, “Artificial Intelligence Applications and Innovations. IFIP Advances in Information and Communication Technology, vol 584. Springer”, 2020.
2. Reem Alotaibi, Ahlam Ali, Haya Alharthi, Renad Almehamadi, “AI Chatbot for Tourism Recommendations”, “International Journal of Interactive Mobile Technologies, Vol. 14, No. 19”, 2020.
3. Lalwani, Tarun Bhalotia, Shashank Pal, AshishRathod, Vasundhara Bisen, Shreya, “Implementation of a Chatbot System using AI and NLP”, “International Journal of Innovative Research in Computer Science & Technology (IJIRCST) Vol. 6, Issue-3”, 2018.
4. Anran Jiao, “An Intelligent Chatbot System Based on Entity Extraction Using RASA NLU and Neural Network”, “IOP Conf. Series: Journal of Physics: Conf. Series 1487”,2020.
5. Yurio Windiatmoko, Ahmad Fathan Hidayatullah, Ridho Rahmadi, “Developing FB Chatbot Based on Deep Learning Using RASA Framework for University Enquiries”, “IOP Conference Series: Materials Science and Engineering, 1077”, 2021.
6. Md Imran Pavel, “Comparing Chatbot Frameworks: A Study of RASA and Botkit”, “Master’s Thesis Faculty of Information Technology and Communication Sciences Supervisor: Zheying Zhang”, May 2020