

Conversational Chatbot

For MoTA (Ministry of Tribal Affairs) Scholarship Website

Report

Research Practice

BITS G540

in

Master of Engineering (Software Systems)

Submitted By

Srijon Mallick (2022H1120279P)

Under the supervision of

Dr. Navneet Goel

Professor

Department of Computer Science and Information Systems



**Birla Institute of Technology and Science
Pilani, Pilani Campus, Rajasthan (India) May, 2023**

Project Introduction:

In this project, our task is to build a conversational chatbot for MoTA (Ministry of Tribal Affairs) website [[Click here](#) to go to website]

Chatbot:

There are several types of chatbots that can be developed depending on their functionality and purpose. Here are some of the most common types:

1. **Rule-based chatbots:** These chatbots follow a predefined set of rules or decision trees to respond to user inputs. They can handle simple queries and provide straightforward responses.
2. **Retrieval-Based Chatbots:** These chatbots use predefined responses based on keyword matching or pattern recognition. They analyse the input and provide the best-matched response from a pre-existing database. They are commonly used in customer support and frequently asked questions (FAQ) scenarios.
3. **Generative Chatbots:** Generative chatbots are powered by machine learning algorithms and can generate responses based on the input received. They can understand natural language and produce human-like responses. Open Ai's GPT-3 model, which powers me, is an example of a generative chatbot.
4. **Virtual assistants:** These chatbots are designed to perform specific tasks for users, such as booking appointments, setting reminders, or ordering food. They use AI and machine learning to understand user requests and provide relevant information or perform the desired task.
5. **Voice-activated chatbots:** These chatbots are designed to interact with users through voice commands. They are often integrated into smart speakers or other devices and can perform tasks like playing music, setting reminders, or answering questions.
6. **Hybrid chatbots:** These chatbots combine multiple technologies to provide a more comprehensive experience for users. For example, a hybrid chatbot could use rule-based programming to handle simple queries and AI-powered algorithms to handle AR5R5more complex ones.

Methodology:

In this project we have implemented retrieval based chatbots for MoTA website using LSTM (Long Short Time Memory) sequential network. Let's introduce what is LSTM? – LSTM is advanced RNN (Recurrent Neural Network).

Recurrent Neural Network:

Usually in the domain of deep learning when we implement any neural network model, in such case the network takes one single input and give some output. For example- Convolutional Neural Network (CNN) takes a single image as input and give feature vector of that image as output. But what if we have to handle sequence data like sequence text, audio, or video. In that case we have to retain the output of previous state as the network will give output for entire sequence data.

Recurrent Neural Network is a deep learning model that can handle sequence of data. The model can be used for text generation, text classification and text summarization etc.

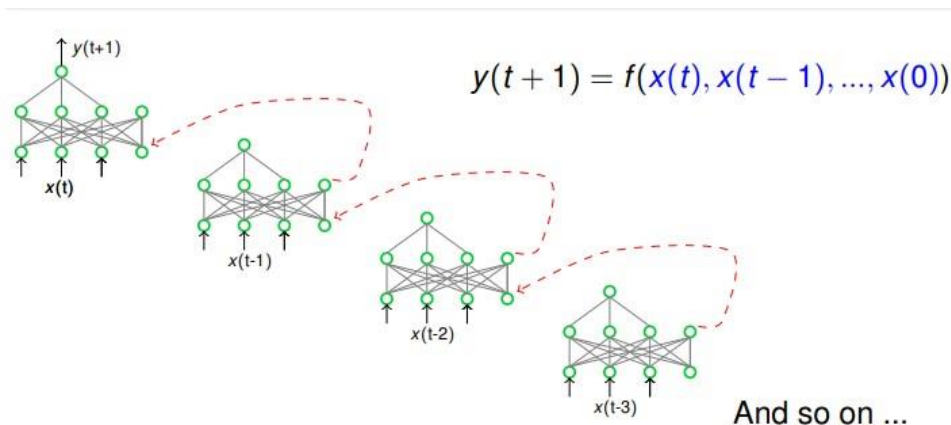


Figure 1. RNN

Above figure [Figure 1] shows how RNN works.

Due to gradient vanishing problem and exploding in RNN, LSTM comes. Now let's see how LSTM works.

LSTM (Long Short Time Memory):

In traditional RNNs, the gradients can become extremely small when backpropagating through time, which can result in a loss of information over

time. LSTM networks, on the other hand, use memory cells, gates, and forget gates to allow for the retention or removal of information from the previous time step, which helps to alleviate the vanishing gradient problem. LSTM uses a special circuit to overcome the problem.

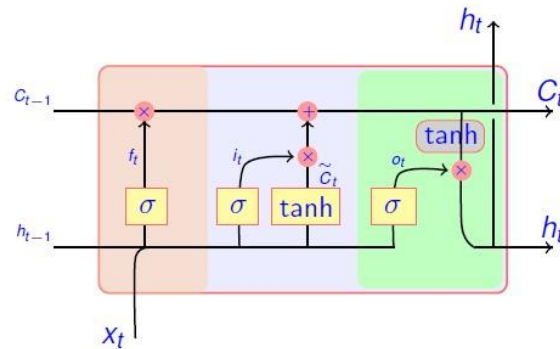


Figure 2 LSTM

Details of circuit is as follows:

1. **Forget gate:** This gate takes the previous cell state and the current input, and it decides which information to keep and which to forget by producing a forget vector. The forget vector is an array of values between 0 and 1 that is multiplied element-wise with the previous cell state, allowing the network to retain important information while discarding irrelevant information.
2. **Input gate:** This gate takes the current input and the previous hidden state and decides which information to store in the cell state by producing an input vector. The input vector is an array of values between 0 and 1 that is multiplied element-wise with the candidate cell state, which is a new vector of information to be added to the cell state.
3. **Cell state:** This is the long-term memory of the LSTM network. It is updated by combining the previous cell state with the forget vector and the input vector.
4. **Output gate:** This gate takes the cell state and the current input and decides which information to output by producing an output vector. The output vector is an array of values between 0 and 1 that is multiplied element-wise with the current cell state to produce the output.

Implementation Details:

We have implemented the chatbot in Keras framework. To implement the chatbot we have followed the following steps:

1. Preparing Dataset: As we have implemented retrieval based chatbot, we need a dataset containing a set of questions and corresponding responses from the website information.

Here we have collected data from the MoTA scholarship (Ministry of Tribal Affairs) website and made a JSON file that contains the data in the below format:

```
Users > Srijon Mallick > OneDrive > Desktop > {} website.json > [] intents
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": ["Hi", "Hey", "Is anyone there?", "Hello", "Hay"],
      "responses": ["Hello", "Hi", "Hi there"]
    },
    {
      "tag": "goodbye",
      "patterns": ["Bye", "See you later", "Goodbye"],
      "responses": ["See you later", "Have a nice day", "Bye! Come back again"]
    },
    {
      "tag": "thanks",
      "patterns": ["Thanks", "Thank you", "That's helpful", "Thanks for the help"],
      "responses": ["Happy to help!", "Any time!", "My pleasure", "You're most welcome!"]
    },
    {
      "tag": "about MOTA scholarship",
      "patterns": ["What are the scholarships schemes available?", "tell me something about scholarships", "tell me something about MOTA scholarship"],
      "responses": ["Ministry of Tribal Affairs is implementing following Scholarship Schemes for ST students in the country-\n1. Pre matric scheme\n2. Post matric scheme\n3. Higher secondary scheme\n4. Degree scheme\n5. Post graduate scheme\n6. Research scheme\n7. Special scholarship for ST students\n8. Special scholarship for ST students\n9. Special scholarship for ST students\n10. Special scholarship for ST students"]
    },
    {
      "tag": "pre matric scheme",
      "patterns": ["What is pre matric scholarship scheme", "tell me something about pre matric scheme", "how is pre matric scheme"],
      "responses": ["This is a Centrally Sponsored Scheme implemented through States/UTs who are responsible for Inviting applications"]
    },
    {
      "tag": "pre matric eligibility",
      "patterns": ["What is eligibility for pre-matric scholarship?", "tell me the eligibility criteria for pre matric scholarship"],
      "responses": ["1. Applicable to students who are studying in Classes IX - X. \n2. Parental income from all sources should be less than Rs. 2.25 lakh per annum"]
    },
    {
      "tag": "pre matric payment",
      "patterns": ["What is the pre matric scholarship amount for this academic year?", "what is the amount of pre matric scholarship"],
      "responses": ["Scholarships are paid @ Rs.225/- per month for Day Scholars and @ Rs.525/- per month for Hostellers, for a period of 10 months"]
    }
  ]
}
```

Figure 3 Dataset

In the above-mentioned dataset, a set of questions and answers is mentioned for each possible intents of users.

2. Required Packages:

Below packages are used for the implementation.

```
[8] import json
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from keras.layers.core import Dense, Activation, Dropout
from keras.layers import LSTM
from keras.models import Sequential
```

Figure 4 Packages

3. Extracting training data & Pre-processing (text vectorizing):

In this step we have extracted all the possible questions mentioned in the dataset and their corresponding intent label (tag as in the dataset).

```
with open('/content/drive/MyDrive/RP Work/website.json') as file:
    data = json.load(file)
    training_sentences = []
    training_labels = []
    labels = []
    responses = []
    for intent in data['intents']:
        for pattern in intent['patterns']:
            training_sentences.append(pattern)
            training_labels.append(intent['tag'])
            responses.append(intent['responses'])

    if intent['tag'] not in labels:
        labels.append(intent['tag'])
    num_classes = len(labels)
    len(training_sentences)
```

Figure 5 Data Loading

As we can see in the above code snippet, The variable “*training_sentences*” holds all the training data (which are the sample messages/questions in each intent category) and the “*training_labels*” variable holds all the target intent labels correspond to each training data.

Then we use “*LabelEncoder ()*” function provided by scikit-learn to convert the target labels into a model understandable form. Here we are assigning a number to each training labels or intents as in the below code snippet:

```
lbl_encoder = LabelEncoder()
lbl_encoder.fit(training_labels)
training_labels = lbl_encoder.transform(training_labels)
training_labels

array([38, 38, 38, 38, 38, 37, 37, 37, 93, 93, 93, 93,  0,  0,  0,
       0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 90, 90, 90, 90, 87,
       87, 87, 87, 87, 87, 87, 87, 87, 89, 89, 89, 89, 89, 89, 89])
```

Figure 6 Label / Intent Encoding

Next, we vectorize our text data corpus by using the “*Tokenizer*” class and it allows us to limit our vocabulary size up to some defined number. When we

use this class for the text pre-processing task, by default all punctuations will be removed, turning the texts into space-separated sequences of words, and these sequences are then split into lists of tokens. They will then be indexed or vectorized. We can also add “oov_token” which is a value for “out of token” to deal with out of vocabulary words(tokens) at inference time as in the below code:

```
[11] vocab_size = 1000
     embedding_dim = 16
     max_len = 20
     oov_token = "<OOV>"

     tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)
     tokenizer.fit_on_texts(training_sentences)
     word_index = tokenizer.word_index
     sequences = tokenizer.texts_to_sequences(training_sentences)
     padded_sequences = pad_sequences(sequences, truncating='post', maxlen=max_len)
```

Figure 7 Text Vectorization

In the above code, vocab size is the total number of unique words in the set of training sentences (all questions set), embedding dimension refers to the number of dimensions used to represent each word in the vector space, and max len defines the maximum number of words that can be included in a sequence. If the sequence length is longer than the maximum length, it needs to be truncated, and if it is shorter, it needs to be padded.

The “*pad_sequences*” method is used to make all the training text sequences into the equal size.

4. Model Preparing:

In this section we will discuss how the model has been made to train it with the gathered dataset.

As said in the previous sections we have used LSTM model in the implementation. Below is the code snippet of the model summary.

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(32, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

model.summary()
```

In the above code sequential architecture has been used.

Below is the detailed explanation of the code.

Creating the model:

- ``model = Sequential ()`` initializes a sequential model, which is a linear stack of layers.
- Layers are added to the model one by one, defining the model's architecture.

Embedding layer:

- ``model.add (Embedding (vocab_size, embedding_dim, input_length=max_len))`` adds an embedding layer as the first layer of the model.
- The embedding layer is responsible for converting input sequences into dense vectors (embeddings) of fixed size.
- Parameters:
- ``vocab_size``: The size of the vocabulary, i.e., the number of unique words in the input.
- ``embedding_dim``: The dimensionality of the dense embedding vectors.
- ``input_length``: The length of input sequences (the maximum sequence length).

LSTM layers:

- ``model.add (LSTM (128, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))`` adds the first LSTM layer.
- ``model.add (LSTM (128, dropout=0.2, recurrent_dropout=0.2))`` adds the second LSTM layer.
- LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) layer that is particularly effective in processing sequential data like text.
- Parameters:
- ``128``: The number of memory units or LSTM cells in the layer. Increasing this number can allow the model to capture more complex patterns but also increases computational requirements.
- ``dropout``: The dropout rate, which helps prevent overfitting by randomly setting a fraction of input units to 0 during training.
- ``recurrent_dropout``: The dropout rate for the recurrent connections of the LSTM layer.
- ``return_sequences=True``: This parameter is set to True in the first LSTM layer to ensure that it returns the output sequences of all time steps.
- The second LSTM layer does not have ``return_sequences=True``, so it only returns the output of the last time step (the final hidden state).

Dense layers:

- ``model.add (Dense (32, activation='relu'))`` adds a dense layer with 32 units and a ReLU activation function.
- ``model.add (Dense (num_classes, activation='softmax'))`` adds the output layer with ``num_classes`` units (the number of classes in the output) and a softmax activation function.

- Dense layers are fully connected layers, which perform a linear operation followed by a non-linear activation.
- The ReLU activation function ('relu') introduces non-linearity and helps the model learn more complex representations.
- The softmax activation in the output layer is commonly used for multi-class classification problems, as it produces a probability distribution over the classes.

Compiling the model:

- `model.compile (loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])` `compiles the model and configures the training process.
- Parameters:
- `loss`:` The loss function to minimize during training. `'sparse_categorical_crossentropy'` is suitable for multi-class classification when the target values are integers.
- `optimizer`:` The optimizer algorithm used for gradient descent and updating the model weights. `'adam'` is a popular choice known for its efficiency and good performance.
- `metrics`:` A list of metrics to evaluate the model's performance during training and testing. Here, we use `'accuracy'` to monitor the accuracy of the model.

Model summary:

- `model.summary ()` `displays a summary of the model, showing the layer types, output shapes, and number of

5. Model Training:

Now the above model trained with the sample questions ['training_sentences'] and their respective intents ['tag']. Code snippet is as below.

```

▶ epochs = 200
  history = model.fit(padded_sequences, np.array(training_labels), epochs=epochs, batch_size=30)

📄 Epoch 1/200
  14/14 [=====] - 9s 115ms/step - loss: 4.5375 - accuracy: 0.0167
  Epoch 2/200
  14/14 [=====] - 2s 116ms/step - loss: 4.4932 - accuracy: 0.0144
  Epoch 3/200
  14/14 [=====] - 2s 116ms/step - loss: 4.4932 - accuracy: 0.0144

```

Here we have used 200 epochs as our dataset is small and to get training accuracy 96%. But the most disadvantage is that due to high epochs the model may get overfitted.

We have implemented Bi-directional LSTM for the same dataset but within 100 epochs accuracy is coming above 90 % but due to small dataset the output is almost same.

6. Output (Deployed Chatbot):

After training the model by our domain specific data, we saved the trained model. Now we have made one android app to make the chatbot usable for user. Below are the use cases for **User & Admin**.

User: User can do conversation with the chatbot and after completion of the conversation user can give rating to the particular conversation.

Admin: One admin can analyse all the conversations rating wise and can see the question answers asked against the particular conversation so that admin can analyse how the chatbot is performing.

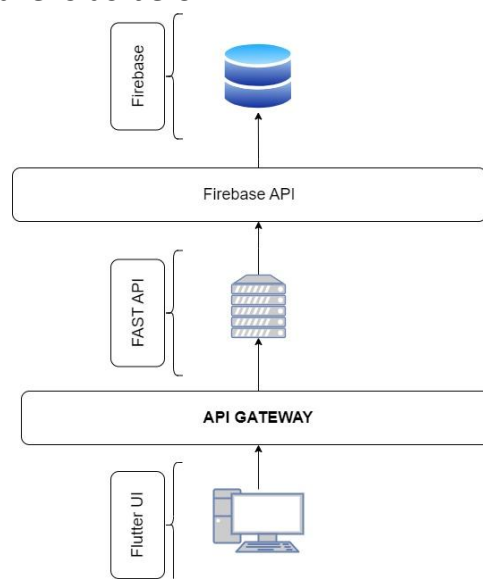
Technology stack used in the application:

UI – Framework: Flutter- Dart

Backend: Python Fast API Server (As microservices)

Database: Firebase.

Application architecture is as below:

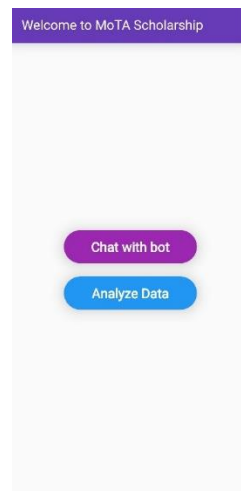


Detailed explanation is as below:

UI – Framework:

UI contains a home page from where user can chat with bot and admin can analyse conversational data (Here two use cases are implemented in the

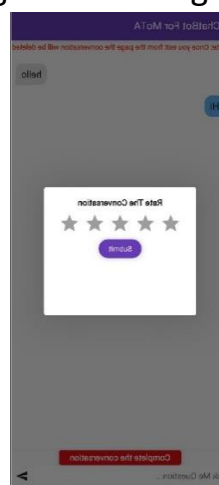
same app- but separate app for user and admin can be made): Screenshot of home is as below:



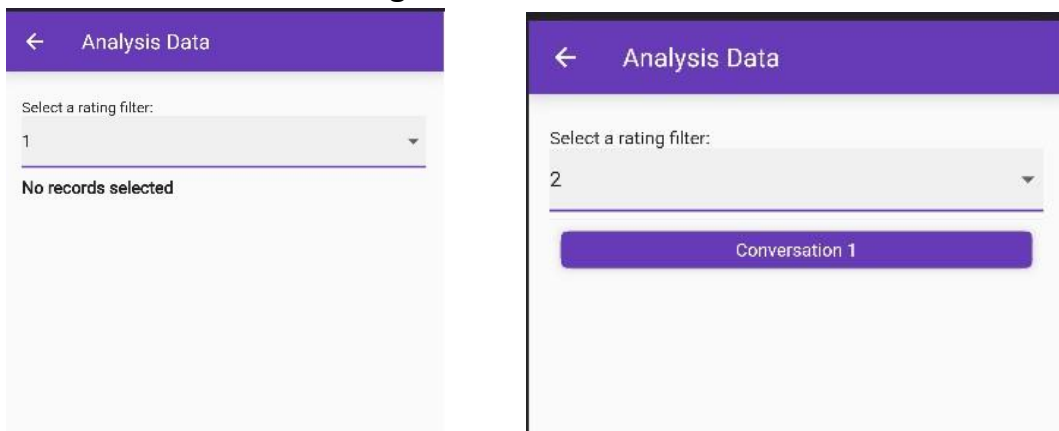
Second page is for conversational chat screen where user can ask questions and chatbot will give answers. Screenshot is as below.



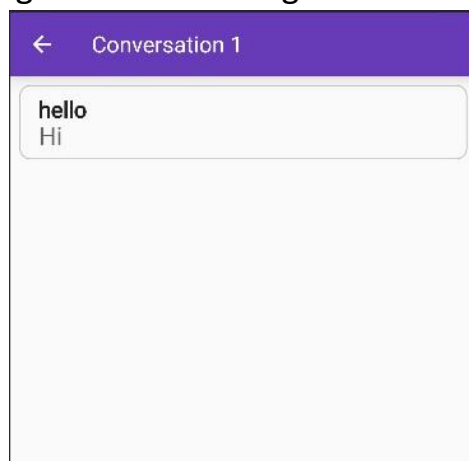
Here user can complete the conversation by pressing the red button. Once the conversation is over the user has to give feedback or rating for the conversation. Follow below figure for rating.



Admin can analyse data by pressing analyse button in the home page as shown in the previous screenshot. Then admin can select rating wise conversations as below figures.



As we can see above that if we select rating as 2 then it will show all the conversations having rating 2. Now we can see the question answers of that conversation by clicking on it as below figure



All the data transfer is happening with UI through https request as API call.

Backend: [Fast API- Python]:

Backend is written in python that runs all the backend functions like fetching data from database and sending to user UI. Sample code snippet/APIs is given below.

```
@app.get("/get/{input}")
@app.put("/add/{question}/{answer}/{conid}")
@app.get("/getrating/{id}")
@app.put("/addconversation/")
@app.post("/update/{id}/{rating}")
@app.get("/getall/{id}")
```

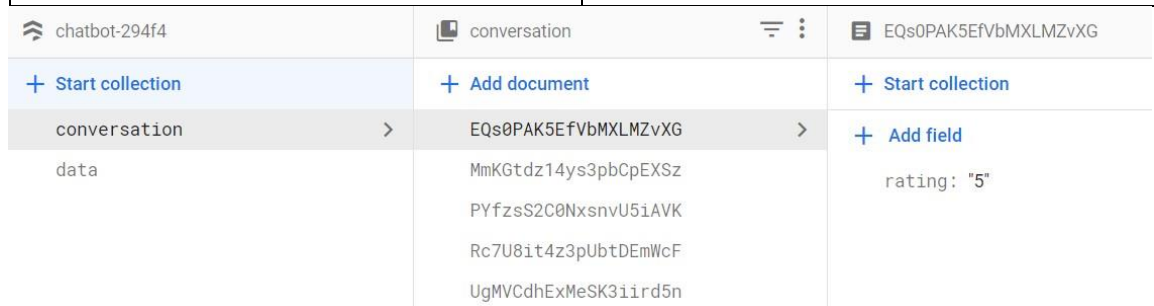
```
@app.get("/loadfilterconversations/{rating}")
```

```
return query.where(field_path, op_string, value)
INFO: 127.0.0.1:62834 - "GET /loadfilterconversations/3 HTTP/1.1" 200 OK
INFO: 127.0.0.1:62844 - "GET /loadfilterconversations/5 HTTP/1.1" 200 OK
INFO: 127.0.0.1:62903 - "GET /loadfilterconversations/0 HTTP/1.1" 200 OK
INFO: 127.0.0.1:62903 - "GET /getall/EQs0PAK5EfVbMXLMZvXG HTTP/1.1" 200 OK
INFO: 127.0.0.1:62937 - "GET /loadfilterconversations/5 HTTP/1.1" 200 OK
1/1 [=====] - 2s 2s/step
INFO: 127.0.0.1:63013 - "GET /get/hi HTTP/1.1" 200 OK
INFO: 127.0.0.1:63013 - "OPTIONS /addconversation/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:63013 - "PUT /addconversation/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:63013 - "OPTIONS /add/hi/Hi%20there/PYfzsS2C0NxsnuU5iAVK HTTP/1.1" 200 OK
INFO: 127.0.0.1:63013 - "PUT /add/hi/Hi%20there/PYfzsS2C0NxsnuU5iAVK HTTP/1.1" 200 OK
INFO: 127.0.0.1:64111 - "PUT /add/hello/Hi/hBQIHvznVEJ92lZCz0s8 HTTP/1.1" 200 OK
INFO: 127.0.0.1:64630 - "POST /update/hBQIHvznVEJ92lZCz0s8/2 HTTP/1.1" 200 OK
```

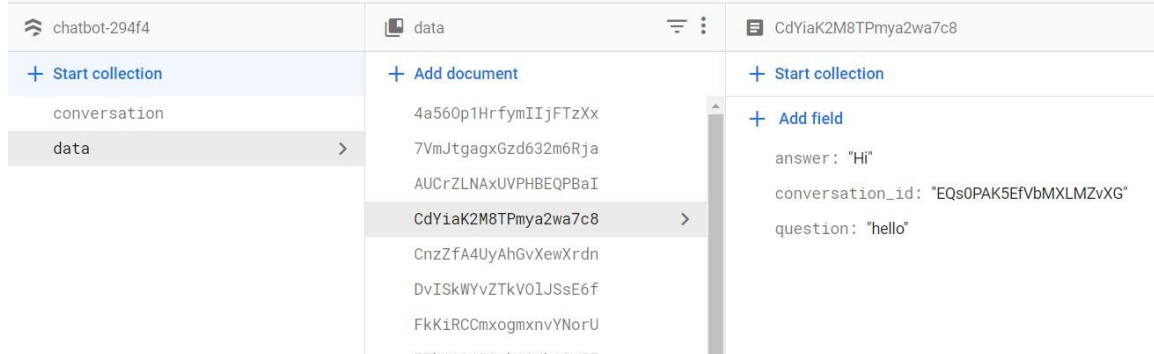
Database [Firestore]: Data is physically stored in the firebase. Fast API connects to firebase API to fetch data, add data and update data. Database table structure is as follows:

Two tables- 1. Conversation 2. Data

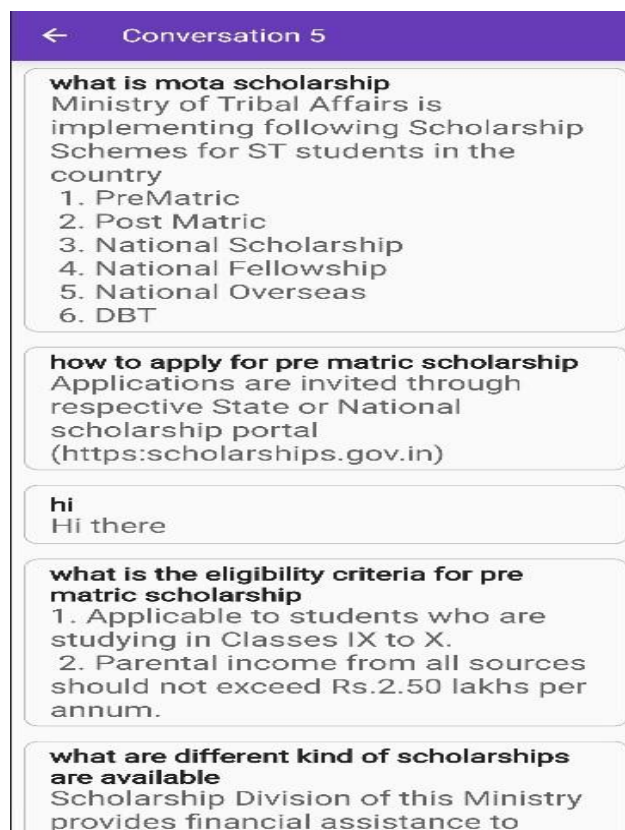
Conversation		
Document id	Rating	
EQs0PAK5EfVbMXLMZvXG	5	



Data			
Document id	Question	Answer	Conversation id
CdYiaK2M8TPmya2wa7c8	Hello	hi	EQs0PAK5EfVbMXLMZvXG



Sample question answers are shown as below:



Limitations:

- This chatbot is trained with very a smaller number of data, that's why some kind of model overfitting is there. So, it cannot handle any question that are in the known context but the user question has some difficult words.
- If we give out of context question to the chatbot, instead of giving a smart answer like "I can't understand your question", the chatbot will give some known answer that is totally wrong.
- This chatbot cannot handle the situation if user made a spelling mistake in asking question.

Future Scope:

Our first target is to overcome all the limitations that the chatbot is currently having and secondly, we will make our chatbot more advanced. Our next move may be as follow:

Approach 1:

As we have small amount of data for our domain, then we will use pre trained model like – BERT, GPT which are already trained on a large set of English words. And we can easily fine tune the models according to our dataset using Hugging Face libraries.

In this approach, dataset will contain a set of question answers against one intent. By training the BERT model on our data, the model will tell the intent for user input and according to that intent the answers will be selected randomly.

Approach 2:

As approach 1, we will use the dataset that contains a set of question answers pairs under one intent. We will use pre trained model like BERT, GPT to train the same on our dataset by finetuning.

In this approach, the BERT or GPT model will give the intent of user input and then we will check the cosine similarity between user input and all the question under that intent. The most similar question will be selected along with its answer.