PROJECT REPORT

➢ ABSTRACT:

This report outlines the implementation of a chatbot using Python and machine learning techniques, as well as creating a chatbot with Dialogflow. The objective is to create a conversational agent capable of understanding and responding to user queries effectively. The methodology involves leveraging natural language processing (NLP) algorithms and training models on relevant datasets. By using Python libraries for machine learning and Dialogflow's intuitive interface, we aim to develop a robust and user-friendly chatbot solution.

➢ OBJECTIVE:

The primary objective of this project is to implement a chatbot system that can understand natural language input from users and provide relevant responses. Specifically, we aim to achieve the following:

1. Develop a Python-based chatbot using machine learning algorithms for natural language processing.
2. Train the chatbot on a diverse dataset to improve its understanding and response accuracy.
3. Create chatbot with Dialogflow to enhance its conversational abilities and provide a seamless user experience.
4. Evaluate the performance of the chatbot in terms of response accuracy, user satisfaction, and scalability.

➢ INTRODUCTION:

Chatbots have become increasingly prevalent in various applications, ranging from customer service to personal assistants. These intelligent systems use natural language processing techniques to interpret user queries and generate appropriate responses. In this project, we focus on implementing a chatbot using Python and machine learning, with the additional integration of Dialogflow to enhance its capabilities.

Python provides a rich ecosystem of libraries for machine learning and natural language processing, making it an ideal choice for developing chatbots. The libraries used in this project are numpy ,pickle, keras, tensorflow. By leveraging these tools, we can train models to understand the nuances of human language and respond intelligently.

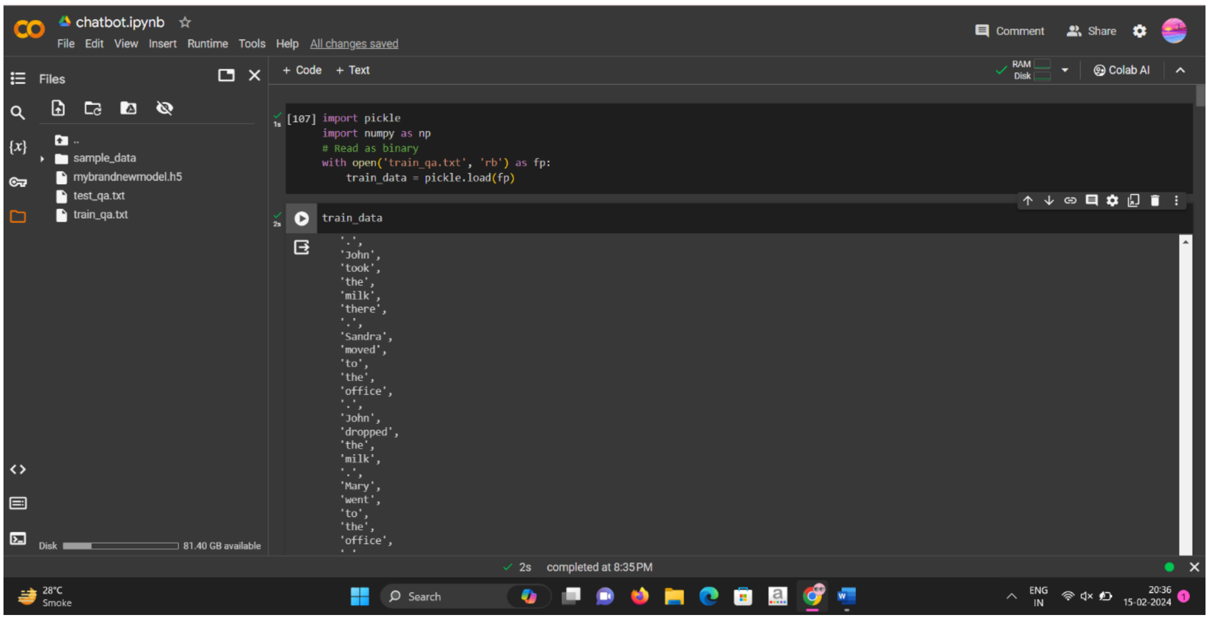
Dialogflow, on the other hand, offers a user-friendly platform for building conversational agents with minimal coding effort. It provides features such as intent recognition, context management, and entity extraction, which are essential for creating engaging chatbot experiences.

Using the strengths of Python and Dialogflow, we aim to develop a chatbot that can effectively interact with users, understand their queries, and provide accurate responses. This report outlines the methodology employed to achieve this goal, code and presents the findings and conclusions from the implementation process.

➢ METHODOLOGY:

1. Data Collection: Gather a diverse dataset of conversational examples to train the chatbot model. This dataset should cover a wide range of topics and user queries to ensure robustness. The dataset used in this project is from meta and is divided into story, questions and answer.
2. Preprocessing: Clean and preprocess the dataset to remove noise, tokenize text.
3. Feature Engineering: Extract relevant features from the preprocessed text data to represent input queries in a format suitable for machine learning algorithms.
4. Model Training: Train machine learning models, on the preprocessed data to learn patterns and relationships between user queries and responses.
5. Evaluation: Evaluate the performance of the trained models using metrics such as accuracy, precision, recall, and F1-score. Fine-tune the models as needed to improve performance. We have used accuracy and loss function for this.
6. Testing and Deployment: Test the integrated chatbot system to ensure functionality and usability. Deploy the chatbot to the desired platform or environment for public use.
7. Using Dialogflow: Use Dialogflow to enable real-time interactions with users. Define intents, entities, within Dialogflow to handle user queries and generate responses.

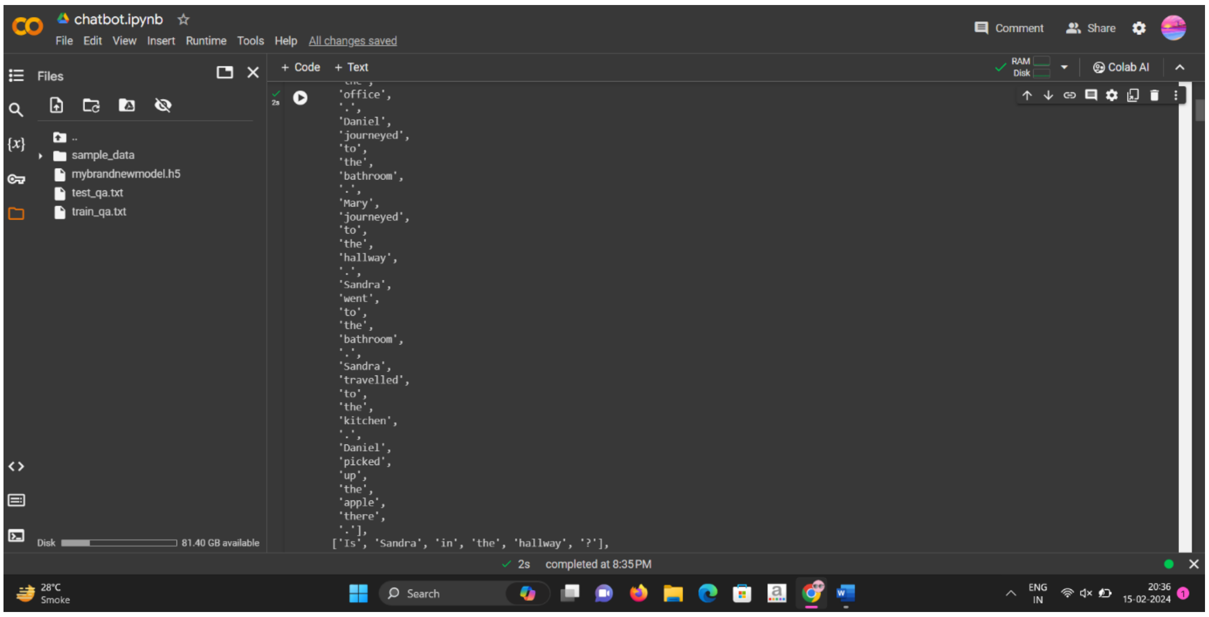
Code: (Python Chatbot implementation):



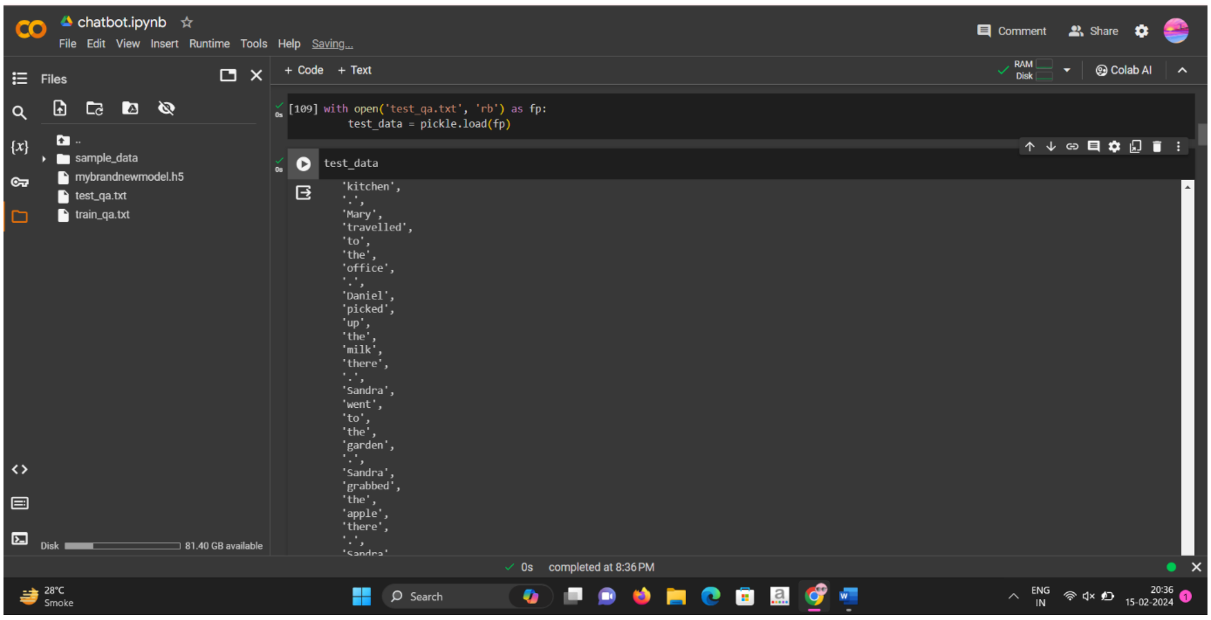
```
[107] import pickle
      import numpy as np
      # Read as binary
      with open('train_qa.txt', 'rb') as fp:
          train_data = pickle.load(fp)
```

```
train_data
```

```
  '.',
  'John',
  'took',
  'the',
  'milk',
  'there',
  '.',
  'Sandra',
  'moved',
  'to',
  'the',
  'office',
  '.',
  'John',
  'dropped',
  'the',
  'milk',
  '.',
  'Mary',
  'went',
  'to',
  'the',
  'office',
```



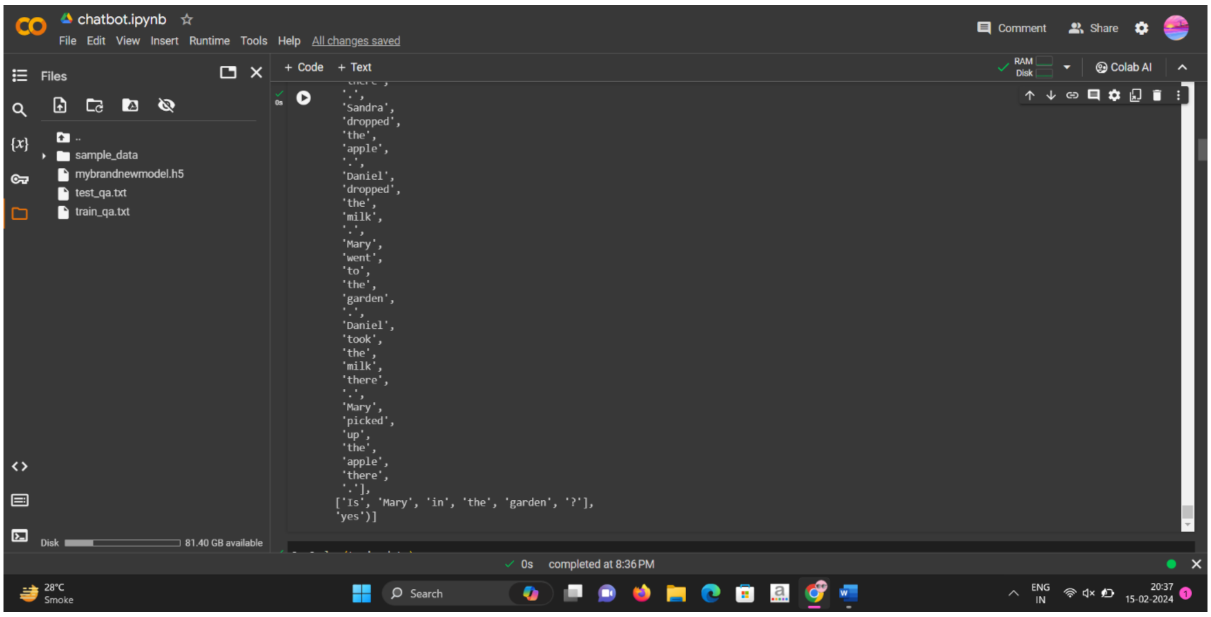
```
  'office',
  '.',
  'Daniel',
  'journeyed',
  'to',
  'the',
  'bathroom',
  '.',
  'Mary',
  'journeyed',
  'to',
  'the',
  'hallway',
  '.',
  'Sandra',
  'went',
  'to',
  'the',
  'bathroom',
  '.',
  'Sandra',
  'travelled',
  'to',
  'the',
  'kitchen',
  '.',
  'Daniel',
  'picked',
  'up',
  'the',
  'apple',
  'there',
  '.'],
 ['Is', 'Sandra', 'in', 'the', 'hallway', '?'],
```



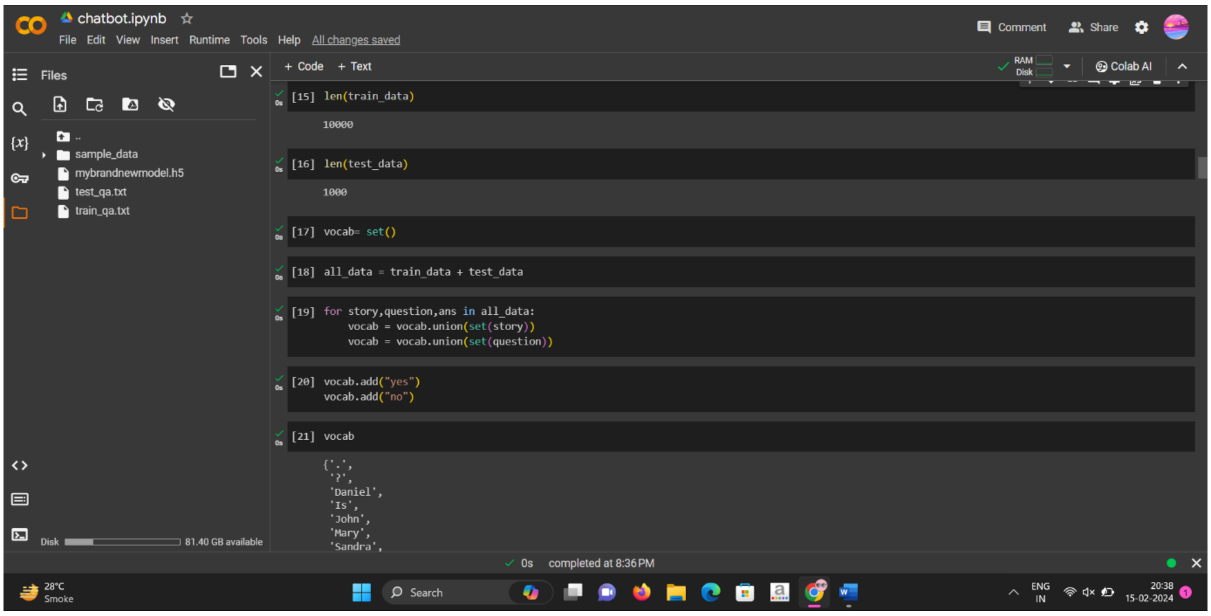
```
[109] with open('test_qa.txt', 'rb') as fp:
          test_data = pickle.load(fp)
```

```
test_data
```

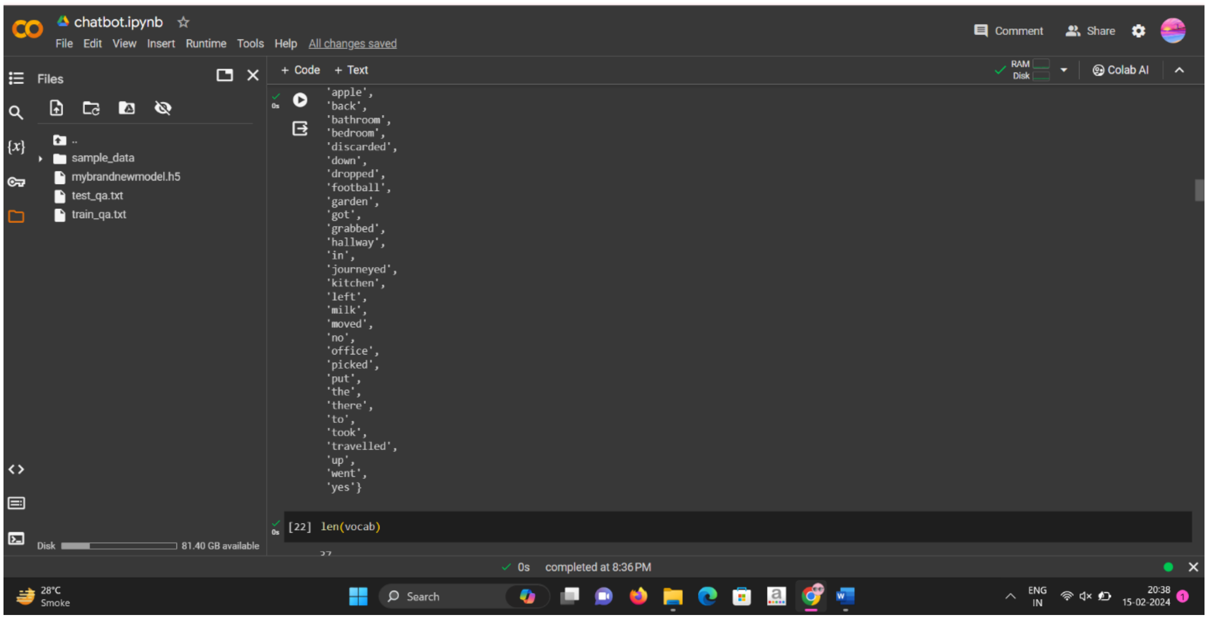
```
   'kitchen',
   '.',
   'Mary',
   'travelled',
   'to',
   'the',
   'office',
   '.',
   'Daniel',
   'picked',
   'up',
   'the',
   'milk',
   'there',
   '.',
   'Sandra',
   'went',
   'to',
   'the',
   'garden',
   '.',
   'Sandra',
   'grabbed',
   'the',
   'apple',
   'there',
   '.',
```



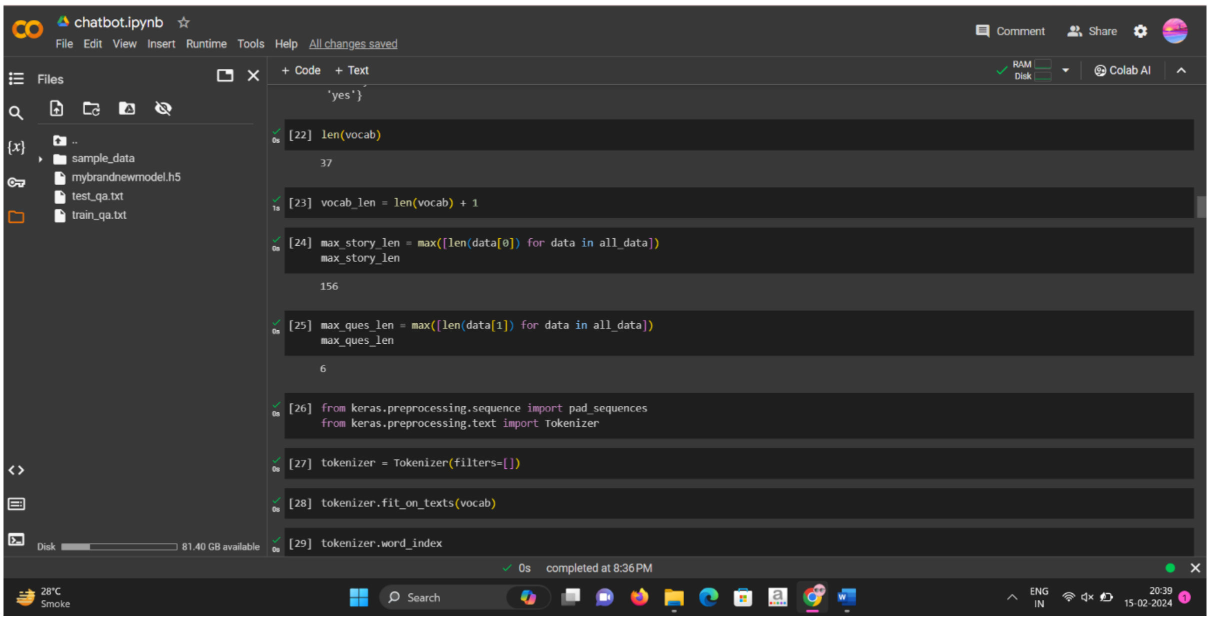
```
   '.',
   'Sandra',
   'dropped',
   'the',
   'apple',
   '.',
   'Daniel',
   'dropped',
   'the',
   'milk',
   '.',
   'Mary',
   'went',
   'to',
   'the',
   'garden',
   '.',
   'Daniel',
   'took',
   'the',
   'milk',
   'there',
   '.',
   'Mary',
   'picked',
   'up',
   'the',
   'apple',
   'there',
   '.'],
  ['Is', 'Mary', 'in', 'the', 'garden', '?'],
  'yes')]
```



```
[15] len(train_data)

     10000

[16] len(test_data)

     1000

[17] vocab= set()

[18] all_data = train_data + test_data

[19] for story,question,ans in all_data:
         vocab = vocab.union(set(story))
         vocab = vocab.union(set(question))

[20] vocab.add("yes")
     vocab.add("no")

[21] vocab

     {'.',
      '?',
      'Daniel',
      'Is',
      'John',
      'Mary',
      'Sandra',
```



```
      'apple',
      'back',
      'bathroom',
      'bedroom',
      'discarded',
      'down',
      'dropped',
      'football',
      'garden',
      'got',
      'grabbed',
      'hallway',
      'in',
      'journeyed',
      'kitchen',
      'left',
      'milk',
      'moved',
      'no',
      'office',
      'picked',
      'put',
      'the',
      'there',
      'to',
      'took',
      'travelled',
      'up',
      'went',
      'yes'}

[22] len(vocab)
```



```
'yes'}
```

```
[22] len(vocab)
```

```
37
```

```
[23] vocab_len = len(vocab) + 1
```

```
[24] max_story_len = max([len(data[0]) for data in all_data])
     max_story_len
```

```
156
```

```
[25] max_ques_len = max([len(data[1]) for data in all_data])
     max_ques_len
```

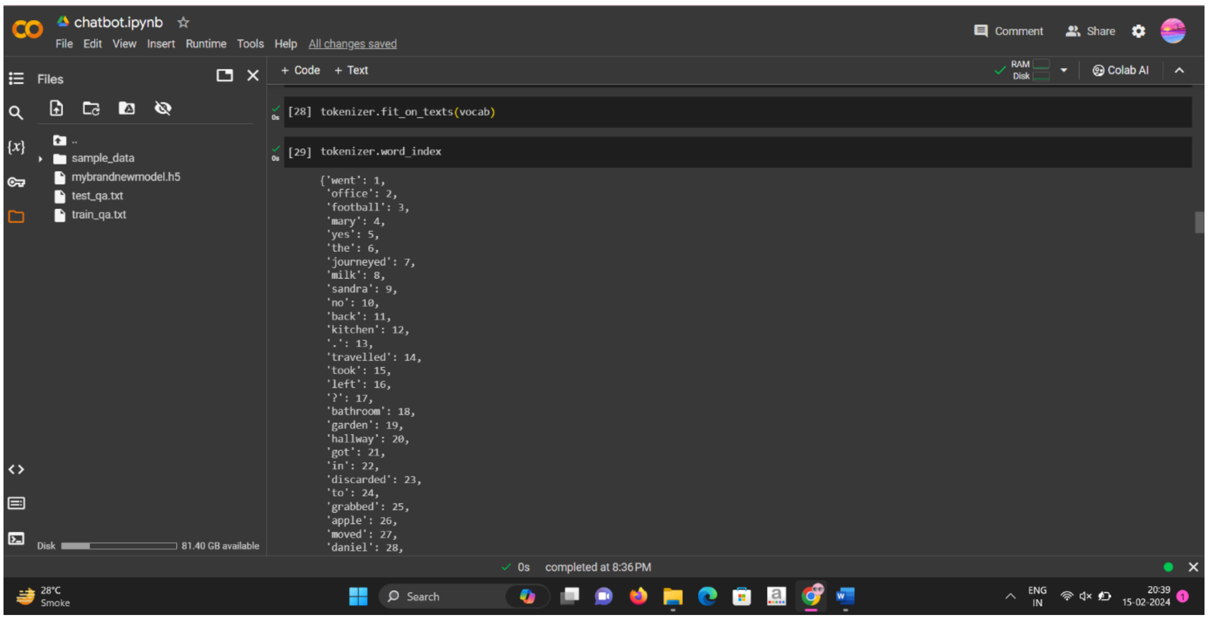
```
6
```

```
[26] from keras.preprocessing.sequence import pad_sequences
     from keras.preprocessing.text import Tokenizer
```

```
[27] tokenizer = Tokenizer(filters=[])
```

```
[28] tokenizer.fit_on_texts(vocab)
```

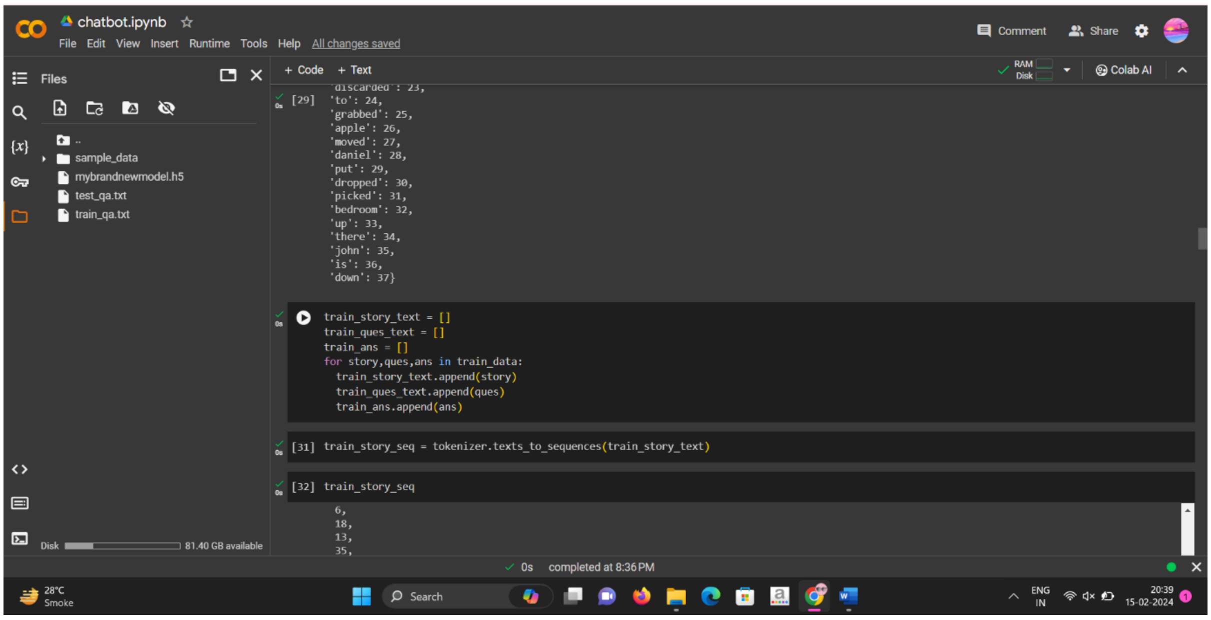
```
[29] tokenizer.word_index
```



```
[28] tokenizer.fit_on_texts(vocab)
```

```
[29] tokenizer.word_index
```

```
{'went': 1,
 'office': 2,
 'football': 3,
 'mary': 4,
 'yes': 5,
 'the': 6,
 'journeyed': 7,
 'milk': 8,
 'sandra': 9,
 'no': 10,
 'back': 11,
 'kitchen': 12,
 '.': 13,
 'travelled': 14,
 'took': 15,
 'left': 16,
 '?': 17,
 'bathroom': 18,
 'garden': 19,
 'hallway': 20,
 'got': 21,
 'in': 22,
 'discarded': 23,
 'to': 24,
 'grabbed': 25,
 'apple': 26,
 'moved': 27,
 'daniel': 28,
```



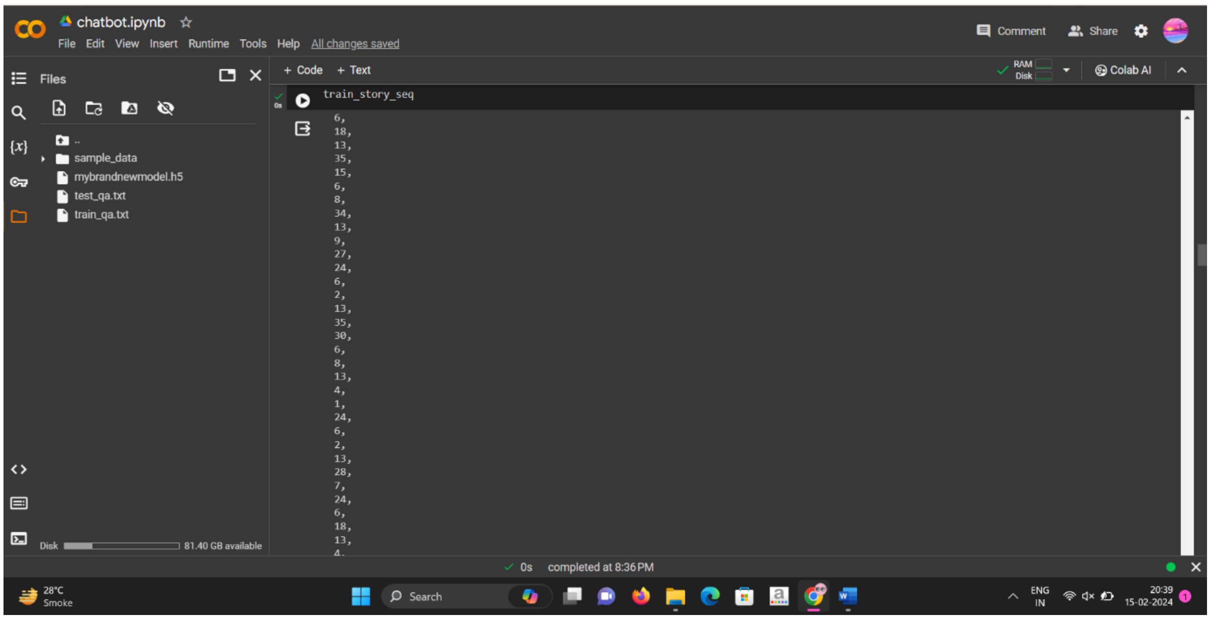
```
 'discarded': 23,
[29] 'to': 24,
 'grabbed': 25,
 'apple': 26,
 'moved': 27,
 'daniel': 28,
 'put': 29,
 'dropped': 30,
 'picked': 31,
 'bedroom': 32,
 'up': 33,
 'there': 34,
 'john': 35,
 'is': 36,
 'down': 37}
```

```
train_story_text = []
train_ques_text = []
train_ans = []
for story,ques,ans in train_data:
  train_story_text.append(story)
  train_ques_text.append(ques)
  train_ans.append(ans)
```

```
[31] train_story_seq = tokenizer.texts_to_sequences(train_story_text)
```

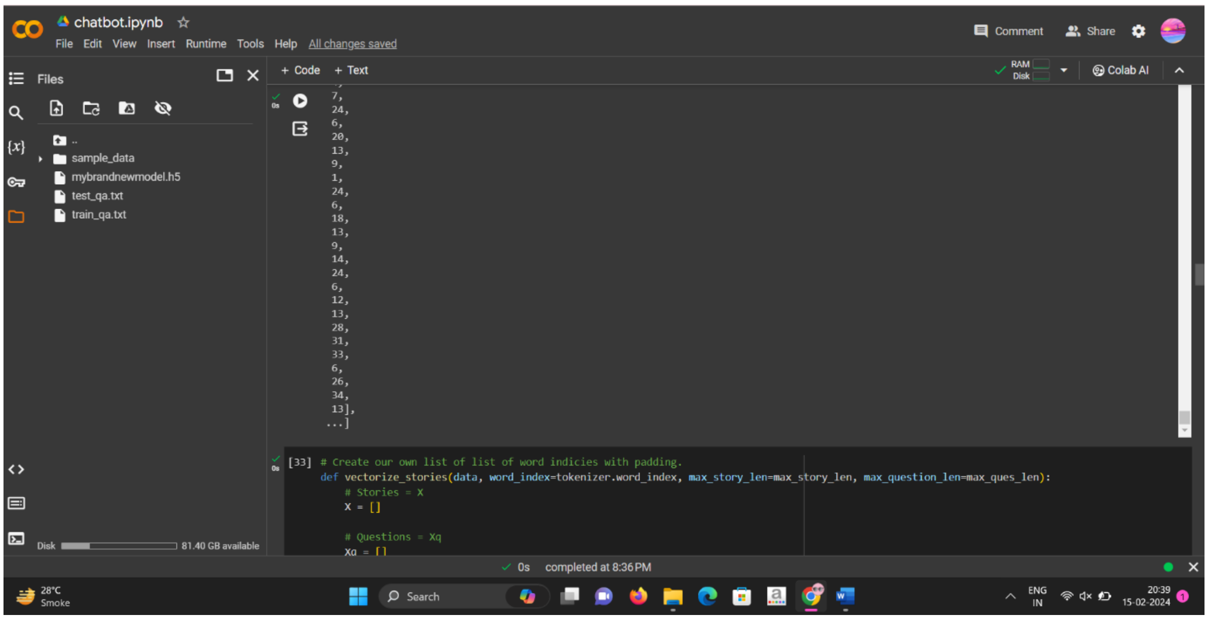
```
[32] train_story_seq
```

```
  6,
  18,
  13,
  35,
```



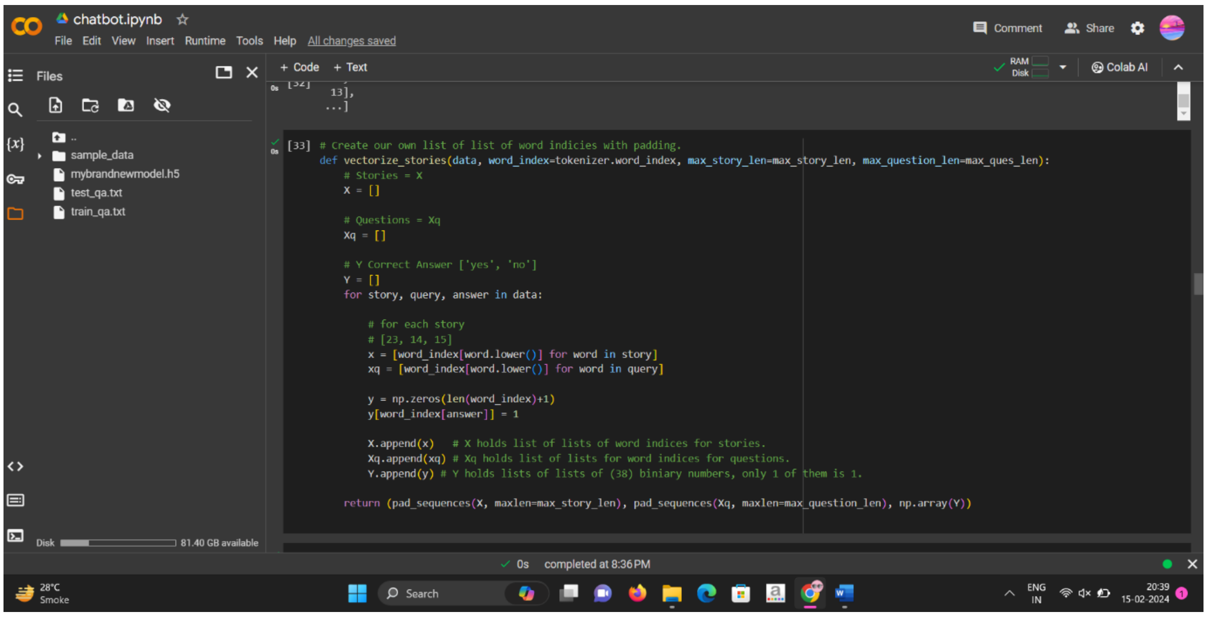
```
train_story_seq
```

```
  6,
  18,
  13,
  35,
  15,
  6,
  8,
  34,
  13,
  9,
  27,
  24,
  6,
  2,
  13,
  35,
  30,
  6,
  8,
  13,
  4,
  1,
  24,
  6,
  2,
  13,
  28,
  7,
  24,
  6,
  18,
  13,
  4,
```



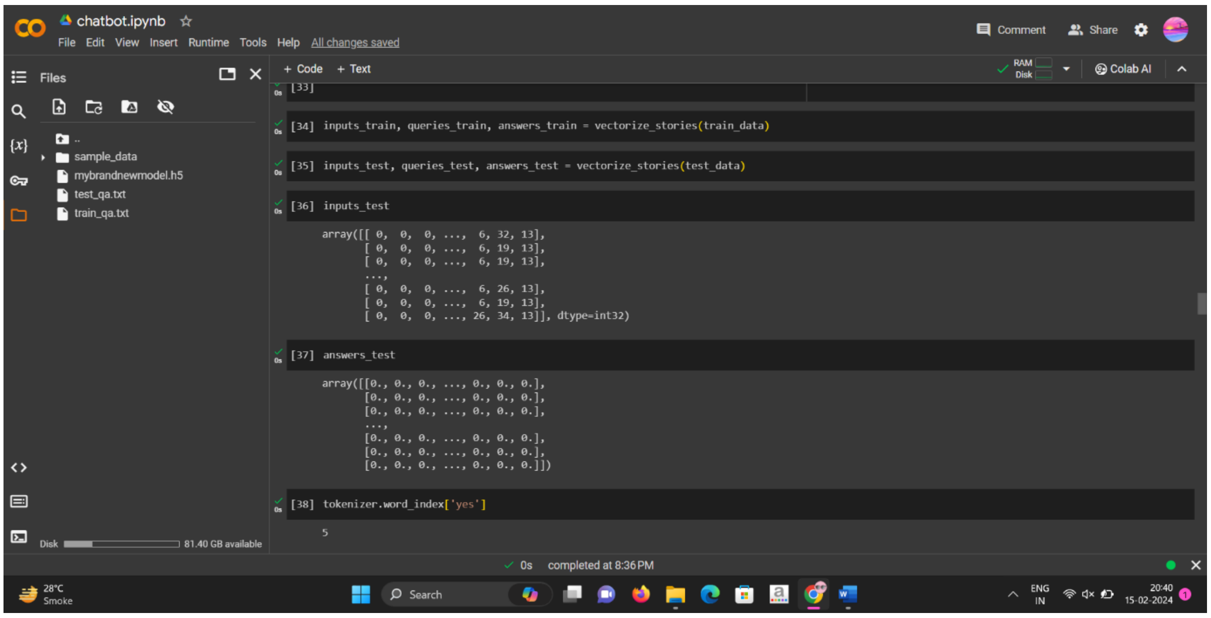
```
   7,
   24,
   6,
   20,
   13,
   9,
   1,
   24,
   6,
   18,
   13,
   9,
   14,
   24,
   6,
   12,
   13,
   28,
   31,
   33,
   6,
   26,
   34,
   13],
  ...]
```

```
[33] # Create our own list of list of word indicies with padding.
     def vectorize_stories(data, word_index=tokenizer.word_index, max_story_len=max_story_len, max_question_len=max_ques_len):
         # Stories = X
         X = []

         # Questions = Xq
         Xq = []
```

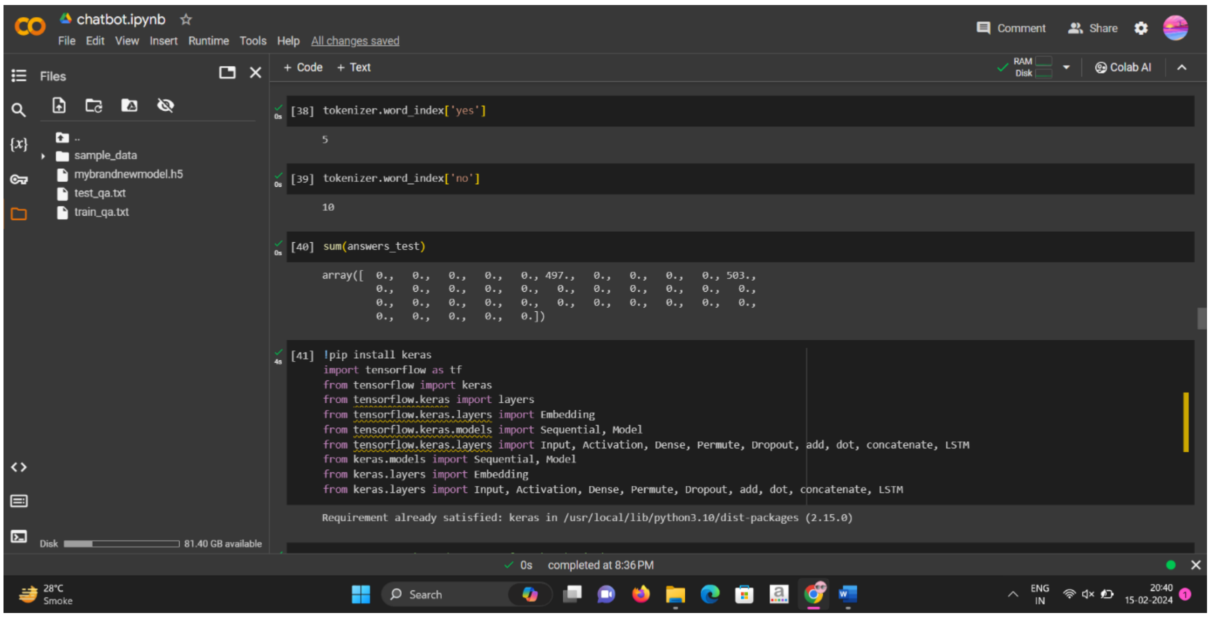


```
  13],
  ...]
```

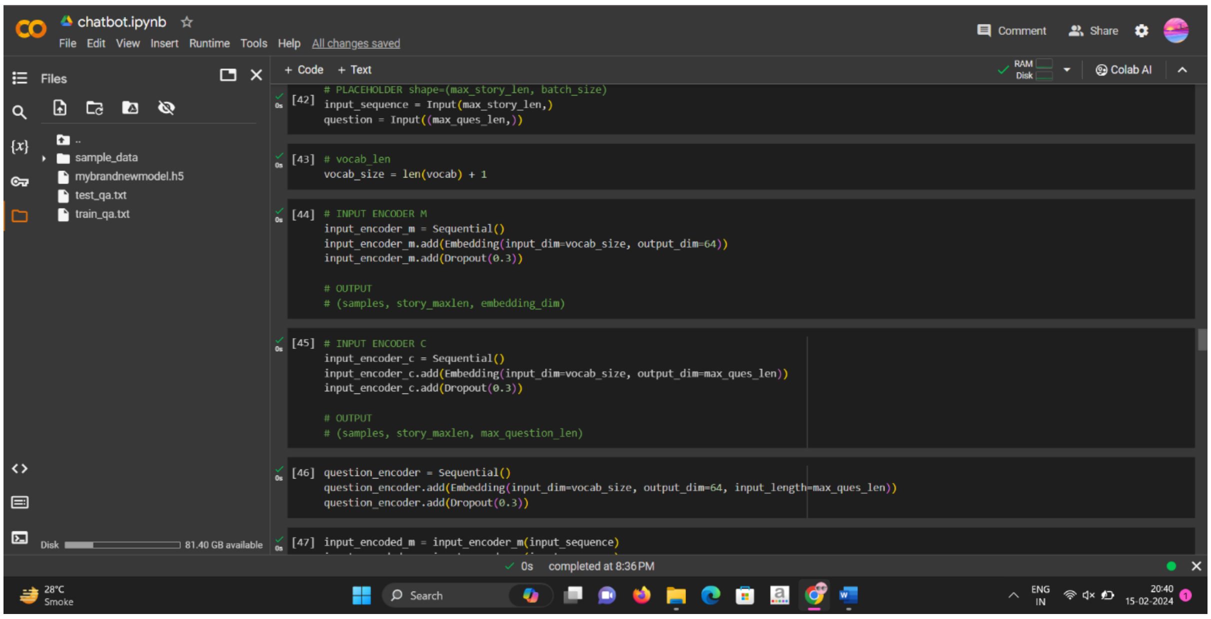
```
[33] # Create our own list of list of word indicies with padding.
     def vectorize_stories(data, word_index=tokenizer.word_index, max_story_len=max_story_len, max_question_len=max_ques_len):
         # Stories = X
         X = []

         # Questions = Xq
         Xq = []

         # Y Correct Answer ['yes', 'no']
         Y = []
         for story, query, answer in data:

             # for each story
             # [23, 14, 15]
             x = [word_index[word.lower()] for word in story]
             xq = [word_index[word.lower()] for word in query]

             y = np.zeros(len(word_index)+1)
             y[word_index[answer]] = 1

             X.append(x)   # X holds list of lists of word indices for stories.
             Xq.append(xq) # Xq holds list of lists for word indices for questions.
             Y.append(y) # Y holds lists of lists of (38) biniary numbers, only 1 of them is 1.

         return (pad_sequences(X, maxlen=max_story_len), pad_sequences(Xq, maxlen=max_question_len), np.array(Y))
```



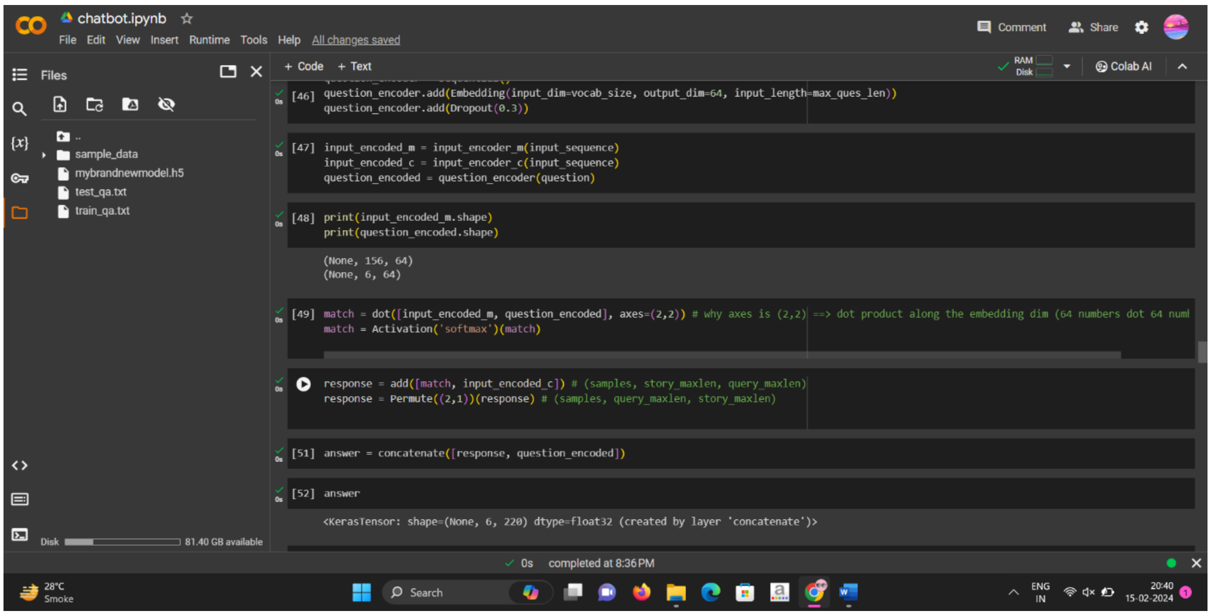
```
[33]

[34] inputs_train, queries_train, answers_train = vectorize_stories(train_data)

[35] inputs_test, queries_test, answers_test = vectorize_stories(test_data)

[36] inputs_test

     array([[ 0,  0,  0, ...,  6, 32, 13],
            [ 0,  0,  0, ...,  6, 19, 13],
            [ 0,  0,  0, ...,  6, 19, 13],
            ...,
            [ 0,  0,  0, ...,  6, 26, 13],
            [ 0,  0,  0, ...,  6, 19, 13],
            [ 0,  0,  0, ..., 26, 34, 13]], dtype=int32)

[37] answers_test

     array([[0., 0., 0., ..., 0., 0., 0.],
            [0., 0., 0., ..., 0., 0., 0.],
            [0., 0., 0., ..., 0., 0., 0.],
            ...,
            [0., 0., 0., ..., 0., 0., 0.],
            [0., 0., 0., ..., 0., 0., 0.],
            [0., 0., 0., ..., 0., 0., 0.]])

[38] tokenizer.word_index['yes']

     5
```



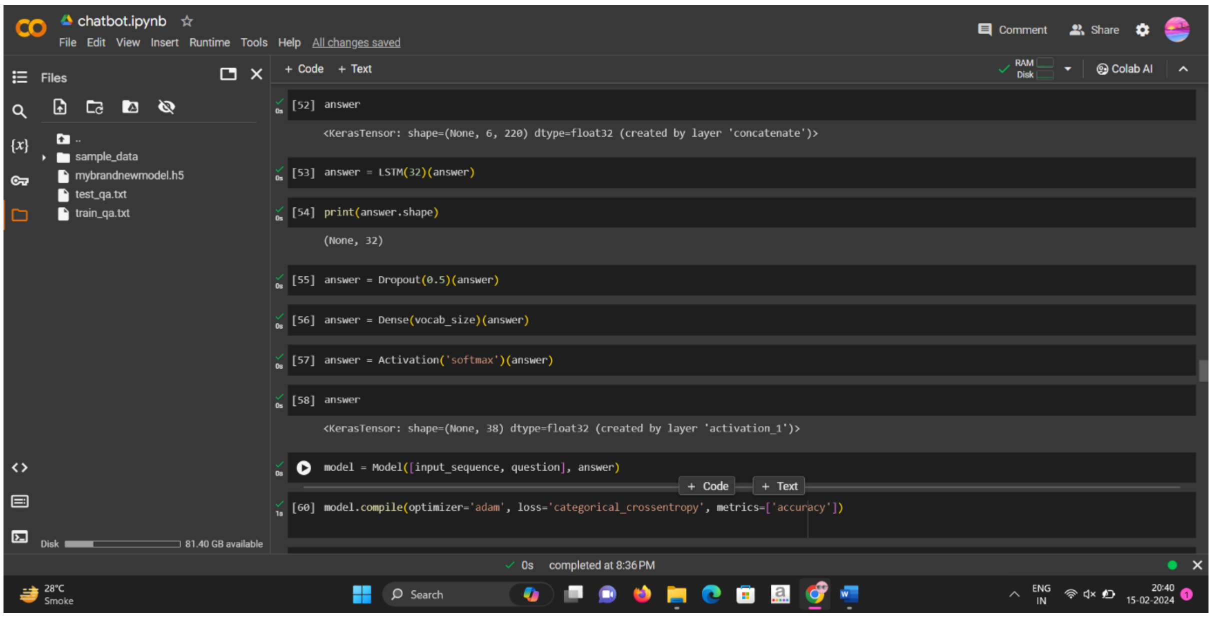
```
[38] tokenizer.word_index['yes']

     5

[39] tokenizer.word_index['no']

     10

[40] sum(answers_test)

     array([  0.,   0.,   0.,   0.,   0., 497.,   0.,   0.,   0.,   0., 503.,
              0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
              0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
              0.,   0.,   0.,   0.,   0.])

[41] !pip install keras
     import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras import layers
     from tensorflow.keras.layers import Embedding
     from tensorflow.keras.models import Sequential, Model
     from tensorflow.keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM
     from keras.models import Sequential, Model
     from keras.layers import Embedding
     from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM

     Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
```



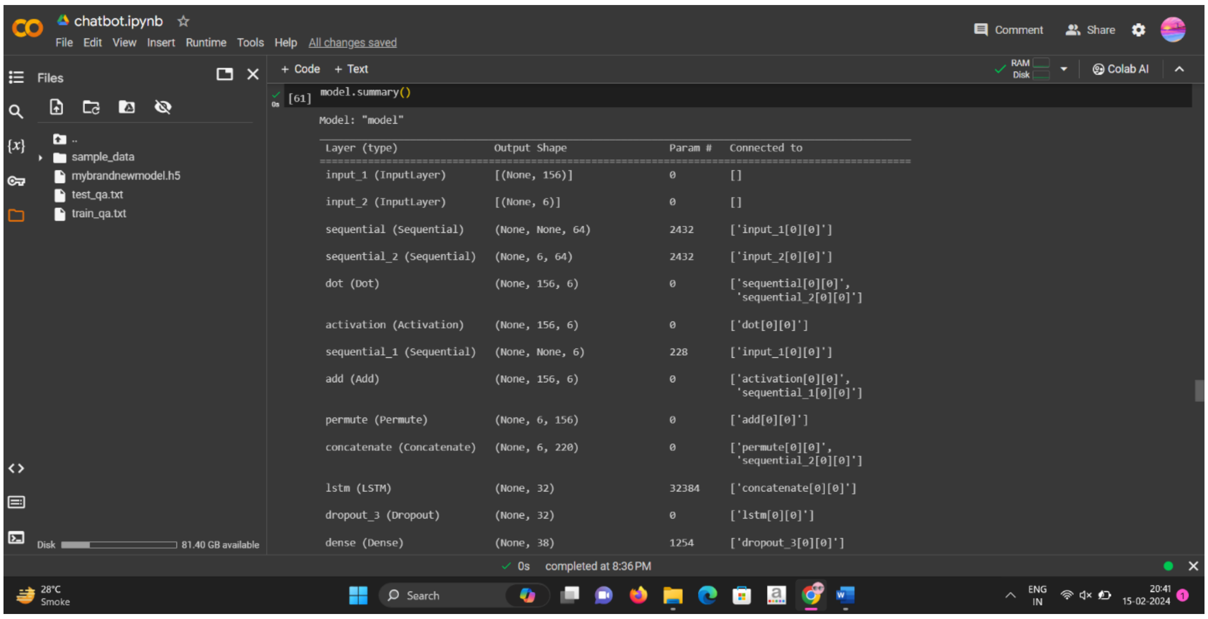
```
[42] # PLACEHOLDER shape=(max_story_len, batch_size)
     input_sequence = Input(max_story_len,)
     question = Input((max_ques_len,))

[43] # vocab_len
     vocab_size = len(vocab) + 1

[44] # INPUT ENCODER M
     input_encoder_m = Sequential()
     input_encoder_m.add(Embedding(input_dim=vocab_size, output_dim=64))
     input_encoder_m.add(Dropout(0.3))

     # OUTPUT
     # (samples, story_maxlen, embedding_dim)

[45] # INPUT ENCODER C
     input_encoder_c = Sequential()
     input_encoder_c.add(Embedding(input_dim=vocab_size, output_dim=max_ques_len))
     input_encoder_c.add(Dropout(0.3))

     # OUTPUT
     # (samples, story_maxlen, max_question_len)

[46] question_encoder = Sequential()
     question_encoder.add(Embedding(input_dim=vocab_size, output_dim=64, input_length=max_ques_len))
     question_encoder.add(Dropout(0.3))

[47] input_encoded_m = input_encoder_m(input_sequence)
```



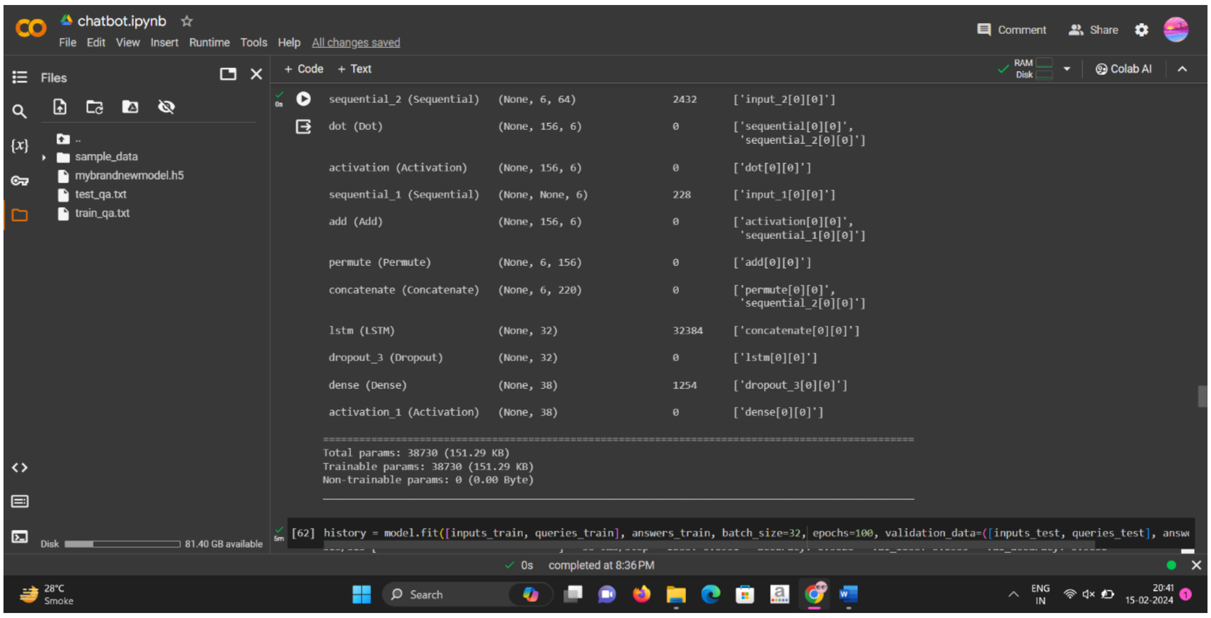
```
[46] question_encoder.add(Embedding(input_dim=vocab_size, output_dim=64, input_length=max_ques_len))
     question_encoder.add(Dropout(0.3))

[47] input_encoded_m = input_encoder_m(input_sequence)
     input_encoded_c = input_encoder_c(input_sequence)
     question_encoded = question_encoder(question)

[48] print(input_encoded_m.shape)
     print(question_encoded.shape)

     (None, 156, 64)
     (None, 6, 64)

[49] match = dot([input_encoded_m, question_encoded], axes=(2,2)) # why axes is (2,2) ==> dot product along the embedding dim (64 numbers dot 64 numl
     match = Activation('softmax')(match)

     response = add([match, input_encoded_c]) # (samples, story_maxlen, query_maxlen)
     response = Permute((2,1))(response) # (samples, query_maxlen, story_maxlen)

[51] answer = concatenate([response, question_encoded])

[52] answer

     <KerasTensor: shape=(None, 6, 220) dtype=float32 (created by layer 'concatenate')>
```



```
[52] answer

     <KerasTensor: shape=(None, 6, 220) dtype=float32 (created by layer 'concatenate')>

[53] answer = LSTM(32)(answer)

[54] print(answer.shape)

     (None, 32)

[55] answer = Dropout(0.5)(answer)

[56] answer = Dense(vocab_size)(answer)

[57] answer = Activation('softmax')(answer)

[58] answer

     <KerasTensor: shape=(None, 38) dtype=float32 (created by layer 'activation_1')>

     model = Model([input_sequence, question], answer)

[60] model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

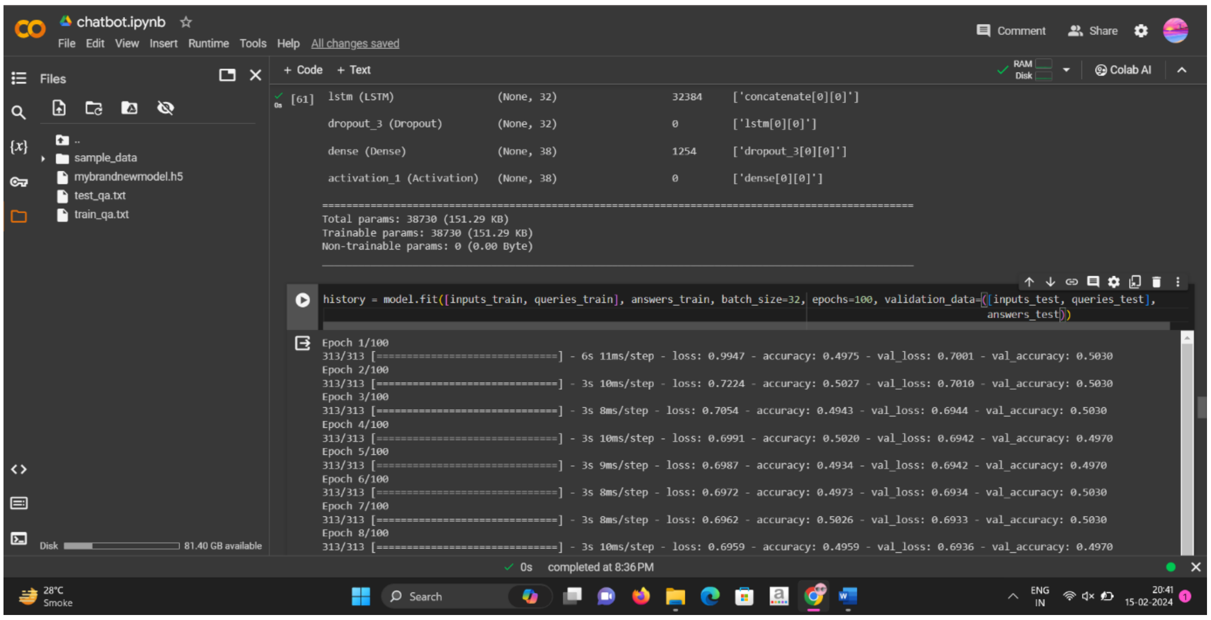


```
[61] model.summary()

Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 156)]                0         []

 input_2 (InputLayer)        [(None, 6)]                  0         []

 sequential (Sequential)     (None, None, 64)             2432      ['input_1[0][0]']

 sequential_2 (Sequential)   (None, 6, 64)                2432      ['input_2[0][0]']

 dot (Dot)                   (None, 156, 6)               0         ['sequential[0][0]',
                                                                     'sequential_2[0][0]']

 activation (Activation)     (None, 156, 6)               0         ['dot[0][0]']

 sequential_1 (Sequential)   (None, None, 6)              228       ['input_1[0][0]']

 add (Add)                   (None, 156, 6)               0         ['activation[0][0]',
                                                                     'sequential_1[0][0]']

 permute (Permute)           (None, 6, 156)               0         ['add[0][0]']

 concatenate (Concatenate)   (None, 6, 220)               0         ['permute[0][0]',
                                                                     'sequential_2[0][0]']

 lstm (LSTM)                 (None, 32)                   32384     ['concatenate[0][0]']

 dropout_3 (Dropout)         (None, 32)                   0         ['lstm[0][0]']

 dense (Dense)               (None, 38)                   1254      ['dropout_3[0][0]']
```



```
 sequential_2 (Sequential)   (None, 6, 64)             2432      ['input_2[0][0]']

 dot (Dot)                   (None, 156, 6)            0         ['sequential[0][0]',
                                                                  'sequential_2[0][0]']

 activation (Activation)     (None, 156, 6)            0         ['dot[0][0]']

 sequential_1 (Sequential)   (None, None, 6)           228       ['input_1[0][0]']

 add (Add)                   (None, 156, 6)            0         ['activation[0][0]',
                                                                  'sequential_1[0][0]']

 permute (Permute)           (None, 6, 156)            0         ['add[0][0]']

 concatenate (Concatenate)   (None, 6, 220)            0         ['permute[0][0]',
                                                                  'sequential_2[0][0]']

 lstm (LSTM)                 (None, 32)                32384     ['concatenate[0][0]']

 dropout_3 (Dropout)         (None, 32)                0         ['lstm[0][0]']

 dense (Dense)               (None, 38)                1254      ['dropout_3[0][0]']

 activation_1 (Activation)   (None, 38)                0         ['dense[0][0]']

==================================================================================================
Total params: 38730 (151.29 KB)
Trainable params: 38730 (151.29 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

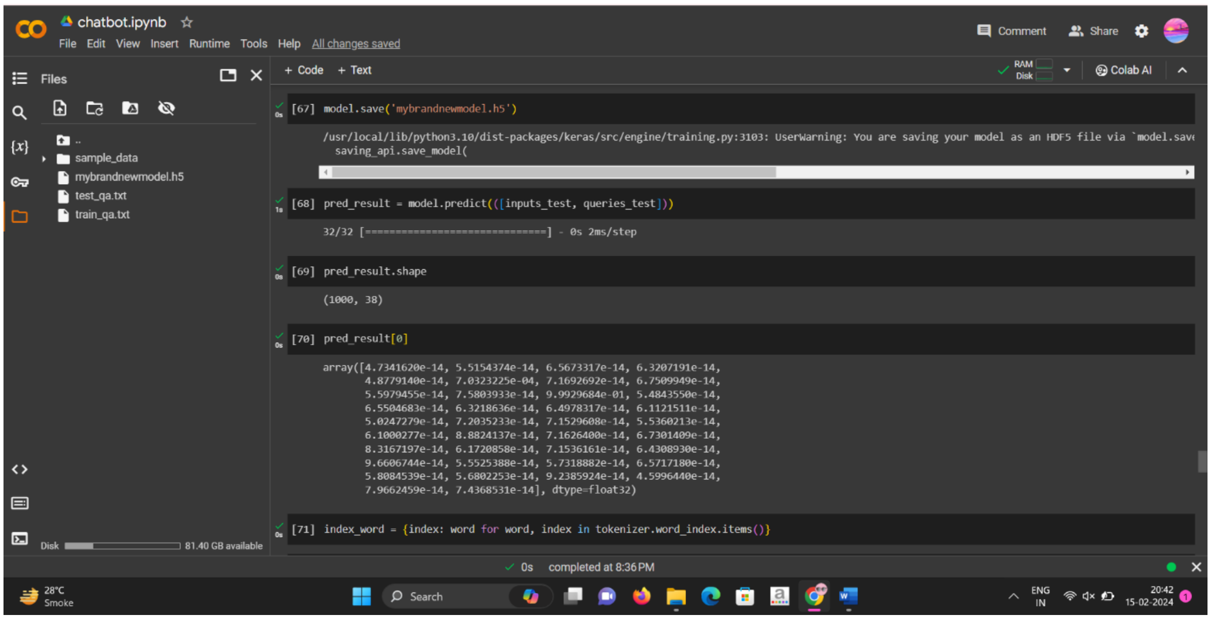
```
[62] history = model.fit([inputs_train, queries_train], answers_train, batch_size=32, epochs=100, validation_data=([inputs_test, queries_test], answ
```



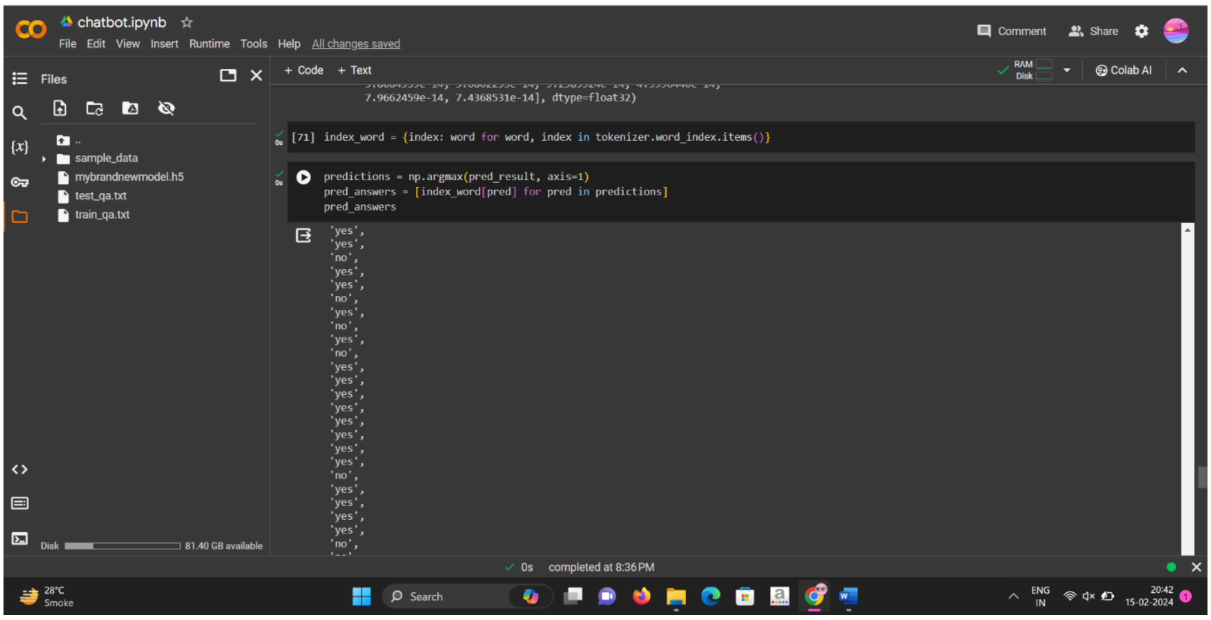
```
[61] lstm (LSTM)             (None, 32)                32384     ['concatenate[0][0]']

 dropout_3 (Dropout)         (None, 32)                0         ['lstm[0][0]']

 dense (Dense)               (None, 38)                1254      ['dropout_3[0][0]']

 activation_1 (Activation)   (None, 38)                0         ['dense[0][0]']

==================================================================================================
Total params: 38730 (151.29 KB)
Trainable params: 38730 (151.29 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

```
history = model.fit([inputs_train, queries_train], answers_train, batch_size=32, epochs=100, validation_data=([inputs_test, queries_test],
                                                                                                answers_test))
```

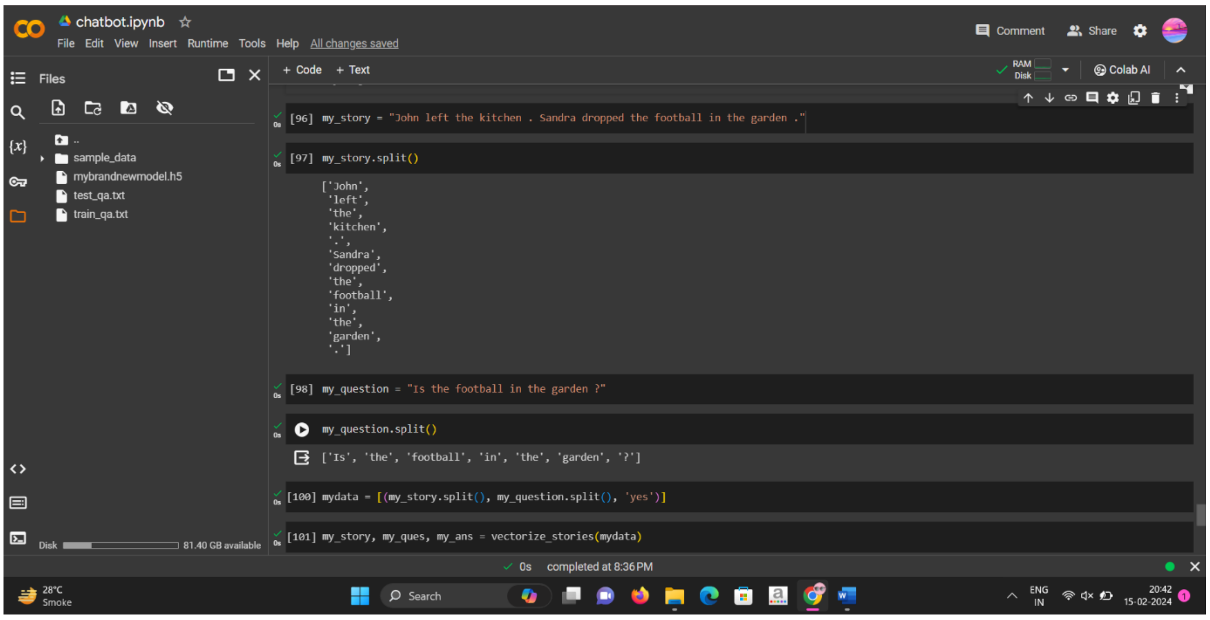
```
Epoch 1/100
313/313 [==============================] - 6s 11ms/step - loss: 0.9947 - accuracy: 0.4975 - val_loss: 0.7001 - val_accuracy: 0.5030
Epoch 2/100
313/313 [==============================] - 3s 10ms/step - loss: 0.7224 - accuracy: 0.5027 - val_loss: 0.7010 - val_accuracy: 0.5030
Epoch 3/100
313/313 [==============================] - 3s 8ms/step - loss: 0.7054 - accuracy: 0.4943 - val_loss: 0.6944 - val_accuracy: 0.5030
Epoch 4/100
313/313 [==============================] - 3s 10ms/step - loss: 0.6991 - accuracy: 0.5020 - val_loss: 0.6942 - val_accuracy: 0.4970
Epoch 5/100
313/313 [==============================] - 3s 9ms/step - loss: 0.6987 - accuracy: 0.4934 - val_loss: 0.6942 - val_accuracy: 0.4970
Epoch 6/100
313/313 [==============================] - 3s 8ms/step - loss: 0.6972 - accuracy: 0.4973 - val_loss: 0.6934 - val_accuracy: 0.5030
Epoch 7/100
313/313 [==============================] - 3s 8ms/step - loss: 0.6962 - accuracy: 0.5026 - val_loss: 0.6933 - val_accuracy: 0.5030
Epoch 8/100
313/313 [==============================] - 3s 10ms/step - loss: 0.6959 - accuracy: 0.4959 - val_loss: 0.6936 - val_accuracy: 0.4970
```



```
[67] model.save('mybrandnewmodel.h5')

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save
  saving_api.save_model(

[68] pred_result = model.predict(([inputs_test, queries_test]))

32/32 [==============================] - 0s 2ms/step

[69] pred_result.shape

(1000, 38)

[70] pred_result[0]

array([4.7341620e-14, 5.5154374e-14, 6.5673317e-14, 6.3207191e-14,
       4.8779140e-14, 7.0323225e-04, 7.1692692e-14, 6.7509949e-14,
       5.5979455e-14, 7.5803933e-14, 9.9929684e-01, 5.4843550e-14,
       6.5504683e-14, 6.3218636e-14, 6.4978317e-14, 6.1121511e-14,
       5.0247279e-14, 7.2035233e-14, 7.1529608e-14, 5.5360213e-14,
       6.1000277e-14, 8.8824137e-14, 7.1626400e-14, 6.7301409e-14,
       8.3167197e-14, 6.1720858e-14, 7.1536161e-14, 6.4308930e-14,
       9.6606744e-14, 5.5525388e-14, 5.7318882e-14, 6.5717180e-14,
       5.8084539e-14, 5.6802253e-14, 9.2385924e-14, 4.5996440e-14,
       7.9662459e-14, 7.4368531e-14], dtype=float32)

[71] index_word = {index: word for word, index in tokenizer.word_index.items()}
```



```
       7.9662459e-14, 7.4368531e-14], dtype=float32)

[71] index_word = {index: word for word, index in tokenizer.word_index.items()}

predictions = np.argmax(pred_result, axis=1)
pred_answers = [index_word[pred] for pred in predictions]
pred_answers

 'yes',
 'yes',
 'no',
 'yes',
 'yes',
 'no',
 'yes',
 'no',
 'yes',
 'no',
 'yes',
 'yes',
 'yes',
 'yes',
 'yes',
 'yes',
 'yes',
 'yes',
 'no',
 'yes',
 'yes',
 'yes',
 'yes',
 'no',
```



```
[96] my_story = "John left the kitchen . Sandra dropped the football in the garden ."
```

```
[97] my_story.split()
```

```
['John',
 'left',
 'the',
 'kitchen',
 '.',
 'Sandra',
 'dropped',
 'the',
 'football',
 'in',
 'the',
 'garden',
 '.']
```

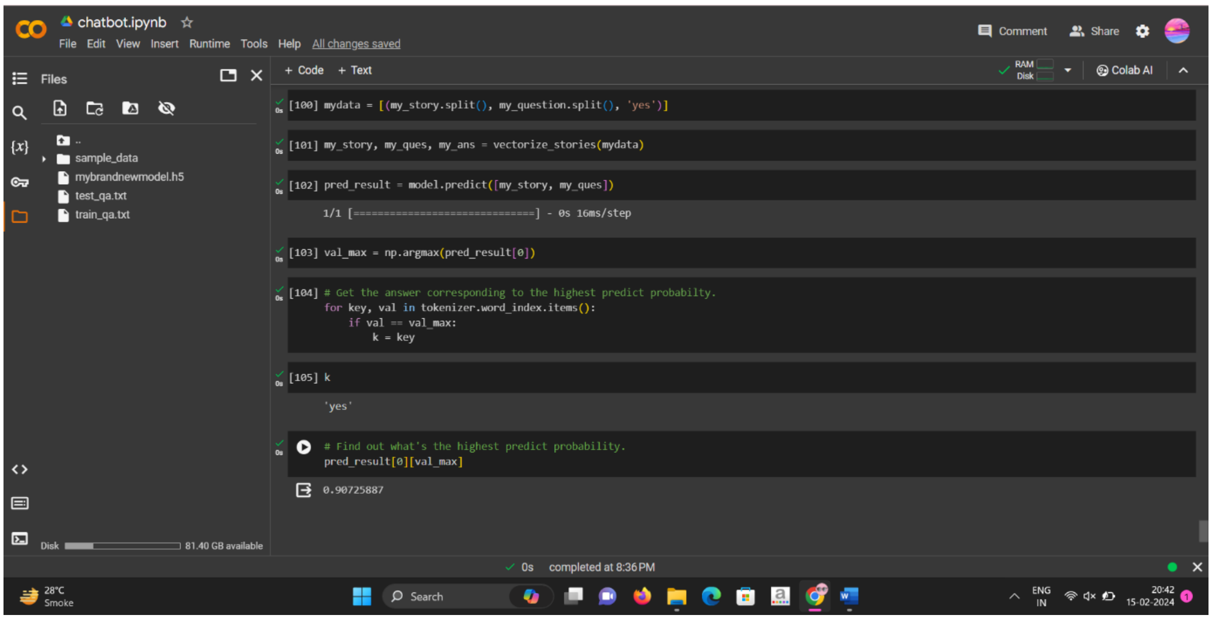
```
[98] my_question = "Is the football in the garden ?"
```

```
my_question.split()
```

```
['Is', 'the', 'football', 'in', 'the', 'garden', '?']
```

```
[100] mydata = [(my_story.split(), my_question.split(), 'yes')]
```

```
[101] my_story, my_ques, my_ans = vectorize_stories(mydata)
```



```
[100] mydata = [(my_story.split(), my_question.split(), 'yes')]
```

```
[101] my_story, my_ques, my_ans = vectorize_stories(mydata)
```

```
[102] pred_result = model.predict([my_story, my_ques])
```

```
1/1 [==============================] - 0s 16ms/step
```

```
[103] val_max = np.argmax(pred_result[0])
```

```
[104] # Get the answer corresponding to the highest predict probabilty.
      for key, val in tokenizer.word_index.items():
          if val == val_max:
              k = key
```

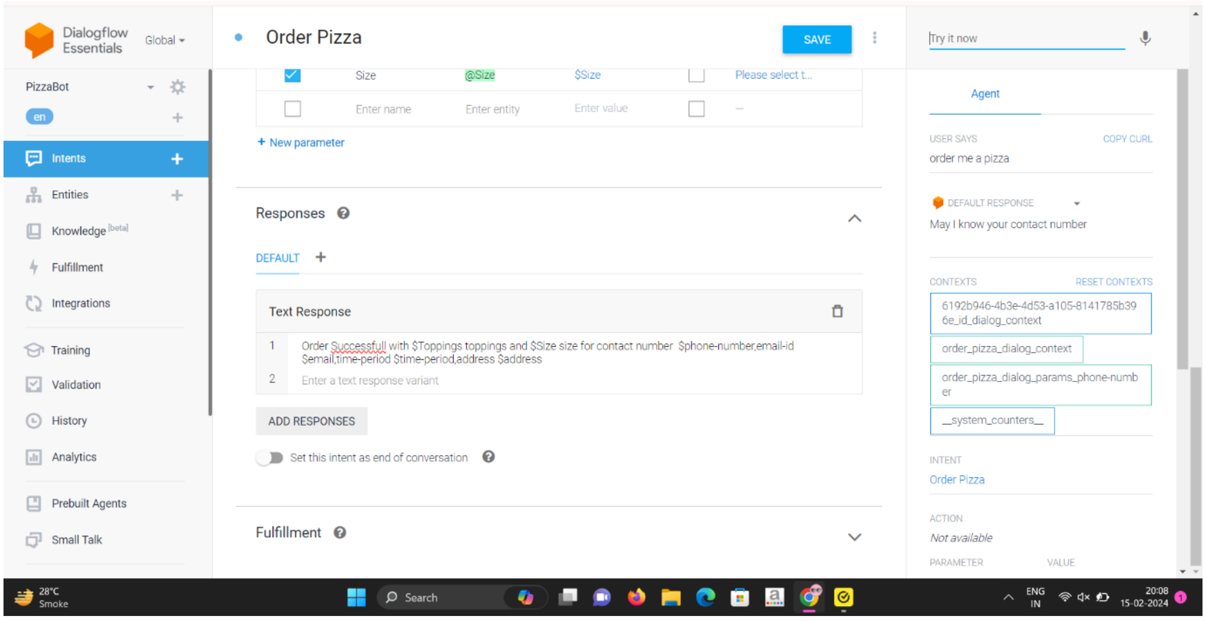
```
[105] k
```

```
'yes'
```

```
# Find out what's the highest predict probability.
pred_result[0][val_max]
```

```
0.90725887
```

Dialogflow Chatbot Implementation:



Order Pizza

✓	Size	@Size	$Size		Please select t...
	Enter name	Enter entity	Enter value		—

+ New parameter

Responses

DEFAULT +

Text Response

1 Order Successfull with $Toppings toppings and $Size size for contact number $phone-number,email-id $email,time-period $time-period,address $address

2 Enter a text response variant

ADD RESPONSES

Set this intent as end of conversation

Fulfillment

USER SAYS COPY CURL

order me a pizza

DEFAULT RESPONSE

May I know your contact number

CONTEXTS RESET CONTEXTS

6192b946-4b3e-4d53-a105-8141785b396e_id_dialog_context

order_pizza_dialog_context

order_pizza_dialog_params_phone-number

__system_counters__

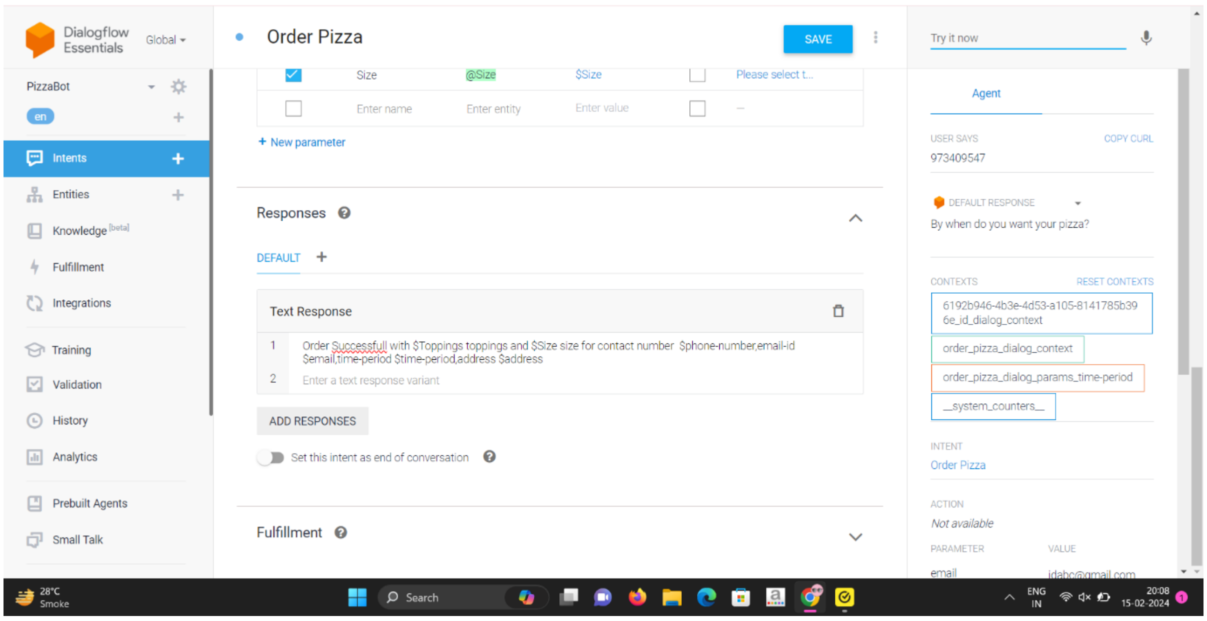
INTENT

Order Pizza

ACTION

Not available

PARAMETER VALUE



Order Pizza

✓	Size	@Size	$Size		Please select t...
	Enter name	Enter entity	Enter value		—

+ New parameter

Responses

DEFAULT +

Text Response

1 Order Successfull with $Toppings toppings and $Size size for contact number $phone-number,email-id $email,time-period $time-period,address $address

2 Enter a text response variant

ADD RESPONSES

Set this intent as end of conversation

Fulfillment

USER SAYS COPY CURL

973409547

DEFAULT RESPONSE

By when do you want your pizza?

CONTEXTS RESET CONTEXTS

6192b946-4b3e-4d53-a105-8141785b396e_id_dialog_context

order_pizza_dialog_context

order_pizza_dialog_params_time-period

__system_counters__

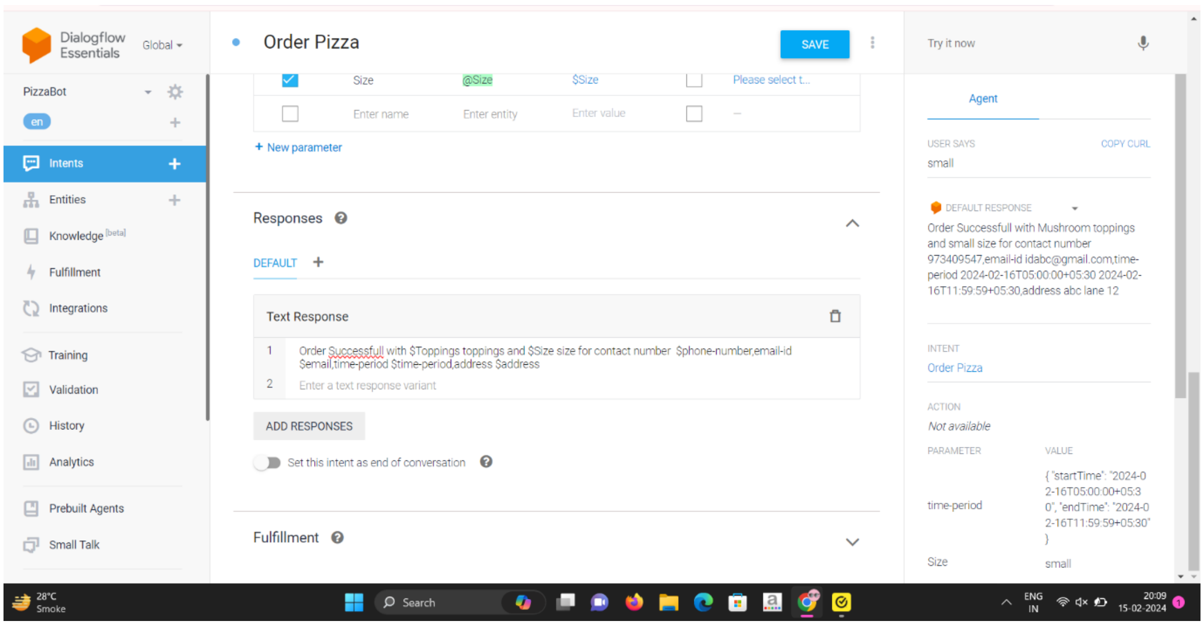
INTENT

Order Pizza

ACTION

Not available

PARAMETER	VALUE
email	idabc@gmail.com



Order Pizza

✓	Size	@Size	$Size		Please select t...
	Enter name	Enter entity	Enter value		–

+ New parameter

Responses

Text Response

1 Order Successfull with $Toppings toppings and $Size size for contact number $phone-number,email-id $email,time-period $time-period,address $address

2 Enter a text response variant

Set this intent as end of conversation

Fulfillment

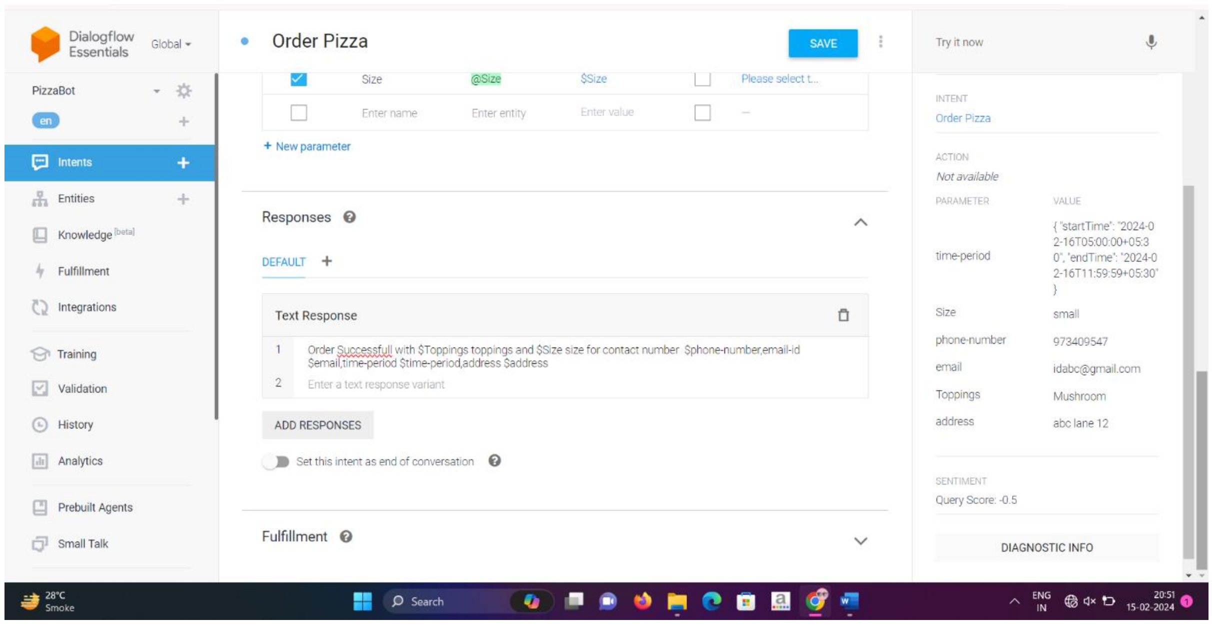
USER SAYS
small

DEFAULT RESPONSE
Order Successfull with Mushroom toppings and small size for contact number 973409547,email-id idabc@gmail.com,time-period 2024-02-16T05:00:00+05:30 2024-02-16T11:59:59+05:30,address abc lane 12

INTENT
Order Pizza

ACTION
Not available

PARAMETER	VALUE
time-period	{ "startTime": "2024-02-16T05:00:00+05:30", "endTime": "2024-02-16T11:59:59+05:30" }
Size	small



Order Pizza

✓	Size	@Size	$Size		Please select t...
	Enter name	Enter entity	Enter value		–

+ New parameter

Responses

Text Response

1 Order Successfull with $Toppings toppings and $Size size for contact number $phone-number,email-id $email,time-period $time-period,address $address

2 Enter a text response variant

Set this intent as end of conversation

Fulfillment

INTENT
Order Pizza

ACTION
Not available

PARAMETER	VALUE
time-period	{ "startTime": "2024-02-16T05:00:00+05:30", "endTime": "2024-02-16T11:59:59+05:30" }
Size	small
phone-number	973409547
email	idabc@gmail.com
Toppings	Mushroom
address	abc lane 12

SENTIMENT
Query Score: -0.5

➢ Conclusion:

In conclusion, this project demonstrates the implementation of a chatbot using Python and machine learning techniques(using supervised machine learning), also with Dialogflow for enhanced conversational capabilities. By following a systematic methodology of data collection, preprocessing, model training, and integration, we have developed a robust and effective chatbot solution.
Dialogflow provides additional benefits such as intent recognition and context management, allowing the chatbot to understand user queries more accurately and provide relevant responses. Through thorough evaluation and testing, we have ensured that the chatbot meets the desired performance criteria in terms of response accuracy and user satisfaction.

Overall, this project highlights the potential of combining Python-based machine learning with platforms like Dialogflow to create advanced chatbot systems capable of engaging in natural and meaningful conversations with users.