BVRIT HYDERABAD College of Engineering for Women

Department of Information Technology

ML POWERED TEXT AUTO-COMPLETION AND GENERATION

Team - 3
J.Nikhitha -19WH1A1207
P.Pooja Sri – 19WH1A1216
G.Pavithra – 19WH1A1218
Ch.Sai Meghana – 19WH1A1246

Under the guidance of
Dr. P. Kayal
Associate Professor

Agenda

- Summary of Stage 1
- Architecture
- Implementation of Experimental Design
- Execution video
- Analysis of Results
- Conclusion & Future Scope
- References

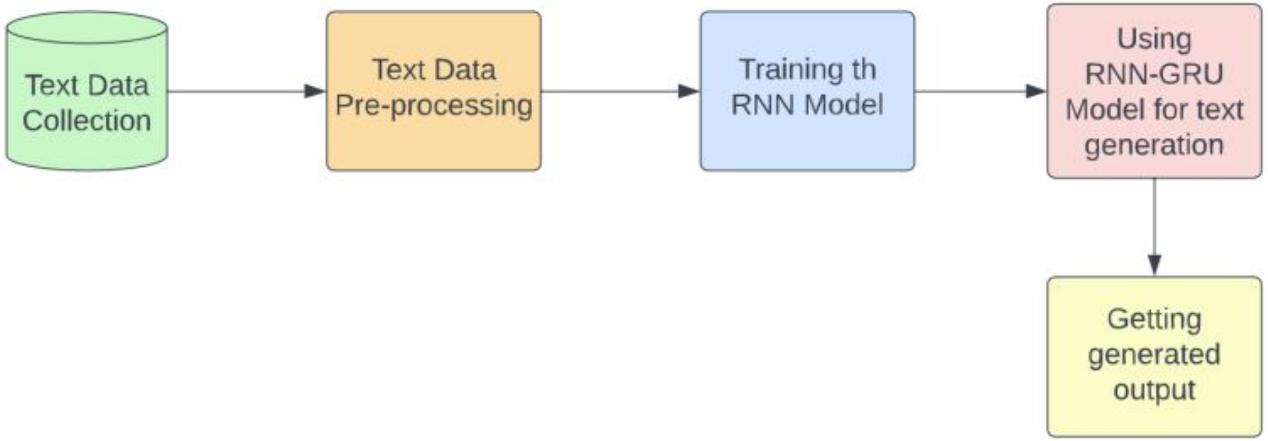
Summary of Stage 1

- Our system helps in minimizing the human effort by providing the features like text auto-completion and generation. The auto-complete completes the upcoming words from predicting and text generation is responsible for generating the body of the mail through the subject line.

- In Stage -1 We worked on our Auto-Complete model.. For building this model we used LSTM and N-Grams models.Here we were able to predict the next word when given an input.

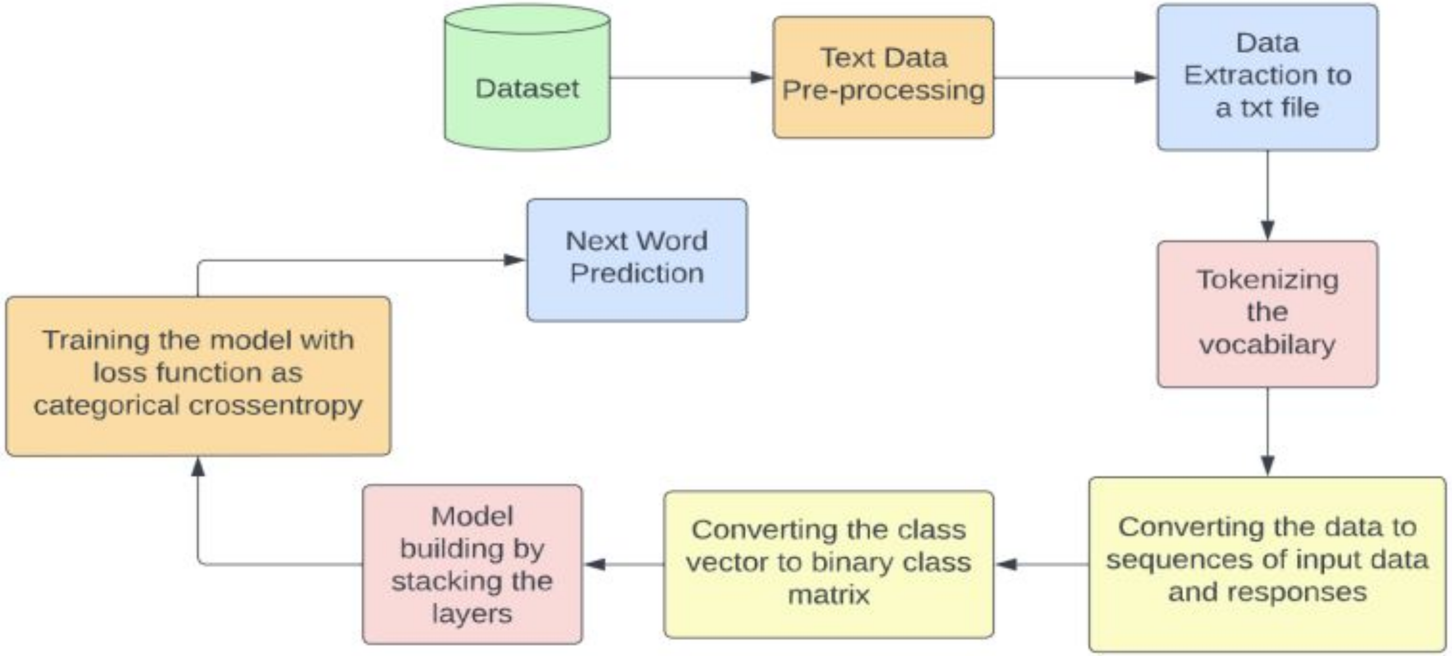
Architecture

Text Generation Module



Architecture

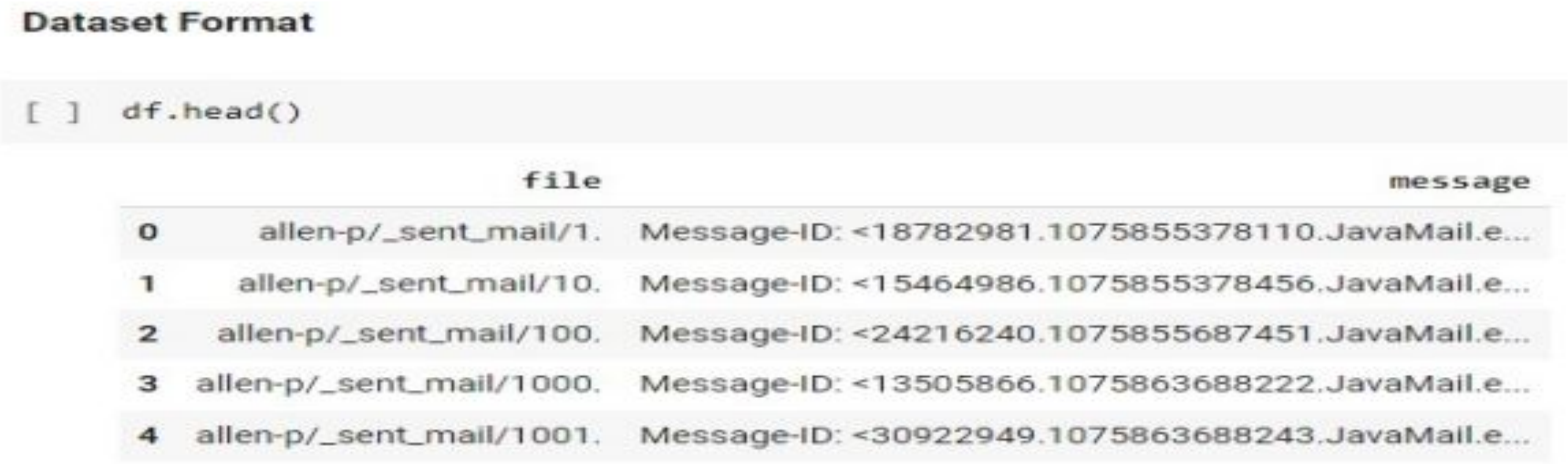
Text Auto-Completion Module



Dataset
Text Data Pre-processing
Data Extraction to a txt file
Tokenizing the vocabilary
Converting the data to sequences of input data and responses
Converting the class vector to binary class matrix
Model building by stacking the layers
Training the model with loss function as categorical crossentropy
Next Word Prediction

Implementation

Dataset Format used for Text Auto-Completion

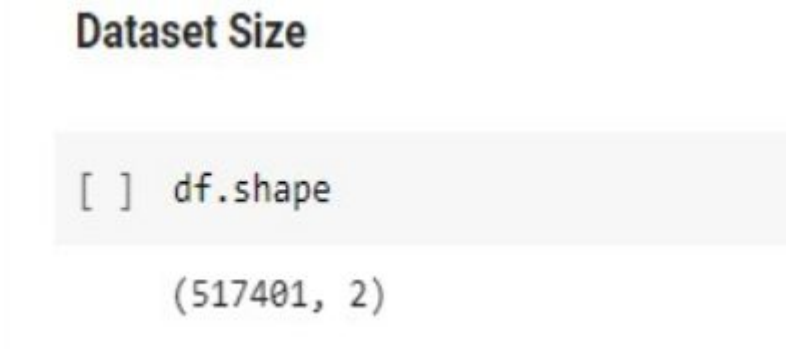


Dataset Format

```
[ ]  df.head()
```

	file	message
0	allen-p/_sent_mail/1.	Message-ID: <18782981.1075855378110.JavaMail.e...
1	allen-p/_sent_mail/10.	Message-ID: <15464986.1075855378456.JavaMail.e...
2	allen-p/_sent_mail/100.	Message-ID: <24216240.1075855687451.JavaMail.e...
3	allen-p/_sent_mail/1000.	Message-ID: <13505866.1075863688222.JavaMail.e...
4	allen-p/_sent_mail/1001.	Message-ID: <30922949.1075863688243.JavaMail.e...

Dataset Shape

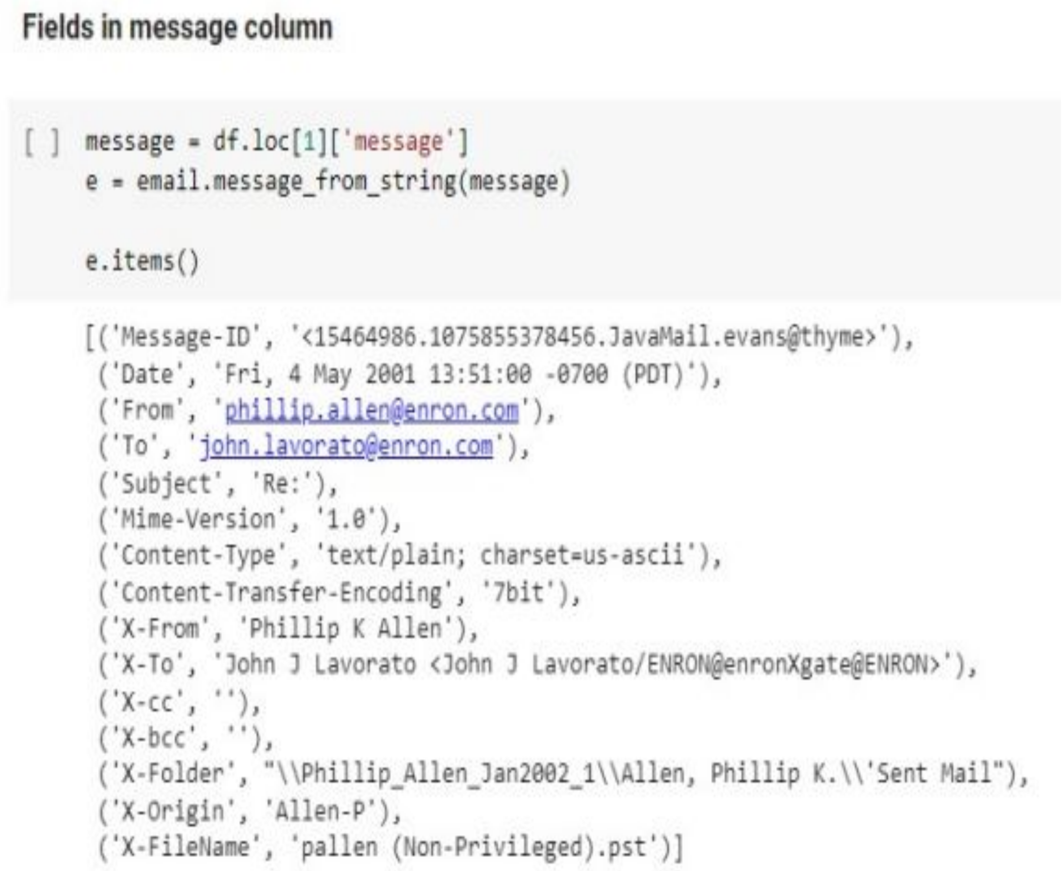


Dataset Size

```
[ ]  df.shape

(517401, 2)
```

Implementation

Actual Data Format

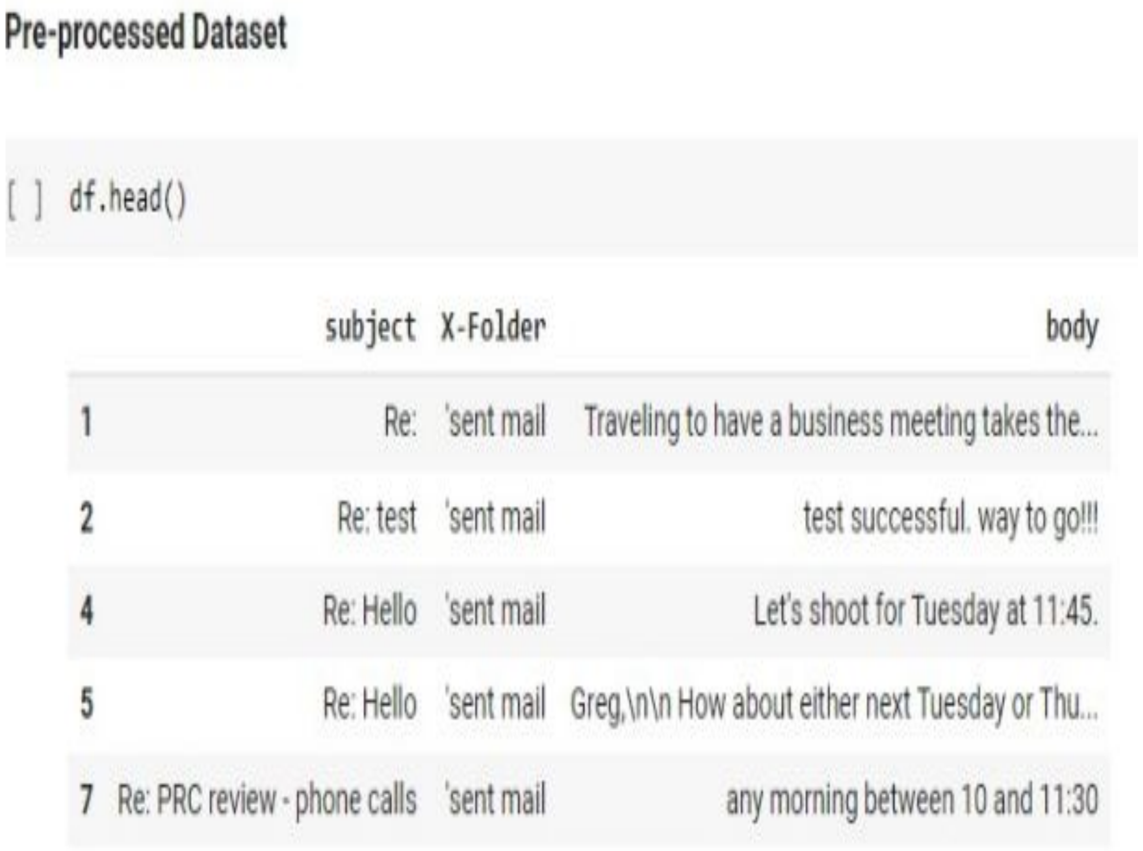


Fields in message column

```
[ ]  message = df.loc[1]['message']
     e = email.message_from_string(message)

     e.items()
```

```
[('Message-ID', '<15464986.1075855378456.JavaMail.evans@thyme>'),
 ('Date', 'Fri, 4 May 2001 13:51:00 -0700 (PDT)'),
 ('From', 'phillip.allen@enron.com'),
 ('To', 'john.lavorato@enron.com'),
 ('Subject', 'Re:'),
 ('Mime-Version', '1.0'),
 ('Content-Type', 'text/plain; charset=us-ascii'),
 ('Content-Transfer-Encoding', '7bit'),
 ('X-From', 'Phillip K Allen'),
 ('X-To', 'John J Lavorato <John J Lavorato/ENRON@enronXgate@ENRON>'),
 ('X-cc', ''),
 ('X-bcc', ''),
 ('X-Folder', "\\Phillip_Allen_Jan2002_1\\Allen, Phillip K.\\'Sent Mail"),
 ('X-Origin', 'Allen-P'),
 ('X-FileName', 'pallen (Non-Privileged).pst')]
```

Dataset after pre-processing



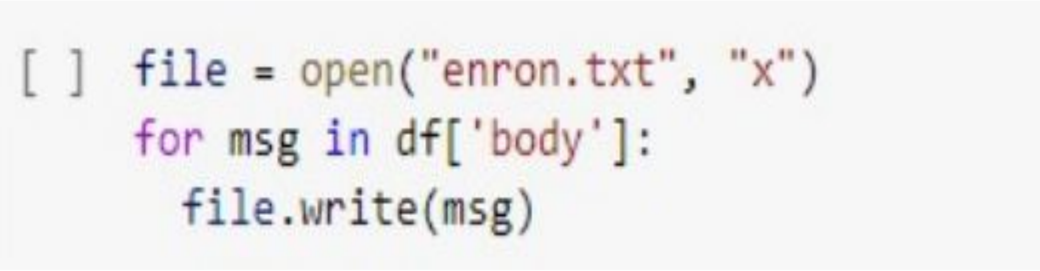
Pre-processed Dataset

```
[ ]  df.head()
```

	subject	X-Folder	body
1	Re:	'sent mail	Traveling to have a business meeting takes the...
2	Re: test	'sent mail	test successful. way to go!!!
4	Re: Hello	'sent mail	Let's shoot for Tuesday at 11:45.
5	Re: Hello	'sent mail	Greg,\n\n How about either next Tuesday or Thu...
7	Re: PRC review - phone calls	'sent mail	any morning between 10 and 11:30

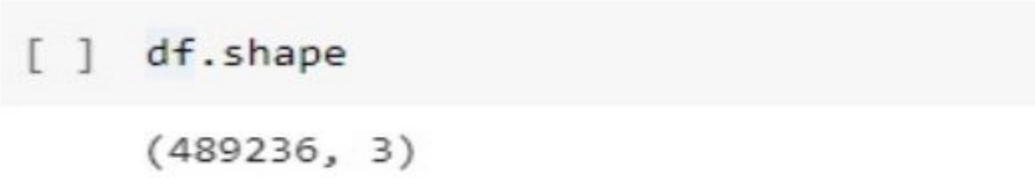
Implementation

Storing the mail content in a txt file



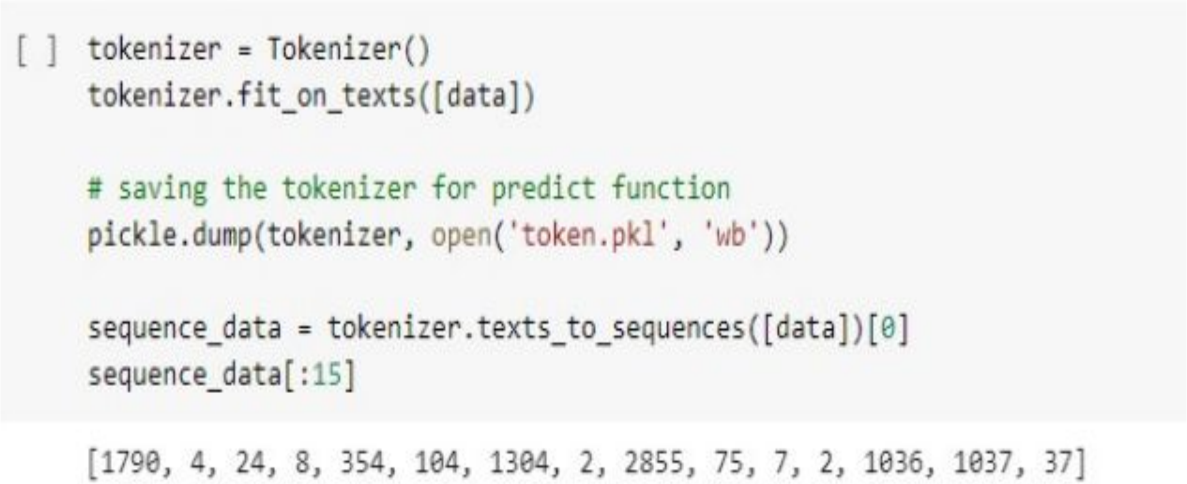
```
[ ]  file = open("enron.txt", "x")
     for msg in df['body']:
       file.write(msg)
```

Dataset shape after pre-processing



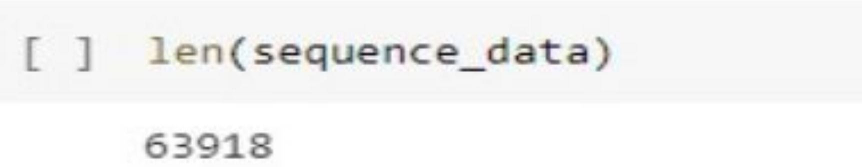
```
[ ]  df.shape

     (489236, 3)
```

Tokenizing the vocabulary



```
[ ]  tokenizer = Tokenizer()
     tokenizer.fit_on_texts([data])

     # saving the tokenizer for predict function
     pickle.dump(tokenizer, open('token.pkl', 'wb'))

     sequence_data = tokenizer.texts_to_sequences([data])[0]
     sequence_data[:15]

     [1790, 4, 24, 8, 354, 104, 1304, 2, 2855, 75, 7, 2, 1036, 1037, 37]
```

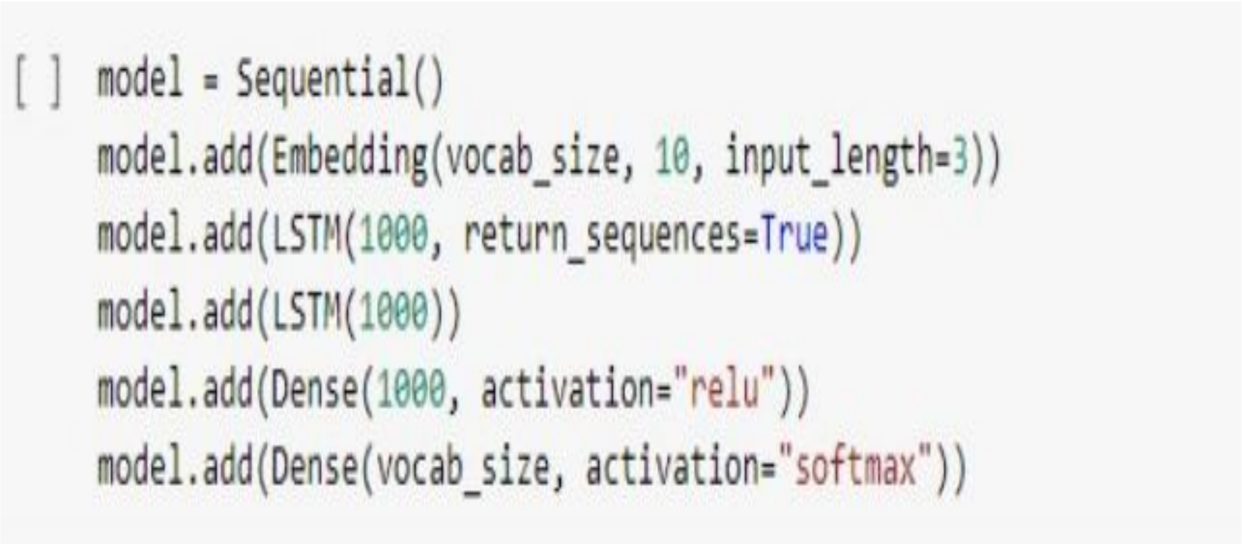
Number of unique words



```
[ ]  len(sequence_data)

     63918
```

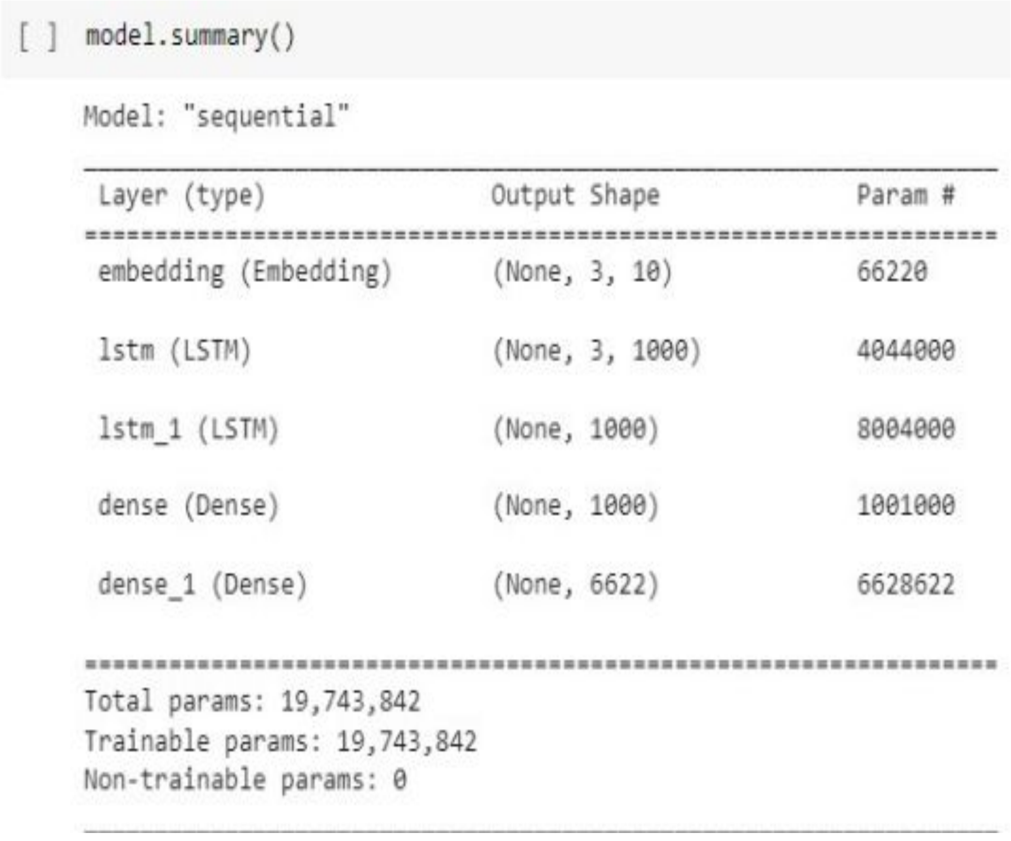
Implementation

Building the Ngrams Model



```
[ ]  model = Sequential()
     model.add(Embedding(vocab_size, 10, input_length=3))
     model.add(LSTM(1000, return_sequences=True))
     model.add(LSTM(1000))
     model.add(Dense(1000, activation="relu"))
     model.add(Dense(vocab_size, activation="softmax"))
```

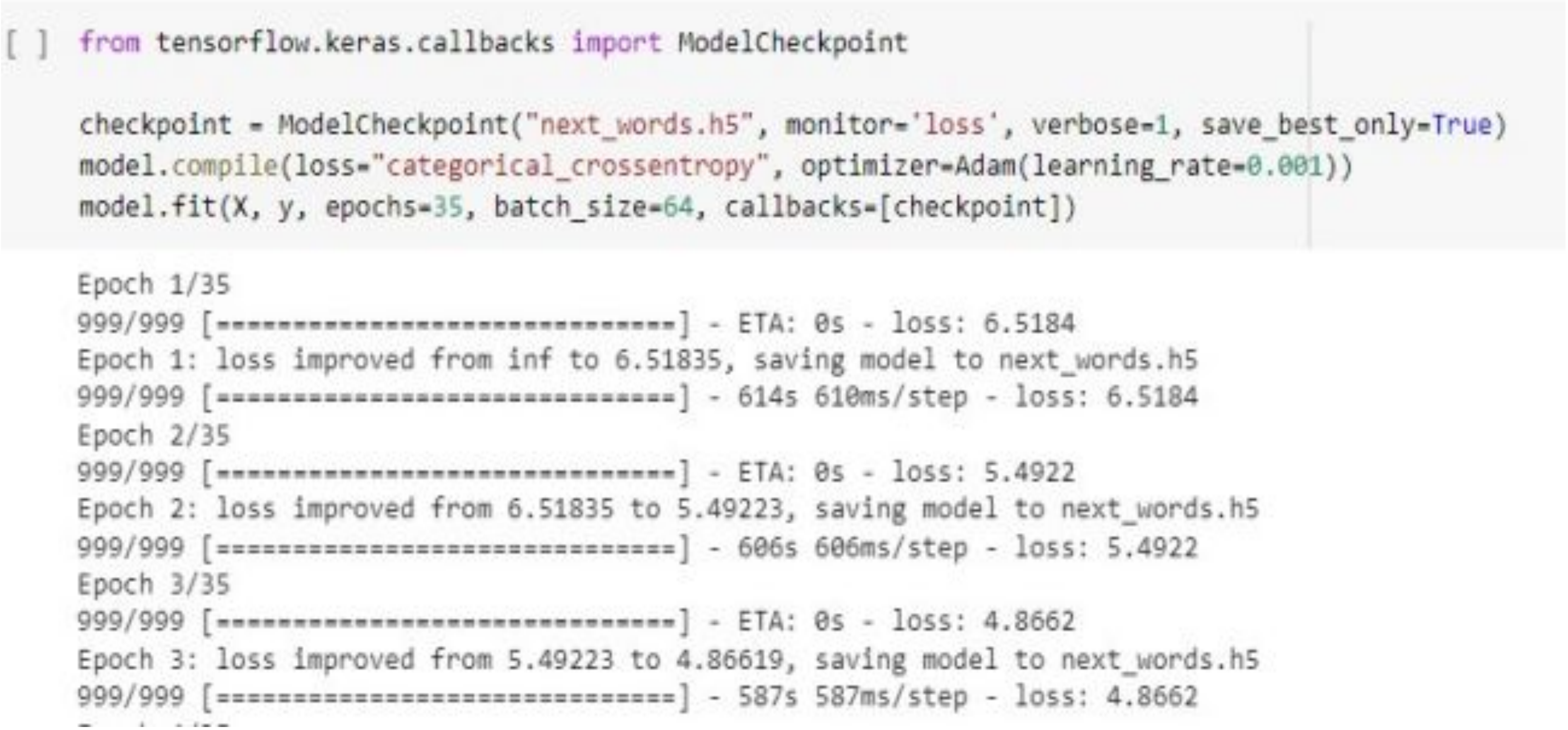
Model Summary



```
[ ]  model.summary()

     Model: "sequential"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      embedding (Embedding)       (None, 3, 10)             66220

      lstm (LSTM)                 (None, 3, 1000)           4044000

      lstm_1 (LSTM)               (None, 1000)              8004000

      dense (Dense)               (None, 1000)              1001000

      dense_1 (Dense)             (None, 6622)              6628622

     =================================================================
     Total params: 19,743,842
     Trainable params: 19,743,842
     Non-trainable params: 0
     _________________________________________________________________
```

Implementation

Training the Ngrams Model

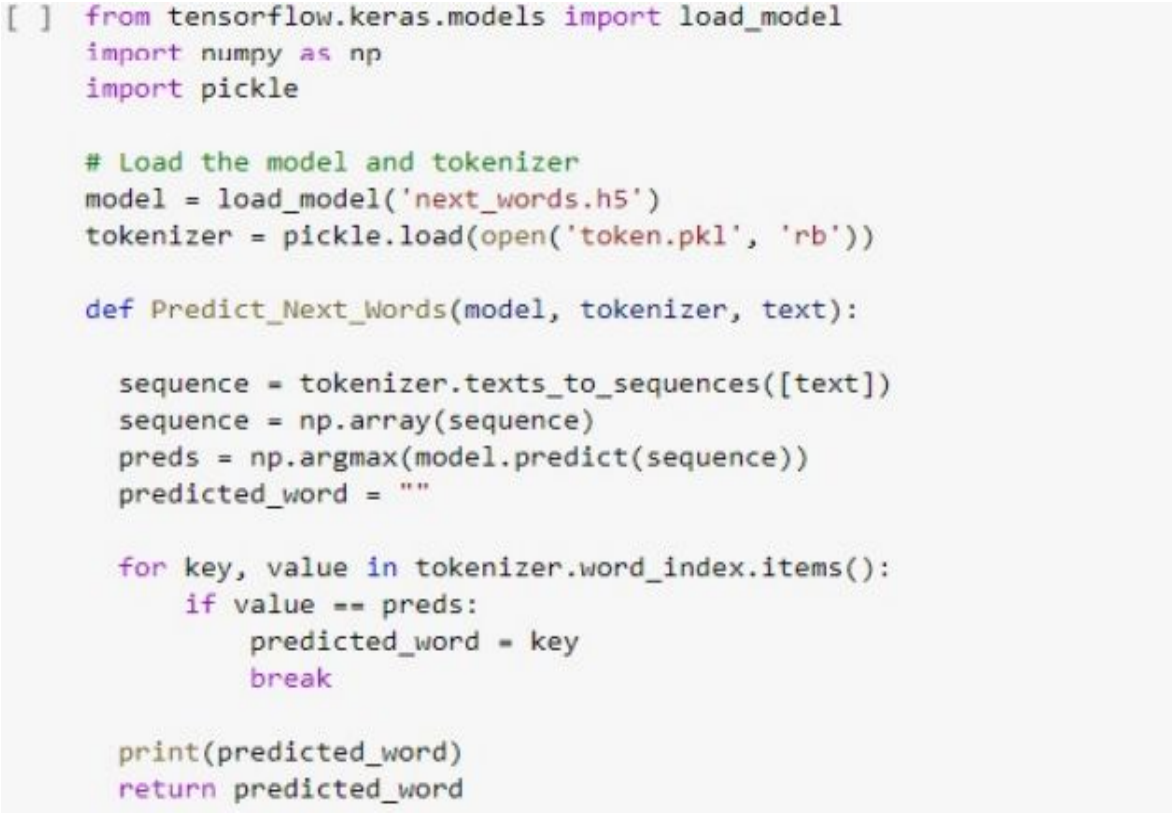


```
[ ]  from tensorflow.keras.callbacks import ModelCheckpoint

     checkpoint = ModelCheckpoint("next_words.h5", monitor='loss', verbose=1, save_best_only=True)
     model.compile(loss="categorical_crossentropy", optimizer=Adam(learning_rate=0.001))
     model.fit(X, y, epochs=35, batch_size=64, callbacks=[checkpoint])
```

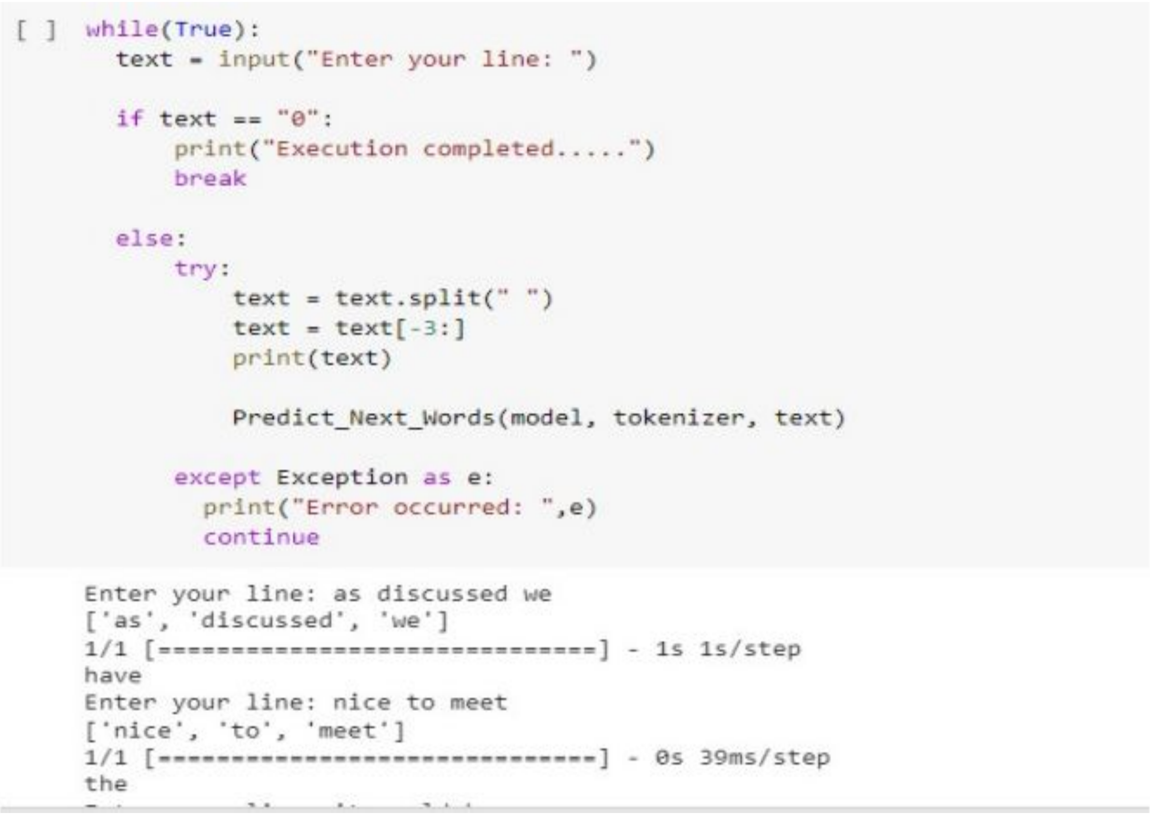
```
Epoch 1/35
999/999 [==============================] - ETA: 0s - loss: 6.5184
Epoch 1: loss improved from inf to 6.51835, saving model to next_words.h5
999/999 [==============================] - 614s 610ms/step - loss: 6.5184
Epoch 2/35
999/999 [==============================] - ETA: 0s - loss: 5.4922
Epoch 2: loss improved from 6.51835 to 5.49223, saving model to next_words.h5
999/999 [==============================] - 606s 606ms/step - loss: 5.4922
Epoch 3/35
999/999 [==============================] - ETA: 0s - loss: 4.8662
Epoch 3: loss improved from 5.49223 to 4.86619, saving model to next_words.h5
999/999 [==============================] - 587s 587ms/step - loss: 4.8662
```

Implementation

Testing the Ngrams Model



```
[ ]  from tensorflow.keras.models import load_model
     import numpy as np
     import pickle

     # Load the model and tokenizer
     model = load_model('next_words.h5')
     tokenizer = pickle.load(open('token.pkl', 'rb'))

     def Predict_Next_Words(model, tokenizer, text):

       sequence = tokenizer.texts_to_sequences([text])
       sequence = np.array(sequence)
       preds = np.argmax(model.predict(sequence))
       predicted_word = ""

       for key, value in tokenizer.word_index.items():
           if value == preds:
               predicted_word = key
               break

       print(predicted_word)
       return predicted_word
```

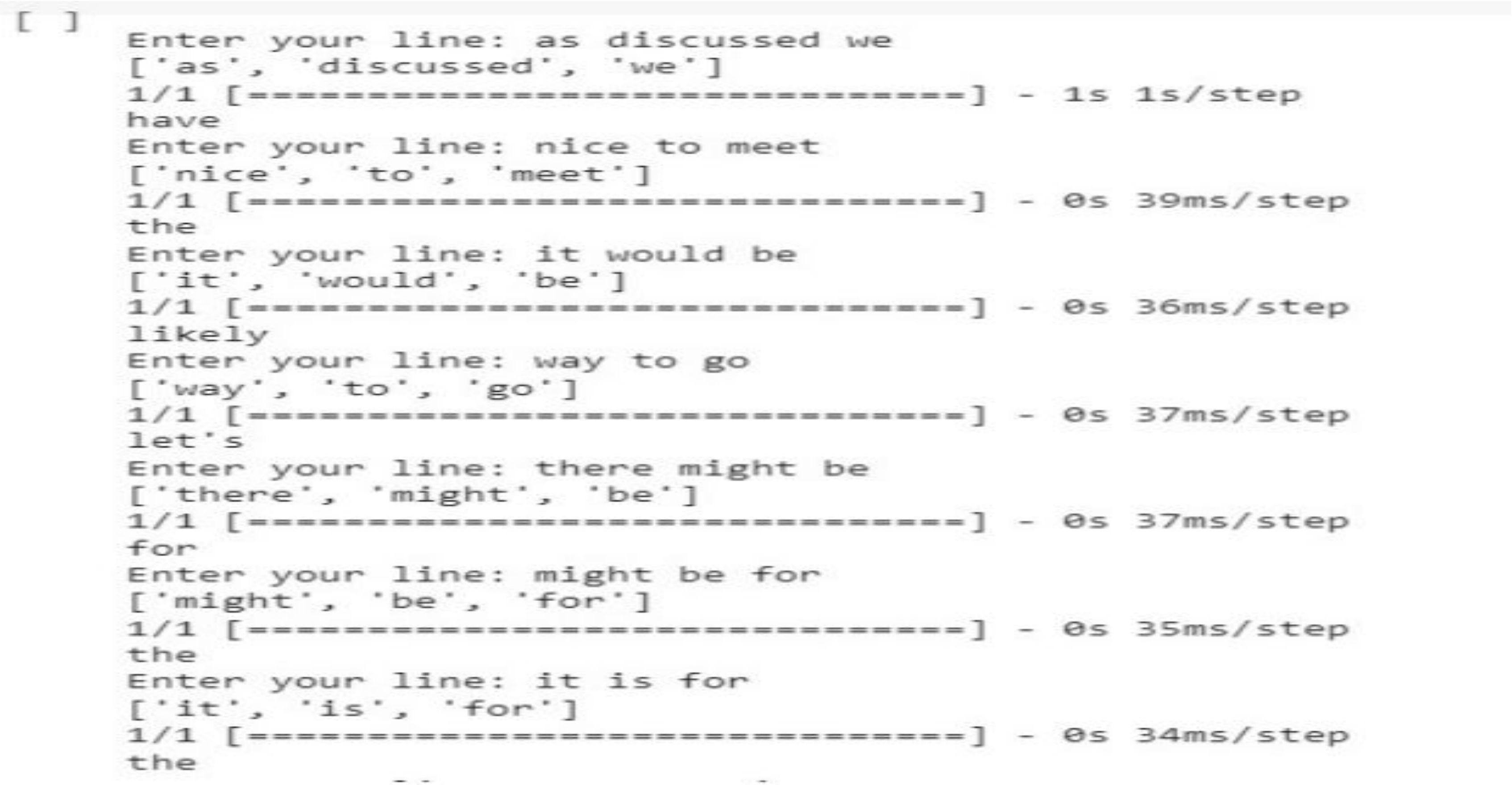


```
[ ]  while(True):
       text = input("Enter your line: ")

       if text == "0":
           print("Execution completed.....")
           break

       else:
           try:
               text = text.split(" ")
               text = text[-3:]
               print(text)

               Predict_Next_Words(model, tokenizer, text)

           except Exception as e:
             print("Error occurred: ",e)
             continue
```

```
Enter your line: as discussed we
['as', 'discussed', 'we']
1/1 [==============================] - 1s 1s/step
have
Enter your line: nice to meet
['nice', 'to', 'meet']
1/1 [==============================] - 0s 39ms/step
the
```

Implementation

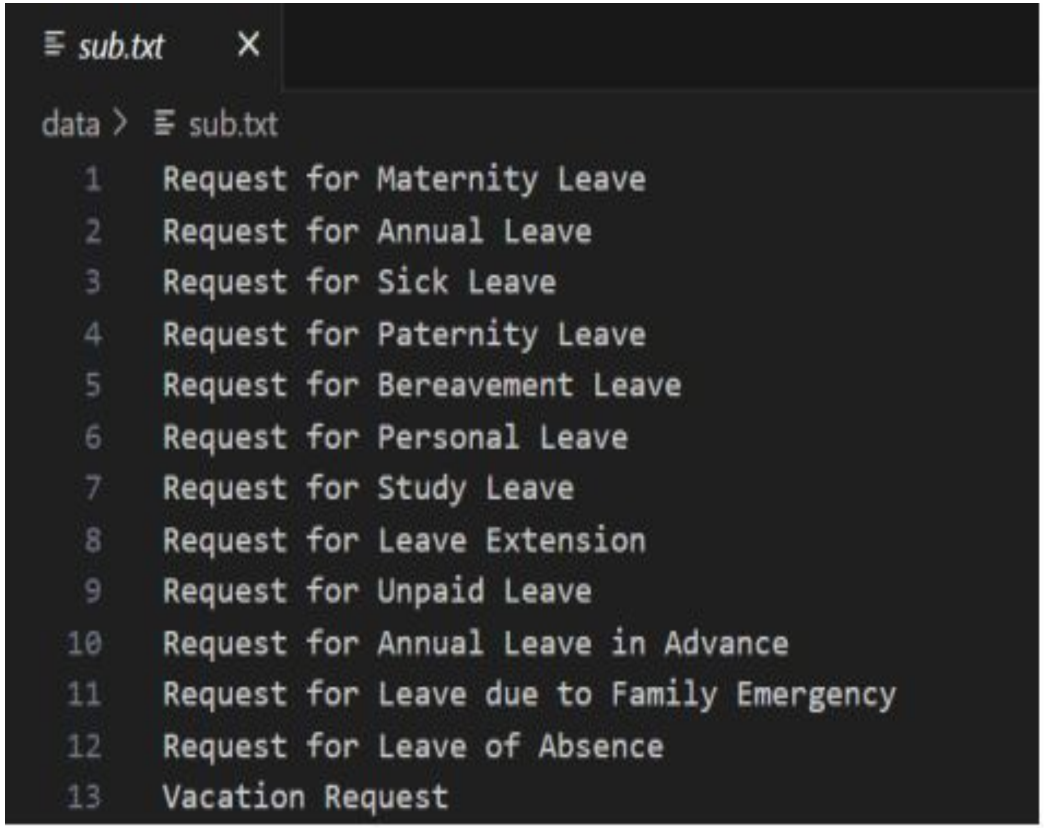
Next Word Prediction



```
[ ]
     Enter your line: as discussed we
     ['as', 'discussed', 'we']
     1/1 [==============================] - 1s 1s/step
     have
     Enter your line: nice to meet
     ['nice', 'to', 'meet']
     1/1 [==============================] - 0s 39ms/step
     the
     Enter your line: it would be
     ['it', 'would', 'be']
     1/1 [==============================] - 0s 36ms/step
     likely
     Enter your line: way to go
     ['way', 'to', 'go']
     1/1 [==============================] - 0s 37ms/step
     let's
     Enter your line: there might be
     ['there', 'might', 'be']
     1/1 [==============================] - 0s 37ms/step
     for
     Enter your line: might be for
     ['might', 'be', 'for']
     1/1 [==============================] - 0s 35ms/step
     the
     Enter your line: it is for
     ['it', 'is', 'for']
     1/1 [==============================] - 0s 34ms/step
     the
```

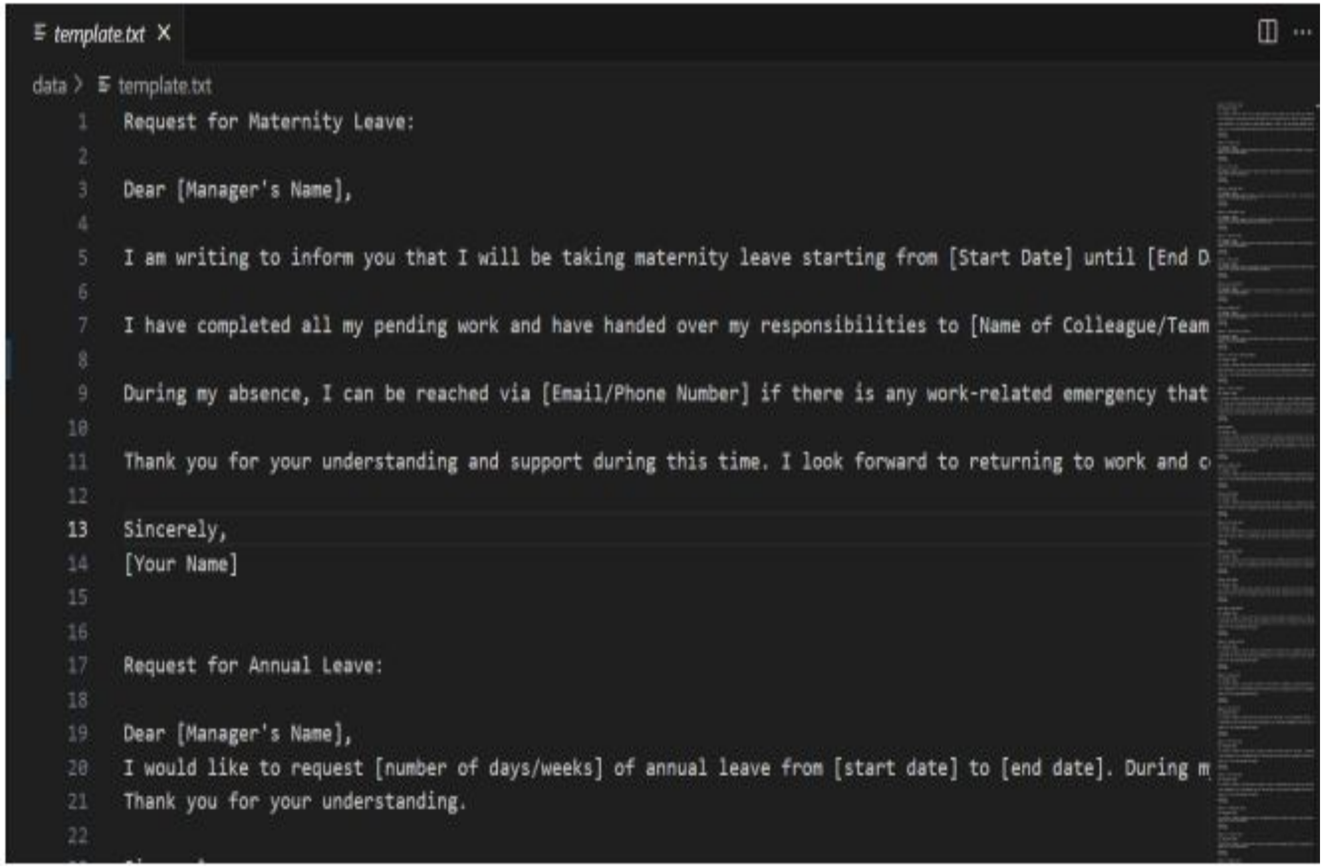
Implementation

Subject Lines Format



```
Request for Maternity Leave
Request for Annual Leave
Request for Sick Leave
Request for Paternity Leave
Request for Bereavement Leave
Request for Personal Leave
Request for Study Leave
Request for Leave Extension
Request for Unpaid Leave
Request for Annual Leave in Advance
Request for Leave due to Family Emergency
Request for Leave of Absence
Vacation Request
```

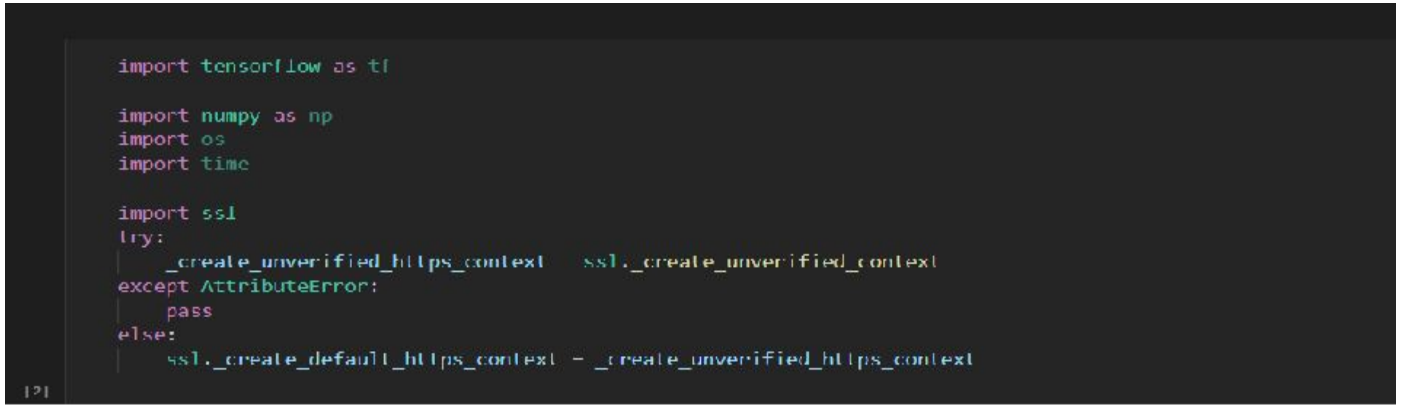
Email Template Format



```
Request for Maternity Leave:

Dear [Manager's Name],

I am writing to inform you that I will be taking maternity leave starting from [Start Date] until [End D

I have completed all my pending work and have handed over my responsibilities to [Name of Colleague/Team

During my absence, I can be reached via [Email/Phone Number] if there is any work-related emergency that

Thank you for your understanding and support during this time. I look forward to returning to work and c

Sincerely,
[Your Name]


Request for Annual Leave:

Dear [Manager's Name],
I would like to request [number of days/weeks] of annual leave from [start date] to [end date]. During m
Thank you for your understanding.

```

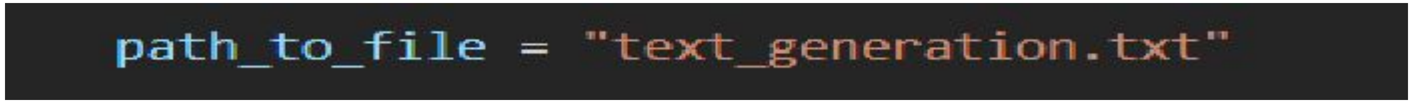
Implementation

Import Necessary Modules



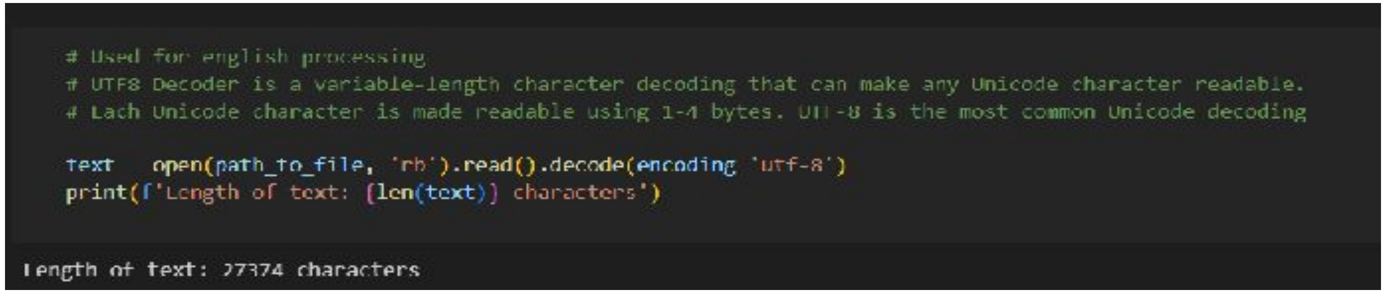
```
import tensorflow as tf

import numpy as np
import os
import time

import ssl
try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context
```

Email Template Format



```
path_to_file = "text_generation.txt"
```

Read the Dataset



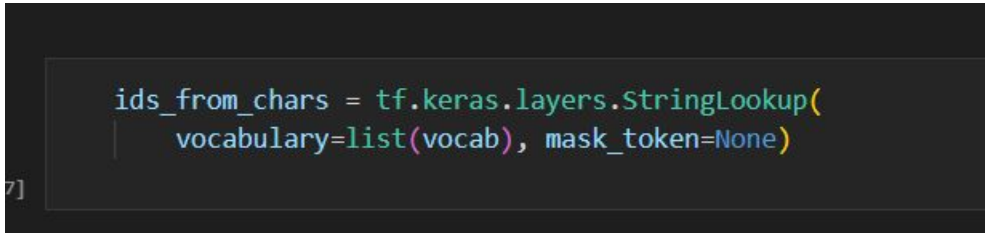
```
# Used for english processing
# UTF8 Decoder is a variable-length character decoding that can make any Unicode character readable.
# Each Unicode character is made readable using 1-4 bytes. UTF-8 is the most common Unicode decoding

text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
print(f'Length of text: {len(text)} characters')
```

```
Length of text: 27374 characters
```

Implementation

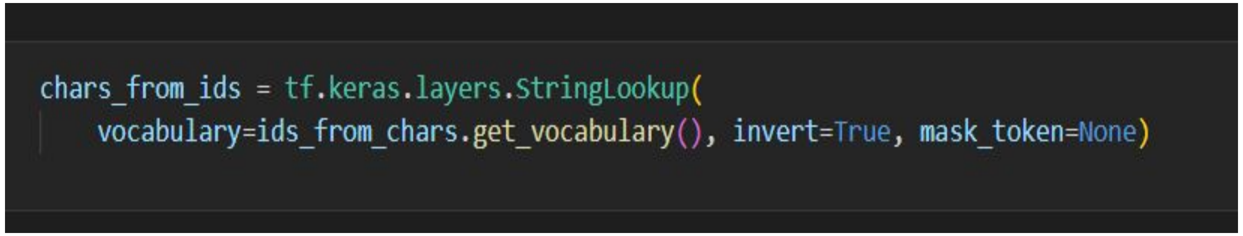
Data Preprocessing

Tokenization



```
ids_from_chars = tf.keras.layers.StringLookup(
    vocabulary=list(vocab), mask_token=None)
```

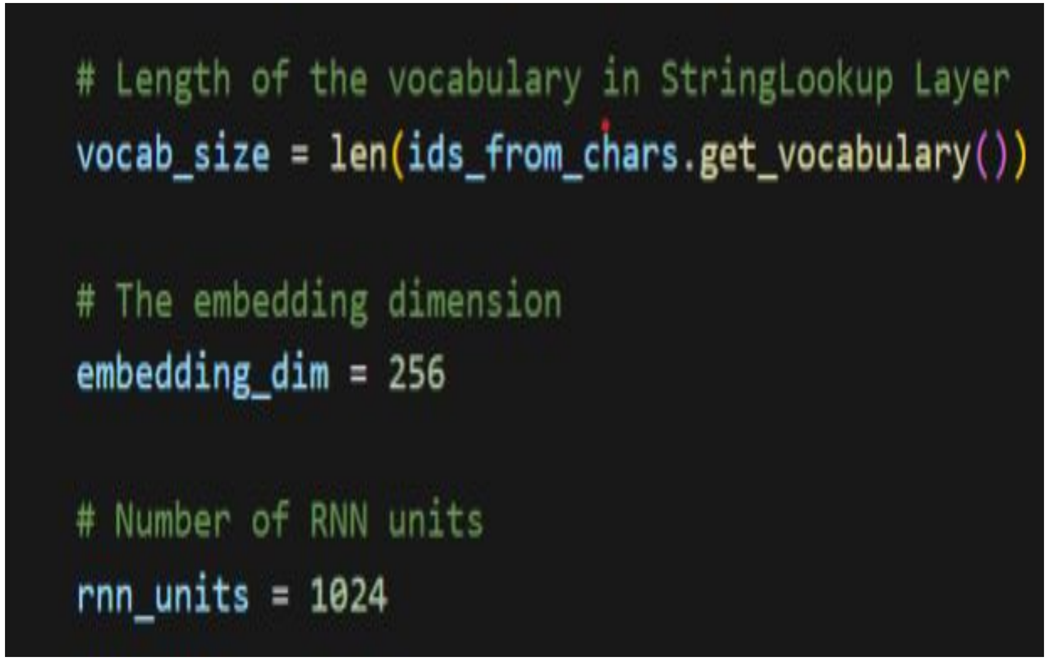
Vectorization



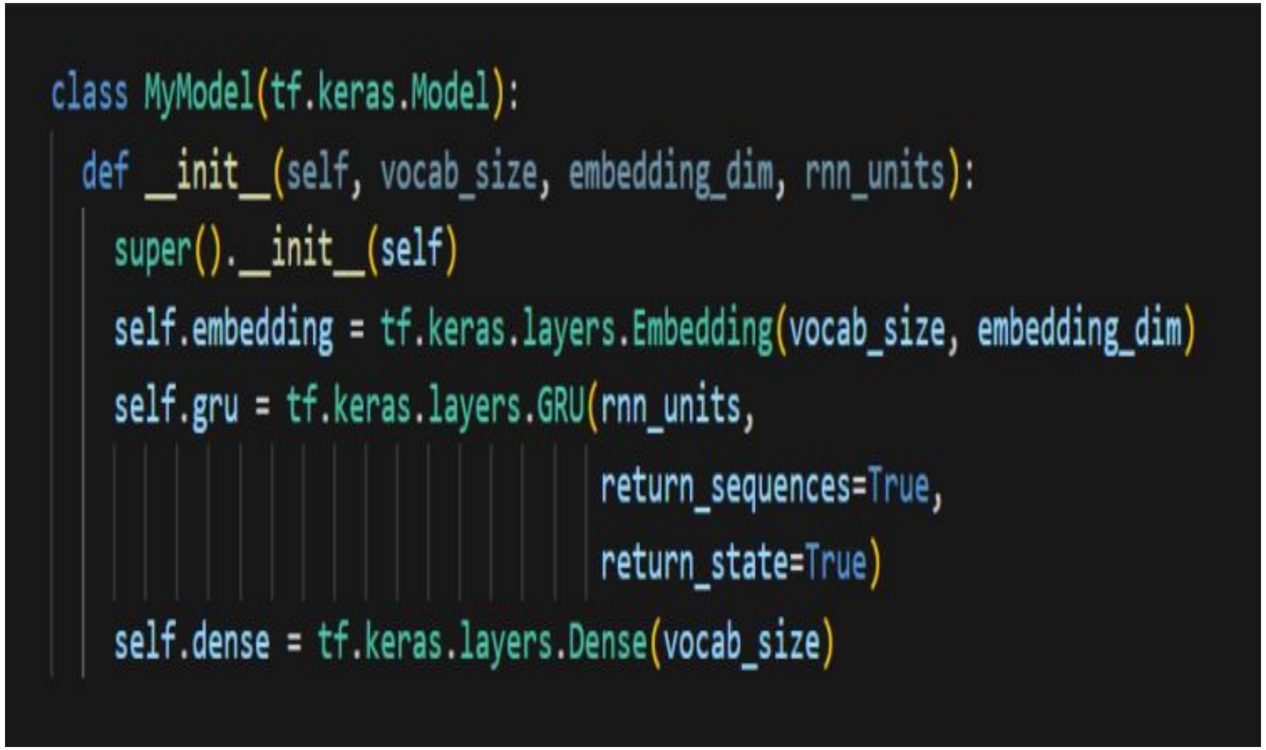
```
chars_from_ids = tf.keras.layers.StringLookup(
    vocabulary=ids_from_chars.get_vocabulary(), invert=True, mask_token=None)
```

Implementation

Building the Model



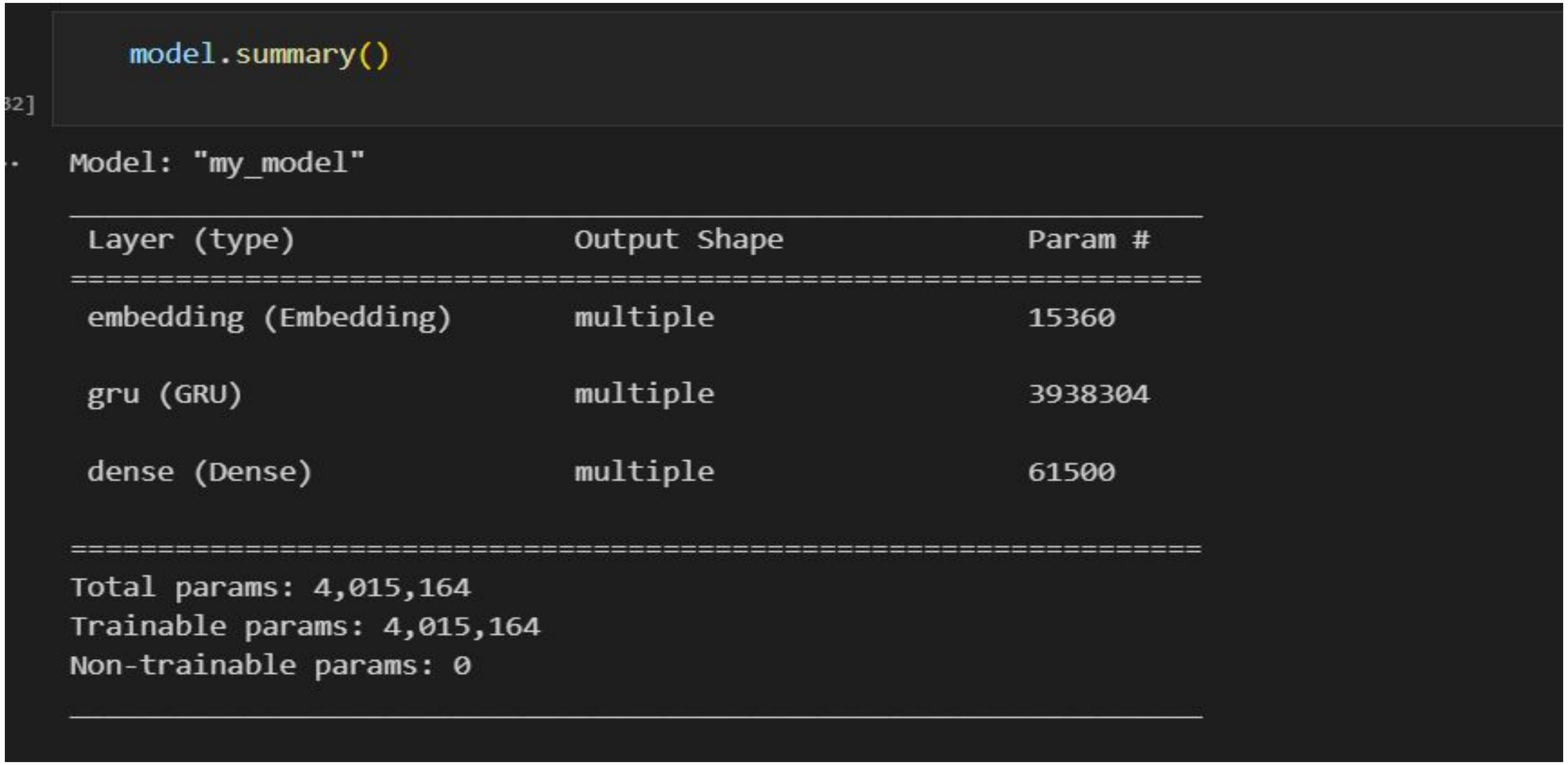
```
# Length of the vocabulary in StringLookup Layer
vocab_size = len(ids_from_chars.get_vocabulary())

# The embedding dimension
embedding_dim = 256

# Number of RNN units
rnn_units = 1024
```



```
class MyModel(tf.keras.Model):
  def __init__(self, vocab_size, embedding_dim, rnn_units):
    super().__init__(self)
    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(rnn_units,
                                   return_sequences=True,
                                   return_state=True)
    self.dense = tf.keras.layers.Dense(vocab_size)
```

Implementation

Model Summary

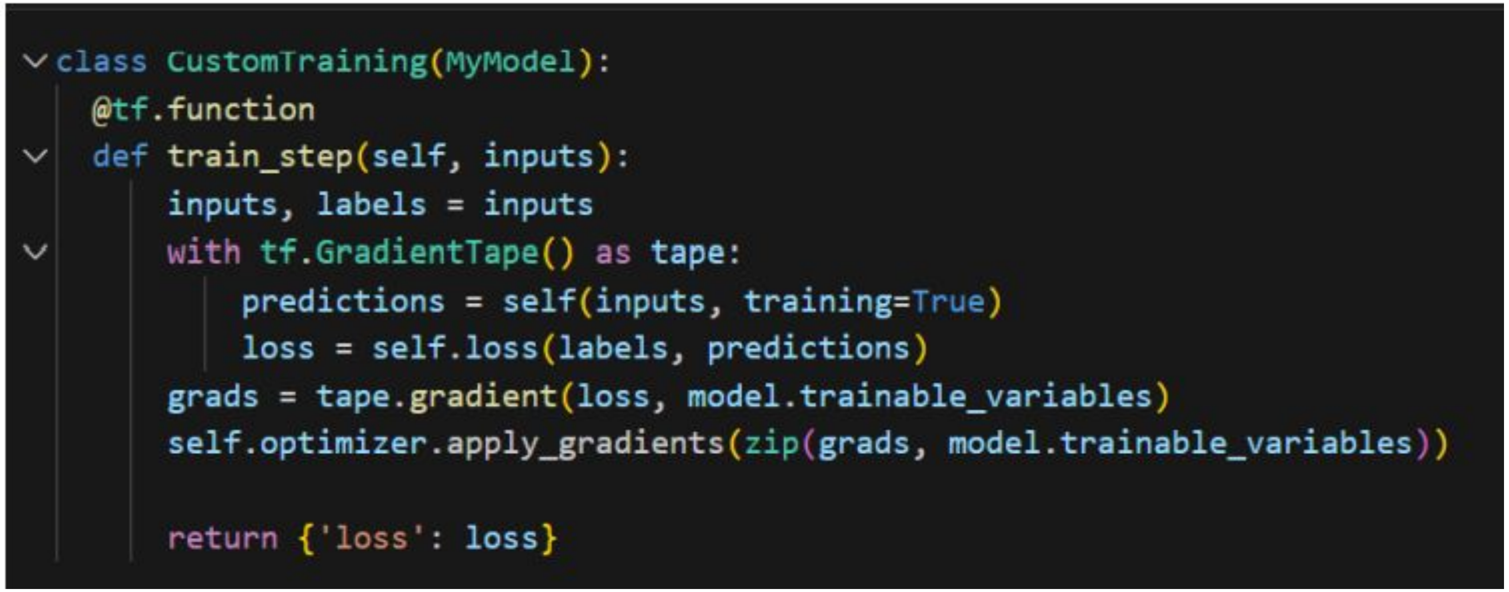


```
model.summary()
```

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       multiple                  15360

 gru (GRU)                   multiple                  3938304

 dense (Dense)               multiple                  61500

=================================================================
Total params: 4,015,164
Trainable params: 4,015,164
Non-trainable params: 0
_________________________________________________________________
```

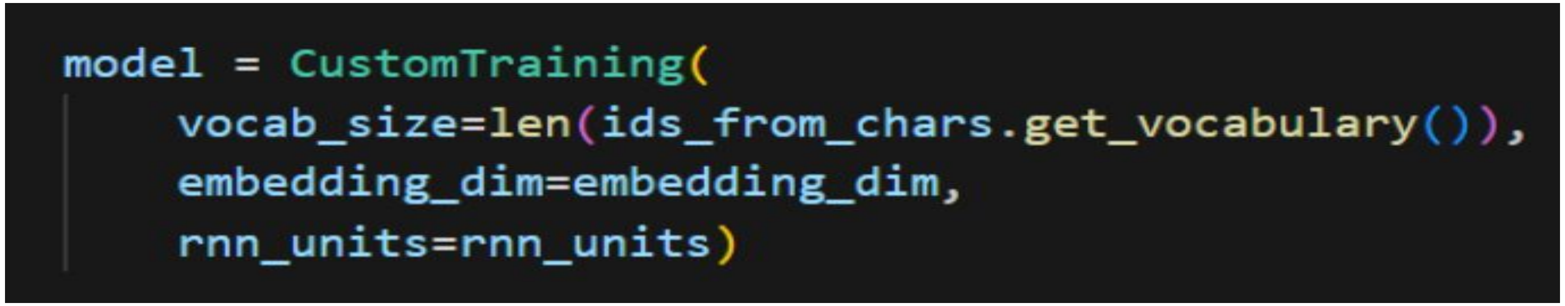
Implementation

Custom Training Model



```
class CustomTraining(MyModel):
  @tf.function
  def train_step(self, inputs):
      inputs, labels = inputs
      with tf.GradientTape() as tape:
          predictions = self(inputs, training=True)
          loss = self.loss(labels, predictions)
      grads = tape.gradient(loss, model.trainable_variables)
      self.optimizer.apply_gradients(zip(grads, model.trainable_variables))

      return {'loss': loss}
```

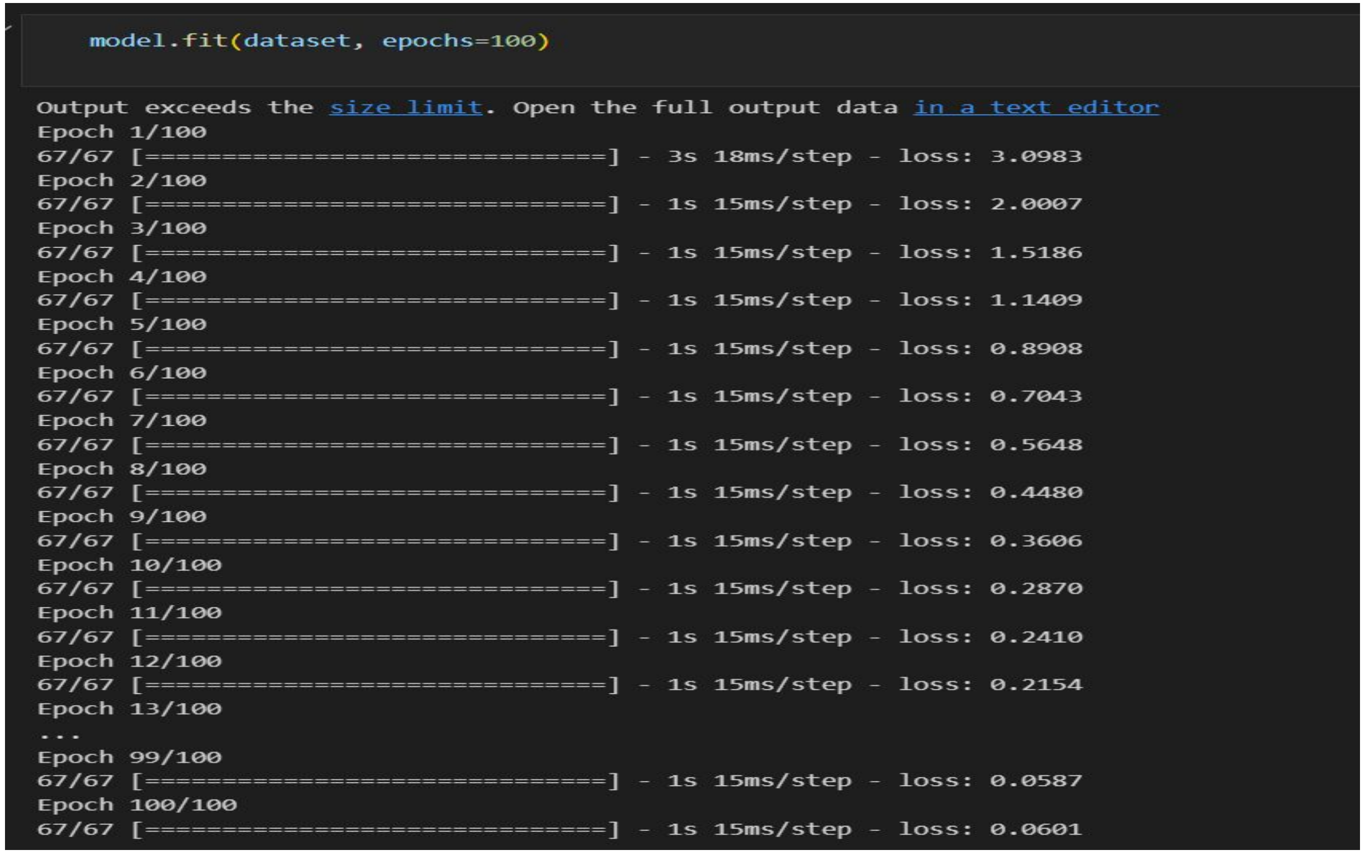
Initialize the Model



```
model = CustomTraining(
    vocab_size=len(ids_from_chars.get_vocabulary()),
    embedding_dim=embedding_dim,
    rnn_units=rnn_units)
```

Implementation

Training The Model

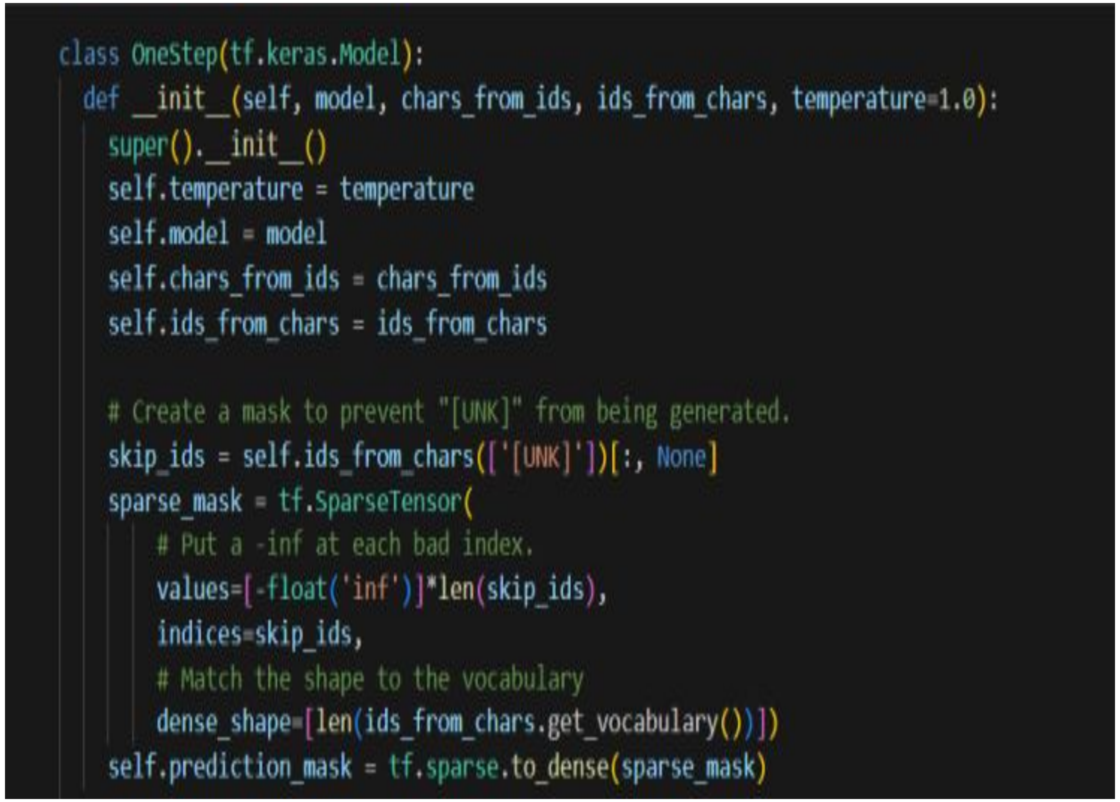


```
model.fit(dataset, epochs=100)
```

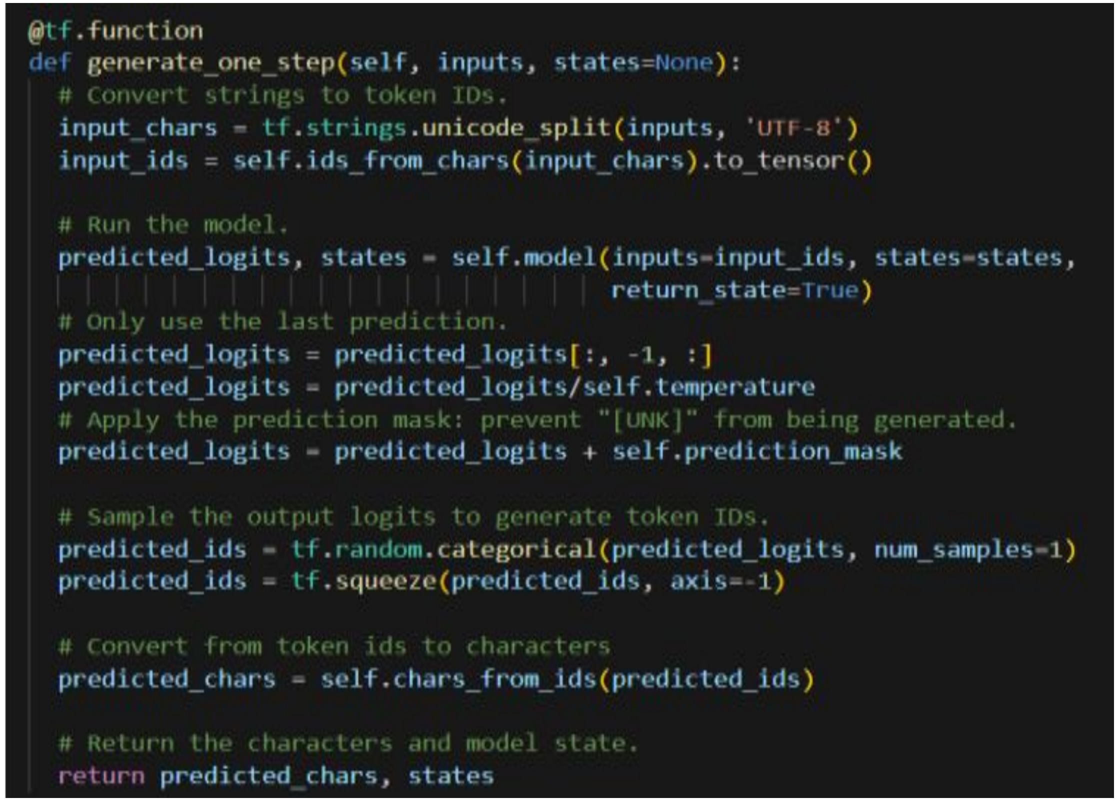
```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/100
67/67 [==============================] - 3s 18ms/step - loss: 3.0983
Epoch 2/100
67/67 [==============================] - 1s 15ms/step - loss: 2.0007
Epoch 3/100
67/67 [==============================] - 1s 15ms/step - loss: 1.5186
Epoch 4/100
67/67 [==============================] - 1s 15ms/step - loss: 1.1409
Epoch 5/100
67/67 [==============================] - 1s 15ms/step - loss: 0.8908
Epoch 6/100
67/67 [==============================] - 1s 15ms/step - loss: 0.7043
Epoch 7/100
67/67 [==============================] - 1s 15ms/step - loss: 0.5648
Epoch 8/100
67/67 [==============================] - 1s 15ms/step - loss: 0.4480
Epoch 9/100
67/67 [==============================] - 1s 15ms/step - loss: 0.3606
Epoch 10/100
67/67 [==============================] - 1s 15ms/step - loss: 0.2870
Epoch 11/100
67/67 [==============================] - 1s 15ms/step - loss: 0.2410
Epoch 12/100
67/67 [==============================] - 1s 15ms/step - loss: 0.2154
Epoch 13/100
...
Epoch 99/100
67/67 [==============================] - 1s 15ms/step - loss: 0.0587
Epoch 100/100
67/67 [==============================] - 1s 15ms/step - loss: 0.0601
```

Implementation

Predict Function



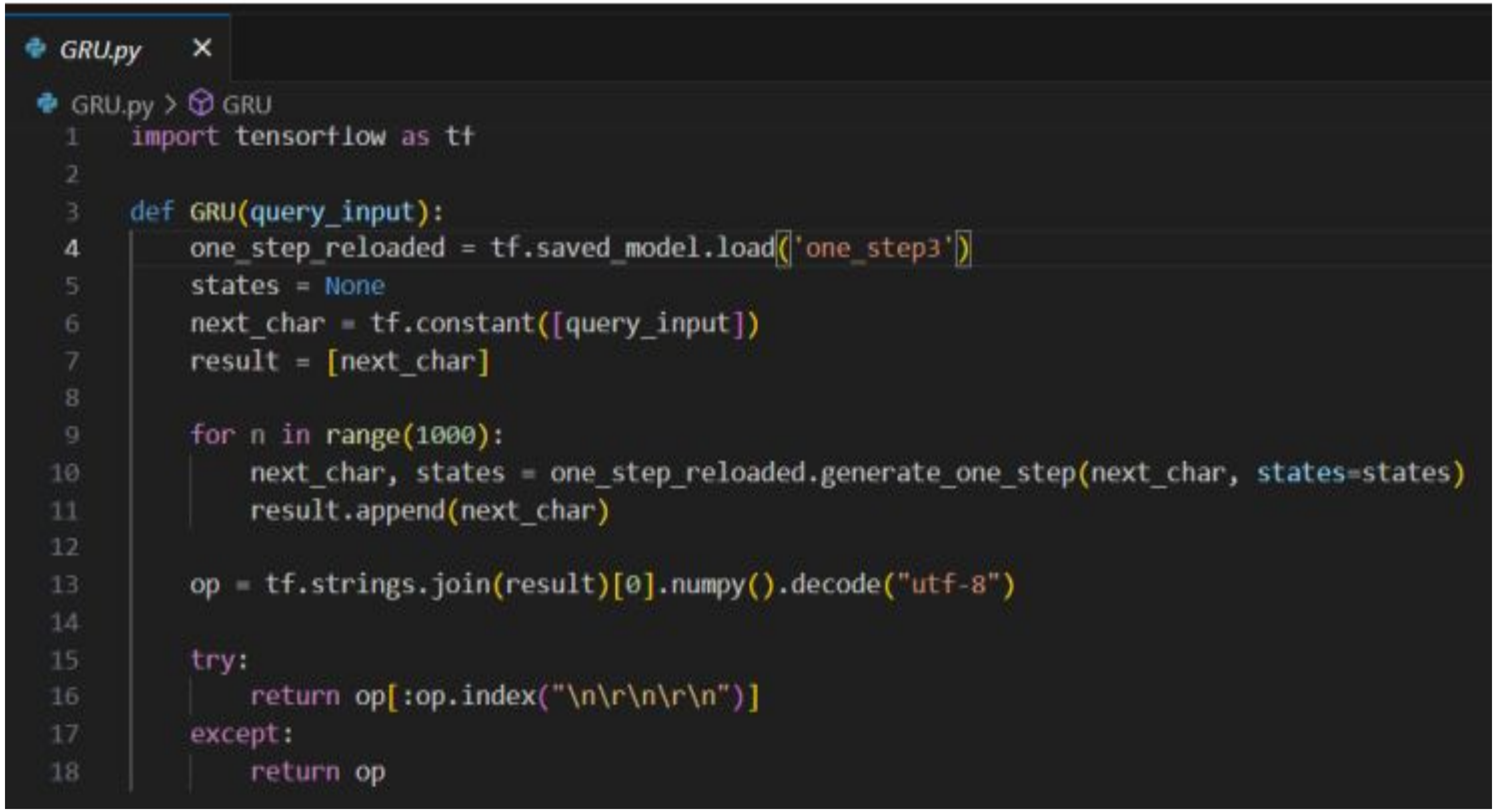
```
class OneStep(tf.keras.Model):
  def __init__(self, model, chars_from_ids, ids_from_chars, temperature=1.0):
    super().__init__()
    self.temperature = temperature
    self.model = model
    self.chars_from_ids = chars_from_ids
    self.ids_from_chars = ids_from_chars

    # Create a mask to prevent "[UNK]" from being generated.
    skip_ids = self.ids_from_chars(['[UNK]'])[:, None]
    sparse_mask = tf.SparseTensor(
        # Put a -inf at each bad index.
        values=[-float('inf')]*len(skip_ids),
        indices=skip_ids,
        # Match the shape to the vocabulary
        dense_shape=[len(ids_from_chars.get_vocabulary())])
    self.prediction_mask = tf.sparse.to_dense(sparse_mask)
```



```
@tf.function
def generate_one_step(self, inputs, states=None):
  # Convert strings to token IDs.
  input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
  input_ids = self.ids_from_chars(input_chars).to_tensor()

  # Run the model.
  predicted_logits, states = self.model(inputs=input_ids, states=states,
                                        return_state=True)
  # Only use the last prediction.
  predicted_logits = predicted_logits[:, -1, :]
  predicted_logits = predicted_logits/self.temperature
  # Apply the prediction mask: prevent "[UNK]" from being generated.
  predicted_logits = predicted_logits + self.prediction_mask

  # Sample the output logits to generate token IDs.
  predicted_ids = tf.random.categorical(predicted_logits, num_samples=1)
  predicted_ids = tf.squeeze(predicted_ids, axis=-1)

  # Convert from token ids to characters
  predicted_chars = self.chars_from_ids(predicted_ids)

  # Return the characters and model state.
  return predicted_chars, states
```

Implementation

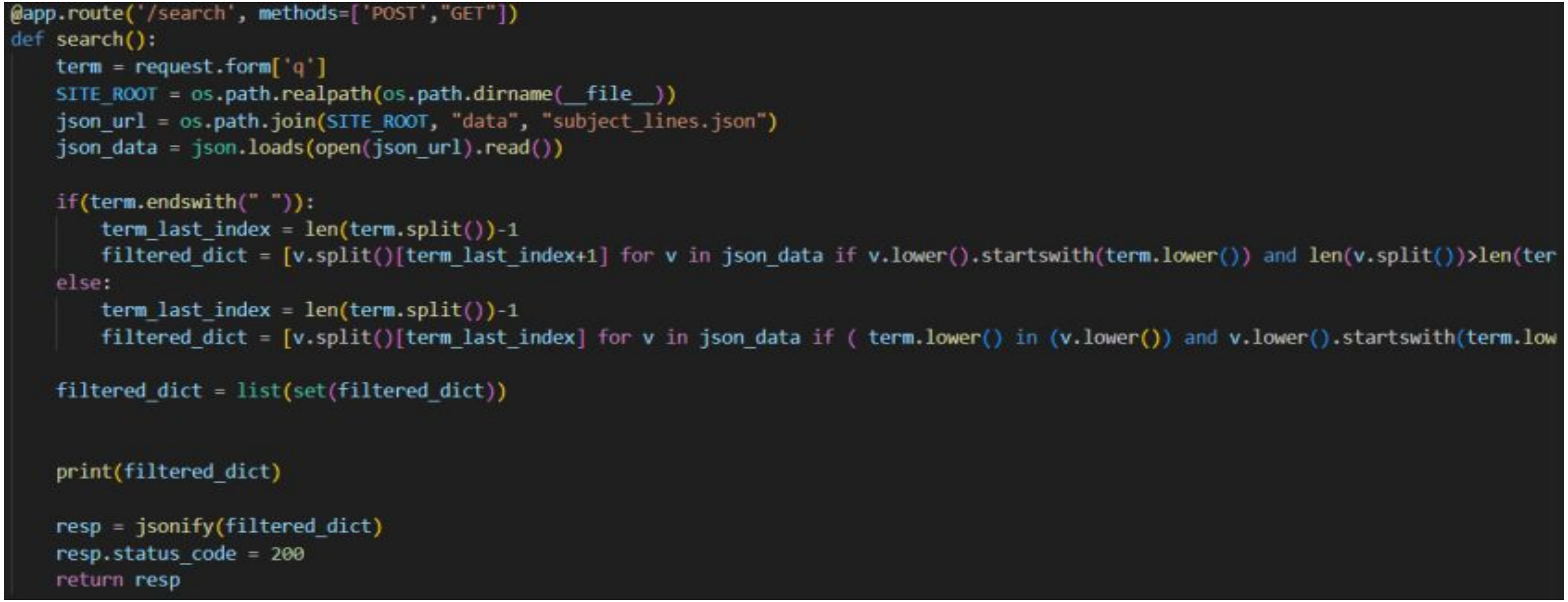
Calling Result



```python
import tensorflow as tf

def GRU(query_input):
    one_step_reloaded = tf.saved_model.load('one_step3')
    states = None
    next_char = tf.constant([query_input])
    result = [next_char]

    for n in range(1000):
        next_char, states = one_step_reloaded.generate_one_step(next_char, states=states)
        result.append(next_char)

    op = tf.strings.join(result)[0].numpy().decode("utf-8")

    try:
        return op[:op.index("\n\r\n\r\n")]
    except:
        return op
```

Implementation

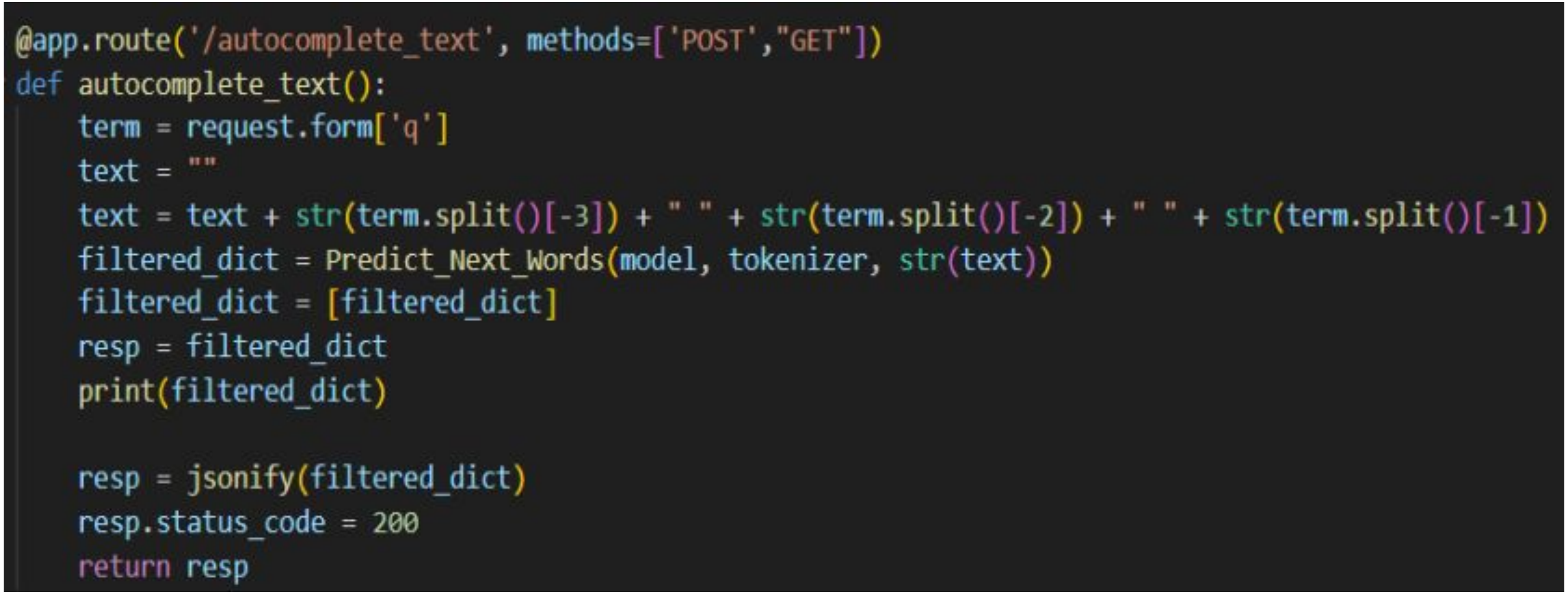
Auto Suggestion in Search Box



```
@app.route('/search', methods=['POST',"GET"])
def search():
    term = request.form['q']
    SITE_ROOT = os.path.realpath(os.path.dirname(__file__))
    json_url = os.path.join(SITE_ROOT, "data", "subject_lines.json")
    json_data = json.loads(open(json_url).read())

    if(term.endswith(" ")):
        term_last_index = len(term.split())-1
        filtered_dict = [v.split()[term_last_index+1] for v in json_data if v.lower().startswith(term.lower()) and len(v.split())>len(ter
    else:
        term_last_index = len(term.split())-1
        filtered_dict = [v.split()[term_last_index] for v in json_data if ( term.lower() in (v.lower()) and v.lower().startswith(term.low

    filtered_dict = list(set(filtered_dict))


    print(filtered_dict)

    resp = jsonify(filtered_dict)
    resp.status_code = 200
    return resp
```

Implementation

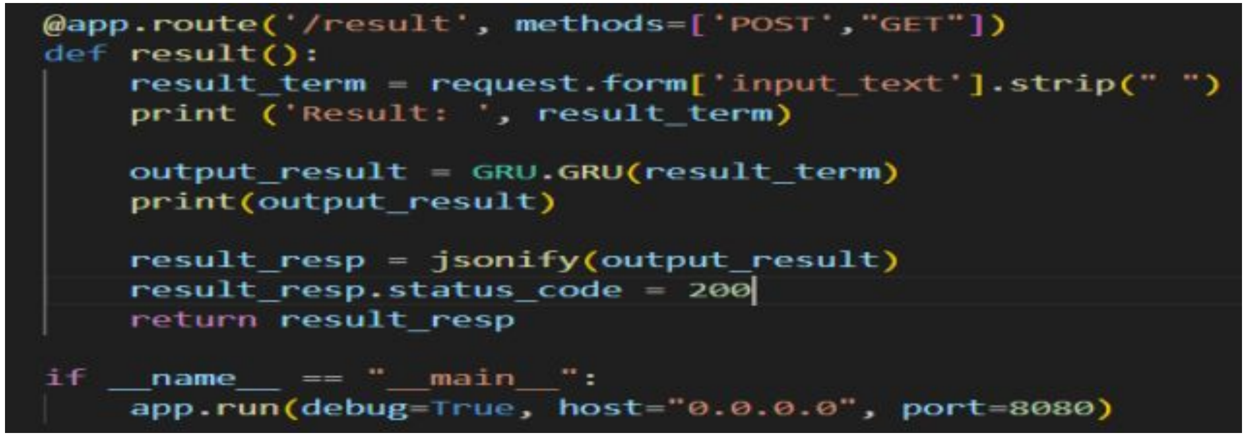
Predicting Next Word in the Template



```
@app.route('/autocomplete_text', methods=['POST',"GET"])
def autocomplete_text():
    term = request.form['q']
    text = ""
    text = text + str(term.split()[-3]) + " " + str(term.split()[-2]) + " " + str(term.split()[-1])
    filtered_dict = Predict_Next_Words(model, tokenizer, str(text))
    filtered_dict = [filtered_dict]
    resp = filtered_dict
    print(filtered_dict)

    resp = jsonify(filtered_dict)
    resp.status_code = 200
    return resp
```

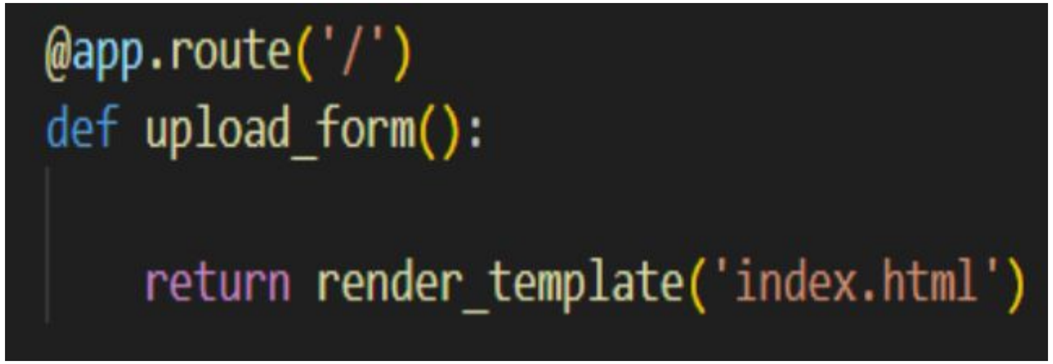
Implementation

Generating Template



```
@app.route('/result', methods=['POST',"GET"])
def result():
    result_term = request.form['input_text'].strip(" ")
    print ('Result: ', result_term)

    output_result = GRU.GRU(result_term)
    print(output_result)

    result_resp = jsonify(output_result)
    result_resp.status_code = 200
    return result_resp

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=8080)
```

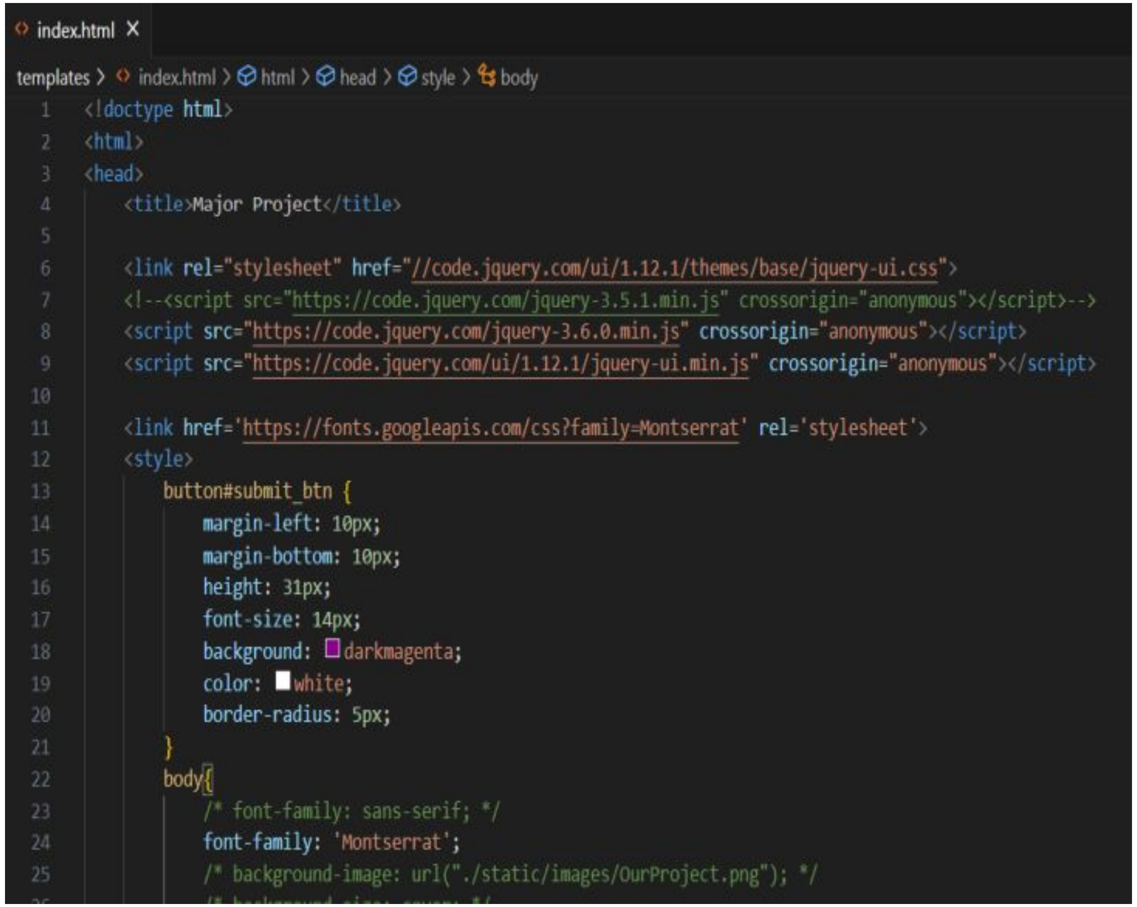
Rendering the html template on visiting the url



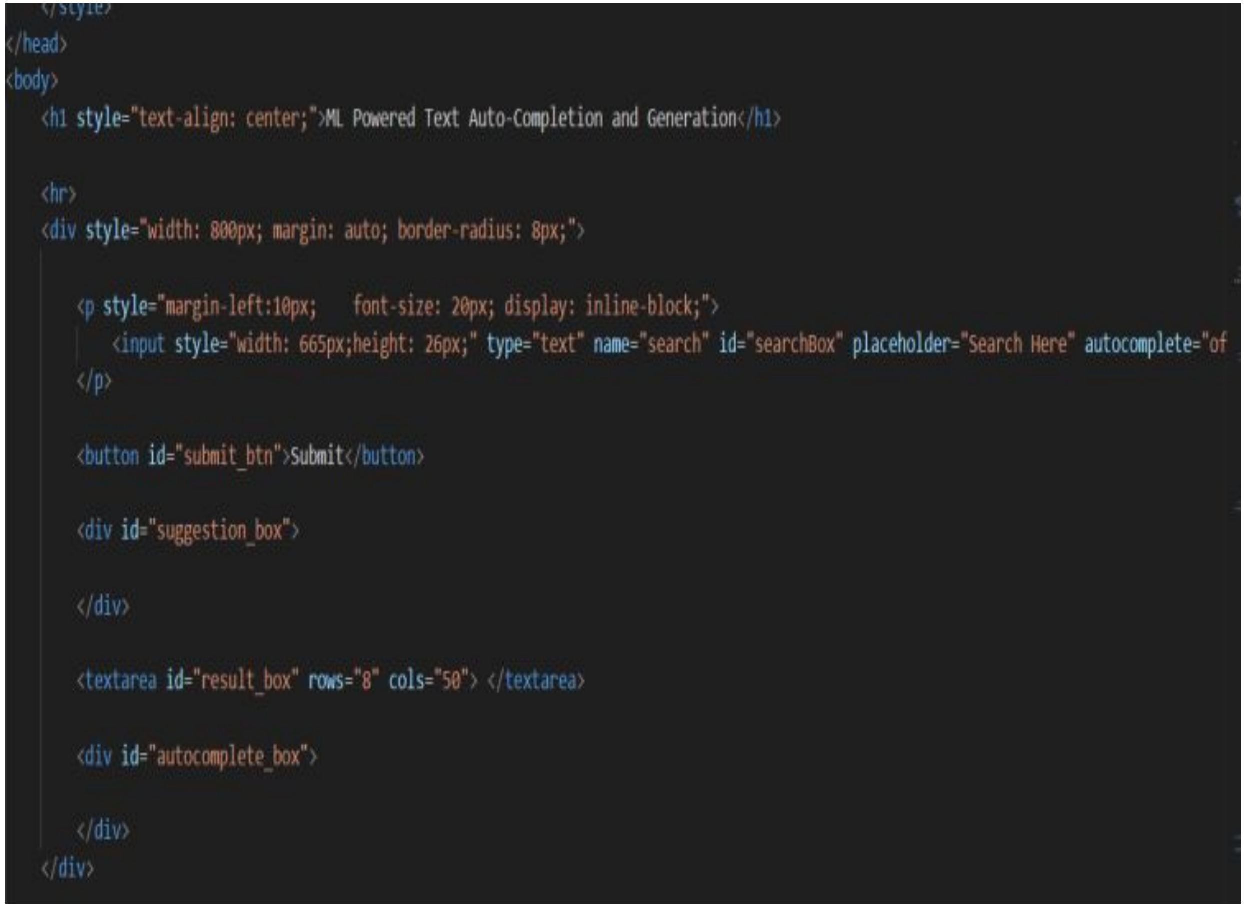
```
@app.route('/')
def upload_form():


    return render_template('index.html')
```

Implementation

HTML Template



```html
<!doctype html>
<html>
<head>
    <title>Major Project</title>

    <link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
    <!--<script src="https://code.jquery.com/jquery-3.5.1.min.js" crossorigin="anonymous"></script>-->
    <script src="https://code.jquery.com/jquery-3.6.0.min.js" crossorigin="anonymous"></script>
    <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js" crossorigin="anonymous"></script>

    <link href='https://fonts.googleapis.com/css?family=Montserrat' rel='stylesheet'>
    <style>
        button#submit_btn {
            margin-left: 10px;
            margin-bottom: 10px;
            height: 31px;
            font-size: 14px;
            background: ■darkmagenta;
            color: □white;
            border-radius: 5px;
        }
        body{
            /* font-family: sans-serif; */
            font-family: 'Montserrat';
            /* background-image: url("./static/images/OurProject.png"); */
```

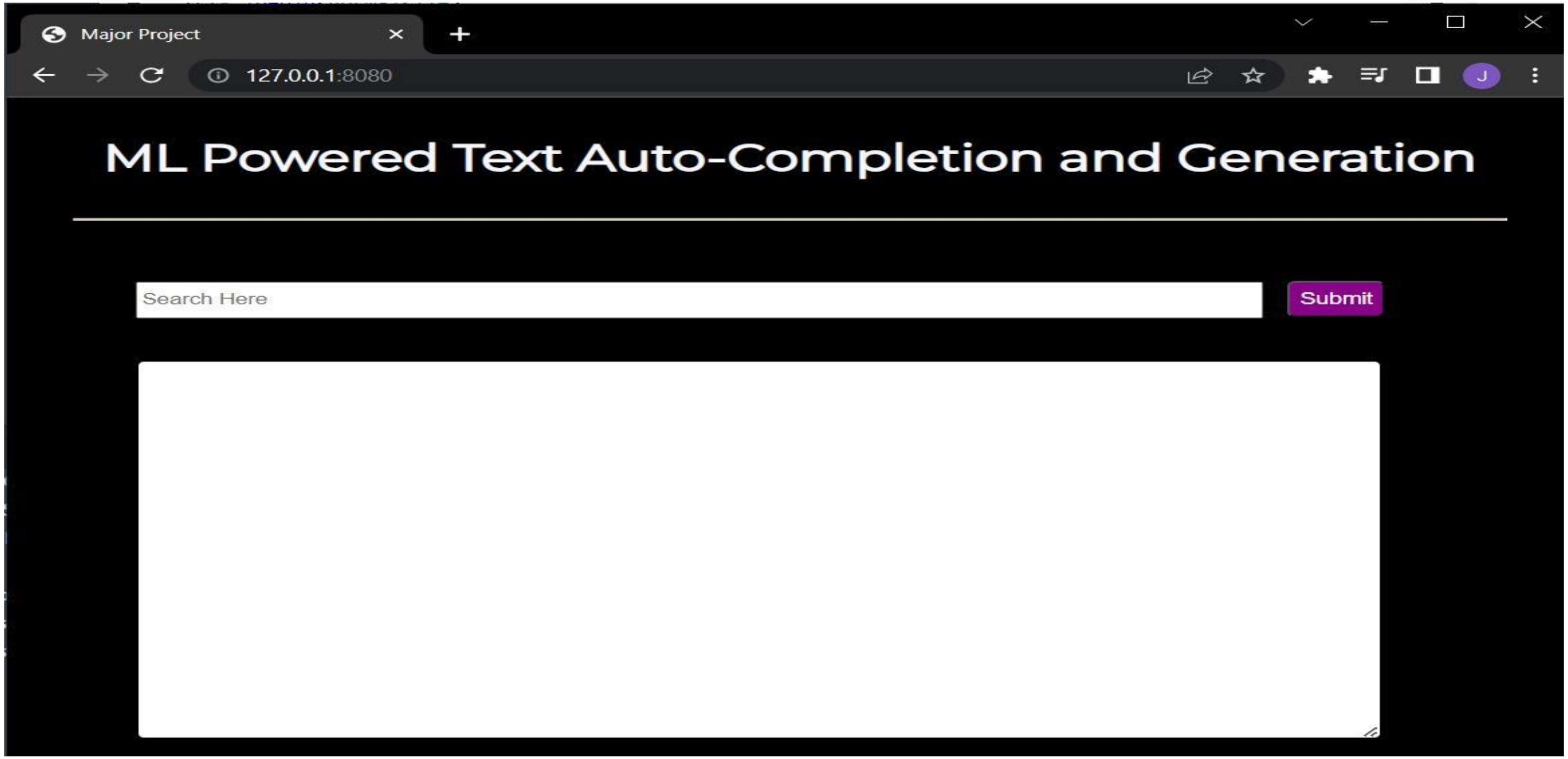


```html
    </style>
</head>
<body>
    <h1 style="text-align: center;">ML Powered Text Auto-Completion and Generation</h1>

    <hr>
    <div style="width: 800px; margin: auto; border-radius: 8px;">

        <p style="margin-left:10px;    font-size: 20px; display: inline-block;">
            <input style="width: 665px;height: 26px;" type="text" name="search" id="searchBox" placeholder="Search Here" autocomplete="of
        </p>

        <button id="submit_btn">Submit</button>

        <div id="suggestion_box">

        </div>

        <textarea id="result_box" rows="8" cols="50"> </textarea>

        <div id="autocomplete_box">

        </div>
    </div>
```

Execution video

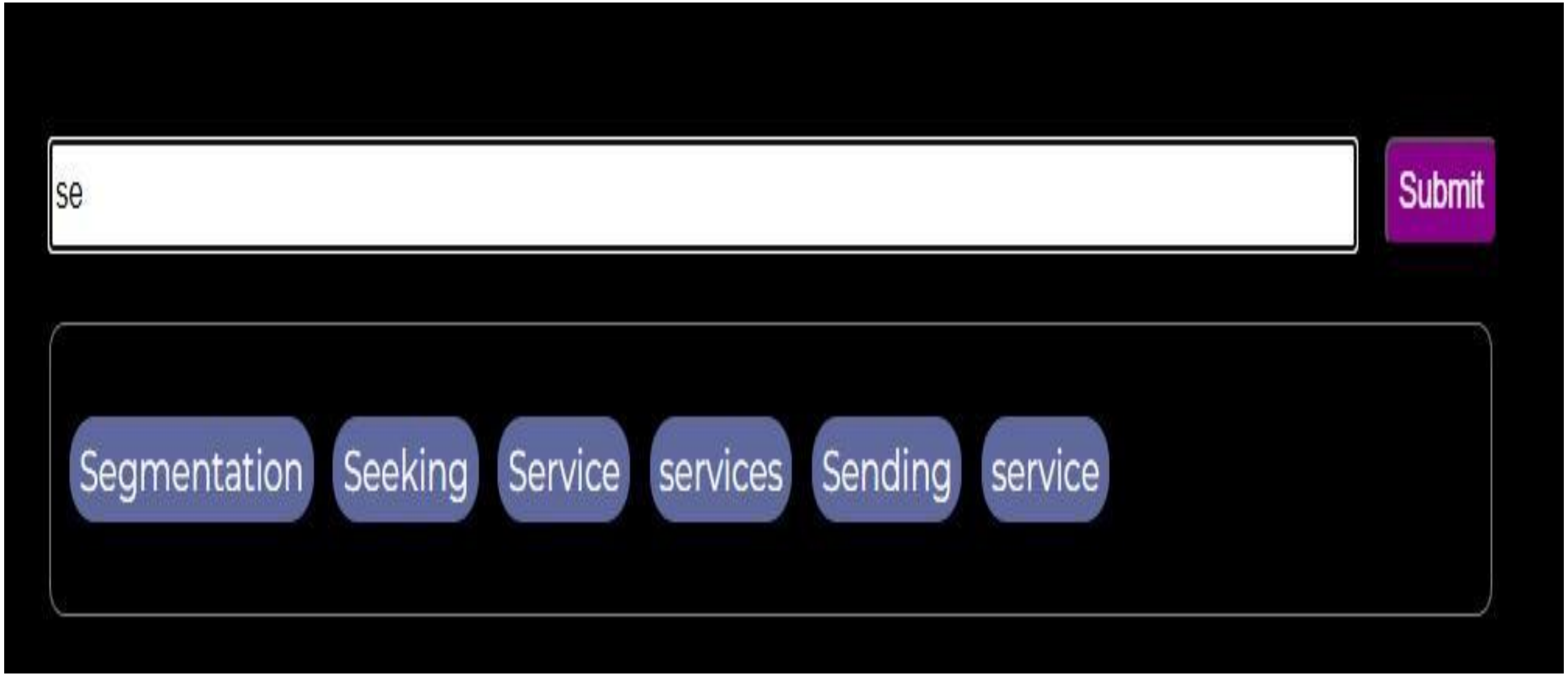
https://drive.google.com/file/d/1vTzvOnBFmeTrqKabbwxgaxOzmXmG9lJ8/view?usp=sharing

Analysis of Results

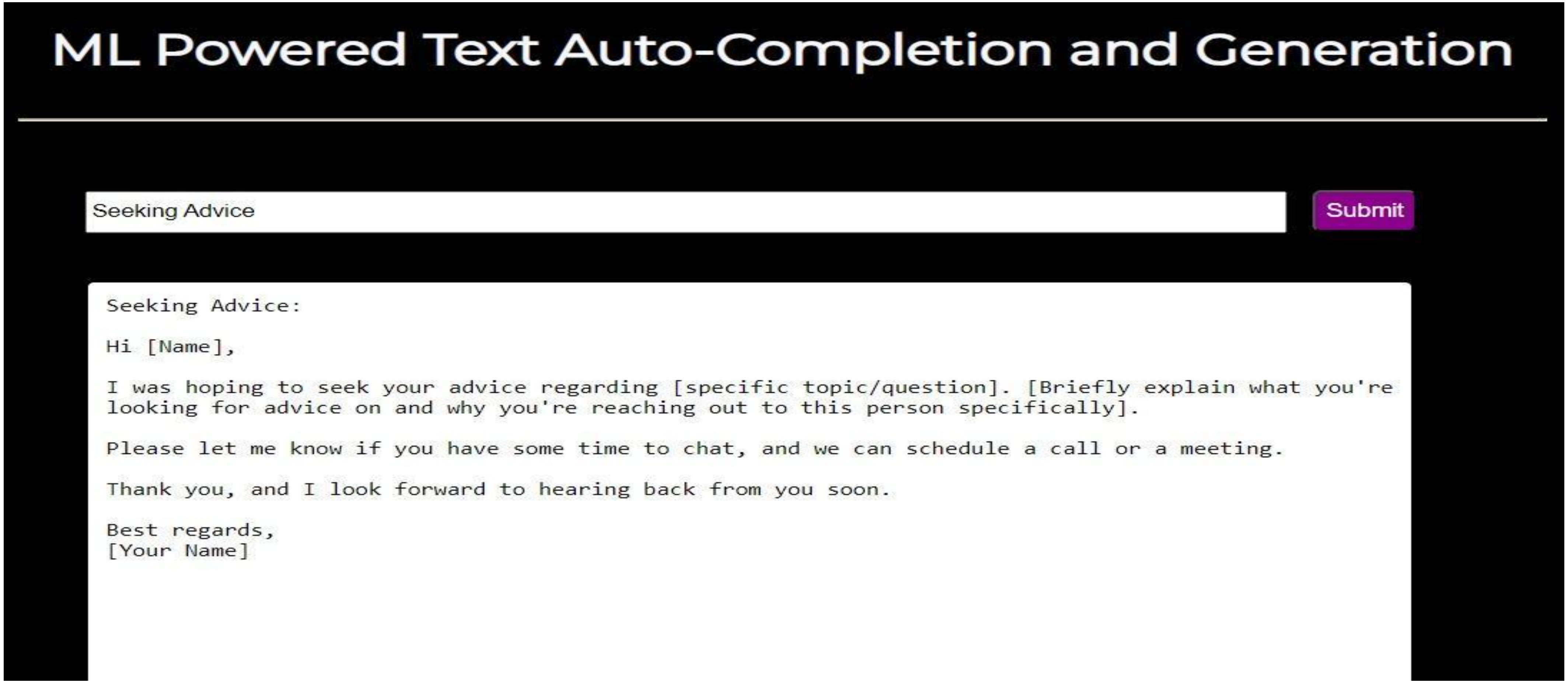


ML Powered Text Auto-Completion and Generation

Analysis of Results

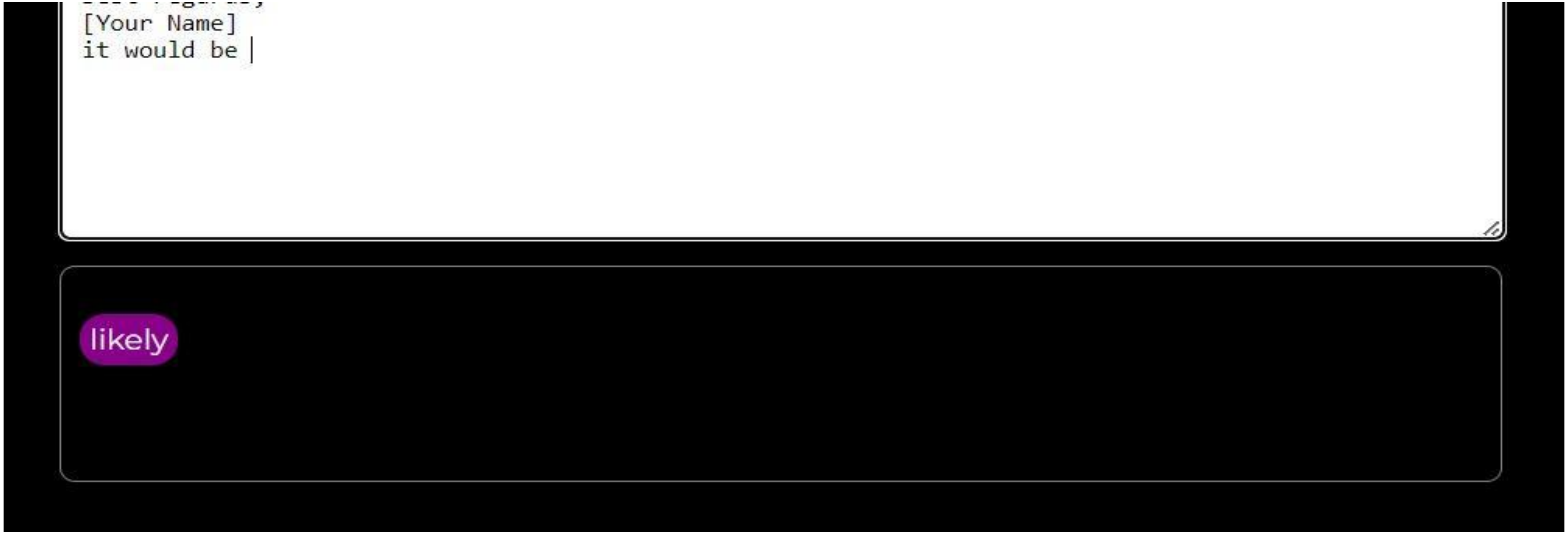


Analysis of Results



```
Seeking Advice:

Hi [Name],

I was hoping to seek your advice regarding [specific topic/question]. [Briefly explain what you're
looking for advice on and why you're reaching out to this person specifically].

Please let me know if you have some time to chat, and we can schedule a call or a meeting.

Thank you, and I look forward to hearing back from you soon.

Best regards,
[Your Name]
```

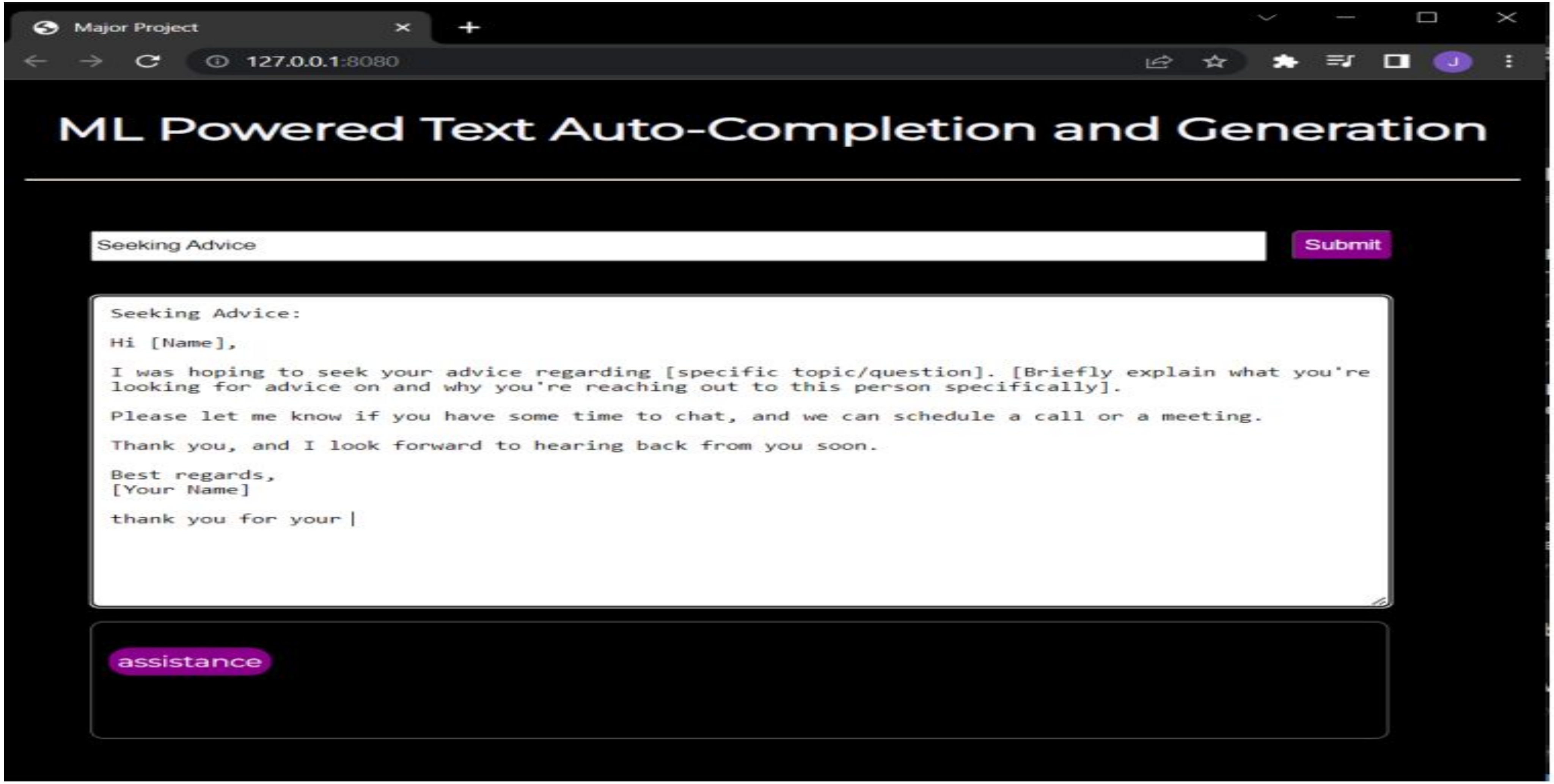
Analysis of Results



```
[Your Name]
it would be
```

likely

Analysis of Results



ML Powered Text Auto-Completion and Generation

Seeking Advice

```
Seeking Advice:

Hi [Name],

I was hoping to seek your advice regarding [specific topic/question]. [Briefly explain what you're
looking for advice on and why you're reaching out to this person specifically].

Please let me know if you have some time to chat, and we can schedule a call or a meeting.

Thank you, and I look forward to hearing back from you soon.

Best regards,
[Your Name]

thank you for your
```

Conclusion & Future Scope

- This text Generation is done by classifying the subject line in a email entered by the user. When the user enters the subject line then our model would be classifying it ,and would generate the text i.e body of the email.

- Multilingual support, collaboration features, integration with voice assistants and integration with other communication platforms will be considered as future scope.

References

- Mia Xu Chen, Benjamin N Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, Timothy Sohn, and Yonghui Wu. 2019. “Gmail Smart Compose: Real-Time Assisted Writing”. In The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ’19), August 4–8, 2019, Anchorage, AK, USA.
- Yue Weng, Huaixiu Zheng, Franziska Bell, and Gokhan Tur. 2019. “OCC: A Smart Reply System for Efficient In-App Communications". In Proceedings of The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Anchorage, AK, USA, August 4–8, 2019 (KDD ’19).
- Rajeev Gupta, Ranganath Kondapally, Chakrapani Ravi Kiran. “Impersonation: Modeling Persona in Smart Responses to Email”. IJCAI 2018 conference. Workshop on Humanizing AI
- Shao T., Chen H., Chen W. “Query auto-completion based on word2vec semantic similarity”. J. Phys. Conf. Ser. 1004(1), 12–18 (2018)
- Aaron Jaech and Mari Ostendorf. 2018. “Personalized Language Model for Query Auto-Completion”. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics.

Thank You