Fall 2023 DATA 225 Deep Learning Technologies

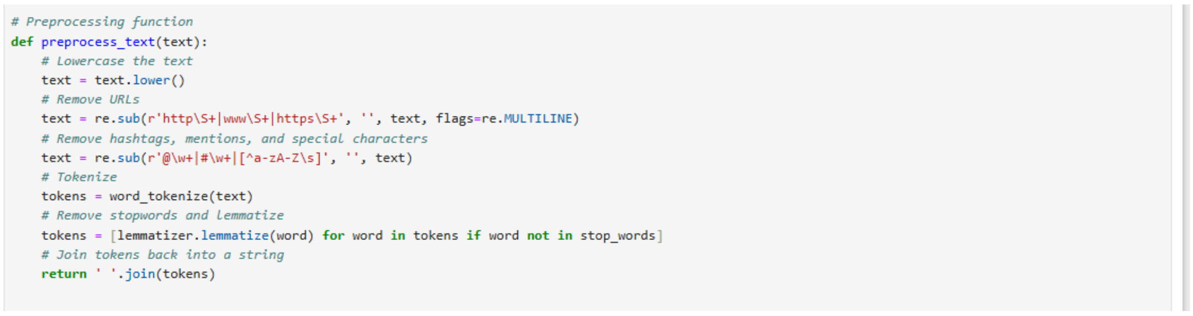
<u>Homework – 4</u>

Name :- Prayag Nikul Purani

SJSU Id :- 017416737

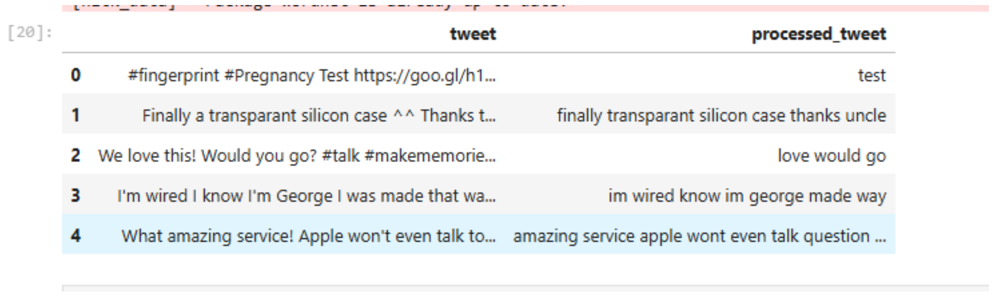
Problem 1 - (Coding):-

a.



```python
# Preprocessing function
def preprocess_text(text):
    # Lowercase the text
    text = text.lower()
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # Remove hashtags, mentions, and special characters
    text = re.sub(r'@\w+|#\w+|[^a-zA-Z\s]', '', text)
    # Tokenize
    tokens = word_tokenize(text)
    # Remove stopwords and lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    # Join tokens back into a string
    return ' '.join(tokens)
```

The first image shows a Python function, preprocess_text, designed to clean and prepare text data for analysis. It performs several preprocessing steps, including converting text to lowercase for uniformity, removing URLs, hashtags, mentions, and special characters using regular expressions. It then tokenizes the text into individual words, removes stopwords (common, less meaningful words), and lemmatizes the words to their base forms (e.g., "running" → "run"). Finally, the cleaned tokens are joined back into a single string and returned as the processed text.

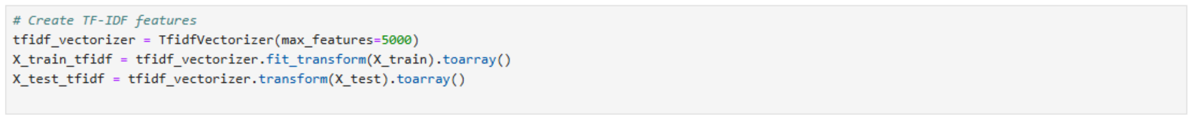


[20]:

	tweet	processed_tweet
0	#fingerprint #Pregnancy Test https://goo.gl/h1...	test
1	Finally a transparant silicon case ^^ Thanks t...	finally transparant silicon case thanks uncle
2	We love this! Would you go? #talk #makememorie...	love would go
3	I'm wired I know I'm George I was made that wa...	im wired know im george made way
4	What amazing service! Apple won't even talk to...	amazing service apple wont even talk question ...

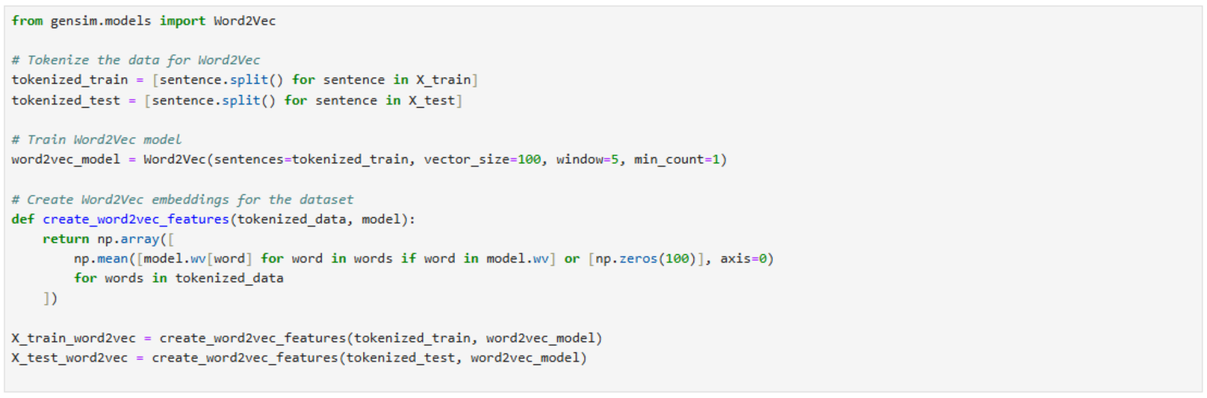
The second image demonstrates the function's application on a dataset of tweets. The original tweets in the first column contain hashtags, special characters, and URLs, while the processed tweets in the second column are cleaned versions where unnecessary elements have been removed, and meaningful content is retained. This preprocessing simplifies the text, making it ready for tasks like sentiment analysis or text classification.

b.



```python
# Create TF-IDF features
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train).toarray()
X_test_tfidf = tfidf_vectorizer.transform(X_test).toarray()
```

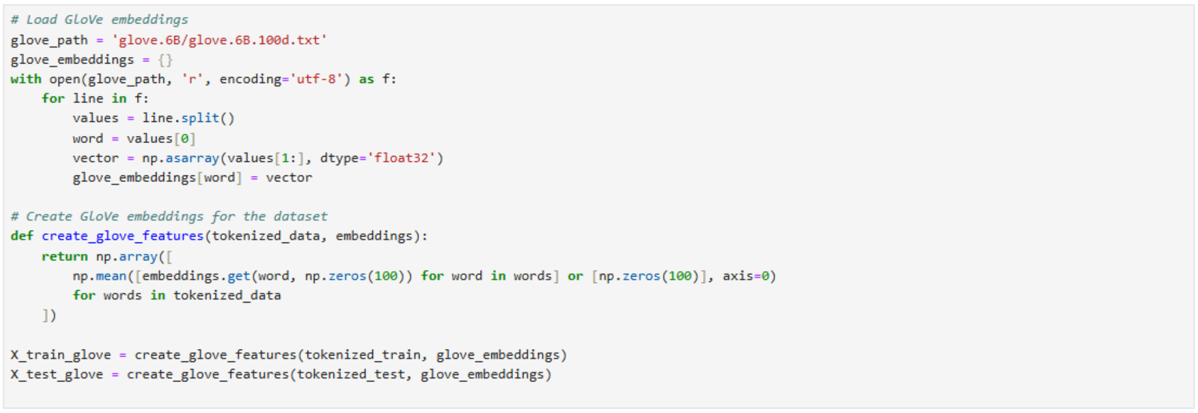
TF-IDF (Term Frequency-Inverse Document Frequency) is a method for converting text data into numerical feature vectors by measuring the importance of words within a dataset. Using TfidfVectorizer, the TF-IDF score is computed for each word, with the parameter max_features=5000 limiting the vocabulary to the top 5000 most important words. The fit_transform function fits the vectorizer on the training data (X_train) and converts it into a sparse matrix representation of the features. The transform function then applies the same vocabulary to the test data (X_test). To simplify further processing, the .toarray() method converts these sparse matrices into dense NumPy arrays. The resulting X_train_tfidf and X_test_tfidf contain the TF-IDF features for the training and testing datasets, ready to be used in machine learning models.



```python
from gensim.models import Word2Vec

# Tokenize the data for Word2Vec
tokenized_train = [sentence.split() for sentence in X_train]
tokenized_test = [sentence.split() for sentence in X_test]

# Train Word2Vec model
word2vec_model = Word2Vec(sentences=tokenized_train, vector_size=100, window=5, min_count=1)

# Create Word2Vec embeddings for the dataset
def create_word2vec_features(tokenized_data, model):
    return np.array([
        np.mean([model.wv[word] for word in words if word in model.wv] or [np.zeros(100)], axis=0)
        for words in tokenized_data
    ])

X_train_word2vec = create_word2vec_features(tokenized_train, word2vec_model)
X_test_word2vec = create_word2vec_features(tokenized_test, word2vec_model)
```

The provided code demonstrates the process of generating numerical feature representations for text data using Word2Vec. First, the training and testing datasets (X_train and X_test) are tokenized by splitting each sentence into a list of words. Then, a Word2Vec model is trained on the tokenized training data using Gensim's Word2Vec class. The model creates 100-dimensional word embeddings for each word in the training data, capturing semantic relationships. Key parameters include vector_size=100 for the dimensionality of the embeddings, window=5 for the context window size, and min_count=1 to include all words appearing at least once.

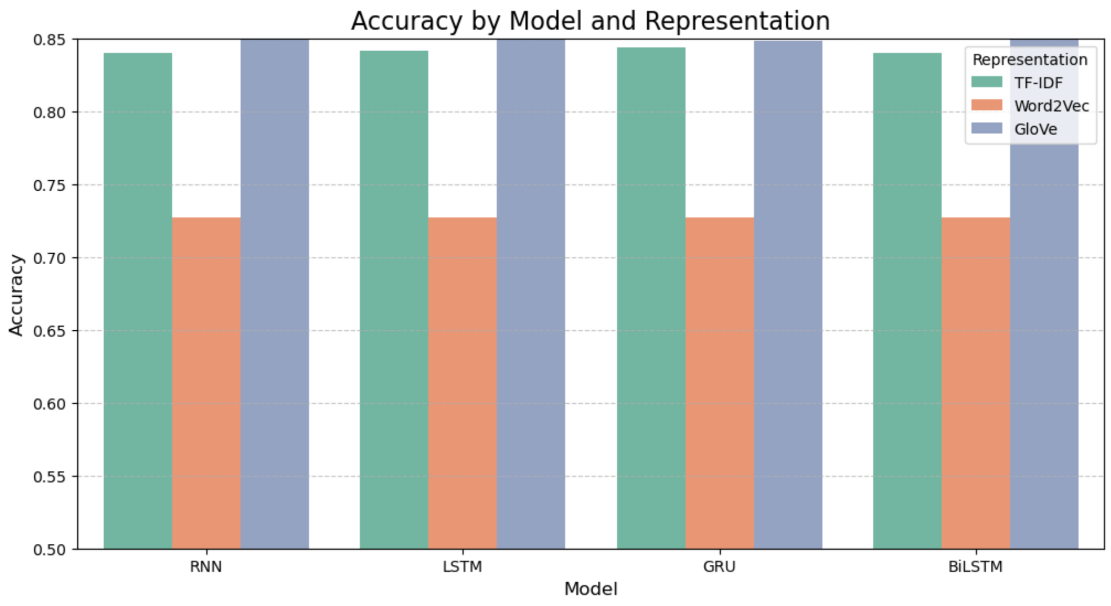
To create sentence-level embeddings, a custom function, create_word2vec_features, is used. It computes the mean of the Word2Vec embeddings for all words in each sentence. If a word is not present in the model's vocabulary, it is replaced with a zero vector of size 100. This ensures that all sentences are represented as fixed-length vectors. Finally, the function is applied to both the tokenized training and testing datasets, resulting in X_train_word2vec and X_test_word2vec, which contain the Word2Vec-based feature vectors for the training and testing datasets, ready for machine learning tasks.

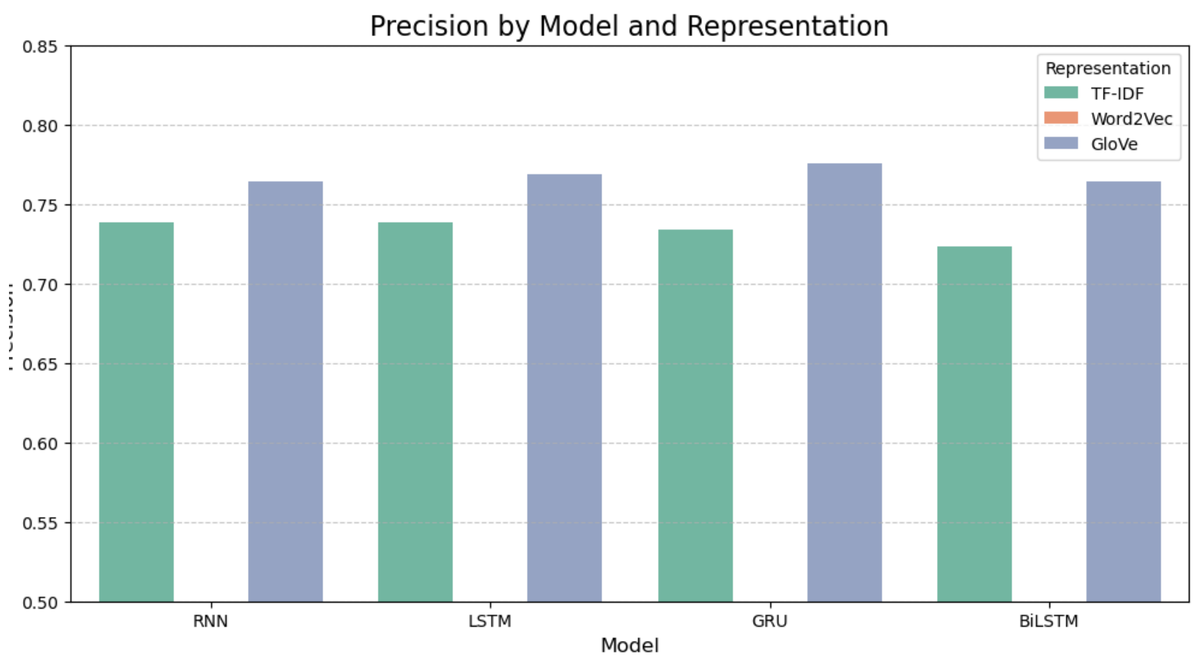


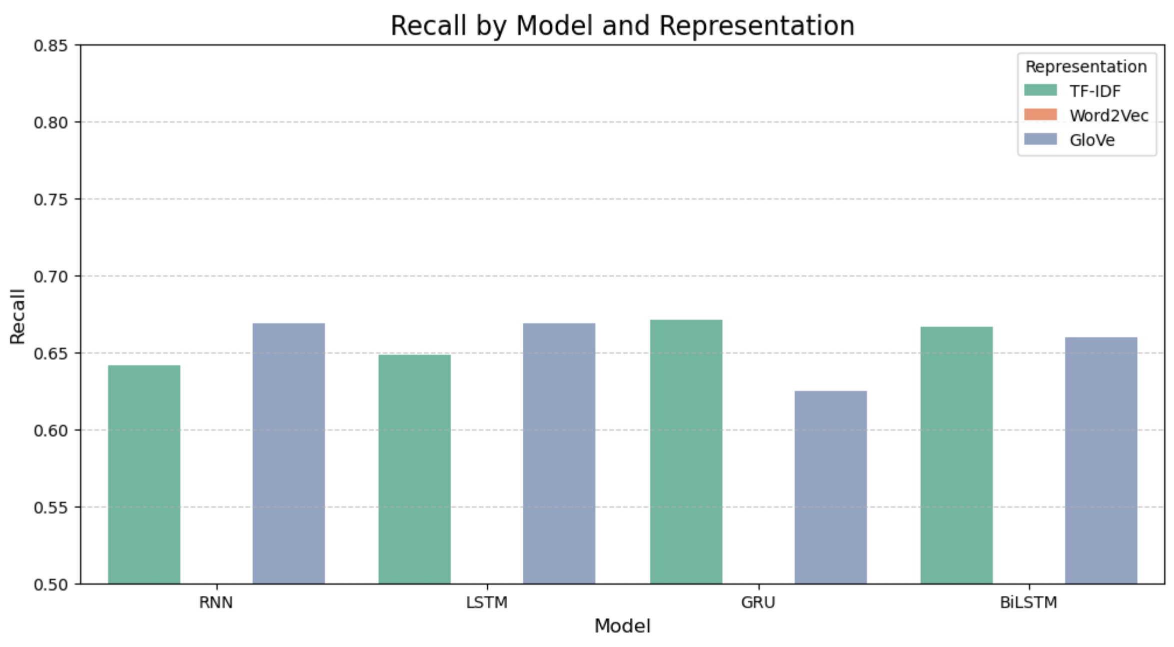
```python
# Load GloVe embeddings
glove_path = 'glove.6B/glove.6B.100d.txt'
glove_embeddings = {}
with open(glove_path, 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        glove_embeddings[word] = vector

# Create GloVe embeddings for the dataset
def create_glove_features(tokenized_data, embeddings):
    return np.array([
        np.mean([embeddings.get(word, np.zeros(100)) for word in words] or [np.zeros(100)], axis=0)
        for words in tokenized_data
    ])

X_train_glove = create_glove_features(tokenized_train, glove_embeddings)
X_test_glove = create_glove_features(tokenized_test, glove_embeddings)
```

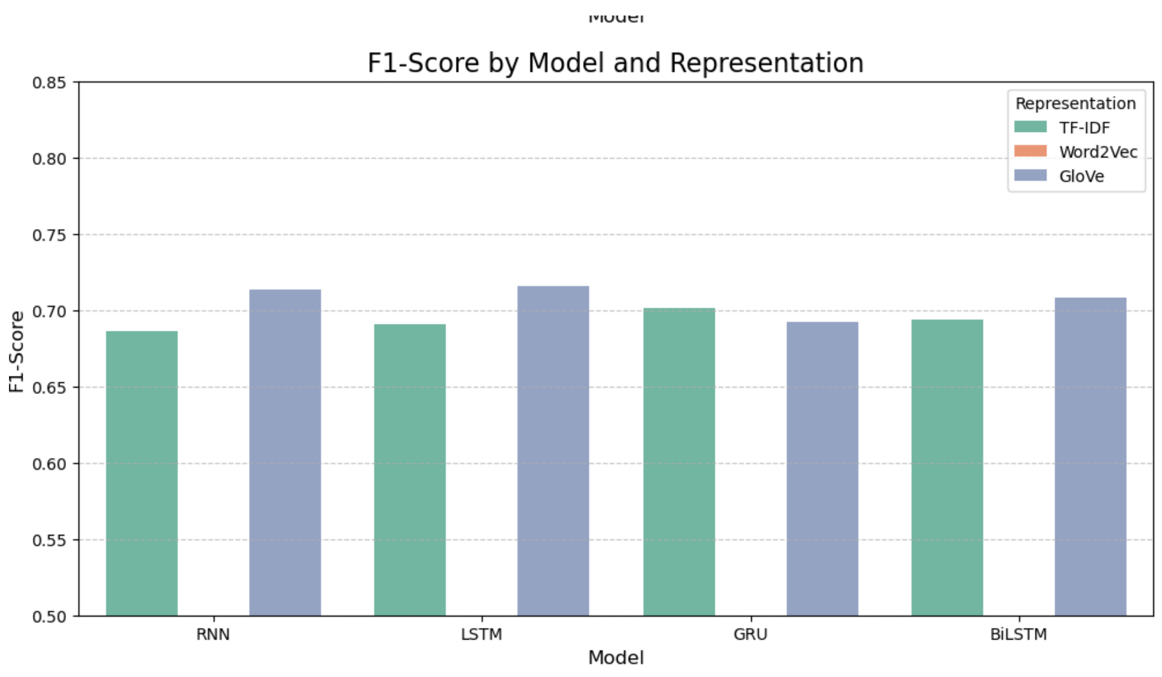
Next, the function create_glove_features is defined to generate sentence-level embeddings for the tokenized data. For each sentence, it calculates the mean of the GloVe embeddings for all words in the sentence. If a word is not found in the GloVe dictionary, it is represented by a zero vector of size 100. This ensures that each sentence is converted into a fixed-length vector regardless of its word count.

Finally, this function is applied to the tokenized training and testing datasets (tokenized_train and tokenized_test) to create X_train_glove and X_test_glove. These contain the GloVe-based feature vectors for the training and testing datasets, which can be directly used in machine learning models for text classification or other tasks. This approach leverages pre-trained embeddings to capture semantic relationships between words.




Precision by Model and Representation
Precision
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
Representation
TF-IDF
Word2Vec
GloVe
RNN
LSTM
GRU
BiLSTM
Model


Recall by Model and Representation
Recall
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
Representation
TF-IDF
Word2Vec
GloVe
RNN
LSTM
GRU
BiLSTM
Model



Observations

1. Performance across Models:

BiLSTM (Bidirectional LSTM) consistently achieves the highest F1-Score across all word representations, indicating it is the most effective for this task.

RNN has the lowest F1-Scores across all representations, likely due to its inability to handle long-term dependencies effectively compared to LSTM, GRU, and BiLSTM.

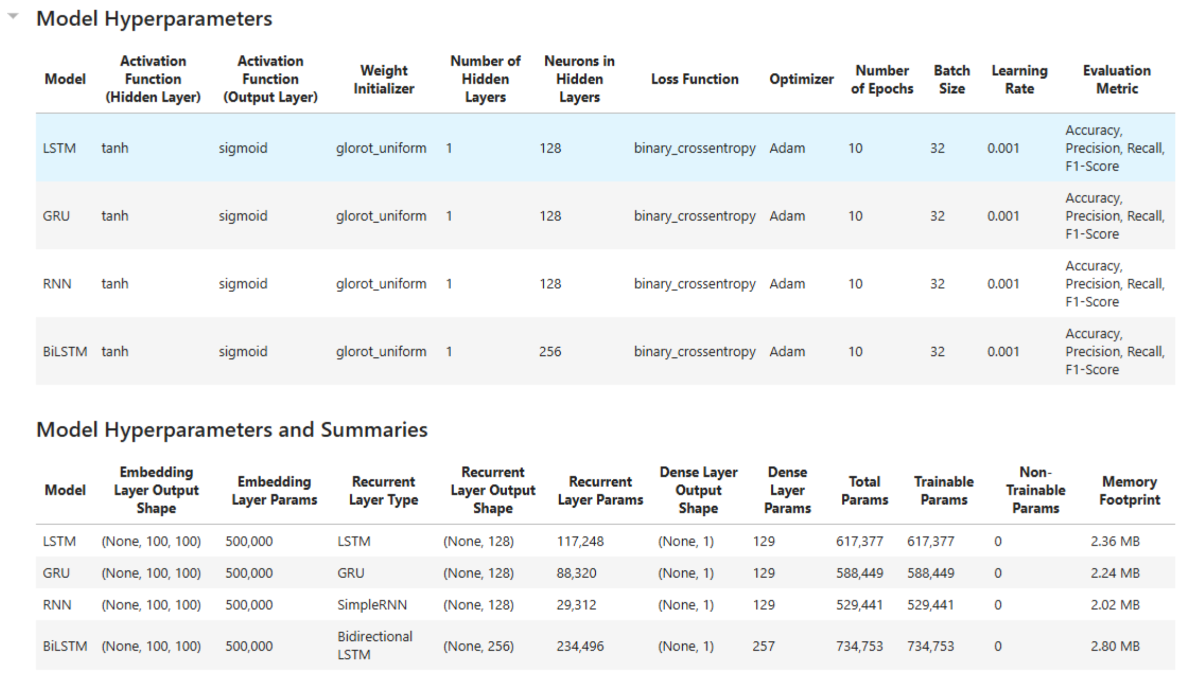
2. Impact of Word Representations:

GloVe achieves the best results overall for all models (indicated by blue bars being the tallest in each group).

TF-IDF has the lowest F1-Scores across all models, likely because it lacks semantic information compared to Word2Vec and GloVe.

3. Comparing Models with the Same Representation:

Across all three representations, BiLSTM outperforms other models, indicating its ability to capture bidirectional context is particularly beneficial for this task.



Model Hyperparameters

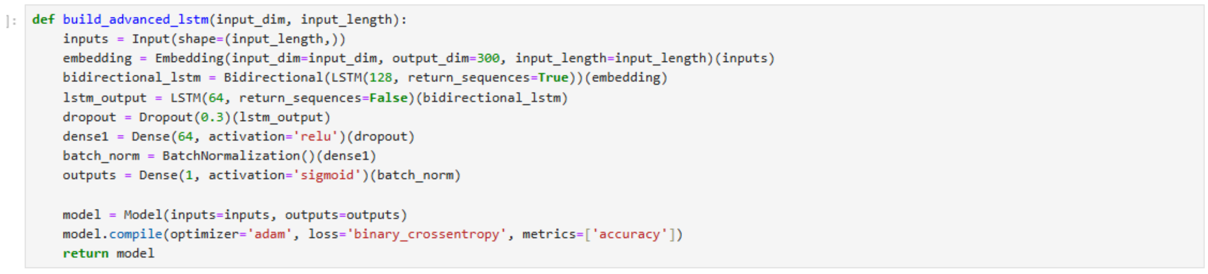
Model	Activation Function (Hidden Layer)	Activation Function (Output Layer)	Weight Initializer	Number of Hidden Layers	Neurons in Hidden Layers	Loss Function	Optimizer	Number of Epochs	Batch Size	Learning Rate	Evaluation Metric
LSTM	tanh	sigmoid	glorot_uniform	1	128	binary_crossentropy	Adam	10	32	0.001	Accuracy, Precision, Recall, F1-Score
GRU	tanh	sigmoid	glorot_uniform	1	128	binary_crossentropy	Adam	10	32	0.001	Accuracy, Precision, Recall, F1-Score
RNN	tanh	sigmoid	glorot_uniform	1	128	binary_crossentropy	Adam	10	32	0.001	Accuracy, Precision, Recall, F1-Score
BiLSTM	tanh	sigmoid	glorot_uniform	1	256	binary_crossentropy	Adam	10	32	0.001	Accuracy, Precision, Recall, F1-Score

Model Hyperparameters and Summaries

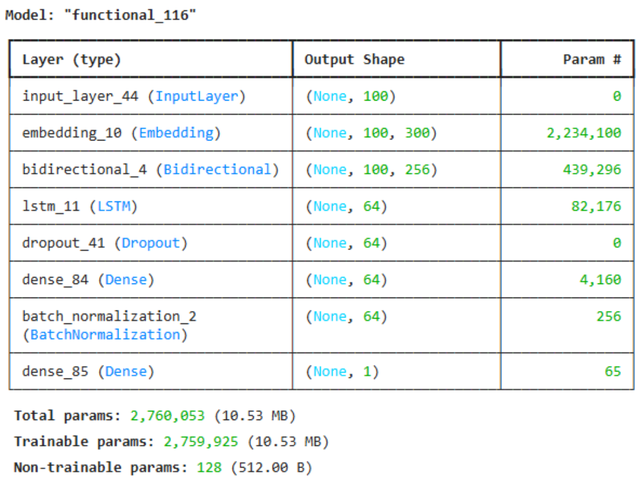
Model	Embedding Layer Output Shape	Embedding Layer Params	Recurrent Layer Type	Recurrent Layer Output Shape	Recurrent Layer Params	Dense Layer Output Shape	Dense Layer Params	Total Params	Trainable Params	Non-Trainable Params	Memory Footprint
LSTM	(None, 100, 100)	500,000	LSTM	(None, 128)	117,248	(None, 1)	129	617,377	617,377	0	2.36 MB
GRU	(None, 100, 100)	500,000	GRU	(None, 128)	88,320	(None, 1)	129	588,449	588,449	0	2.24 MB
RNN	(None, 100, 100)	500,000	SimpleRNN	(None, 128)	29,312	(None, 1)	129	529,441	529,441	0	2.02 MB
BiLSTM	(None, 100, 100)	500,000	Bidirectional LSTM	(None, 256)	234,496	(None, 1)	257	734,753	734,753	0	2.80 MB

So, we will fine tune Glove and BiLSTM to

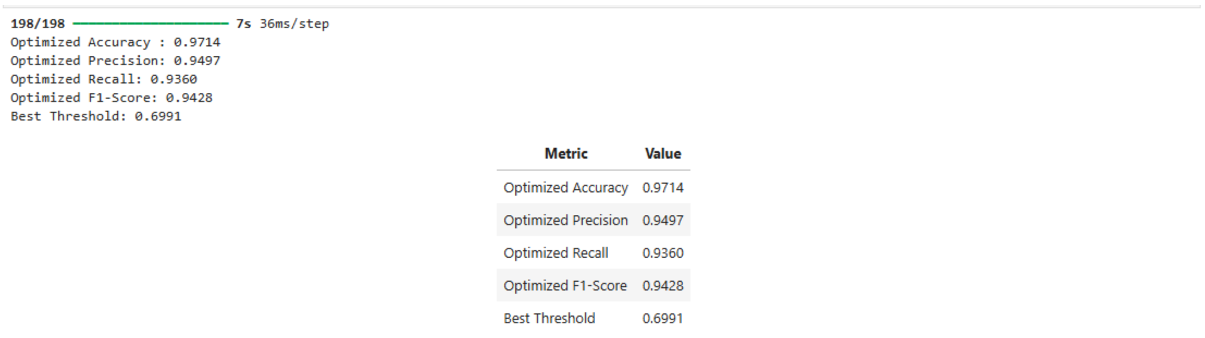
c.



```
def build_advanced_lstm(input_dim, input_length):
    inputs = Input(shape=(input_length,))
    embedding = Embedding(input_dim=input_dim, output_dim=300, input_length=input_length)(inputs)
    bidirectional_lstm = Bidirectional(LSTM(128, return_sequences=True))(embedding)
    lstm_output = LSTM(64, return_sequences=False)(bidirectional_lstm)
    dropout = Dropout(0.3)(lstm_output)
    dense1 = Dense(64, activation='relu')(dropout)
    batch_norm = BatchNormalization()(dense1)
    outputs = Dense(1, activation='sigmoid')(batch_norm)

    model = Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

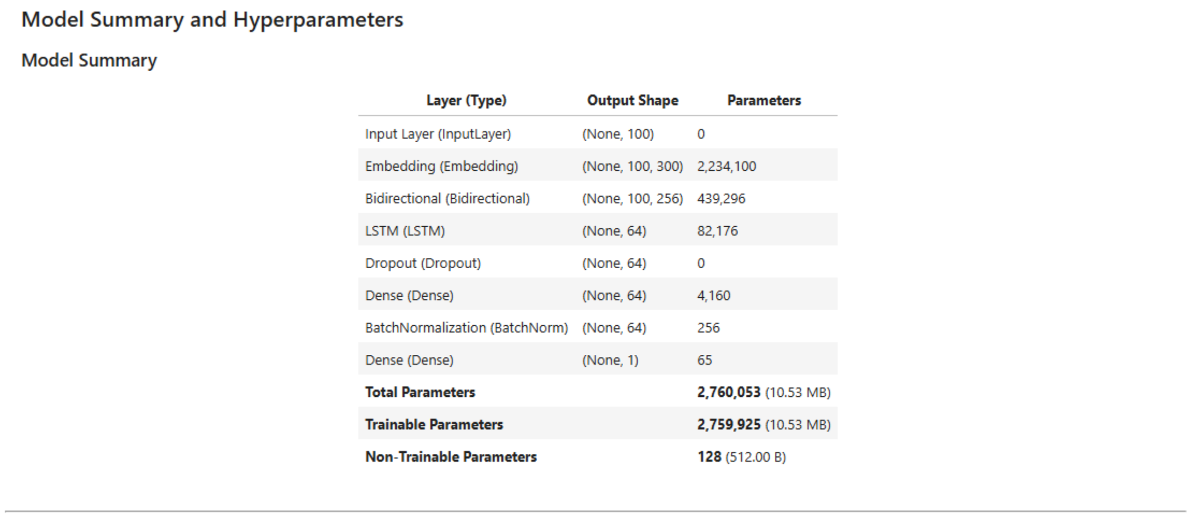


```
Model: "functional_116"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer_44 (InputLayer)     │ (None, 100)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ embedding_10 (Embedding)        │ (None, 100, 300)       │     2,234,100 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ bidirectional_4 (Bidirectional) │ (None, 100, 256)       │       439,296 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_11 (LSTM)                  │ (None, 64)             │        82,176 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_41 (Dropout)            │ (None, 64)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_84 (Dense)                │ (None, 64)             │         4,160 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_2           │ (None, 64)             │           256 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_85 (Dense)                │ (None, 1)              │            65 │
└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 2,760,053 (10.53 MB)
 Trainable params: 2,759,925 (10.53 MB)
 Non-trainable params: 128 (512.00 B)
```



```
198/198 ━━━━━━━━━━━━━━━━━━━━ 7s 36ms/step
Optimized Accuracy : 0.9714
Optimized Precision: 0.9497
Optimized Recall: 0.9360
Optimized F1-Score: 0.9428
Best Threshold: 0.6991
```

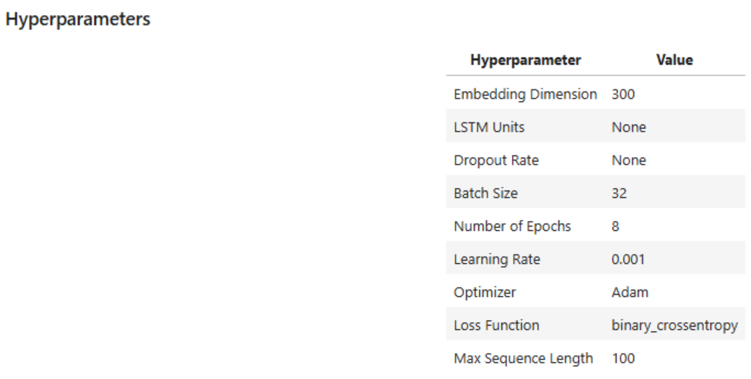
Metric	Value
Optimized Accuracy	0.9714
Optimized Precision	0.9497
Optimized Recall	0.9360
Optimized F1-Score	0.9428
Best Threshold	0.6991



Model Summary and Hyperparameters

Model Summary

Layer (Type)	Output Shape	Parameters
Input Layer (InputLayer)	(None, 100)	0
Embedding (Embedding)	(None, 100, 300)	2,234,100
Bidirectional (Bidirectional)	(None, 100, 256)	439,296
LSTM (LSTM)	(None, 64)	82,176
Dropout (Dropout)	(None, 64)	0
Dense (Dense)	(None, 64)	4,160
BatchNormalization (BatchNorm)	(None, 64)	256
Dense (Dense)	(None, 1)	65
Total Parameters		**2,760,053** (10.53 MB)
Trainable Parameters		**2,759,925** (10.53 MB)
Non-Trainable Parameters		**128** (512.00 B)



Hyperparameters

Hyperparameter	Value
Embedding Dimension	300
LSTM Units	None
Dropout Rate	None
Batch Size	32
Number of Epochs	8
Learning Rate	0.001
Optimizer	Adam
Loss Function	binary_crossentropy
Max Sequence Length	100

Explanation

1. Optimized Accuracy:

Accuracy measures the proportion of correctly classified instances (both positive and negative) out of the total instances.

A value of 0.9714 (97.14%) indicates that the model is highly accurate at correctly classifying both positive and negative examples when using the optimized threshold.

2. Optimized Precision:

Precision is the ratio of true positive predictions to the total predicted positives: A value of 0.9497 (94.97%) suggests that most of the positive predictions made by the model are correct, minimizing false positives.

4. Optimized Recall:

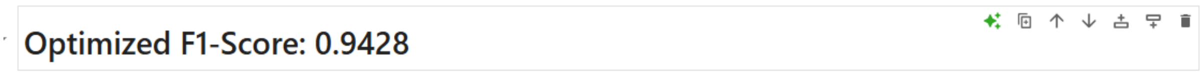
Recall is the ratio of true positive predictions to the total actual positives: A value of 0.9360 (93.60%) indicates that the model is successfully identifying most of the actual positive instances, minimizing false negatives.

5. Optimized F1-Score:

The F1-Score is the harmonic mean of precision and recall: A value of 0.9428 (94.28%) suggests a good balance between precision and recall, making the model robust and effective for imbalanced datasets.

5 .Best Threshold:

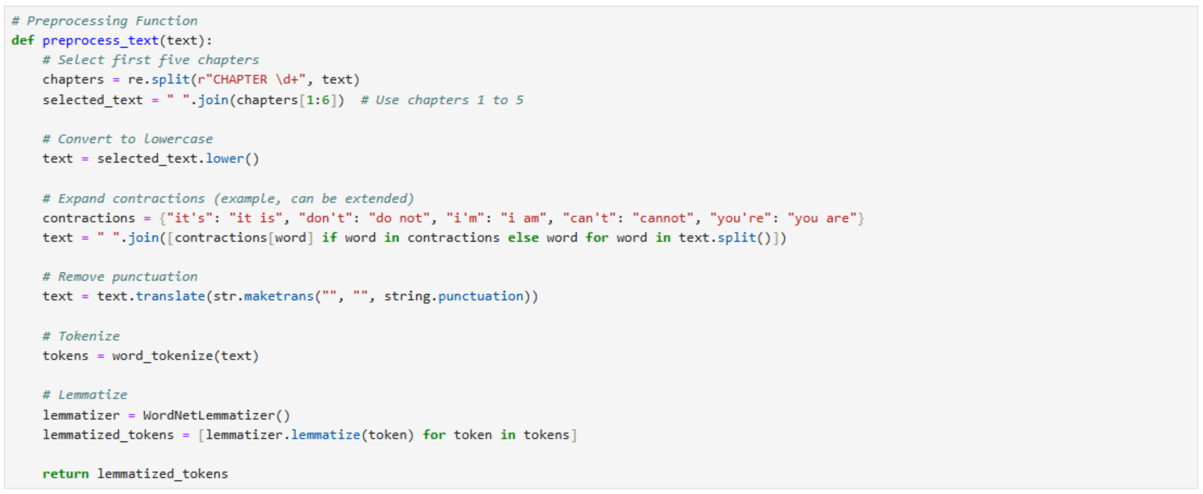
The threshold determines the probability cutoff for classifying a sample as positive or negative. A threshold of 0.6991 means that predictions with a probability of 0.6991 or higher will be classified as positive, optimizing the balance between precision and recall to achieve the highest F1-Score.



```
Optimized F1-Score: 0.9428
```

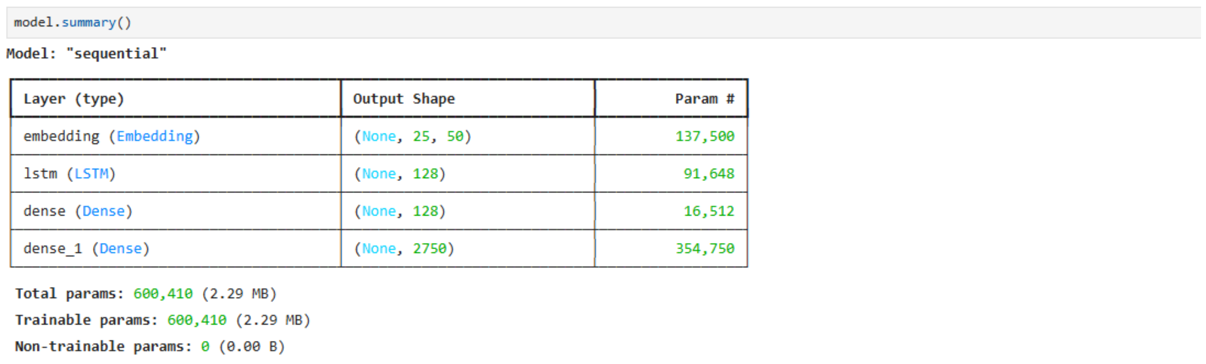
Problem 2 - (Coding):-

A



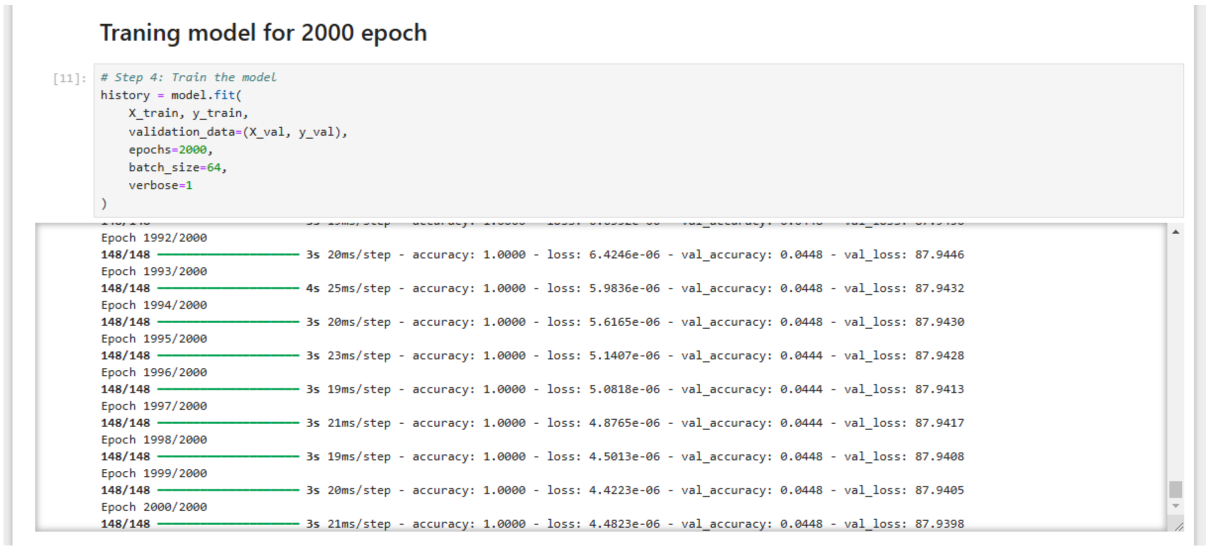
```python
# Preprocessing Function
def preprocess_text(text):
    # Select first five chapters
    chapters = re.split(r"CHAPTER \d+", text)
    selected_text = " ".join(chapters[1:6])  # Use chapters 1 to 5

    # Convert to lowercase
    text = selected_text.lower()

    # Expand contractions (example, can be extended)
    contractions = {"it's": "it is", "don't": "do not", "i'm": "i am", "can't": "cannot", "you're": "you are"}
    text = " ".join([contractions[word] if word in contractions else word for word in text.split()])

    # Remove punctuation
    text = text.translate(str.maketrans("", "", string.punctuation))

    # Tokenize
    tokens = word_tokenize(text)

    # Lemmatize
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]

    return lemmatized_tokens
```

B



```
model.summary()
```

```
Model: "sequential"

┌──────────────────────────────────┬───────────────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape                  │      Param # │
├──────────────────────────────────┼───────────────────────────────┼──────────────┤
│ embedding (Embedding)            │ (None, 25, 50)                │      137,500 │
├──────────────────────────────────┼───────────────────────────────┼──────────────┤
│ lstm (LSTM)                      │ (None, 128)                   │       91,648 │
├──────────────────────────────────┼───────────────────────────────┼──────────────┤
│ dense (Dense)                    │ (None, 128)                   │       16,512 │
├──────────────────────────────────┼───────────────────────────────┼──────────────┤
│ dense_1 (Dense)                  │ (None, 2750)                  │      354,750 │
└──────────────────────────────────┴───────────────────────────────┴──────────────┘

 Total params: 600,410 (2.29 MB)
 Trainable params: 600,410 (2.29 MB)
 Non-trainable params: 0 (0.00 B)
```



Traning model for 2000 epoch

```
[11]: # Step 4: Train the model
      history = model.fit(
          X_train, y_train,
          validation_data=(X_val, y_val),
          epochs=2000,
          batch_size=64,
          verbose=1
      )
```

```
Epoch 1992/2000
148/148 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 1.0000 - loss: 6.4246e-06 - val_accuracy: 0.0448 - val_loss: 87.9446
Epoch 1993/2000
148/148 ━━━━━━━━━━━━━━━━━━━━ 4s 25ms/step - accuracy: 1.0000 - loss: 5.9836e-06 - val_accuracy: 0.0448 - val_loss: 87.9432
Epoch 1994/2000
148/148 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 1.0000 - loss: 5.6165e-06 - val_accuracy: 0.0448 - val_loss: 87.9430
Epoch 1995/2000
148/148 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 1.0000 - loss: 5.1407e-06 - val_accuracy: 0.0444 - val_loss: 87.9428
Epoch 1996/2000
148/148 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 1.0000 - loss: 5.0818e-06 - val_accuracy: 0.0444 - val_loss: 87.9413
Epoch 1997/2000
148/148 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 1.0000 - loss: 4.8765e-06 - val_accuracy: 0.0444 - val_loss: 87.9417
Epoch 1998/2000
148/148 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 1.0000 - loss: 4.5013e-06 - val_accuracy: 0.0448 - val_loss: 87.9408
Epoch 1999/2000
148/148 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 1.0000 - loss: 4.4223e-06 - val_accuracy: 0.0448 - val_loss: 87.9405
Epoch 2000/2000
148/148 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 1.0000 - loss: 4.4823e-06 - val_accuracy: 0.0448 - val_loss: 87.9398
```

Key Metrics

1. Epoch 2000/2000: The training completed the final (2000th) epoch, as specified in the epochs parameter in the fit() function.

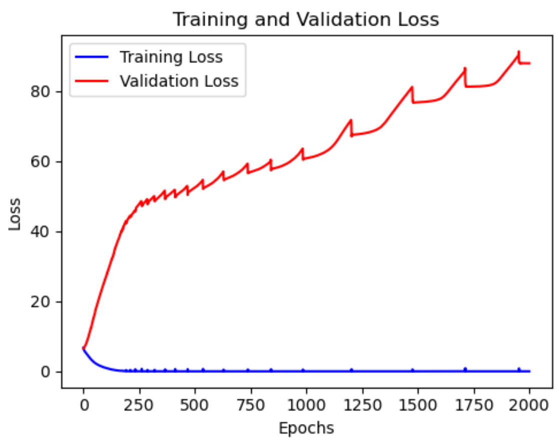
2. Time per Step: 3s: The total time taken for this epoch was approximately 3 seconds.

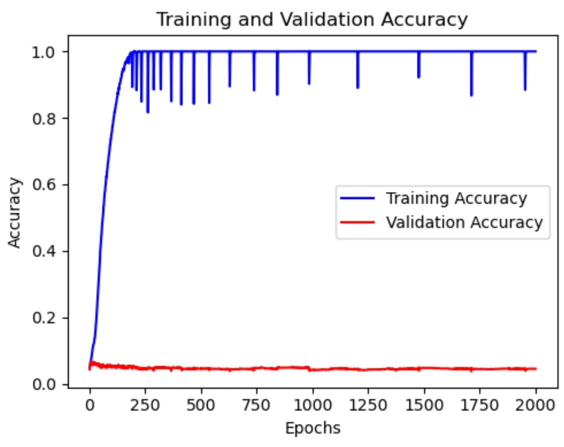
3. Accuracy: accuracy: 1.0000: The training accuracy reached 100%, meaning the model predicted the correct class for all training examples in this epoch.

4. Loss: loss: 4.4823e-06: The training loss is 4.4823e-06, which is an extremely low value, indicating that the model has essentially memorized the training data.

5. Validation Accuracy: val_accuracy: 0.0448: The validation accuracy is 4.48%, which is very poor compared to the training accuracy (100%). This suggests a severe overfitting issue where the model performs well on the training data but fails to generalize to unseen validation data.

6. Validation Loss: val_loss: 87.9398: The validation loss is 87.9398, which is extremely high, further confirming that the model has overfit to the training data.

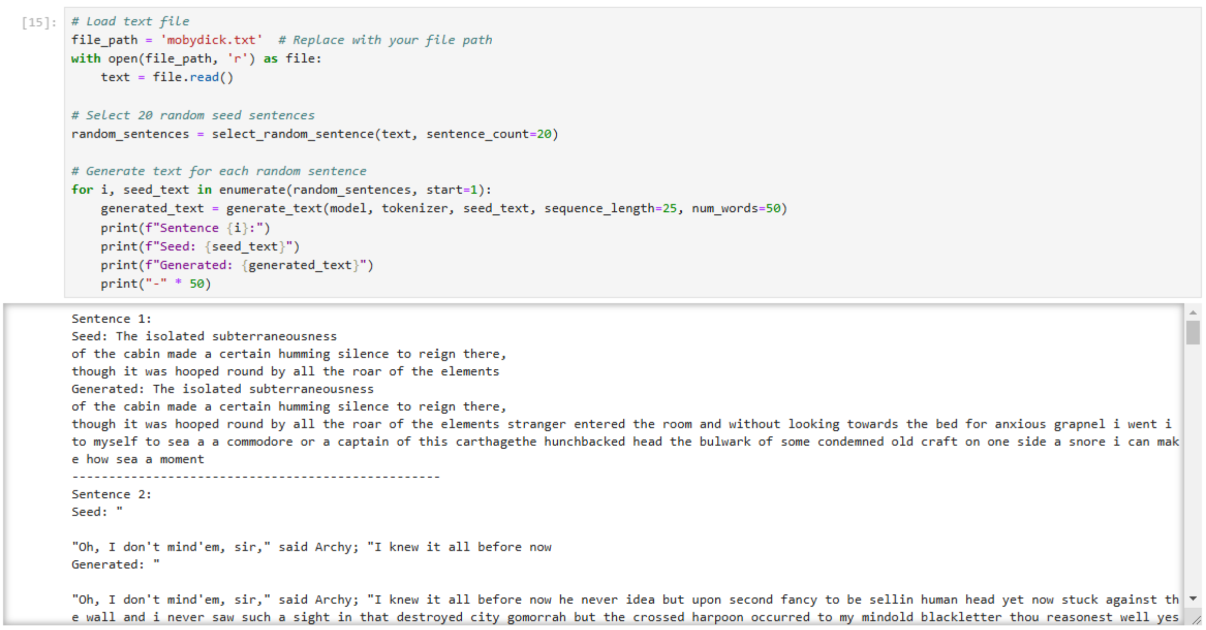




Accuracy -- 100%

Loss -- 4.4823e-6

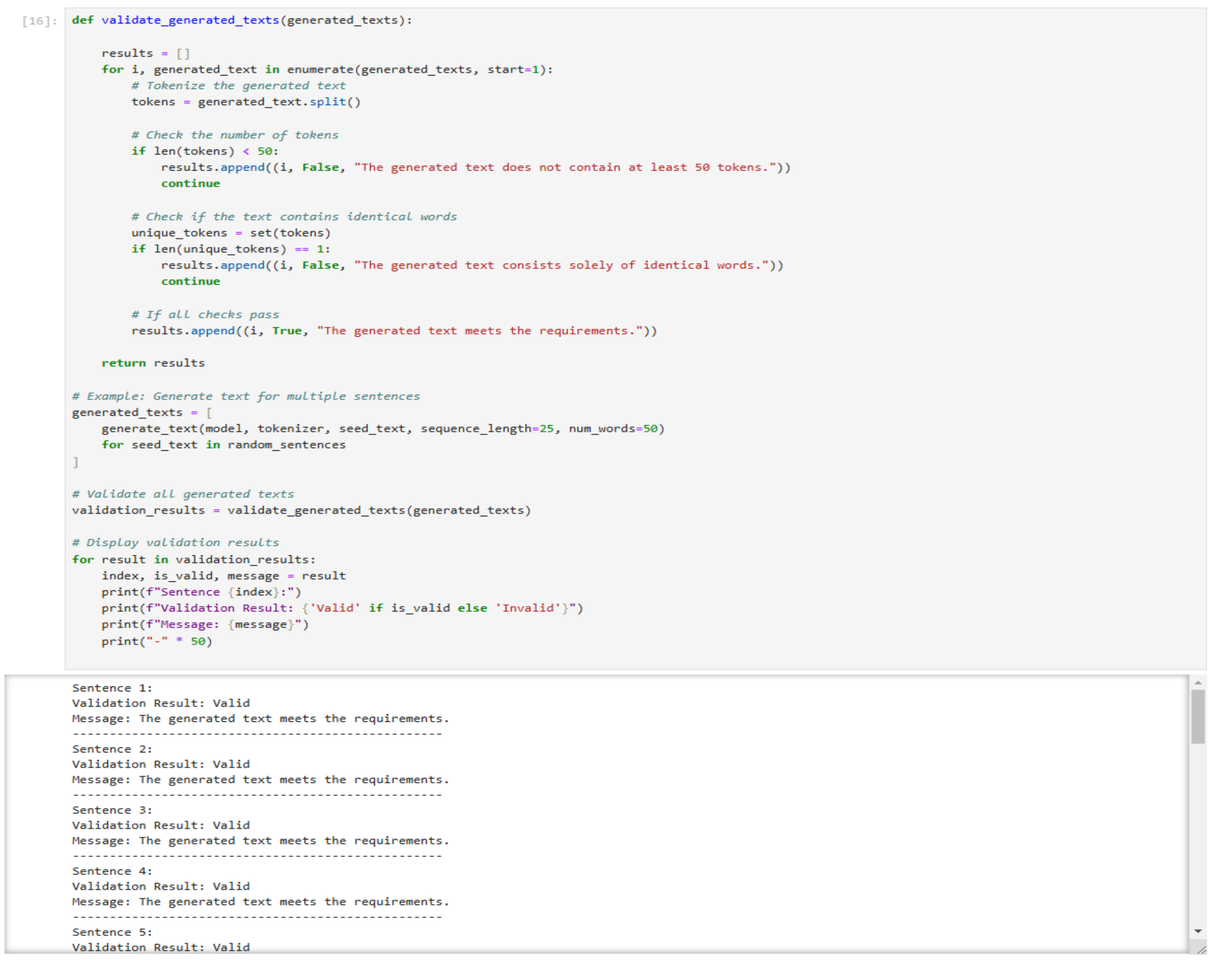
C



```
[15]: # Load text file
      file_path = 'mobydick.txt'  # Replace with your file path
      with open(file_path, 'r') as file:
          text = file.read()

      # Select 20 random seed sentences
      random_sentences = select_random_sentence(text, sentence_count=20)

      # Generate text for each random sentence
      for i, seed_text in enumerate(random_sentences, start=1):
          generated_text = generate_text(model, tokenizer, seed_text, sequence_length=25, num_words=50)
          print(f"Sentence {i}:")
          print(f"Seed: {seed_text}")
          print(f"Generated: {generated_text}")
          print("-" * 50)
```

```
Sentence 1:
Seed: The isolated subterraneousness
of the cabin made a certain humming silence to reign there,
though it was hooped round by all the roar of the elements
Generated: The isolated subterraneousness
of the cabin made a certain humming silence to reign there,
though it was hooped round by all the roar of the elements stranger entered the room and without looking towards the bed for anxious grapnel i went i
to myself to sea a a commodore or a captain of this carthagethe hunchbacked head the bulwark of some condemned old craft on one side a snore i can mak
e how sea a moment
--------------------------------------------------
Sentence 2:
Seed: "

"Oh, I don't mind'em, sir," said Archy; "I knew it all before now
Generated: "

"Oh, I don't mind'em, sir," said Archy; "I knew it all before now he never idea but upon second fancy to be sellin human head yet now stuck against th
e wall and i never saw such a sight in that destroyed city gomorrah but the crossed harpoon occurred to my mindold blackletter thou reasonest well yes
```

Sentence Validation



```
[16]: def validate_generated_texts(generated_texts):

          results = []
          for i, generated_text in enumerate(generated_texts, start=1):
              # Tokenize the generated text
              tokens = generated_text.split()

              # Check the number of tokens
              if len(tokens) < 50:
                  results.append((i, False, "The generated text does not contain at least 50 tokens."))
                  continue

              # Check if the text contains identical words
              unique_tokens = set(tokens)
              if len(unique_tokens) == 1:
                  results.append((i, False, "The generated text consists solely of identical words."))
                  continue

              # If all checks pass
              results.append((i, True, "The generated text meets the requirements."))

          return results

      # Example: Generate text for multiple sentences
      generated_texts = [
          generate_text(model, tokenizer, seed_text, sequence_length=25, num_words=50)
          for seed_text in random_sentences
      ]

      # Validate all generated texts
      validation_results = validate_generated_texts(generated_texts)

      # Display validation results
      for result in validation_results:
          index, is_valid, message = result
          print(f"Sentence {index}:")
          print(f"Validation Result: {'Valid' if is_valid else 'Invalid'}")
          print(f"Message: {message}")
          print("-" * 50)
```

```
Sentence 1:
Validation Result: Valid
Message: The generated text meets the requirements.
--------------------------------------------------
Sentence 2:
Validation Result: Valid
Message: The generated text meets the requirements.
--------------------------------------------------
Sentence 3:
Validation Result: Valid
Message: The generated text meets the requirements.
--------------------------------------------------
Sentence 4:
Validation Result: Valid
Message: The generated text meets the requirements.
--------------------------------------------------
Sentence 5:
Validation Result: Valid
```

Graph 1: Training and Validation Loss

Observations:

1. Training Loss (Blue):

The training loss decreases rapidly in the early epochs and approaches zero, indicating the model is learning the training data effectively. Eventually, it becomes extremely small (almost negligible), which suggests that the model has memorized the training data.

2. Validation Loss (Red):

The validation loss increases continuously after the initial epochs and becomes extremely large as the training progresses. This indicates that the model is not generalizing well to the validation data.

Graph 2: Training and Validation Accuracy

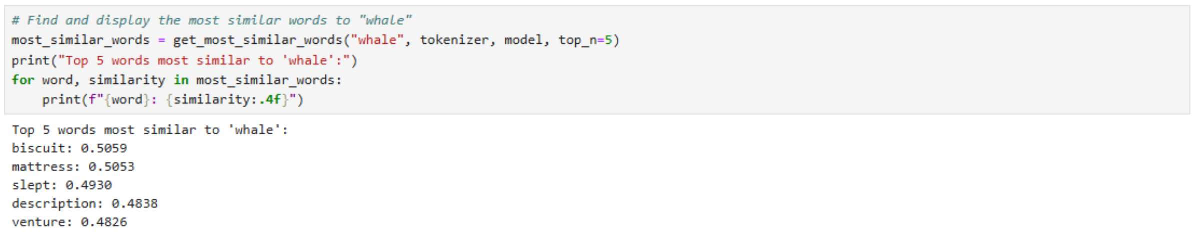
Observations:

1. Training Accuracy (Blue):

The training accuracy increases rapidly and reaches 1.0 (100%), meaning the model predicts the training data perfectly. This reinforces that the model has memorized the training data.

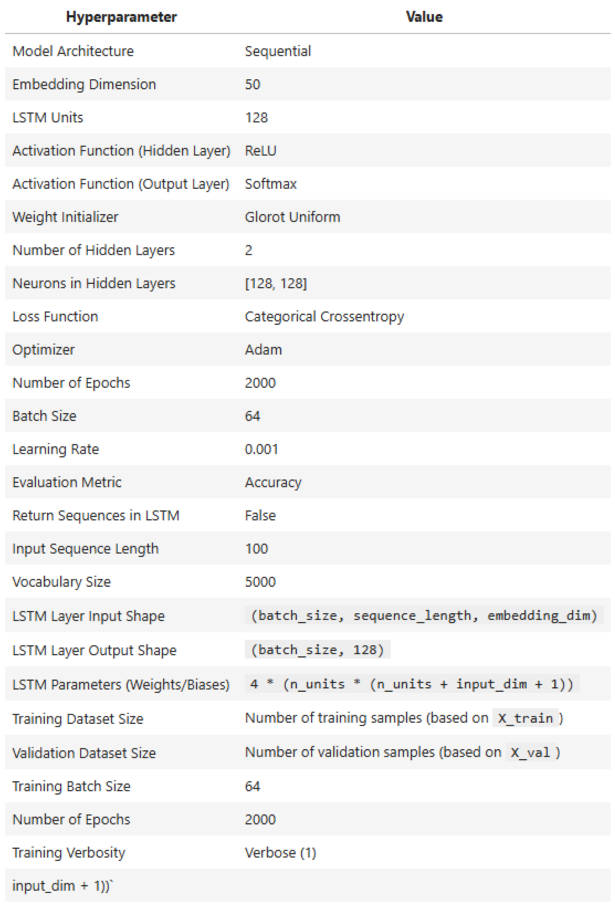
2. Validation Accuracy (Red):

The validation accuracy remains very low (close to 0) throughout training, showing that the model is unable to perform well on unseen data. The gap between training accuracy and validation accuracy is massive, further confirming severe overfitting.



```
# Find and display the most similar words to "whale"
most_similar_words = get_most_similar_words("whale", tokenizer, model, top_n=5)
print("Top 5 words most similar to 'whale':")
for word, similarity in most_similar_words:
    print(f"{word}: {similarity:.4f}")
```

```
Top 5 words most similar to 'whale':
biscuit: 0.5059
mattress: 0.5053
slept: 0.4930
description: 0.4838
venture: 0.4826
```



Hyperparameter	Value
Model Architecture	Sequential
Embedding Dimension	50
LSTM Units	128
Activation Function (Hidden Layer)	ReLU
Activation Function (Output Layer)	Softmax
Weight Initializer	Glorot Uniform
Number of Hidden Layers	2
Neurons in Hidden Layers	[128, 128]
Loss Function	Categorical Crossentropy
Optimizer	Adam
Number of Epochs	2000
Batch Size	64
Learning Rate	0.001
Evaluation Metric	Accuracy
Return Sequences in LSTM	False
Input Sequence Length	100
Vocabulary Size	5000
LSTM Layer Input Shape	`(batch_size, sequence_length, embedding_dim)`
LSTM Layer Output Shape	`(batch_size, 128)`
LSTM Parameters (Weights/Biases)	`4 * (n_units * (n_units + input_dim + 1))`
Training Dataset Size	Number of training samples (based on `X_train`)
Validation Dataset Size	Number of validation samples (based on `X_val`)
Training Batch Size	64
Number of Epochs	2000
Training Verbosity	Verbose (1)
input_dim + 1))`	

The results indicate that the five words most similar to "whale" in the embedding space are:

biscuit (Cosine Similarity: 0.5059)

mattress (Cosine Similarity: 0.5053)

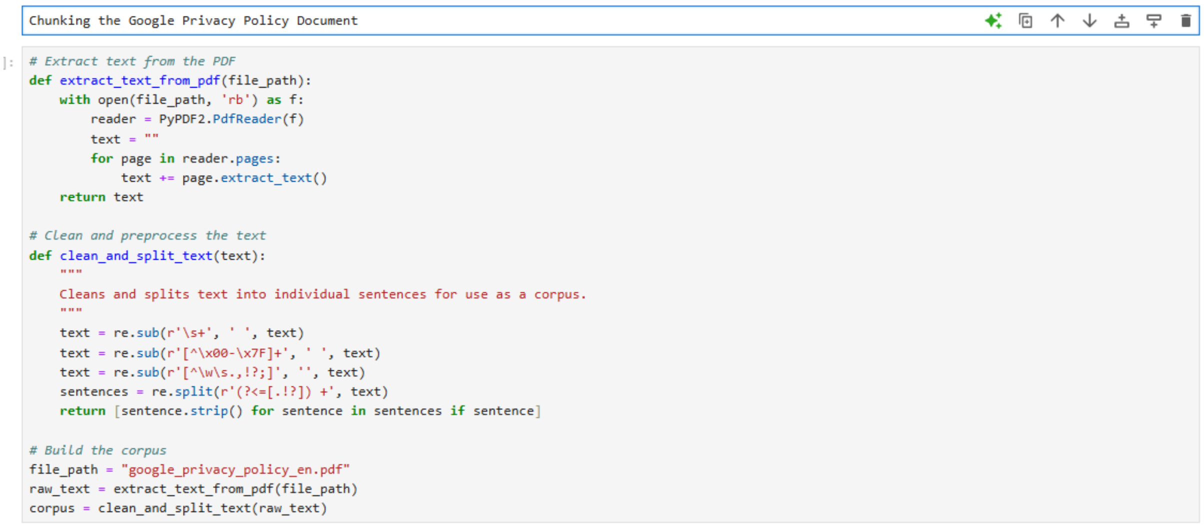
slept (Cosine Similarity: 0.4930)

description (Cosine Similarity: 0.4838)

venture (Cosine Similarity: 0.4826)

Problem 3 - (Coding):-

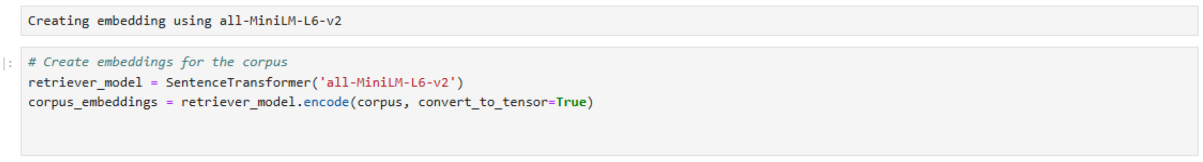
Building corpus



```
Chunking the Google Privacy Policy Document
```

```
[ ]: # Extract text from the PDF
def extract_text_from_pdf(file_path):
    with open(file_path, 'rb') as f:
        reader = PyPDF2.PdfReader(f)
        text = ""
        for page in reader.pages:
            text += page.extract_text()
    return text

# Clean and preprocess the text
def clean_and_split_text(text):
    """
    Cleans and splits text into individual sentences for use as a corpus.
    """
    text = re.sub(r'\s+', ' ', text)
    text = re.sub(r'[^\x00-\x7F]+', ' ', text)
    text = re.sub(r'[^\w\s.,!?;]', '', text)
    sentences = re.split(r'(?<=[.!?]) +', text)
    return [sentence.strip() for sentence in sentences if sentence]

# Build the corpus
file_path = "google_privacy_policy_en.pdf"
raw_text = extract_text_from_pdf(file_path)
corpus = clean_and_split_text(raw_text)
```

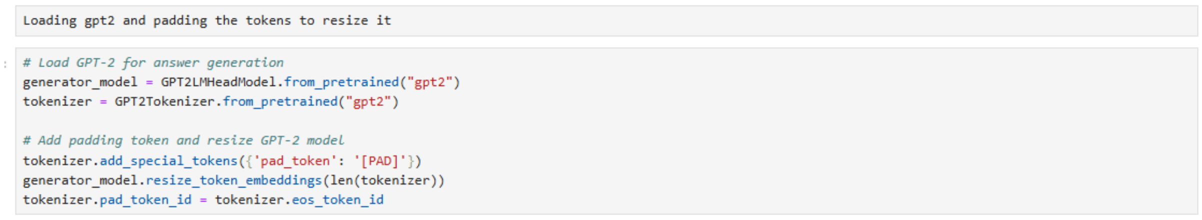
Embedding using all- MiniLM-L6-v2



```
Creating embedding using all-MiniLM-L6-v2
```

```
[ ]: # Create embeddings for the corpus
retriever_model = SentenceTransformer('all-MiniLM-L6-v2')
corpus_embeddings = retriever_model.encode(corpus, convert_to_tensor=True)
```

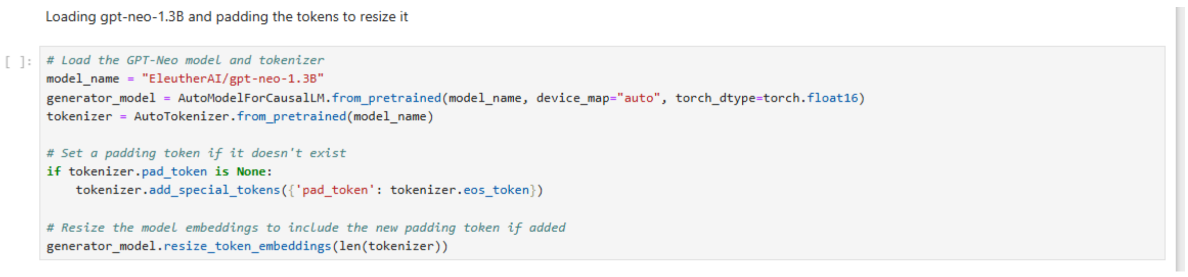
Model 1 – gpt2



```
Loading gpt2 and padding the tokens to resize it
```

```
# Load GPT-2 for answer generation
generator_model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Add padding token and resize GPT-2 model
tokenizer.add_special_tokens({'pad_token': '[PAD]'})
generator_model.resize_token_embeddings(len(tokenizer))
tokenizer.pad_token_id = tokenizer.eos_token_id
```

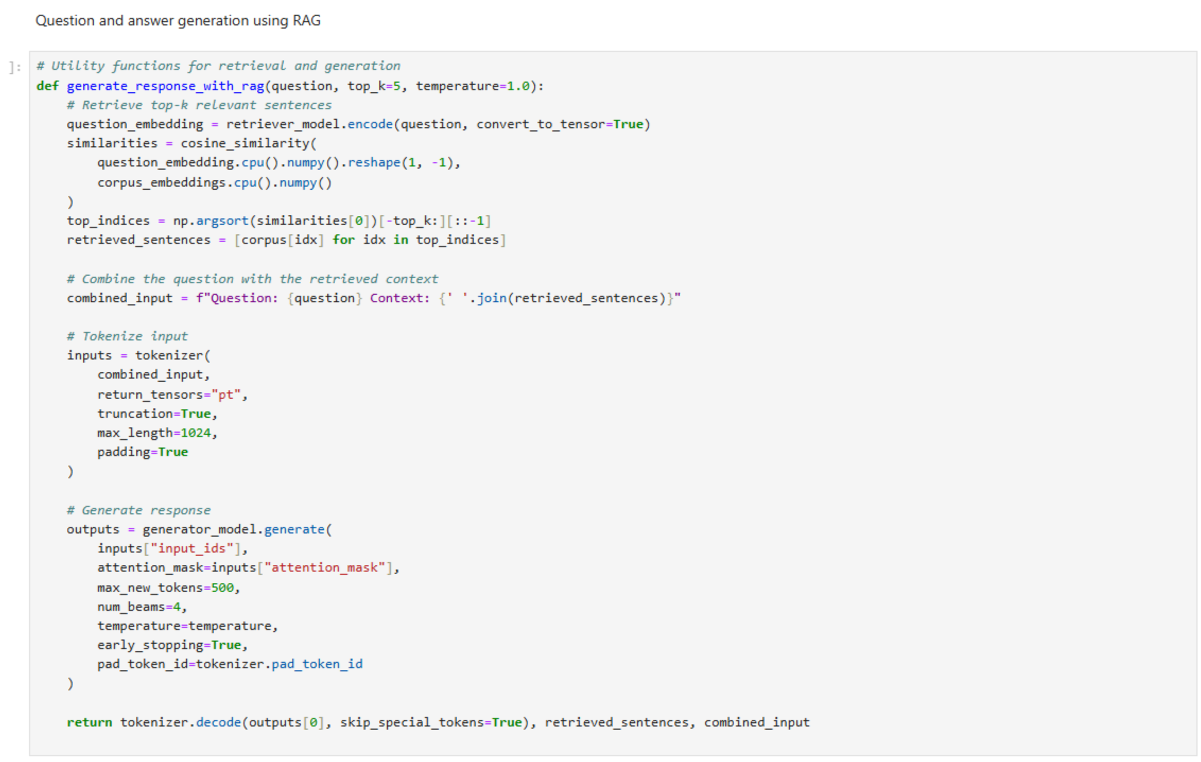
Model 2 – gpt-neo



Loading gpt-neo-1.3B and padding the tokens to resize it

```
[ ]: # Load the GPT-Neo model and tokenizer
model_name = "EleutherAI/gpt-neo-1.3B"
generator_model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto", torch_dtype=torch.float16)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Set a padding token if it doesn't exist
if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({'pad_token': tokenizer.eos_token})

# Resize the model embeddings to include the new padding token if added
generator_model.resize_token_embeddings(len(tokenizer))
```

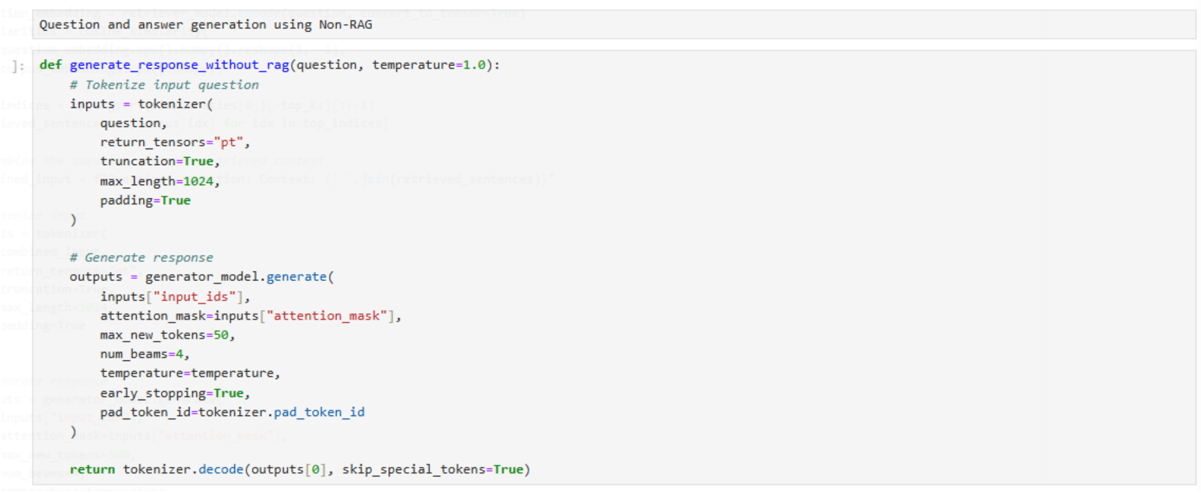
RAG function



Question and answer generation using RAG

```
]: # Utility functions for retrieval and generation
def generate_response_with_rag(question, top_k=5, temperature=1.0):
    # Retrieve top-k relevant sentences
    question_embedding = retriever_model.encode(question, convert_to_tensor=True)
    similarities = cosine_similarity(
        question_embedding.cpu().numpy().reshape(1, -1),
        corpus_embeddings.cpu().numpy()
    )
    top_indices = np.argsort(similarities[0])[-top_k:][::-1]
    retrieved_sentences = [corpus[idx] for idx in top_indices]

    # Combine the question with the retrieved context
    combined_input = f"Question: {question} Context: {' '.join(retrieved_sentences)}"

    # Tokenize input
    inputs = tokenizer(
        combined_input,
        return_tensors="pt",
        truncation=True,
        max_length=1024,
        padding=True
    )

    # Generate response
    outputs = generator_model.generate(
        inputs["input_ids"],
        attention_mask=inputs["attention_mask"],
        max_new_tokens=500,
        num_beams=4,
        temperature=temperature,
        early_stopping=True,
        pad_token_id=tokenizer.pad_token_id
    )

    return tokenizer.decode(outputs[0], skip_special_tokens=True), retrieved_sentences, combined_input
```

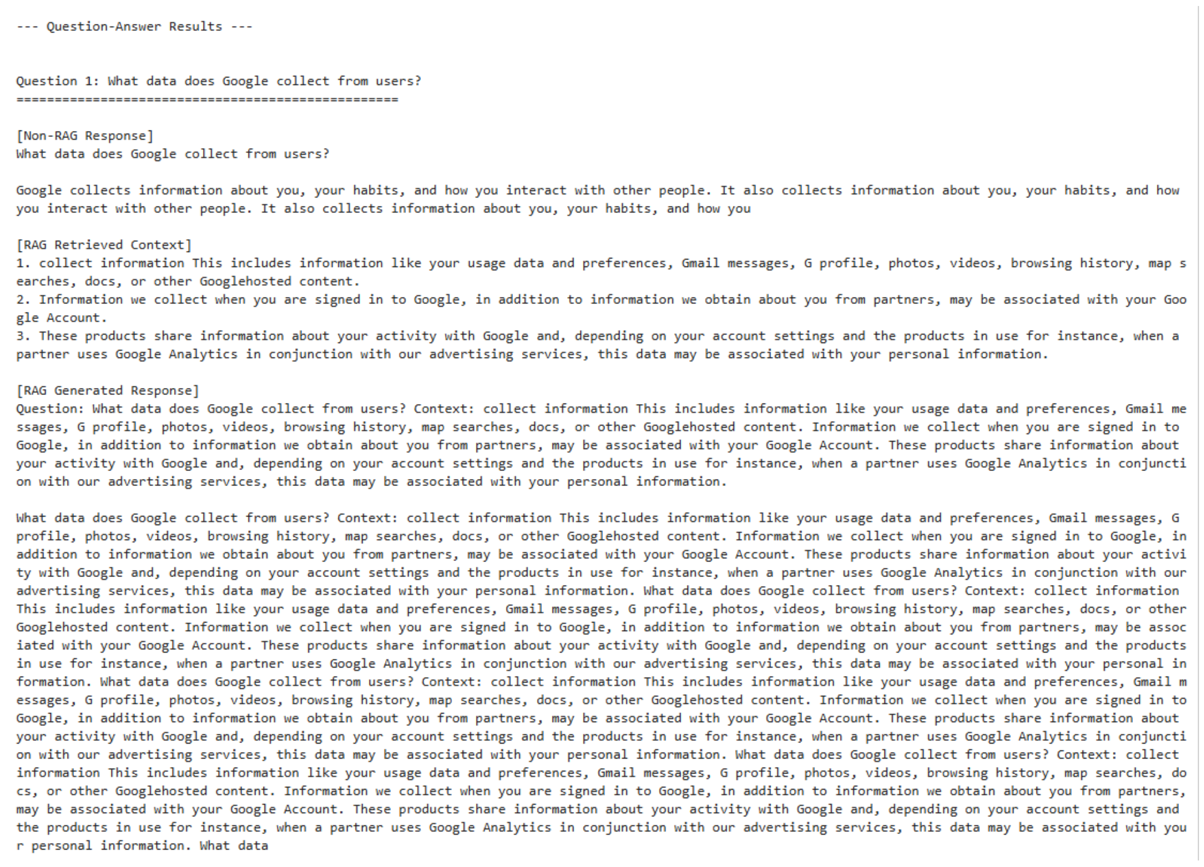
Non RAG function



```
Question and answer generation using Non-RAG
```

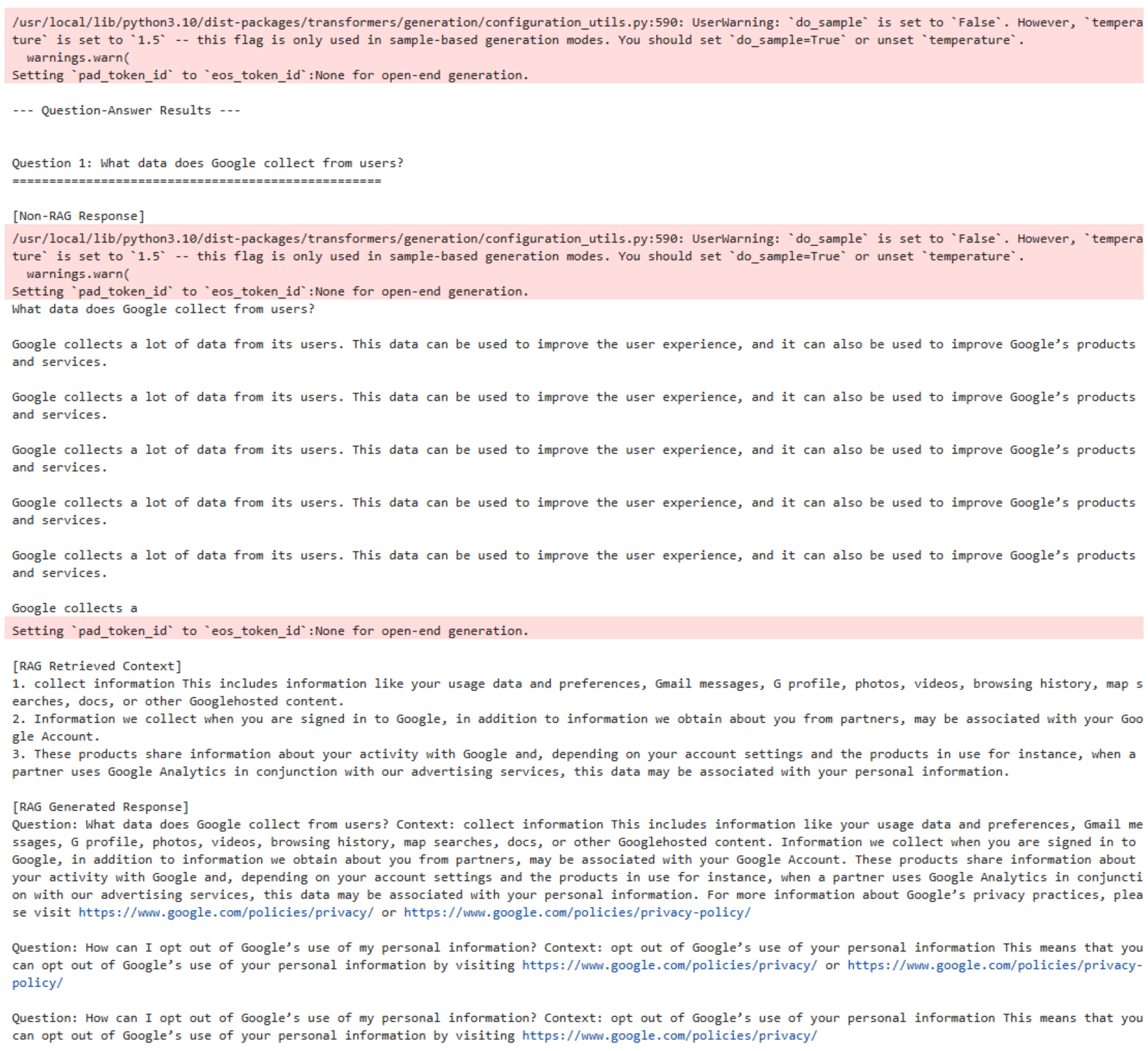
```
]: def generate_response_without_rag(question, temperature=1.0):
    # Tokenize input question
    inputs = tokenizer(
        question,
        return_tensors="pt",
        truncation=True,
        max_length=1024,
        padding=True
    )

    # Generate response
    outputs = generator_model.generate(
        inputs["input_ids"],
        attention_mask=inputs["attention_mask"],
        max_new_tokens=50,
        num_beams=4,
        temperature=temperature,
        early_stopping=True,
        pad_token_id=tokenizer.pad_token_id
    )

    return tokenizer.decode(outputs[0], skip_special_tokens=True)
```

Output from gpt2 (Question 1):-



```
--- Question-Answer Results ---


Question 1: What data does Google collect from users?
==================================================

[Non-RAG Response]
What data does Google collect from users?

Google collects information about you, your habits, and how you interact with other people. It also collects information about you, your habits, and how
you interact with other people. It also collects information about you, your habits, and how you

[RAG Retrieved Context]
1. collect information This includes information like your usage data and preferences, Gmail messages, G profile, photos, videos, browsing history, map s
earches, docs, or other Googlehosted content.
2. Information we collect when you are signed in to Google, in addition to information we obtain about you from partners, may be associated with your Goo
gle Account.
3. These products share information about your activity with Google and, depending on your account settings and the products in use for instance, when a
partner uses Google Analytics in conjunction with our advertising services, this data may be associated with your personal information.

[RAG Generated Response]
Question: What data does Google collect from users? Context: collect information This includes information like your usage data and preferences, Gmail me
ssages, G profile, photos, videos, browsing history, map searches, docs, or other Googlehosted content. Information we collect when you are signed in to
Google, in addition to information we obtain about you from partners, may be associated with your Google Account. These products share information about
your activity with Google and, depending on your account settings and the products in use for instance, when a partner uses Google Analytics in conjuncti
on with our advertising services, this data may be associated with your personal information.

What data does Google collect from users? Context: collect information This includes information like your usage data and preferences, Gmail messages, G
profile, photos, videos, browsing history, map searches, docs, or other Googlehosted content. Information we collect when you are signed in to Google, in
addition to information we obtain about you from partners, may be associated with your Google Account. These products share information about your activi
ty with Google and, depending on your account settings and the products in use for instance, when a partner uses Google Analytics in conjunction with our
advertising services, this data may be associated with your personal information. What data does Google collect from users? Context: collect information
This includes information like your usage data and preferences, Gmail messages, G profile, photos, videos, browsing history, map searches, docs, or other
Googlehosted content. Information we collect when you are signed in to Google, in addition to information we obtain about you from partners, may be assoc
iated with your Google Account. These products share information about your activity with Google and, depending on your account settings and the products
in use for instance, when a partner uses Google Analytics in conjunction with our advertising services, this data may be associated with your personal in
formation. What data does Google collect from users? Context: collect information This includes information like your usage data and preferences, Gmail m
essages, G profile, photos, videos, browsing history, map searches, docs, or other Googlehosted content. Information we collect when you are signed in to
Google, in addition to information we obtain about you from partners, may be associated with your Google Account. These products share information about
your activity with Google and, depending on your account settings and the products in use for instance, when a partner uses Google Analytics in conjuncti
on with our advertising services, this data may be associated with your personal information. What data does Google collect from users? Context: collect
information This includes information like your usage data and preferences, Gmail messages, G profile, photos, videos, browsing history, map searches, do
cs, or other Googlehosted content. Information we collect when you are signed in to Google, in addition to information we obtain about you from partners,
may be associated with your Google Account. These products share information about your activity with Google and, depending on your account settings and
the products in use for instance, when a partner uses Google Analytics in conjunction with our advertising services, this data may be associated with you
r personal information. What data
```

Output from gpt-neo (Question 1):-



```
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:590: UserWarning: `do_sample` is set to `False`. However, `tempera
ture` is set to `1.5` -- this flag is only used in sample-based generation modes. You should set `do_sample=True` or unset `temperature`.
  warnings.warn(
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.

--- Question-Answer Results ---


Question 1: What data does Google collect from users?
==================================================

[Non-RAG Response]
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:590: UserWarning: `do_sample` is set to `False`. However, `tempera
ture` is set to `1.5` -- this flag is only used in sample-based generation modes. You should set `do_sample=True` or unset `temperature`.
  warnings.warn(
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
What data does Google collect from users?

Google collects a lot of data from its users. This data can be used to improve the user experience, and it can also be used to improve Google's products
and services.

Google collects a lot of data from its users. This data can be used to improve the user experience, and it can also be used to improve Google's products
and services.

Google collects a lot of data from its users. This data can be used to improve the user experience, and it can also be used to improve Google's products
and services.

Google collects a lot of data from its users. This data can be used to improve the user experience, and it can also be used to improve Google's products
and services.

Google collects a lot of data from its users. This data can be used to improve the user experience, and it can also be used to improve Google's products
and services.

Google collects a
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.

[RAG Retrieved Context]
1. collect information This includes information like your usage data and preferences, Gmail messages, G profile, photos, videos, browsing history, map s
earches, docs, or other Googlehosted content.
2. Information we collect when you are signed in to Google, in addition to information we obtain about you from partners, may be associated with your Goo
gle Account.
3. These products share information about your activity with Google and, depending on your account settings and the products in use for instance, when a
partner uses Google Analytics in conjunction with our advertising services, this data may be associated with your personal information.

[RAG Generated Response]
Question: What data does Google collect from users? Context: collect information This includes information like your usage data and preferences, Gmail me
ssages, G profile, photos, videos, browsing history, map searches, docs, or other Googlehosted content. Information we collect when you are signed in to
Google, in addition to information we obtain about you from partners, may be associated with your Google Account. These products share information about
your activity with Google and, depending on your account settings and the products in use for instance, when a partner uses Google Analytics in conjuncti
on with our advertising services, this data may be associated with your personal information. For more information about Google's privacy practices, plea
se visit https://www.google.com/policies/privacy/ or https://www.google.com/policies/privacy-policy/

Question: How can I opt out of Google's use of my personal information? Context: opt out of Google's use of your personal information This means that you
can opt out of Google's use of your personal information by visiting https://www.google.com/policies/privacy/ or https://www.google.com/policies/privacy-
policy/

Question: How can I opt out of Google's use of my personal information? Context: opt out of Google's use of your personal information This means that you
can opt out of Google's use of your personal information by visiting https://www.google.com/policies/privacy/
```

Comparative Discussion: RAG vs. Non-RAG Models

1. Accuracy and Relevance

Non-RAG Model:

The Non-RAG responses are often repetitive, generic, and lack specificity. For example: For Question 1: "What data does Google collect from users?" the response loops over phrases without providing concrete details. For Question 2: "When does Google share user data externally?" the response inaccurately claims Google does not share data externally, which is misleading. Non-RAG models fail to extract meaningful context, relying solely on their pre-trained knowledge, which can lead to outdated or irrelevant information.

RAG Model:

RAG responses provide detailed, contextual answers by leveraging retrieved content. For Question 1, RAG contextualizes the response with specific examples of data Google collects, such as usage data, preferences, browsing history, and Google-hosted content. For Question 2, RAG accurately highlights legal and consent-based scenarios where Google shares data, showing alignment with the retrieved context.

2. Contextual Awareness

Non-RAG Model:

Limited to the model's internal knowledge, resulting in responses that are sometimes vague or fail to align with the specific context of the document.

RAG Model:

Retrieves relevant chunks from the document to create responses that are contextually rich and document-specific.

This is evident in Question 3, where RAG discusses privacy controls like the ability to manage account settings and adherence to regulatory frameworks.

3. Redundancy in Outputs

Both RAG and Non-RAG models showed some degree of redundancy in their responses, especially for longer contexts. RAG responses sometimes include the retrieved context verbatim in the generated output, which can feel repetitive but is accurate. Non-RAG responses, on the other hand, tend to repeat themselves due to a lack of external grounding.