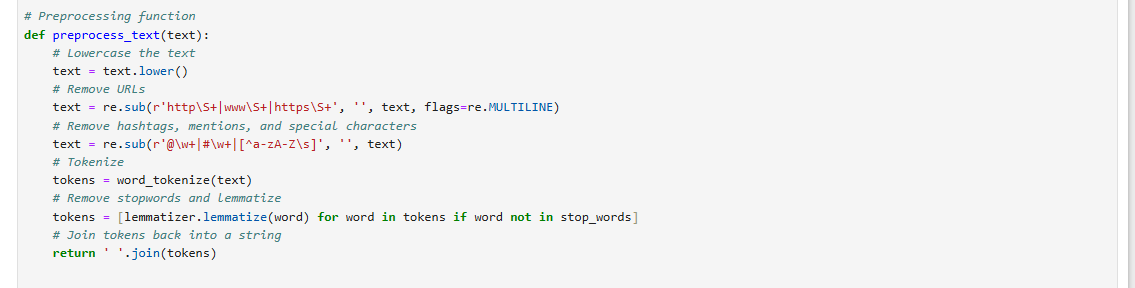
**Fall 2023 DATA 225 Deep Learning Technologies**

**Homework – 4**

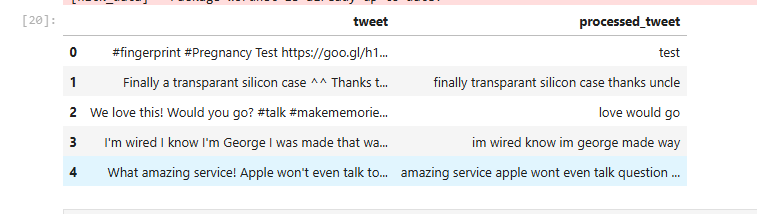
**Name :- Prayag Nikul Purani**

**SJSU Id :- 017416737**

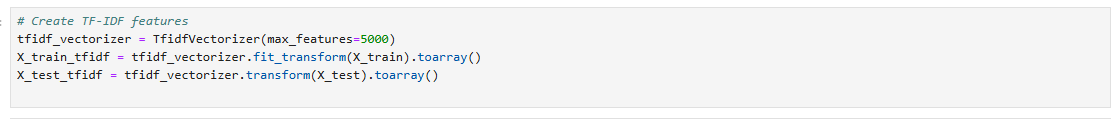
**Problem 1 - (Coding):-**

a.  


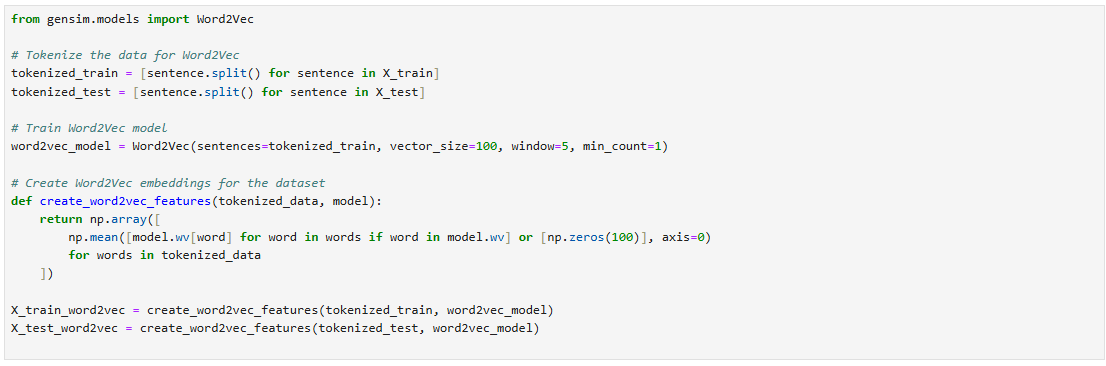
The first image shows a Python function, preprocess\_text, designed to clean and prepare text data for analysis. It performs several preprocessing steps, including converting text to lowercase for uniformity, removing URLs, hashtags, mentions, and special characters using regular expressions. It then 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.



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.  
  


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.



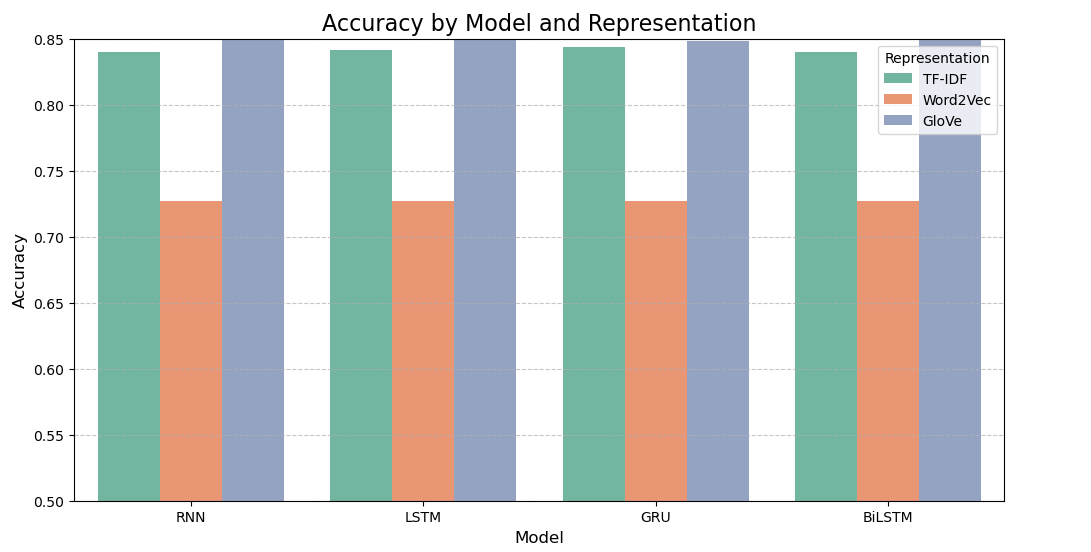
The provided code demonstrates the process of generating numerical feature representations for text data using Word2Vec. First, the training and testing datasets (X\_train and X\_test) are tokenized by splitting each sentence into a list of words. Then, a Word2Vec model is trained on the tokenized training 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.

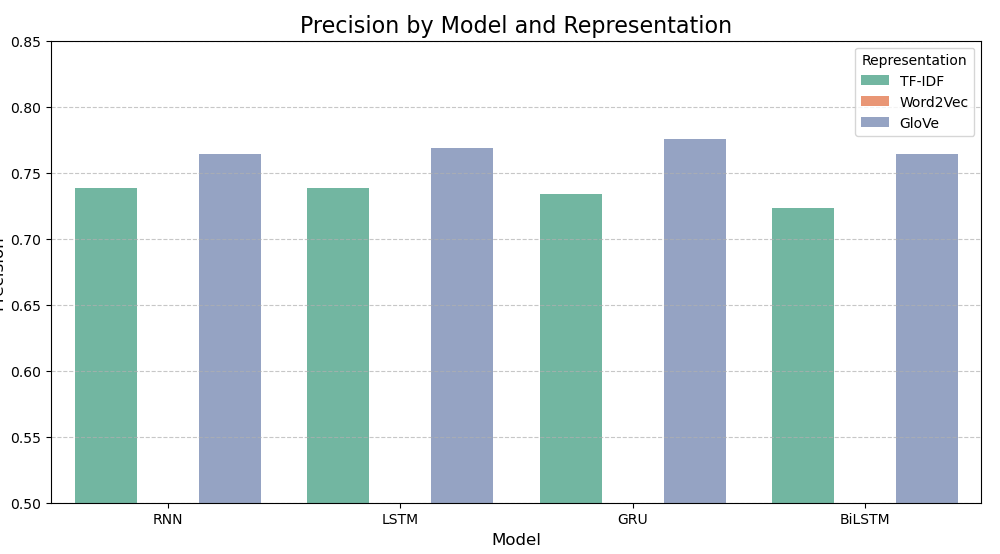
To create sentence-level embeddings, a custom function, create\_word2vec\_features, is used. It computes the mean of the Word2Vec embeddings for all words in each sentence. If a word is not present in the model's vocabulary, it is replaced with a zero vector of size 100. This ensures that all sentences 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.

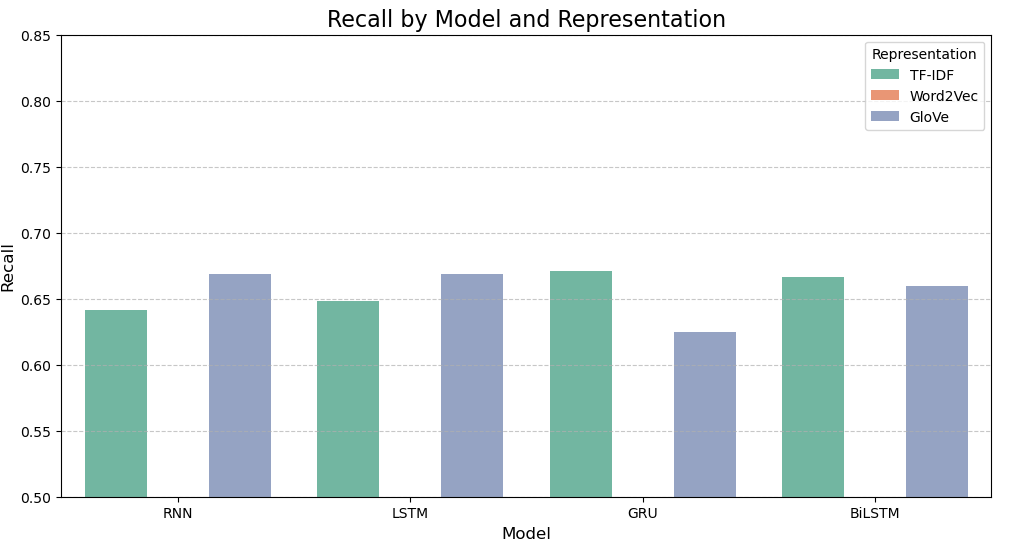


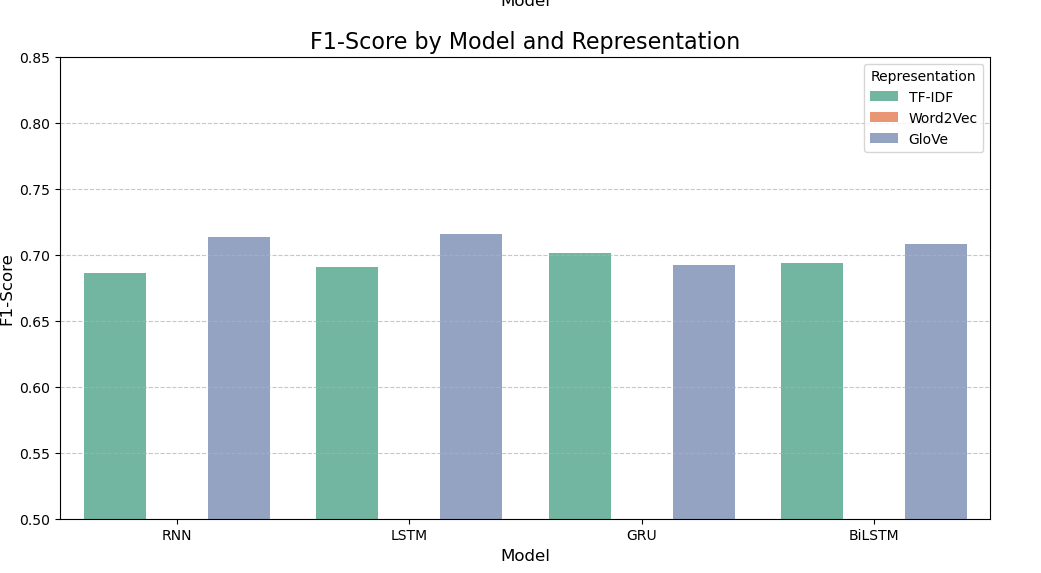
Next, the function create\_glove\_features is defined to generate sentence-level embeddings for the tokenized data. For each sentence, it calculates the mean of the GloVe embeddings for all words in the sentence. If a word is not found in the GloVe dictionary, it is represented by a zero vector of size 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.









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.

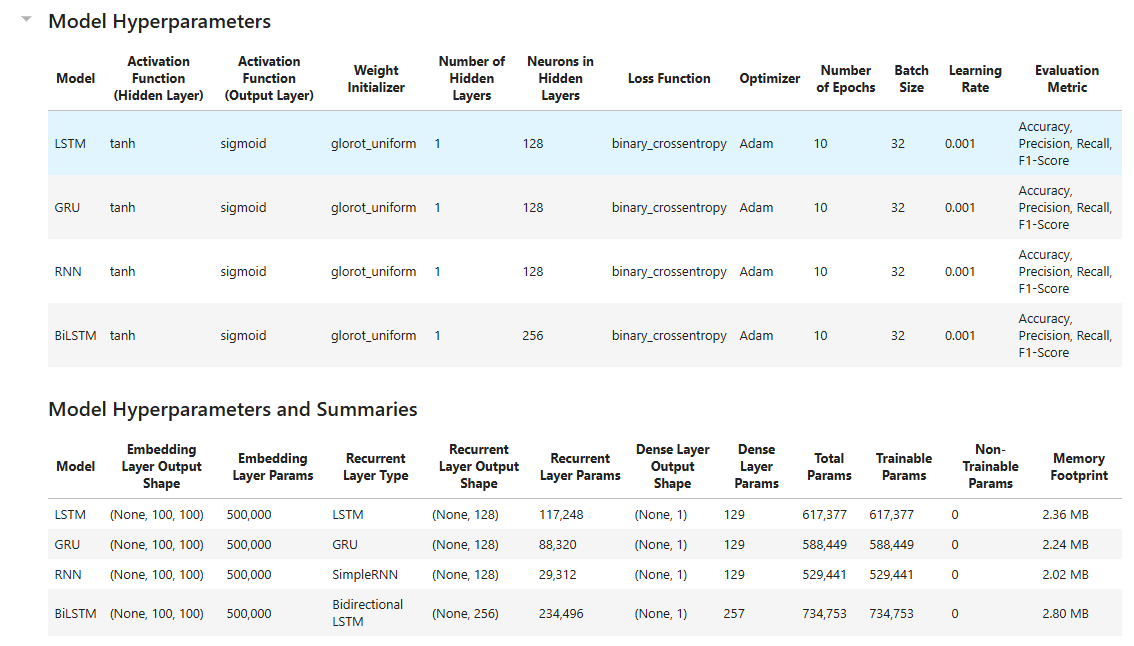
2. Impact of Word Representations:

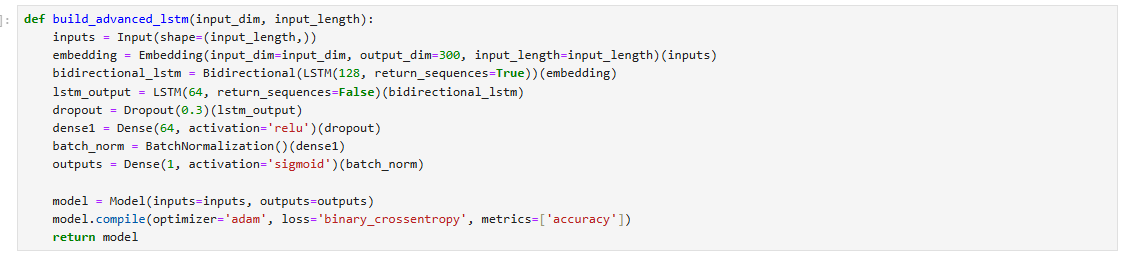
GloVe achieves the best results overall for all models (indicated by blue bars being the tallest in each group).

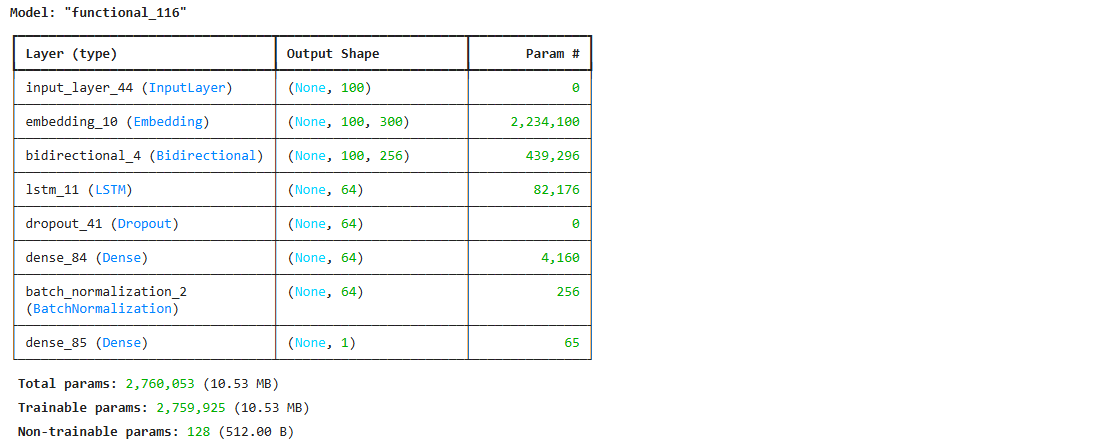
TF-IDF has the lowest F1-Scores across all models, likely because it lacks semantic information compared to Word2Vec and GloVe.

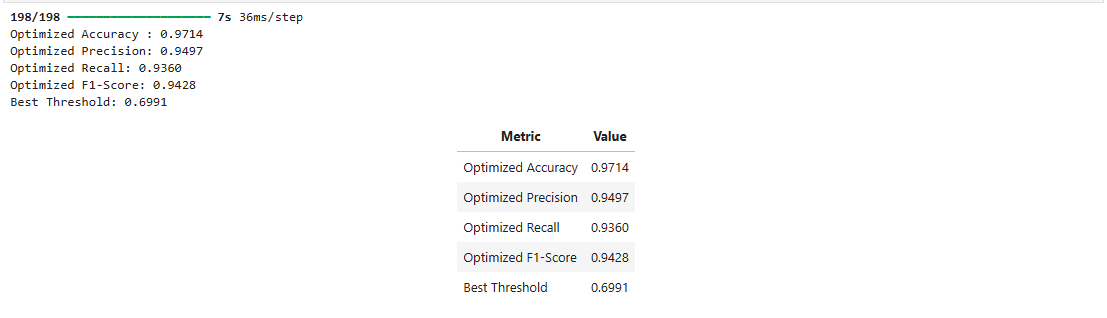
3. Comparing Models with the Same Representation:

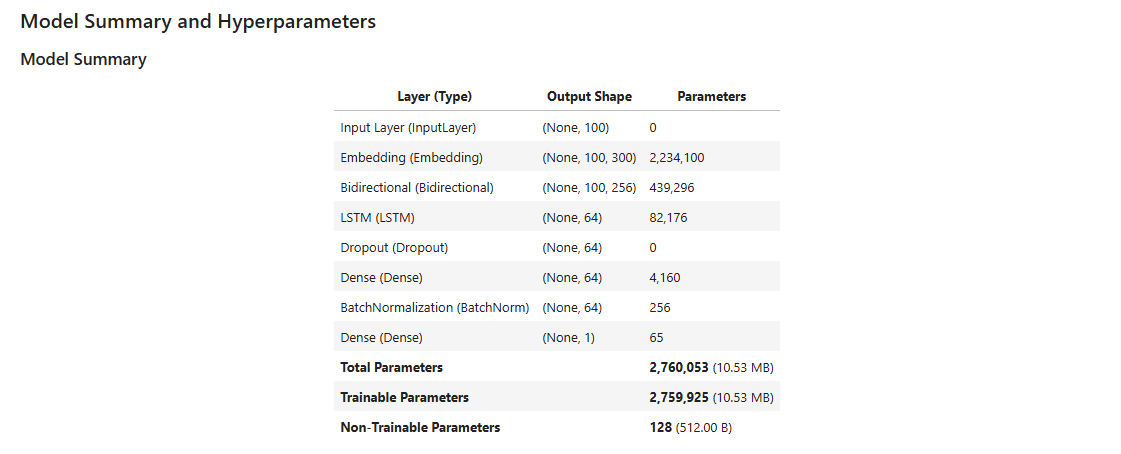
Across all three representations, BiLSTM outperforms other models, indicating its ability to capture bidirectional context is particularly beneficial for this task.

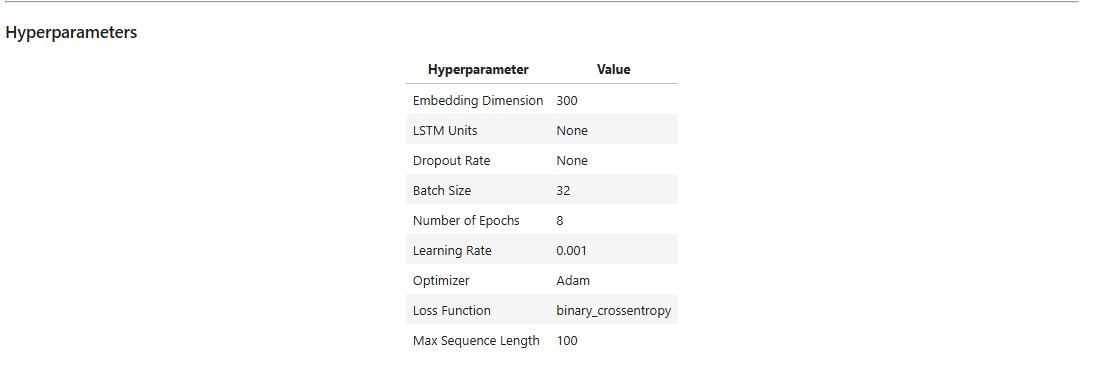
  
So, we will fine tune Glove and BiLSTM to

c.  










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.

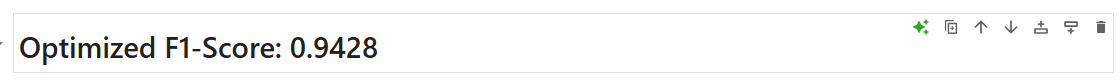
4. Optimized Recall:

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:

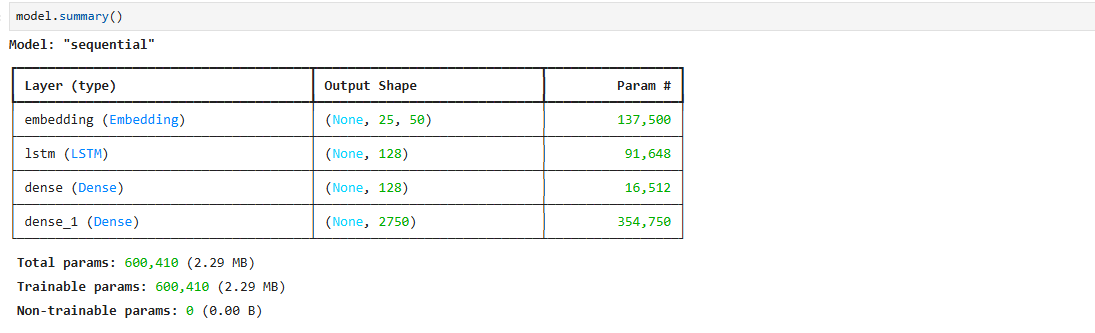
The threshold determines the probability cutoff for classifying a sample as positive or negative. A threshold of 0.6991 means that predictions with a probability of 0.6991 or higher will be classified as positive, optimizing the balance between precision and recall to achieve the highest F1-Score.  


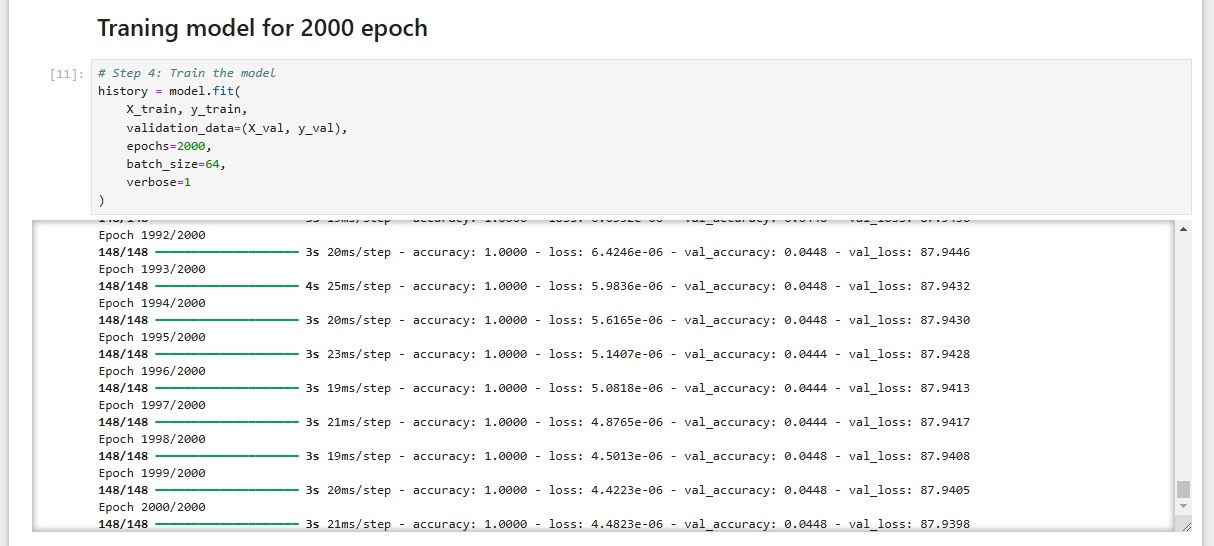
**Problem 2 - (Coding):-**

A



B





Key Metrics

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

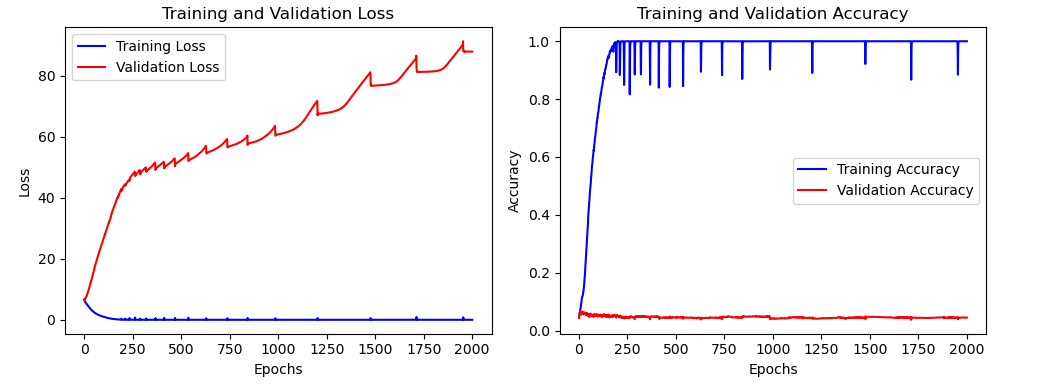
2. Time per Step: 3s: The total time taken for this epoch was approximately 3 seconds.

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.

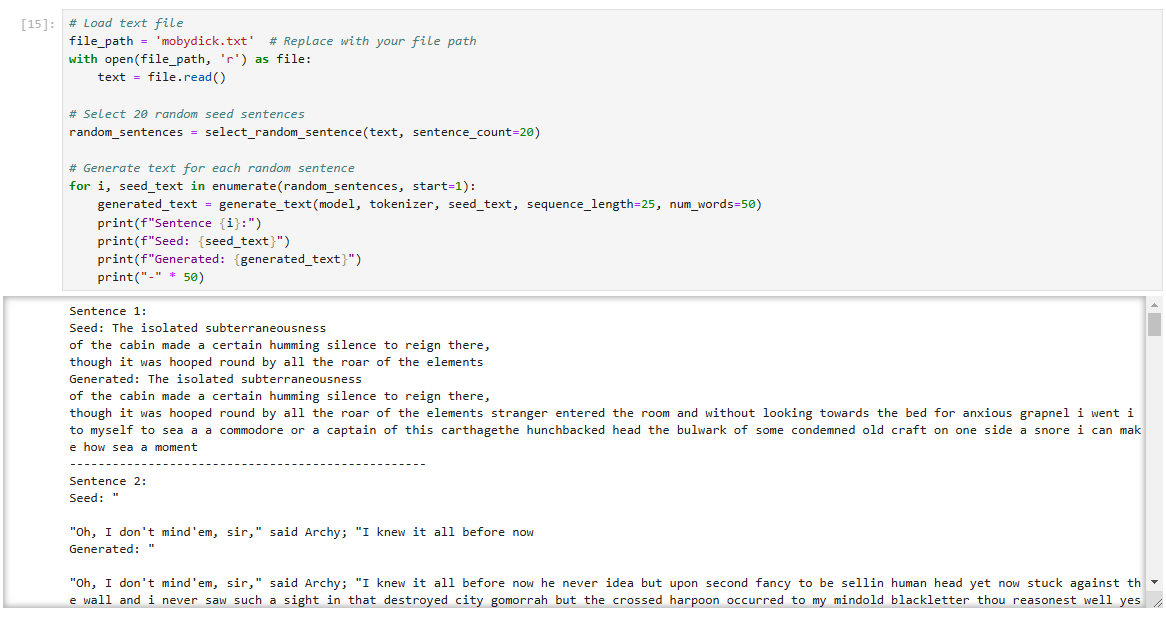
5. Validation Accuracy: val\_accuracy: 0.0448: The validation accuracy is 4.48%, which is very poor compared to the training accuracy (100%). This suggests a severe overfitting issue where the model performs well on the training data but fails to generalize to unseen validation data.

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.





C





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

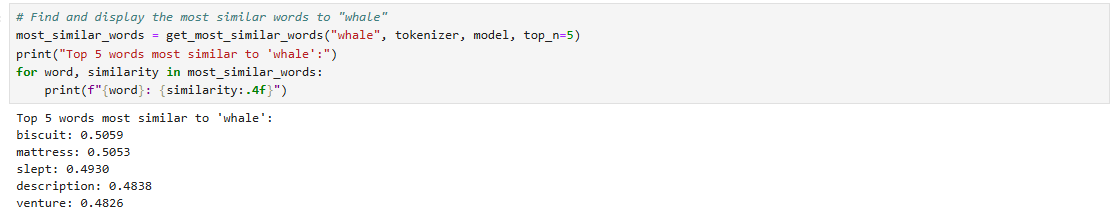
Observations:

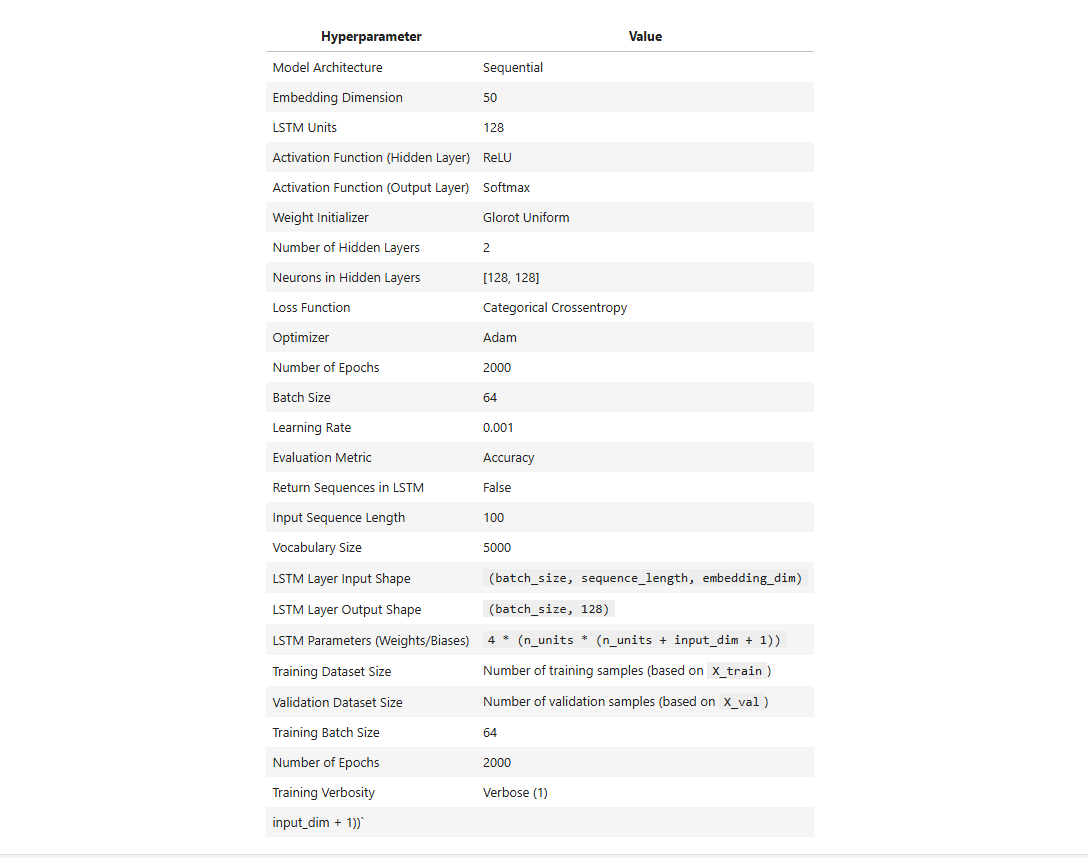
1. Training Accuracy (Blue):

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.





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)

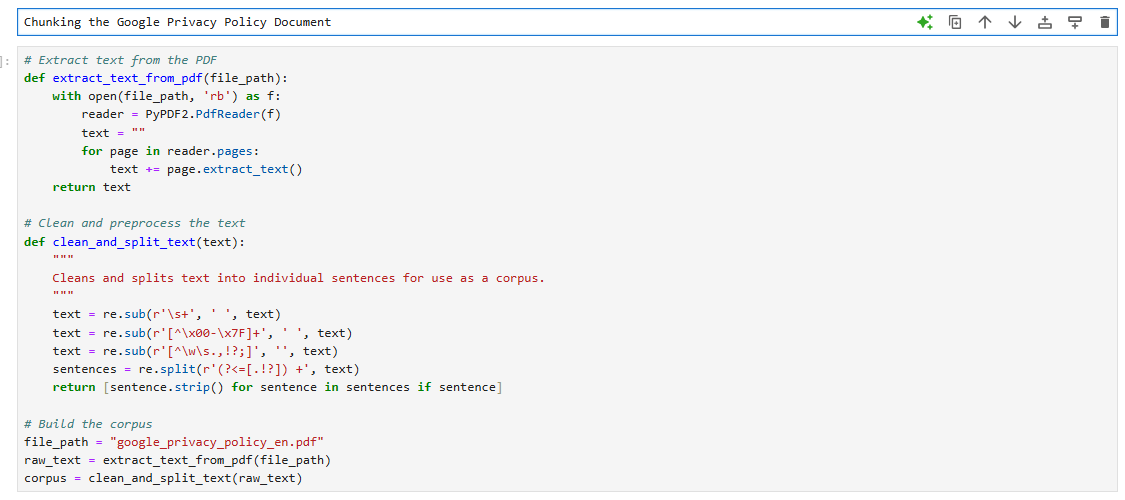
slept (Cosine Similarity: 0.4930)

description (Cosine Similarity: 0.4838)

venture (Cosine Similarity: 0.4826)

**Problem 3 - (Coding):-**

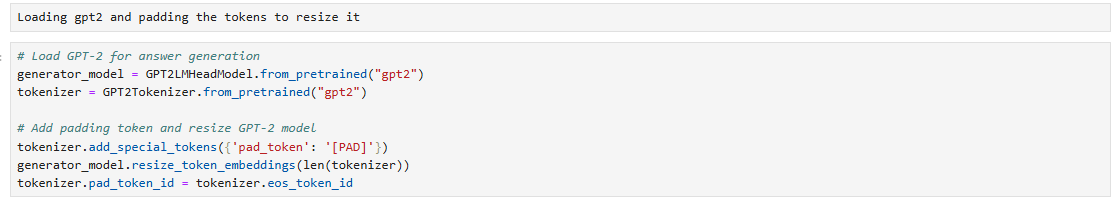
Building corpus



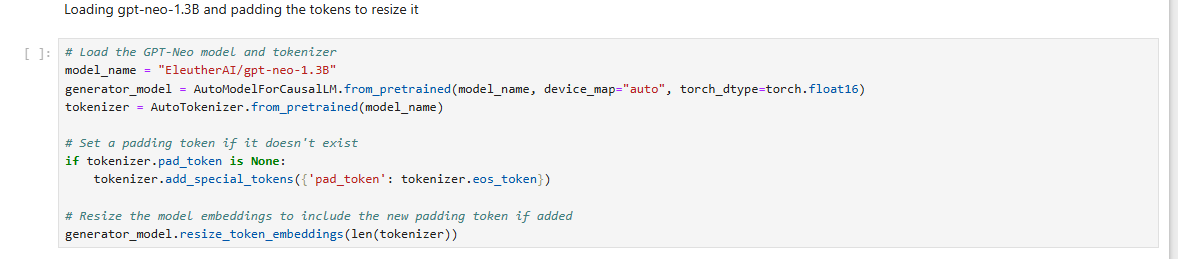
Embedding using all- MiniLM-L6-v2



Model 1 – gpt2



Model 2 – gpt-neo



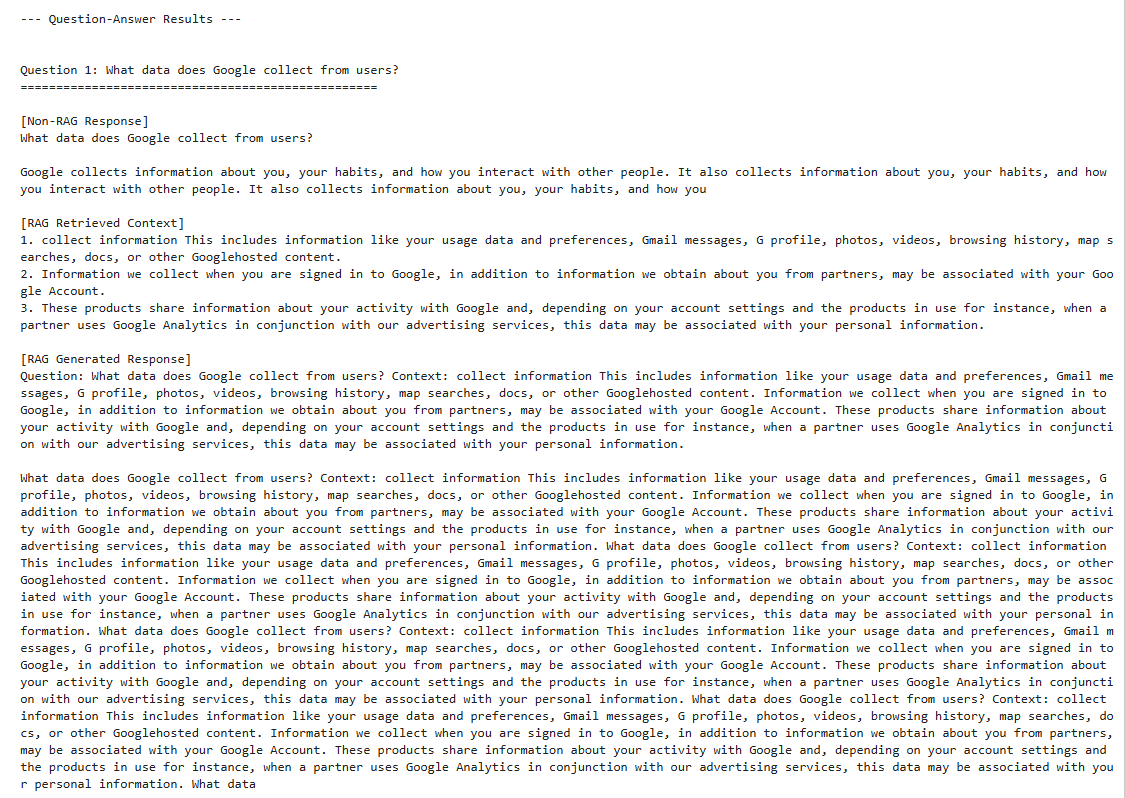
RAG function



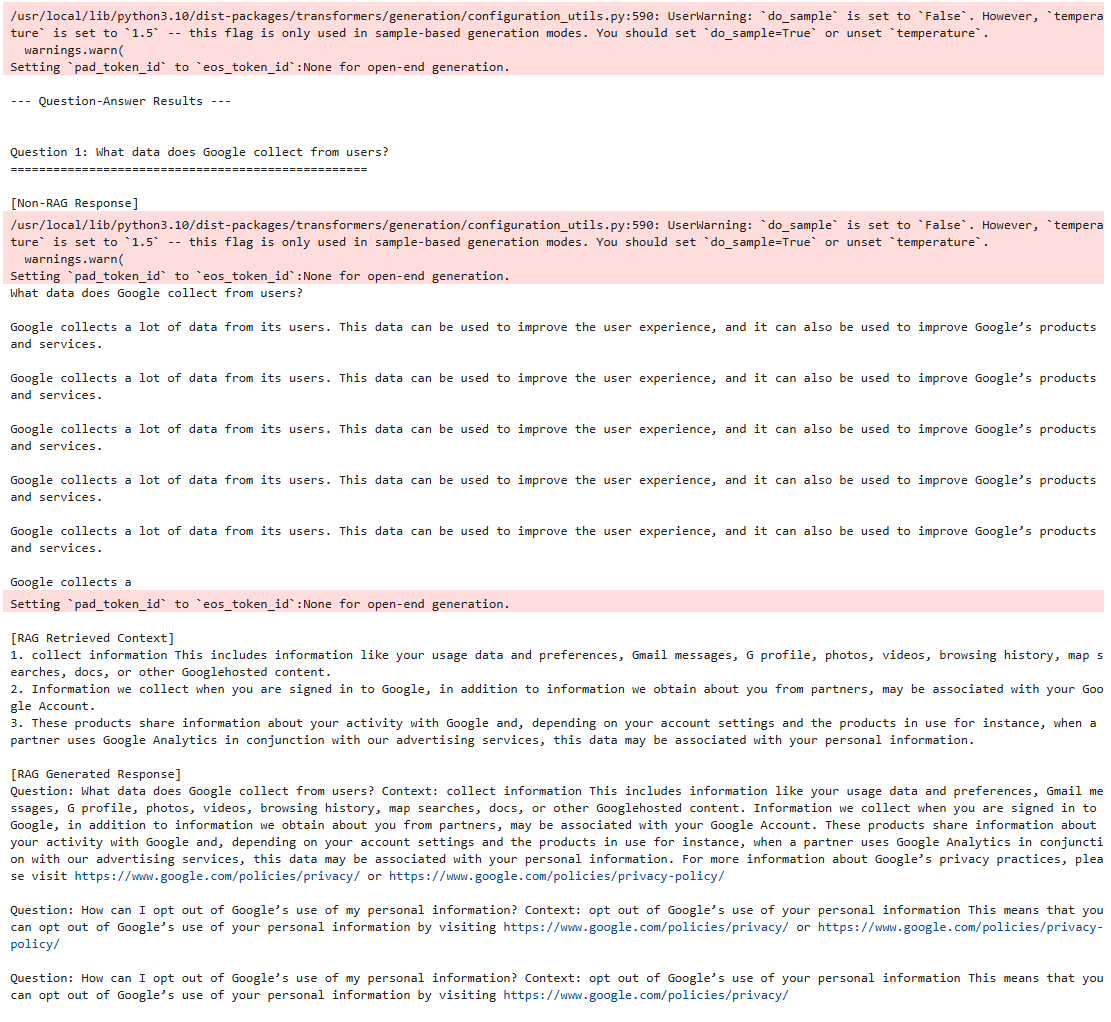
Non RAG function



Output from gpt2 (Question 1):-



Output from gpt-neo (Question 1):-



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.