```
# 📦 Install required libraries
!pip install tensorflow nltk gensim scikit-learn

# 📁 Upload your dataset file
# from google.colab import files
# uploaded = files.upload()

# 📚 Imports
import pandas as pd
import numpy as np
import re
import nltk
import gensim
import tensorflow as tf

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, Bidirectional, Layer
from tensorflow.keras.initializers import Constant

# 📥 Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# 📁 Load the dataset (update filename if needed)
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/NLP_LAB/Training_Essay_Data (1).csv/Training_Essay_Data.csv")

# 🧼 Preprocessing
def preprocess_text(text):
    text = re.sub(r'\W', ' ', str(text))
    tokens = nltk.word_tokenize(text.lower())
    tokens = [w for w in tokens if w not in stopwords.words('english')]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(w) for w in tokens]
    return ' '.join(tokens)

df['clean_text'] = df['text'].apply(preprocess_text)
df['label'] = df['generated'].astype(int)

# 🔡 Tokenization
X = df['clean_text'].values
y = df['label'].values

vocab_size = 10000
max_len = 200

tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=max_len)

X_train, X_test, y_train, y_test = train_test_split(X_pad, y, test_size=0.2, random_state=42)

# 🔤 TF-IDF + Logistic Regression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

tfidf = TfidfVectorizer(max_features=5000)
X_tfidf = tfidf.fit_transform(df['clean_text'])

X_train_tfidf, X_test_tfidf, y_train_tfidf, y_test_tfidf = train_test_split(X_tfidf, y, test_size=0.2)

model_tfidf = LogisticRegression()
model_tfidf.fit(X_train_tfidf, y_train_tfidf)
preds_tfidf = model_tfidf.predict(X_test_tfidf)
print("\n🔍 TF-IDF Accuracy:", accuracy_score(y_test_tfidf, preds_tfidf))
print(classification_report(y_test_tfidf, preds_tfidf))

# 🎯 Attention Layer
class AttentionLayer(Layer):
    def __init__(self):
        super(AttentionLayer, self).__init__()
    def call(self, inputs):
        scores = tf.matmul(inputs, inputs, transpose_b=True)
        weights = tf.nn.softmax(scores, axis=-1)
        context = tf.matmul(weights, inputs)
        return tf.reduce_mean(context, axis=1)

# 🧠 BiLSTM + Attention Model
def build_attention_model(embedding_matrix, trainable=False):
    input_layer = Input(shape=(max_len,))
    embed = Embedding(input_dim=vocab_size, output_dim=embedding_matrix.shape[1],
                      embeddings_initializer=Constant(embedding_matrix),
                      input_length=max_len, trainable=trainable)(input_layer)
    lstm = Bidirectional(LSTM(64, return_sequences=True))(embed)
    attention = AttentionLayer()(lstm)
    output = Dense(1, activation='sigmoid')(attention)
    model = Model(inputs=input_layer, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# 🗒 Word2Vec Embeddings
def get_word2vec_embedding_matrix():
    tokenized_texts = [text.split() for text in df['clean_text']]
    w2v_model = gensim.models.Word2Vec(sentences=tokenized_texts, vector_size=100, window=5, min_count=1)
    embedding_matrix = np.zeros((vocab_size, 100))
    for word, i in tokenizer.word_index.items():
        if i < vocab_size and word in w2v_model.wv:
            embedding_matrix[i] = w2v_model.wv[word]
    return embedding_matrix

print("\n📦 Training Word2Vec + Attention...")
w2v_matrix = get_word2vec_embedding_matrix()
model_w2v = build_attention_model(w2v_matrix)
model_w2v.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test), verbose=1)

preds_w2v = (model_w2v.predict(X_test) > 0.5).astype(int)
print("\n📈 Word2Vec Accuracy:", accuracy_score(y_test, preds_w2v))
print(classification_report(y_test, preds_w2v))

# 📦 Download GloVe embeddings (run once)
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip

# 🗒 GloVe Embeddings
def get_glove_embedding_matrix(glove_file="glove.6B.100d.txt"):
    embeddings_index = {}
    with open(glove_file, encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coeffs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coeffs

    embedding_matrix = np.zeros((vocab_size, 100))
    for word, i in tokenizer.word_index.items():
        if i < vocab_size and word in embeddings_index:
            embedding_matrix[i] = embeddings_index[word]
    return embedding_matrix

print("\n📦 Training GloVe + Attention...")
glove_matrix = get_glove_embedding_matrix()
model_glove = build_attention_model(glove_matrix)
model_glove.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test), verbose=1)

preds_glove = (model_glove.predict(X_test) > 0.5).astype(int)
print("\n📈 GloVe Accuracy:", accuracy_score(y_test, preds_glove))
print(classification_report(y_test, preds_glove))

# ✅ Summary
print("\n✅ Summary of Results:")
print("TF-IDF + Logistic Regression Accuracy: ", accuracy_score(y_test_tfidf, preds_tfidf))
print("Word2Vec + Attention Accuracy:         ", accuracy_score(y_test, preds_w2v))
print("GloVe + Attention Accuracy:            ", accuracy_score(y_test, preds_glove))
```

```
HTTP request sent, awaiting response... 302 Found
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2025-04-11 19:03:15--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2025-04-11 19:03:15--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  3.04MB/s    in 3m 54s

2025-04-11 19:07:09 (3.52 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]


📦 Training GloVe + Attention...
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated
  warnings.warn(
729/729 ━━━━━━━━━━━━━━━━━━━━ 17s 21ms/step - accuracy: 0.9112 - loss: 0.2103 - val_accuracy: 0.9698 - val_loss: 0.1061
Epoch 2/5
729/729 ━━━━━━━━━━━━━━━━━━━━ 20s 20ms/step - accuracy: 0.9642 - loss: 0.1124 - val_accuracy: 0.9679 - val_loss: 0.0952
Epoch 3/5
729/729 ━━━━━━━━━━━━━━━━━━━━ 21s 21ms/step - accuracy: 0.9706 - loss: 0.0879 - val_accuracy: 0.9722 - val_loss: 0.0756
Epoch 4/5
729/729 ━━━━━━━━━━━━━━━━━━━━ 15s 21ms/step - accuracy: 0.9763 - loss: 0.0739 - val_accuracy: 0.9792 - val_loss: 0.0627
Epoch 5/5
729/729 ━━━━━━━━━━━━━━━━━━━━ 15s 21ms/step - accuracy: 0.9805 - loss: 0.0606 - val_accuracy: 0.9816 - val_loss: 0.0552
183/183 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step

📈 GloVe Accuracy: 0.9816435066049065
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      3539
           1       0.99      0.96      0.98      2290

    accuracy                           0.98      5829
   macro avg       0.98      0.98      0.98      5829
weighted avg       0.98      0.98      0.98      5829


✅ Summary of Results:
TF-IDF + Logistic Regression Accuracy:  0.9871332990221308
Word2Vec + Attention Accuracy:          0.9912506433350489
GloVe + Attention Accuracy:             0.9816435066049065
```