In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style='whitegrid')

#from wordcloud import WordCloud
import tensorflow as tf
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.metrics import classification_report,confusion_matrix
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, LSTM,Dense, SpatialDropout1D, Dropout
from keras.initializers import Constant

# Reset individual options to default
pd.reset_option('display.max_columns')
pd.reset_option('display.max_rows')
pd.reset_option('display.max_colwidth')

# Set desired options
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 900)
pd.set_option('display.max_colwidth', 200)

import warnings
warnings.filterwarnings("ignore")
```

```
2024-07-30 10:23:35.795478: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
2024-07-30 10:23:35.795599: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register c
uFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
2024-07-30 10:23:35.964797: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
```

In [2]:
```
train = pd.read_csv('/kaggle/input/sentiment-analysis-dataset/training.csv',header=None)
validation = pd.read_csv('/kaggle/input/sentiment-analysis-dataset/validation.csv',header=None)

train.columns=['Tweet ID','Entity','Sentiment','Tweet Content']
validation.columns=['Tweet ID','Entity','Sentiment','Tweet Content']

print("Training DataSet: \n")
train = train.sample(2000)
display(train.head())
print("Validation DataSet: \n")
#validation = validation.sample(1000)
display(validation.head())
```

Training DataSet:

	Tweet ID	Entity	Sentiment	Tweet Content
64883	7913	MaddenNFL	Negative	<unk>. Easy
24633	4628	Google	Neutral	.
44337	11616	Verizon	Negative	Hey all hi @verizonfios If is my service a 563Mbps down\553Mbps up when I pay for 1Gig symmetric services?. .. Test is performed over a wired connection over CAT8 SFTP 40GB cables.. speedtest.net/...
69136	3842	Cyberpunk2077	Irrelevant	Wife : " What did You do Last All Night? ".. Me : " Bombed a Few Legislative Buildings and a few Business People Meetings Me and You? ".. New Wife : "... Stayed Up All Night Missing About You. ".....
67722	7199	johnson&johnson	Neutral	Johnson & Johnson to Stop Selling Skin-Whitening Creams radio.com/articles/johns… via @Radiodotcom

Validation DataSet:

	Tweet ID	Entity	Sentiment	Tweet Content
0	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling for motivation to go for a run the other day, which has been translated by Tom's great auntie as 'Hayley can't get out of bed' and told to his grandma...
1	352	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects claims company acted like a 'drug dealer' bbc.co.uk/news/av/busine…
2	8312	Microsoft	Negative	@Microsoft Why do I pay for WORD when it functions so poorly on my @SamsungUS Chromebook?
3	4371	CS-GO	Negative	CSGO matchmaking is so full of closet hacking, it's a truly awful game.
4	4433	Google	Neutral	Now the President is slapping Americans in the face that he really did commit an unlawful act after his acquittal! From Discover on Google vanityfair.com/news/2020/02/t…

In [3]:
```
train = train.dropna(subset=['Tweet Content'])
```

```
display(train.isnull().sum())
print("*****"* 5)
display(validation.isnull().sum())
```

```
Tweet ID         0
Entity           0
Sentiment        0
Tweet Content    0
dtype: int64
*************************
Tweet ID         0
Entity           0
Sentiment        0
Tweet Content    0
dtype: int64
```

In [4]:
```
duplicates = train[train.duplicated(subset=['Entity', 'Sentiment', 'Tweet Content'], keep=False)]
train = train.drop_duplicates(subset=['Entity', 'Sentiment', 'Tweet Content'], keep='first')

duplicates = validation[validation.duplicated(subset=['Entity', 'Sentiment', 'Tweet Content'], keep=False)]
validation = validation.drop_duplicates(subset=['Entity', 'Sentiment', 'Tweet Content'], keep='first')
```

In [5]:
```
from tensorflow.keras.layers import Input, Dropout, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.initializers import TruncatedNormal
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy
from tensorflow.keras.utils import to_categorical

import pandas as pd
from sklearn.model_selection import train_test_split
```

In [6]:
```
!pip install plotly
```

```
Requirement already satisfied: plotly in /opt/conda/lib/python3.10/site-packages (5.18.0)
Requirement already satisfied: tenacity>=6.2.0 in /opt/conda/lib/python3.10/site-packages (from plotly) (8.2.3)
Requirement already satisfied: packaging in /opt/conda/lib/python3.10/site-packages (from plotly) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/python3.10/site-packages (from packag
ing->plotly) (3.1.1)
```

In [7]:
```
import pandas as pd
import plotly.graph_objects as go

# Assuming you've already run the data preprocessing steps
data = train[['Tweet Content', 'Sentiment']]

# Set your model output as categorical and save in new label col
data['Sentiment_label'] = pd.Categorical(data['Sentiment'])

# Transform your output to numeric
data['Sentiment'] = data['Sentiment_label'].cat.codes

# Use the entire training data as data_train
data_train = data

# Use validation data as data_test
data_test = validation[['Tweet Content', 'Sentiment']]
data_test['Sentiment_label'] = pd.Categorical(data_test['Sentiment'])
data_test['Sentiment'] = data_test['Sentiment_label'].cat.codes

# Create a colorful table using Plotly
fig = go.Figure(data=[go.Table(
    header=dict(
        values=list(data_train.columns),
        fill_color='paleturquoise',
        align='left',
        font=dict(color='black', size=12)
    ),
    cells=dict(
        values=[data_train[k].tolist()[:10] for k in data_train.columns],
        fill_color=[
            'lightcyan',  # Tweet Content
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
              else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_train['Sentiment_label'][:10]
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
              else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_train['Sentiment_label'][:10]
            'lavender'  # Sentiment (numeric)
        ],
        align='left',
        font=dict(color='black', size=11)
    ))
])

# Update the layout
fig.update_layout(
    title='First 10 Rows of Training Data',
```

```
    width=1000,
    height=400,
)

fig.show()
```

In [8]:

```
import plotly.graph_objects as go

# Create a colorful table using Plotly for the test data
fig = go.Figure(data=[go.Table(
    header=dict(
        values=list(data_test.columns),
        fill_color='paleturquoise',
        align='left',
        font=dict(color='black', size=12)
    ),
    cells=dict(
        values=[data_test[k].tolist()[:5] for k in data_test.columns],  # Show first 5 rows
        fill_color=[
            'lightcyan',  # Tweet Content
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
             else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_test['Sentiment_label'][:5]],
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
             else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_test['Sentiment_label'][:5]],
            'lavender'  # Sentiment (numeric)
        ],
        align='left',
        font=dict(color='black', size=11)
    ))
])

# Update the layout
fig.update_layout(
    title='First 5 Rows of Test Data',
    width=1000,
    height=300,
)

# Show the figure
fig.show()

# If you want to save the figure as an HTML file, uncomment the following line:
# fig.write_html("test_data_sample.html")
```

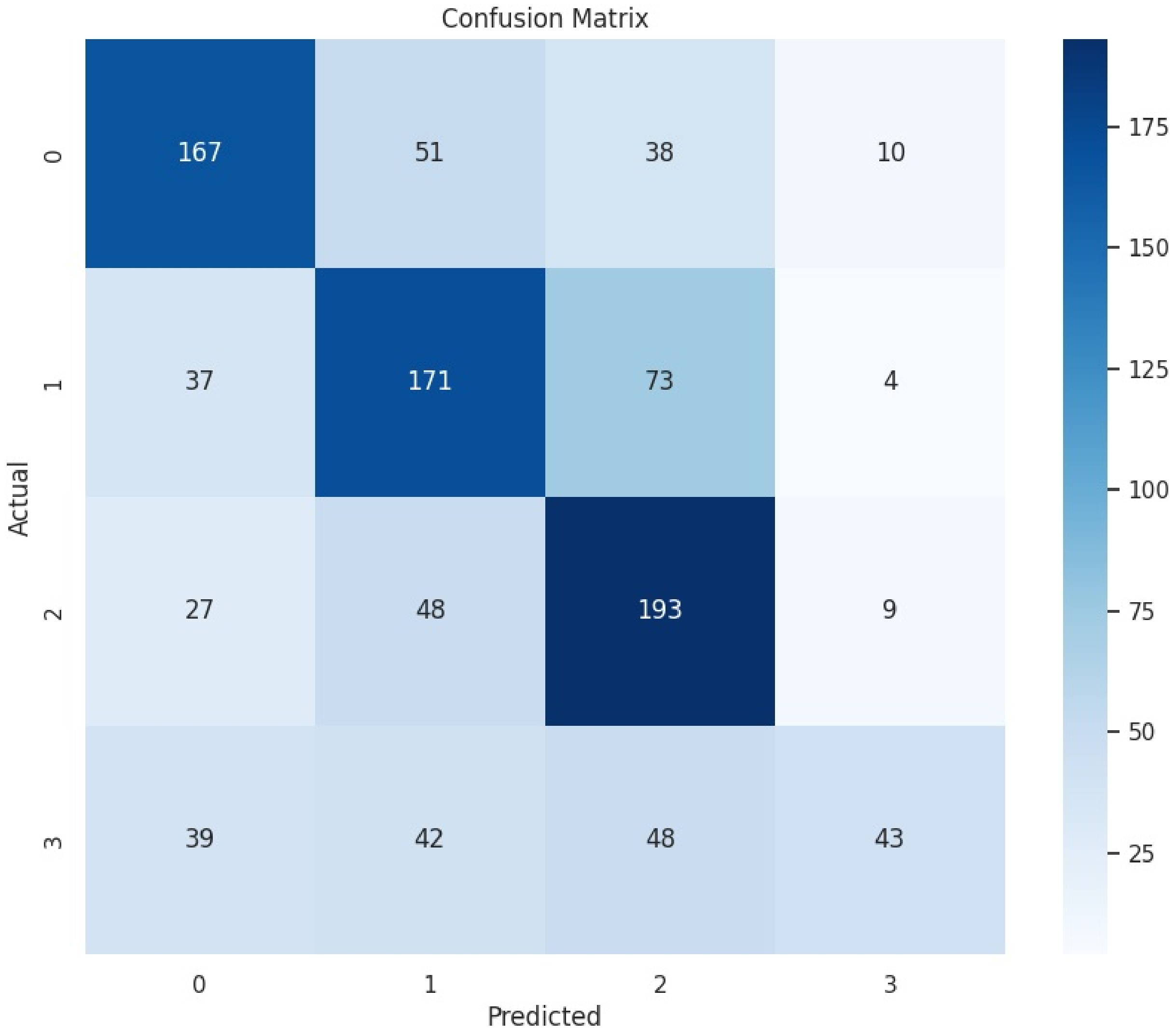
CNN

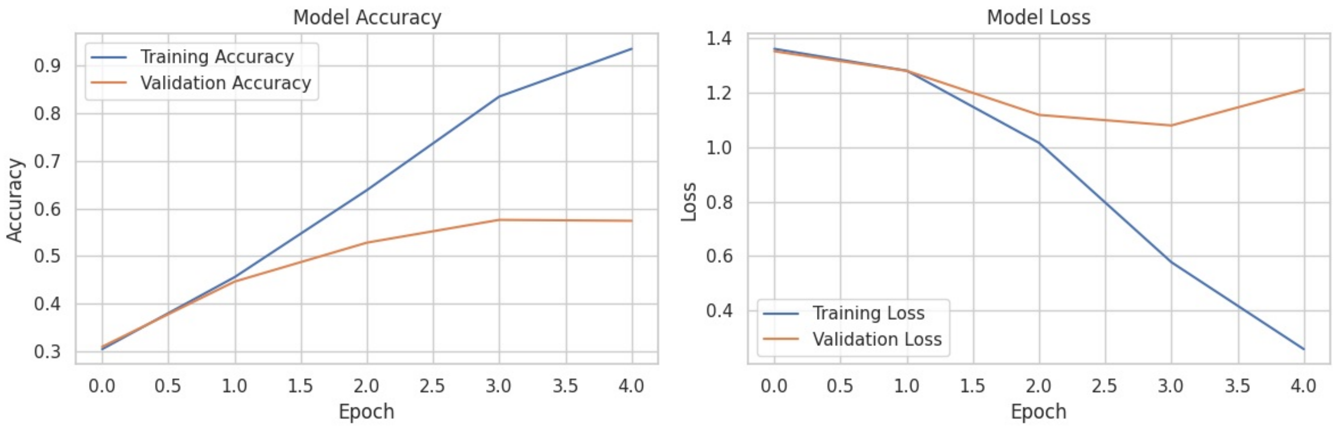
In [9]:

```
%%time

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Dropout
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure we're using GPU if available, otherwise use CPU
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)
else:
    print("No GPU found. Using CPU.")

# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Convert labels to integers
train_labels = train_labels.astype(int)

test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_labels = test_labels.astype(int)

# Preprocess text data
max_len = 100
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(train_texts)
train_sequences = tokenizer.texts_to_sequences(train_texts)
test_sequences = tokenizer.texts_to_sequences(test_texts)
train_padded = pad_sequences(train_sequences, maxlen=max_len, padding="post")
test_padded = pad_sequences(test_sequences, maxlen=max_len, padding="post")

# Create the model with CNN
model = Sequential([
    Embedding(5000, 128, input_length=max_len),
    Conv1D(128, 5, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(4, activation="softmax")
])

# Compile the model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train the model
epochs = 5
batch_size = 32
history = model.fit(
    train_padded, train_labels,
    epochs=epochs,
```

```python
    batch_size=batch_size,
    validation_data=(test_padded, test_labels),
    verbose=1
)

# Make predictions
y_pred = model.predict(test_padded)
y_pred_classes = np.argmax(y_pred, axis=1)

# Print classification report
print(classification_report(test_labels, y_pred_classes))

# Plot confusion matrix
cm = confusion_matrix(test_labels, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 1/5
47/62 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2979 - loss: 1.3724
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1722335048.161148      76 device_compiler.h:186] Compiled cluster using XLA!  This line is logged a
t most once for the lifetime of the process.
62/62 ━━━━━━━━━━━━━━━━━━━━ 10s 77ms/step - accuracy: 0.2992 - loss: 1.3696 - val_accuracy: 0.3090 - val_loss: 1
.3529
Epoch 2/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4457 - loss: 1.2935 - val_accuracy: 0.4460 - val_loss: 1.2
811
Epoch 3/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6291 - loss: 1.0683 - val_accuracy: 0.5280 - val_loss: 1.1
191
Epoch 4/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8264 - loss: 0.6299 - val_accuracy: 0.5760 - val_loss: 1.0
803
Epoch 5/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9354 - loss: 0.2790 - val_accuracy: 0.5740 - val_loss: 1.2
126
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
              precision    recall  f1-score   support

           0       0.62      0.63      0.62       266
           1       0.55      0.60      0.57       285
           2       0.55      0.70      0.61       277
           3       0.65      0.25      0.36       172

    accuracy                           0.57      1000
   macro avg       0.59      0.54      0.54      1000
weighted avg       0.58      0.57      0.56      1000
```






```
CPU times: user 11.6 s, sys: 1.35 s, total: 13 s
Wall time: 14.2 s
```

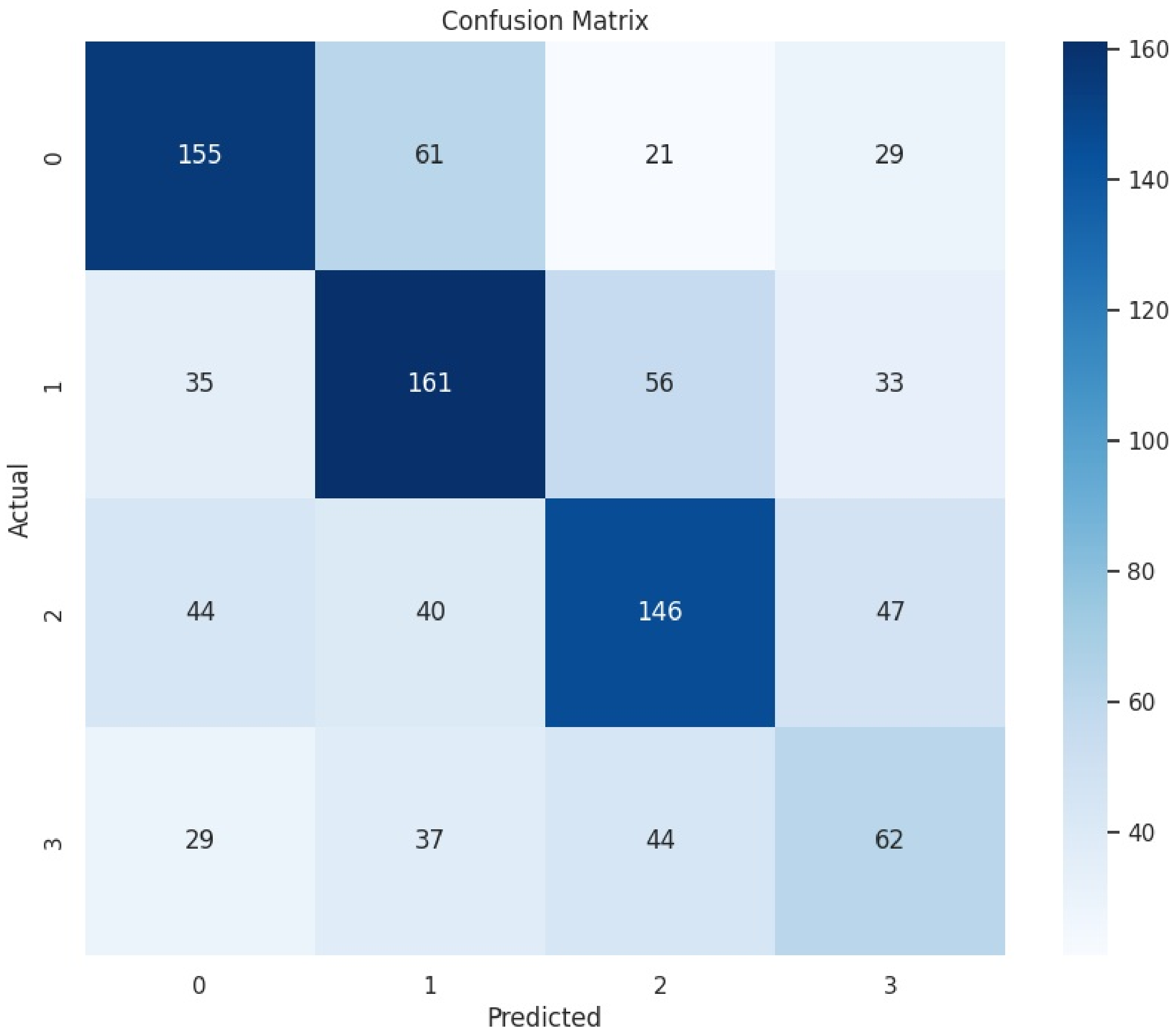
Gated Recurrent Units (GRU)

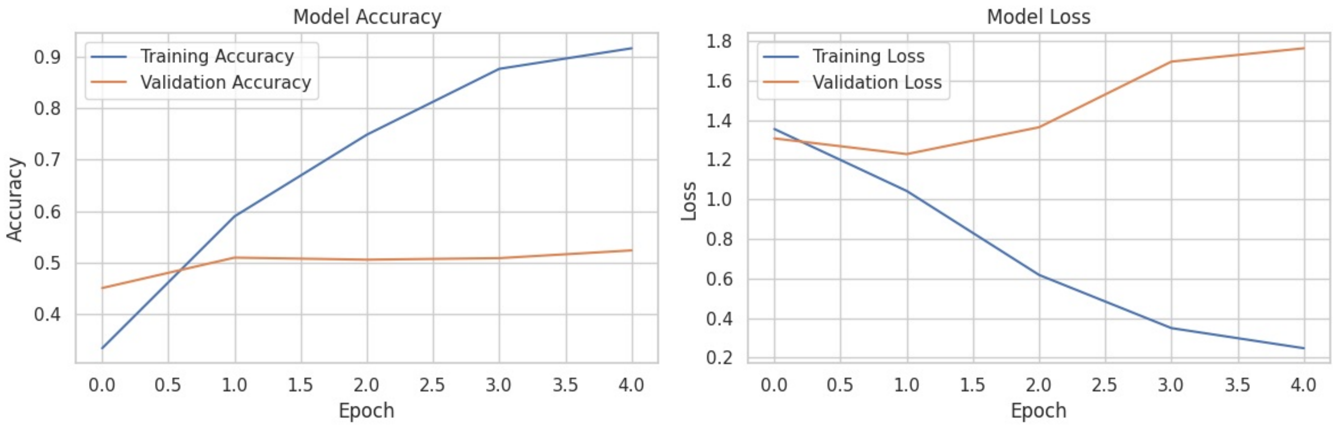
In [10]:
```
%%time

import pandas as pd
```

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, GRU, Dense
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure we're using GPU if available, otherwise use CPU
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)
else:
    print("No GPU found. Using CPU.")

# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Convert labels to integers
train_labels = train_labels.astype(int)

test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_labels = test_labels.astype(int)

# Preprocess text data
max_len = 100
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(train_texts)
train_sequences = tokenizer.texts_to_sequences(train_texts)
test_sequences = tokenizer.texts_to_sequences(test_texts)
train_padded = pad_sequences(train_sequences, maxlen=max_len, padding="post")
test_padded = pad_sequences(test_sequences, maxlen=max_len, padding="post")

# Create the model with BiGRU
model = Sequential([
    Embedding(5000, 128, input_length=max_len),
    Bidirectional(GRU(64, return_sequences=True)),
    Bidirectional(GRU(32)),
    Dense(4, activation="softmax")
])

# Compile the model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train the model
epochs = 5
batch_size = 32  # Add batch size
history = model.fit(
    train_padded, train_labels,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(test_padded, test_labels),
    verbose=1
)

# Make predictions
y_pred = model.predict(test_padded)
y_pred_classes = np.argmax(y_pred, axis=1)

# Print classification report
print(classification_report(test_labels, y_pred_classes))

# Plot confusion matrix
cm = confusion_matrix(test_labels, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 1/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 7s 27ms/step - accuracy: 0.3275 - loss: 1.3633 - val_accuracy: 0.4510 - val_loss: 1.
3075
Epoch 2/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.5775 - loss: 1.1264 - val_accuracy: 0.5100 - val_loss: 1.
2282
Epoch 3/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.7510 - loss: 0.6151 - val_accuracy: 0.5060 - val_loss: 1.
3638
Epoch 4/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.8704 - loss: 0.3772 - val_accuracy: 0.5090 - val_loss: 1.
6953
Epoch 5/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9151 - loss: 0.2492 - val_accuracy: 0.5240 - val_loss: 1.
7625
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step
              precision    recall  f1-score   support

           0       0.59      0.58      0.59       266
           1       0.54      0.56      0.55       285
           2       0.55      0.53      0.54       277
           3       0.36      0.36      0.36       172

    accuracy                           0.52      1000
   macro avg       0.51      0.51      0.51      1000
weighted avg       0.52      0.52      0.52      1000
```






```
CPU times: user 14.8 s, sys: 994 ms, total: 15.8 s
Wall time: 14 s
```

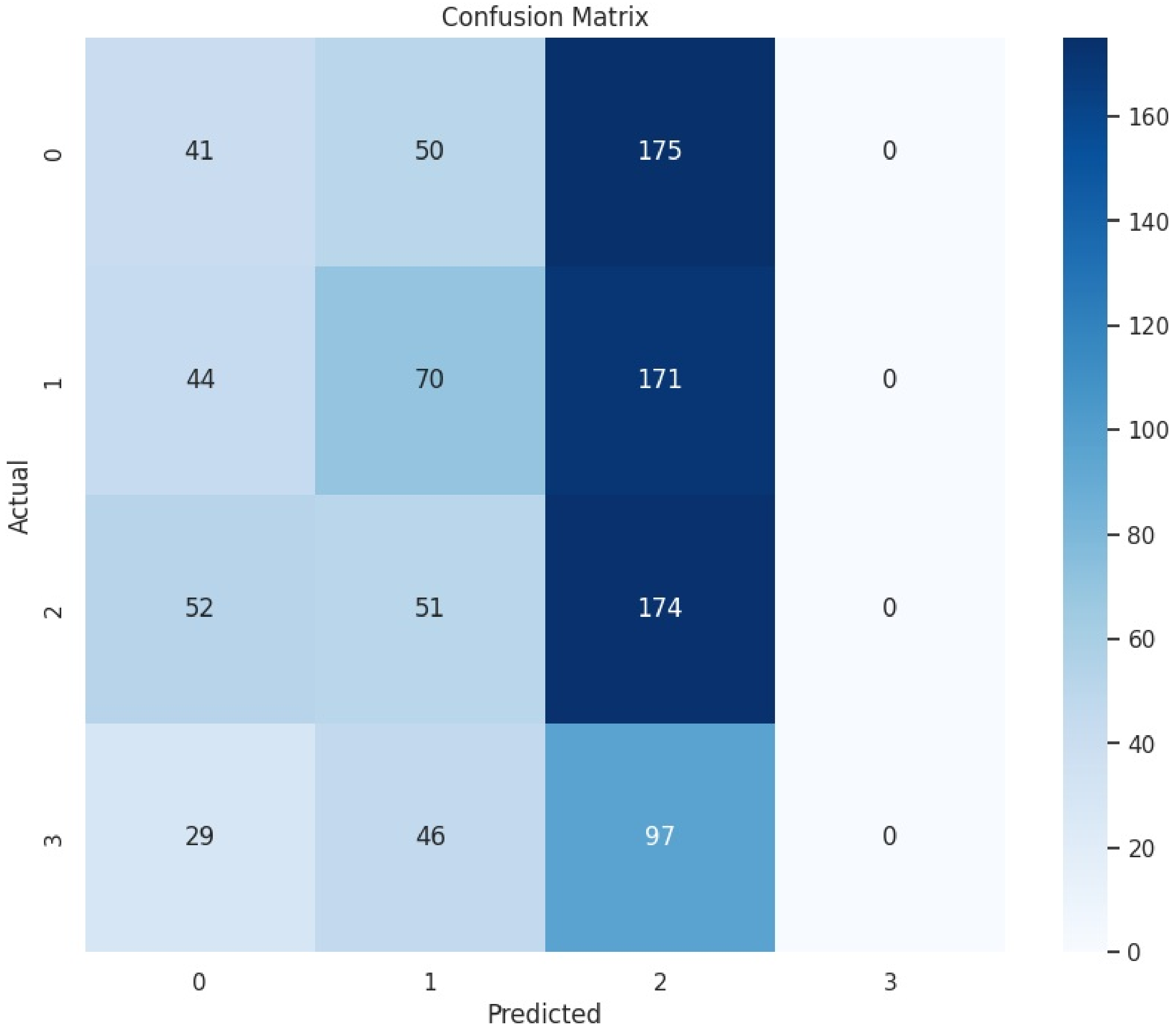
RNN

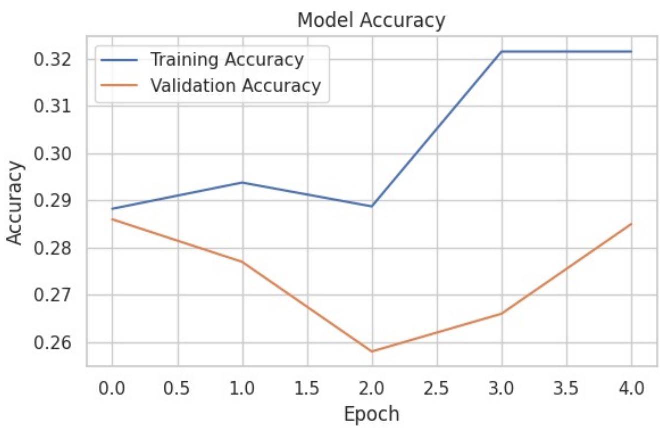
In [11]:

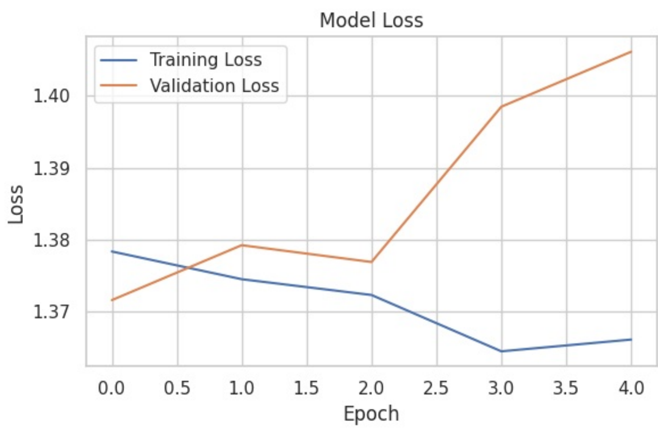
```
%%time

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure we're using GPU if available, otherwise use CPU
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)
else:
    print("No GPU found. Using CPU.")

# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Convert labels to integers
train_labels = train_labels.astype(int)

test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_labels = test_labels.astype(int)

# Preprocess text data
max_len = 100
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(train_texts)
train_sequences = tokenizer.texts_to_sequences(train_texts)
test_sequences = tokenizer.texts_to_sequences(test_texts)
train_padded = pad_sequences(train_sequences, maxlen=max_len, padding="post")
test_padded = pad_sequences(test_sequences, maxlen=max_len, padding="post")

# Create the model with RNN
model = Sequential([
    Embedding(5000, 128, input_length=max_len),
    SimpleRNN(64, return_sequences=True),
    SimpleRNN(32),
    Dense(4, activation="softmax")
])

# Compile the model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train the model
epochs = 5
```

```python
batch_size = 32
history = model.fit(
    train_padded, train_labels,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(test_padded, test_labels),
    verbose=1
)

# Make predictions
y_pred = model.predict(test_padded)
y_pred_classes = np.argmax(y_pred, axis=1)

# Print classification report
print(classification_report(test_labels, y_pred_classes))

# Plot confusion matrix
cm = confusion_matrix(test_labels, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 1/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 8s 67ms/step - accuracy: 0.2899 - loss: 1.3887 - val_accuracy: 0.2860 - val_loss: 1.
3716
Epoch 2/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.2835 - loss: 1.3795 - val_accuracy: 0.2770 - val_loss: 1.
3792
Epoch 3/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.2826 - loss: 1.3699 - val_accuracy: 0.2580 - val_loss: 1.
3769
Epoch 4/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.3448 - loss: 1.3540 - val_accuracy: 0.2660 - val_loss: 1.
3985
Epoch 5/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.3265 - loss: 1.3658 - val_accuracy: 0.2850 - val_loss: 1.
4062
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step
              precision    recall  f1-score   support

           0       0.25      0.15      0.19       266
           1       0.32      0.25      0.28       285
           2       0.28      0.63      0.39       277
           3       0.00      0.00      0.00       172

    accuracy                           0.28      1000
   macro avg       0.21      0.26      0.21      1000
weighted avg       0.24      0.28      0.24      1000
```








```
CPU times: user 14.7 s, sys: 740 ms, total: 15.4 s
Wall time: 15.3 s
```

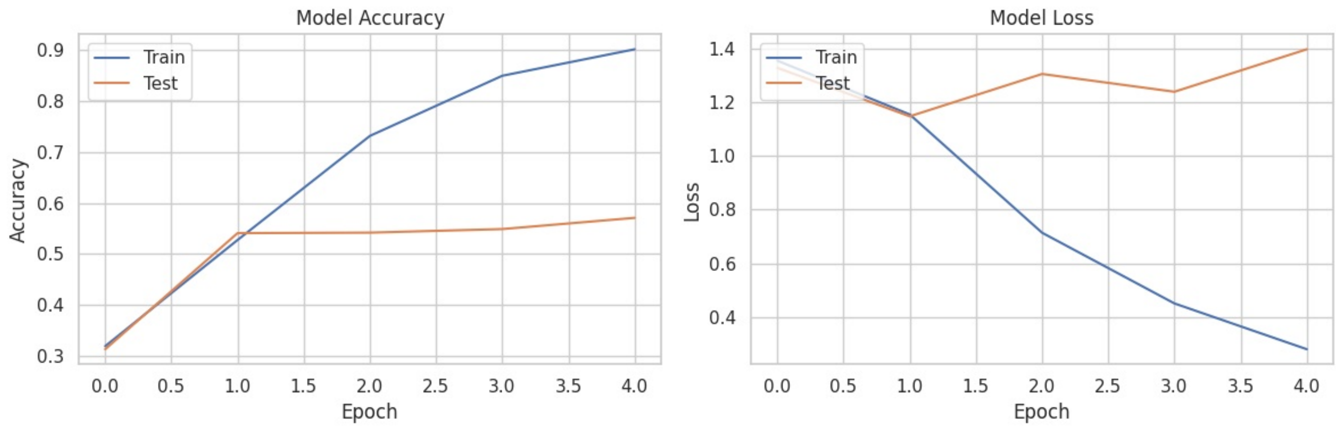
BiLSTM

In [12]:
```
%%time
```

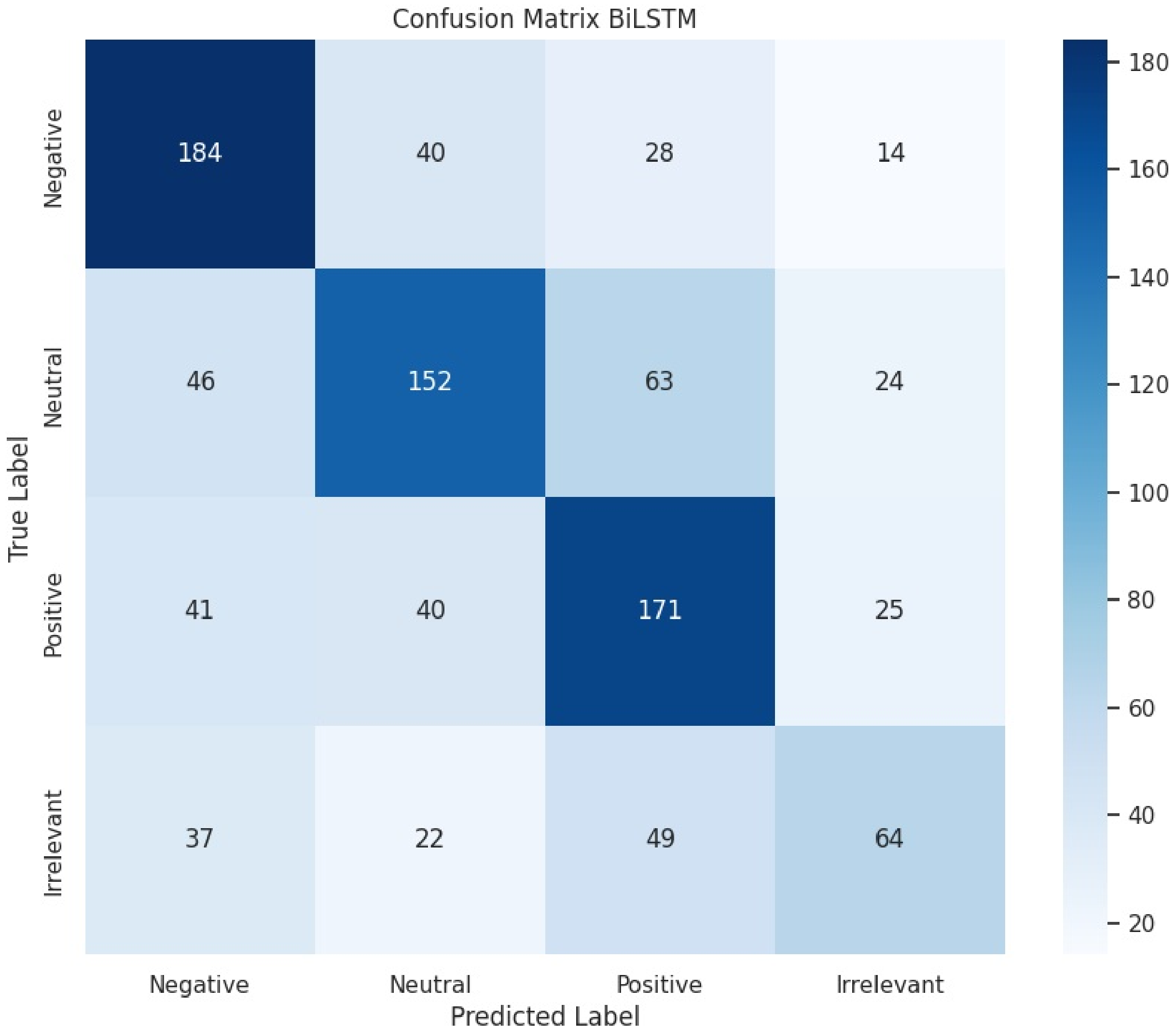
```python
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Convert labels to integers (assuming numerical mapping)
train_labels = train_labels.astype(int)  # Ensure labels are numerical

test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_labels = test_labels.astype(int)  # Ensure labels are numerical

# Preprocess text data
max_len = 100  # Adjust max sequence length as needed
tokenizer = Tokenizer(num_words=5000)  # Adjust vocabulary size as needed
tokenizer.fit_on_texts(train_texts)
train_sequences = tokenizer.texts_to_sequences(train_texts)
test_sequences = tokenizer.texts_to_sequences(test_texts)
train_padded = pad_sequences(train_sequences, maxlen=max_len, padding="post")
test_padded = pad_sequences(test_sequences, maxlen=max_len, padding="post")

# Create the model with BiLSTM
model = Sequential()
model.add(Embedding(5000, 128, input_length=max_len))  # Embedding layer
model.add(Bidirectional(LSTM(64)))  # Bidirectional LSTM layer with 64 units
model.add(Dense(4, activation="softmax"))  # Output layer with softmax activation

# Compile the model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train the model
epochs = 5  # Adjust number of epochs as needed
history = model.fit(train_padded, train_labels, epochs=epochs, validation_data=(test_padded, test_labels))

# Make predictions
y_pred = model.predict(test_padded)
y_pred_classes = np.argmax(y_pred, axis=1)


# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

plt.tight_layout()
plt.show()


# Generate confusion matrix
cm = confusion_matrix(test_labels, y_pred_classes)

# Generate classification report
class_names = ["Negative", "Neutral", "Positive", "Irrelevant"]  # Adjust if needed
cr = classification_report(test_labels, y_pred_classes, target_names=class_names)
print("Classification Report BiLSTM:")
print(cr)


# Make predictions
y_pred = model.predict(test_padded)
y_pred_classes = np.argmax(y_pred, axis=1)

# Define class names
class_names = ["Negative", "Neutral", "Positive", "Irrelevant"]
```

```
# Generate confusion matrix
cm = confusion_matrix(test_labels, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix BiLSTM')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()


# Evaluate the model (on validation data)
loss, accuracy = model.evaluate(test_padded, test_labels)
print("Test Accuracy BiLSTM:", accuracy)
```

```
Epoch 1/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.2796 - loss: 1.3711 - val_accuracy: 0.3130 - val_loss: 1.
3292
Epoch 2/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.4823 - loss: 1.2285 - val_accuracy: 0.5410 - val_loss: 1.
1484
Epoch 3/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.7137 - loss: 0.7426 - val_accuracy: 0.5420 - val_loss: 1.
3064
Epoch 4/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.8463 - loss: 0.4757 - val_accuracy: 0.5490 - val_loss: 1.
2401
Epoch 5/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9061 - loss: 0.2812 - val_accuracy: 0.5710 - val_loss: 1.
3985
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
```




```
Classification Report BiLSTM:
              precision    recall  f1-score   support

    Negative       0.60      0.69      0.64       266
     Neutral       0.60      0.53      0.56       285
    Positive       0.55      0.62      0.58       277
  Irrelevant       0.50      0.37      0.43       172

    accuracy                           0.57      1000
   macro avg       0.56      0.55      0.55      1000
weighted avg       0.57      0.57      0.57      1000

32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```




```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5755 - loss: 1.3273
Test Accuracy BiLSTM: 0.5709999799728394
CPU times: user 10.2 s, sys: 846 ms, total: 11 s
Wall time: 9.31 s
```

LSTM

In [13]:
```
%%time

import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
```

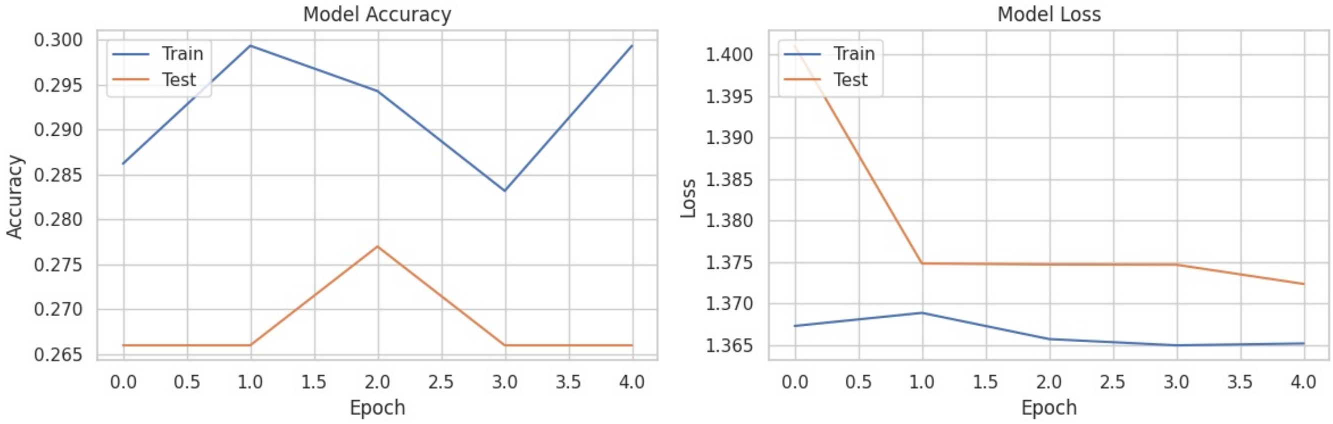
```python
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Convert labels to integers (assuming numerical mapping)
train_labels = train_labels.astype(int)  # Ensure labels are numerical

test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_labels = test_labels.astype(int)  # Ensure labels are numerical

# Preprocess text data
max_len = 100  # Adjust max sequence length as needed
tokenizer = Tokenizer(num_words=5000)  # Adjust vocabulary size as needed
tokenizer.fit_on_texts(train_texts)
train_sequences = tokenizer.texts_to_sequences(train_texts)
test_sequences = tokenizer.texts_to_sequences(test_texts)
train_padded = pad_sequences(train_sequences, maxlen=max_len, padding="post")
test_padded = pad_sequences(test_sequences, maxlen=max_len, padding="post")

# Create the model
model = Sequential()
model.add(Embedding(5000, 128, input_length=max_len))  # Embedding layer
model.add(LSTM(64))  # LSTM layer with 64 units
model.add(Dense(4, activation="softmax"))  # Output layer with softmax activation

# Compile the model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train the model
epochs = 5  # Adjust number of epochs as needed
history = model.fit(train_padded, train_labels, epochs=epochs, validation_data=(test_padded, test_labels))

# Make predictions
y_pred = model.predict(test_padded)
y_pred_classes = np.argmax(y_pred, axis=1)
```

```
Epoch 1/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.2687 - loss: 1.3725 - val_accuracy: 0.2660 - val_loss: 1.
4010
Epoch 2/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.2906 - loss: 1.3779 - val_accuracy: 0.2660 - val_loss: 1.
3748
Epoch 3/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.3067 - loss: 1.3562 - val_accuracy: 0.2770 - val_loss: 1.3
747
Epoch 4/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.2772 - loss: 1.3701 - val_accuracy: 0.2660 - val_loss: 1.3
747
Epoch 5/5
62/62 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3090 - loss: 1.3614 - val_accuracy: 0.2660 - val_loss: 1.3
724
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
CPU times: user 5.56 s, sys: 292 ms, total: 5.85 s
Wall time: 5.07 s
```

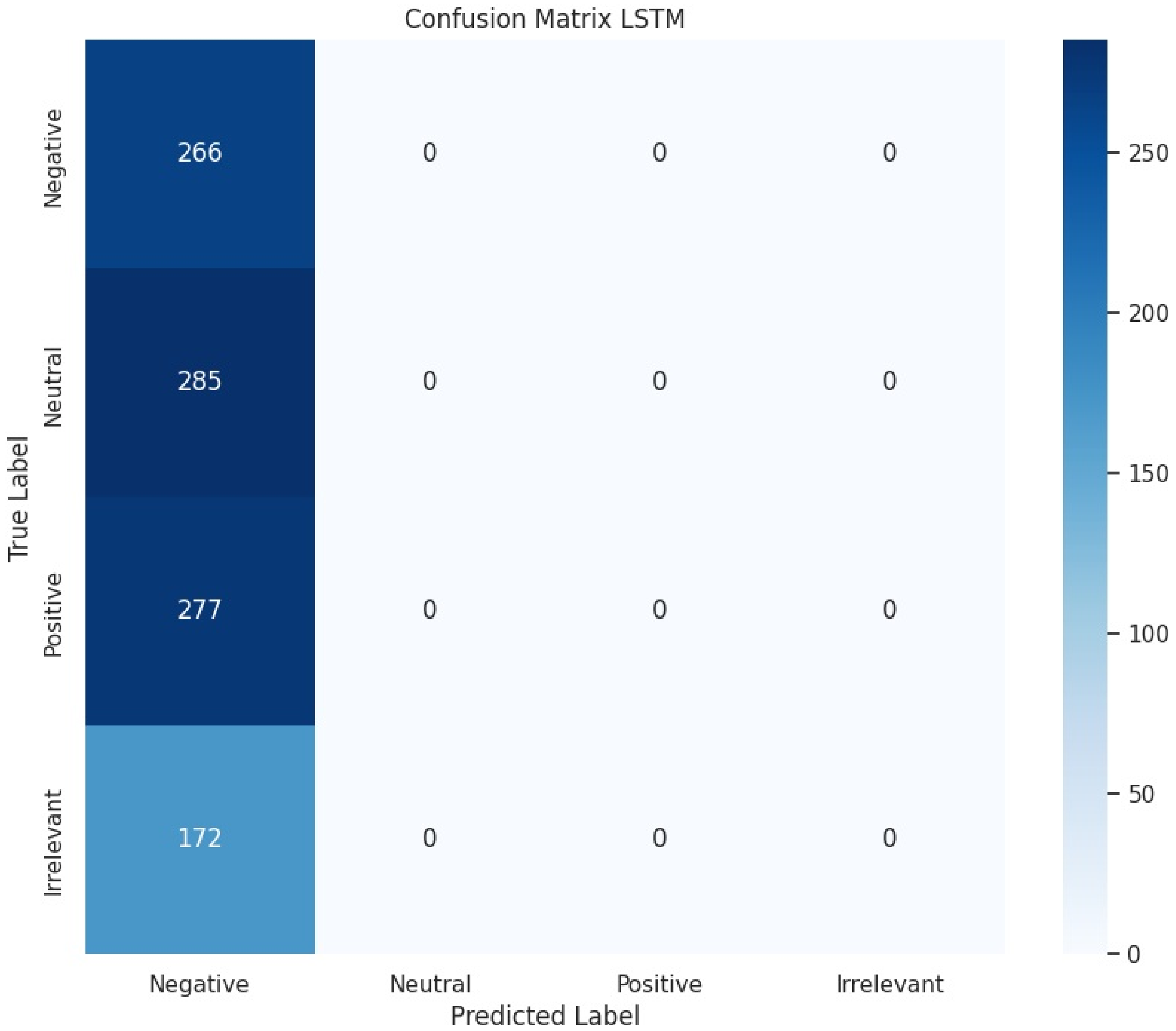
In [14]:

```python
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

plt.tight_layout()
plt.show()


# Make predictions
y_pred = model.predict(test_padded)
y_pred_classes = np.argmax(y_pred, axis=1)

# Define class names
```

```
class_names = ["Negative", "Neutral", "Positive", "Irrelevant"]

# Generate confusion matrix
cm = confusion_matrix(test_labels, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix LSTM')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()

# Evaluate the model (on validation data)
loss, accuracy = model.evaluate(test_padded, test_labels)
print("Test Accuracy LSTM:", accuracy)
```




```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```




```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.2743 - loss: 1.3666
Test Accuracy LSTM: 0.26600000262260437
```

In [15]:

```
# Generate confusion matrix
cm = confusion_matrix(test_labels, y_pred_classes)

# Generate classification report
class_names = ["Negative", "Neutral", "Positive", "Irrelevant"]  # Adjust if needed
```

```python
cr = classification_report(test_labels, y_pred_classes, target_names=class_names)
print("Classification Report:")
print(cr)
```

```
Classification Report:
              precision    recall  f1-score   support

    Negative       0.27      1.00      0.42       266
     Neutral       0.00      0.00      0.00       285
    Positive       0.00      0.00      0.00       277
  Irrelevant       0.00      0.00      0.00       172

    accuracy                           0.27      1000
   macro avg       0.07      0.25      0.11      1000
weighted avg       0.07      0.27      0.11      1000
```

https://www.kaggle.com/code/pythonafroz/neural-network-based-methods-nlp

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js