```
!pip install emoji
```

```
Collecting emoji
  Downloading emoji-2.14.1-py3-none-any.whl.metadata (5.7 kB)
Downloading emoji-2.14.1-py3-none-any.whl (590 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 590.6/590.6 kB 17.6 MB/s eta 0:
Installing collected packages: emoji
Successfully installed emoji-2.14.1
```

```
import random
import numpy as np
import tensorflow as tf
import os
from tensorflow.keras.datasets import imdb
import pandas as pd
import matplotlib.pyplot as plt
import re
import emoji
from tqdm import tqdm
from sklearn.metrics import classification_report
import nltk
from nltk import pos_tag
from nltk.corpus import wordnet
from nltk.tokenize import TreebankWordTokenizer
from nltk.stem import WordNetLemmatizer
from tqdm import tqdm
from sklearn.model_selection import train_test_split
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, Dropout, Leaky
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, LeakyReLU
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import  GRU
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import optuna
from tensorflow.keras.optimizers import Adam


SEED = 42

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger_eng')
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)
os.environ['PYTHONHASHSEED'] = str(SEED)
os.environ['TF_DETERMINISTIC_OPS'] = '1'
```

```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)

word_index = imdb.get_word_index()
reverse_word_index = {value: key for key, value in word_index.items()}

def decode_review(encoded_review):
    return ' '.join([reverse_word_index.get(i - 3, '?') for i in encoded_reviev


train_reviews = [decode_review(x) for x in x_train]
test_reviews = [decode_review(x) for x in x_test]


df = pd.DataFrame({
    "review": train_reviews + test_reviews,
    "label": list(y_train) + list(y_test)
})


df.head()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-da
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-da
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

	review	label
0	? this film was just brilliant casting locatio...	1
1	? big hair big boobs bad music and a giant saf...	0
2	? this has to be one of the worst films of the...	0
3	? the ? ? at storytelling the traditional sort...	1
4	? worst mistake of my life br br i picked this...	0

1-Text EDA

```
df.head()
```

	review	label
0	? this film was just brilliant casting locatio...	1
1	? big hair big boobs bad music and a giant saf...	0
2	? this has to be one of the worst films of the...	0
3	? the ? ? at storytelling the traditional sort...	1
4	? worst mistake of my life br br i picked this...	0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   review  50000 non-null  object
 1   label   50000 non-null  int64
dtypes: int64(1), object(1)
memory usage: 781.4+ KB
```

```
df['label'].value_counts()
```

label	count
1	25000
0	25000

dtype: int64

```
df['review_length'] = df['review'].apply(lambda x: len(x.split()))
df['review_length'].describe()
```

	review_length
count	50000.000000
mean	234.755400
std	172.907439
min	7.000000
25%	129.000000
50%	176.000000
75%	285.000000
max	2494.000000

dtype: float64

```
df.isnull().sum()
```

	0
review	0
label	0
review_length	0

dtype: int64

```
df.sample(3)['review'].values
```

```
array(["? everyone should totally see this movie it's freaking scary but
doesn't resort to lame jump out at you just to surprise you and pass it
off as scary things it really is great see this freaking awesome movie the
director is stanley kubrick easily the greatest director who ever lived
every single one of his movies are masterpieces including this one the
shining is about this family that goes to a hotel in the ? ? as ? for the
winters and get ? in well the house is haunted the kid is psychic the
husband is easily ? by evil haunted ? and well hilarity ensues not really
it becomes this gripping thriller where stuff gets thrown at the viewer
from all different directions and it gets scary not just the classic
here's johnny scene it's memorable but can't speak for the whole movies
it's one of those things where words don't explain it adequately and you
just gotta see it so go on netflix and get it ? ?",
       "? the prey has an interesting history unless you remember the ads
```

```
for it in ? in june of 1984 you might have caught it on the movie channel
back in summer 85 but little else is remembered the plot is your basic
killer in the woods again but ironically this was filmed before friday the
13th the prey was actually shot sometime in 1978 according to one of the
actors in an interview years later but released for about a week at some
drive ins yes jim drive in showed this in june of ? but it has a dated
look to it maybe they released it so later on to cash in on all the other
terror films the market was ? with by 1984 now on the story it has some
kind of back story a forest fire back in the 1940's leaves a lot of ?
burned to death but one of their children survive our monster so flash
forward to present day which would be 1978 we have an older middle age
couple camping only to be ? by the monster the tag line for this picture
claims its not human and its got an axe but an axe was only used in these
first two killings now we have a bunch of teenagers who look like they in
their mid ? camping we all know they are the prey and the monster knocks
them of one by one for an 80 minute movie it seems longer we also have a
lot of ? footage to fill in ? for the 80 mins overall for being out into
an 80 's horror movie it looks way more 70's than ever hey the prey had
potential to be a good horror killer in the woods movie but falls a little
short it does however feature a pretty scary cool looking monster at the
end and we have to wait till the last 2 minutes to see him side note the
monster has gone on to star in the ? family movies in the 1990's",
       "? he pulled the guys guts out his butt that's a spoof right no one
really writes that it just happens like ? gone horribly wrong i think any
way this movie must be a spoof because who would say they wrote that
script otherwise can anyone imagine the entire cast sitting around as the
director and writers go over the ? br br director says next our ? villain
uses his 24 inch ? to ? our token creepy neighbor get this he is going to
pull the guts out his ? br br brilliant the entire cast ? br br no way can
that happen nobody writes that stupid gotta be a spoof br br i loved the
part where the skinny ? gal beats the ? freak to death with the cast iron
? she finds on the floor of the cave i wasn't sure the ? cannibal types
bothered to cook much maybe that explains why the ? was lying on the floor
in the dark at just the right time to kill the ? hulk seems ironic that
after the freaky guy had ? martial arts expert porn queens and a couple
out doors type ? he falls so easily to the ? pan of a skinny ? girl next
door br br what the heck is that richard ? guy doing in this did he fire
his agent or something br br can anyone explain the ending to me please
because i didn't get it either i can't quite figure why the nice hero girl
wanted to kill the funny lady who was making her some tea never mind i
don't want to know"],
      dtype=object)
```

```
df.shape
```

```
(50000, 3)
```

```
df.columns
```

```
Index(['review', 'label', 'review_length'], dtype='object')
```

```
word_index = imdb.get_word_index()
reverse_word_index = {value: key for key, value in word_index.items()}


def decode_review(encoded_review):
    return ' '.join([reverse_word_index.get(i - 3, '?') for i in encoded_reviev


train_reviews = [decode_review(x) for x in x_train]
test_reviews = [decode_review(x) for x in x_test]


df = pd.DataFrame({
    "review": train_reviews + test_reviews,
    "label": list(y_train) + list(y_test)
})


label_map = {0: "Negative", 1: "Positive"}
df['label_name'] = df['label'].map(label_map)
sentiment_counts = df['label_name'].value_counts()

print(" Sentiment Distribution:")
print(sentiment_counts)


df['review_length'] = df['review'].apply(lambda x: len(x.split()))
length_stats = df['review_length'].describe()

print("\n Review Length Statistics (in words):")
print(length_stats)
```

```
 Sentiment Distribution:
label_name
Positive    25000
Negative    25000
Name: count, dtype: int64

 Review Length Statistics (in words):
count    50000.000000
mean       234.755400
std        172.907439
min          7.000000
25%        129.000000
50%        176.000000
75%        285.000000
max       2494.000000
Name: review_length, dtype: float64
```

Best Practice to remove the text les then 10

```
df = df[df['review_length'] >= 10]
```

```
df.shape
```

```
(49997, 4)
```

2-Text Preprocessing

cleaning

```
url_pattern = re.compile(r"http\S+|www\S+|https\S+")
mention_pattern = re.compile(r'@\w+')
hashtag_pattern = re.compile(r'#\w+')
brackets_pattern = re.compile(r'[\(\[\{\<][^()\[\]\{\}<>]*[\)\]\}\>]')
html_pattern = re.compile(r'<.*?>')
special_chars_pattern = re.compile(r'[^a-zA-Z\s]')
digits_pattern = re.compile(r'\d+')
repetition_pattern = re.compile(r'(.)\1{2,}', re.DOTALL)
whitespace_pattern = re.compile(r'\s+')

emoji_word_map = {
    "smiling_face_with_heart_eyes": "love",
    "face_with_tears_of_joy": "funny",
    "red_heart": "love",
    "crying_face": "sad",
    "face_with_symbols_on_mouth": "angry",
    "fire": "hot",
    "clapping_hands": "applause",
    "thumbs_up": "approve",
    "thumbs_down": "disapprove",
    "star_struck": "amazed",
    "grinning_face": "happy",
    "thinking_face": "thinking",
    "broken_heart": "heartbroken",
    "hundred_points": "perfect"
}

def replace_emoji_with_words(text):
```

```
    text = emoji.demojize(text, delimiters=(" ", " "))
    for code, word in emoji_word_map.items():
        text = text.replace(code, word)
    return text

def clean_base_text(text):
    text = str(text).lower()
    text = replace_emoji_with_words(text)
    text = url_pattern.sub(' ', text)
    text = mention_pattern.sub(' ', text)
    text = hashtag_pattern.sub(' ', text)
    text = brackets_pattern.sub(' ', text)
    text = html_pattern.sub(' ', text)
    text = special_chars_pattern.sub(' ', text)
    text = digits_pattern.sub(' ', text)
    text = whitespace_pattern.sub(' ', text).strip()
    return text

def remove_excessive_repetition(text):
    return repetition_pattern.sub(r'\1\1', text)

def remove_useless_short_words(text):
    words = text.split()
    return ' '.join(word for word in words if len(word) > 2)

def full_cleaning_pipeline(text):
    text = clean_base_text(text)
    text = remove_excessive_repetition(text)
    text = remove_useless_short_words(text)
    return text

tqdm.pandas()
df['cleaned_text'] = df['review'].progress_apply(full_cleaning_pipeline)


df[['review', 'cleaned_text']].sample(5)
```

```
100%|██████████| 49997/49997 [00:50<00:00, 989.66it/s]
```

	review	cleaned_text
5480	? i don't know why i keep doing this to myself...	don know why keep doing this myself keep defen...
995	? i think this still is the best routine there...	think this still the best routine there are so...
44278	? i decided to hire out this movie along with ...	decided hire out this movie along with few oth...
[illegible]	? i couldn't believe the eye candy from	couldn believe the eye candy from start

Lemmatization

```
lemmatizer = WordNetLemmatizer()
tokenizer = TreebankWordTokenizer()


def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN


def conservative_lemmatize(text):
    tokens = tokenizer.tokenize(text)
    tagged_tokens = pos_tag(tokens)
    lemmatized_tokens = [
        lemmatizer.lemmatize(token, get_wordnet_pos(tag))
        for token, tag in tagged_tokens
    ]
    return ' '.join(lemmatized_tokens)


tqdm.pandas()
df['lemmatized_text'] = df['cleaned_text'].progress_apply(conservative_lemmatiz
df[['cleaned_text', 'lemmatized_text']].sample(5)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
100%|██████████| 49997/49997 [05:05<00:00, 163.46it/s]
```

	cleaned_text	lemmatized_text
8512	first this movie seems bad that almost fell th...	first this movie seem bad that almost fell the...
5097	the movie starts off setting where not surpris...	the movie start off set where not surprisingly...
28531	this thriller has many twists and turns had th...	this thriller have many twist and turn have th...
[illegible]	know know plan from outer space the	know know plan from out space the bad

Text Vectorization – Tokenization and Padding

```
texts = df['lemmatized_text'].values
labels = df['label'].values



X_train_texts, X_test_texts, y_train, y_test = train_test_split(
    texts, labels, test_size=0.2, random_state=42, stratify=labels
)

tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train_texts)

X_train_seq = tokenizer.texts_to_sequences(X_train_texts)
X_test_seq = tokenizer.texts_to_sequences(X_test_texts)

MAXLEN = 300
X_train_pad = pad_sequences(X_train_seq, maxlen=MAXLEN, padding='post', truncat
X_test_pad = pad_sequences(X_test_seq, maxlen=MAXLEN, padding='post', truncatir

print(" Vocabulary size:", len(tokenizer.word_index))
print(" X_train shape:", X_train_pad.shape)
print(" X_test shape:", X_test_pad.shape)
print(" Example padded sequence:", X_train_pad[0])
```

```
Vocabulary size: 9179
X_train shape: (39997, 300)
X_test shape: (10000, 300)
Example padded sequence: [   4  120    2    8 1588    3  188 4688   11  77
  157  411   41  269  243   56   12 1143    4  358  246   30    2    8
    4 2519 1260    2  370    4 2201  308   93  364    2  374    4   64
  364   83  503 2410    3 3558  104  647  223  112    4  991  120    5
 1080  419   82 1532  232   39   43   12   66   22   67  226  301   84
  194    5    9   19   39 1389    3  735  767   70  246  364    5    9
    7   71   11    2   48  678   78   30    5    8    4    4  450   72
   87  316 2824   13  626   17 1070  302    2  164    6   82   19   11
  191   12   24  140 1326   30   17    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0]
```

Build and Train Simple RNN Model

```
model_rnn = Sequential([
    Embedding(input_dim=10000, output_dim=64),

    Bidirectional(SimpleRNN(units=64, return_sequences=True)),
    Dropout(0.3),

    Bidirectional(SimpleRNN(units=32)),
    Dropout(0.3),

    Dense(32),
    LeakyReLU(alpha=0.01),

    Dense(1, activation='sigmoid')
])

model_rnn.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

history_rnn = model_rnn.fit(
    X_train_pad, y_train,
    validation_data=(X_test_pad, y_test),
    epochs=60,
    batch_size=128,
    callbacks=[EarlyStopping(patience=5, restore_best_weights=True)],
    verbose=1
)
```

```
Epoch 1/60
/usr/local/lib/python3.11/dist-packages/keras/src/layers/activations/leaky_
  warnings.warn(
313/313 ━━━━━━━━━━━━━━━━━━━━ 193s 594ms/step - accuracy: 0.5175 - loss: 0.6
Epoch 2/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 193s 565ms/step - accuracy: 0.7435 - loss: 0.5
Epoch 3/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 195s 542ms/step - accuracy: 0.7161 - loss: 0.5
Epoch 4/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 173s 553ms/step - accuracy: 0.7779 - loss: 0.4
Epoch 5/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 214s 593ms/step - accuracy: 0.6188 - loss: 0.6
Epoch 6/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 214s 632ms/step - accuracy: 0.6730 - loss: 0.5
Epoch 7/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 220s 688ms/step - accuracy: 0.7350 - loss: 0.5
Epoch 8/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 172s 548ms/step - accuracy: 0.8301 - loss: 0.4
Epoch 9/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 203s 552ms/step - accuracy: 0.8580 - loss: 0.3
Epoch 10/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 203s 555ms/step - accuracy: 0.8498 - loss: 0.3
Epoch 11/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 199s 546ms/step - accuracy: 0.7366 - loss: 0.5
Epoch 12/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 199s 538ms/step - accuracy: 0.7349 - loss: 0.5
Epoch 13/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 171s 547ms/step - accuracy: 0.8355 - loss: 0.4
Epoch 14/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 199s 539ms/step - accuracy: 0.8318 - loss: 0.3
```

```python
y_pred_prob = model_rnn.predict(X_test_pad).flatten()
y_pred = (y_pred_prob > 0.5).astype("int32")
```

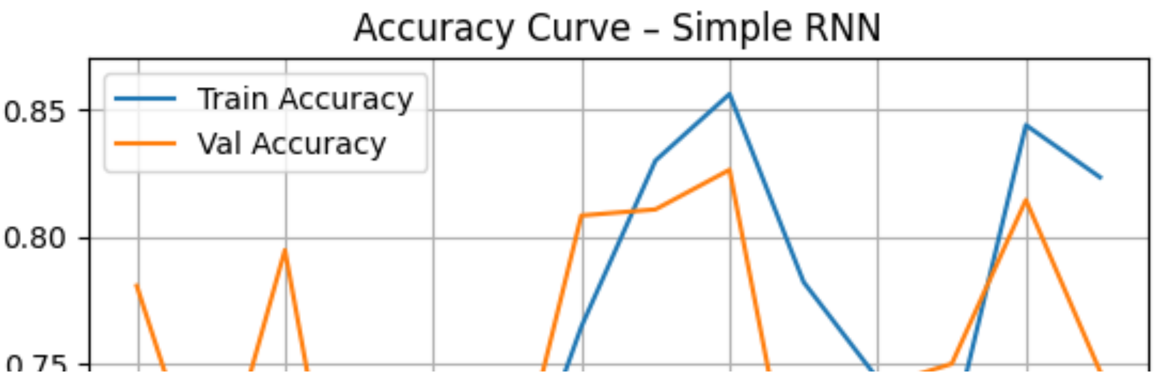
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 27s 85ms/step
```

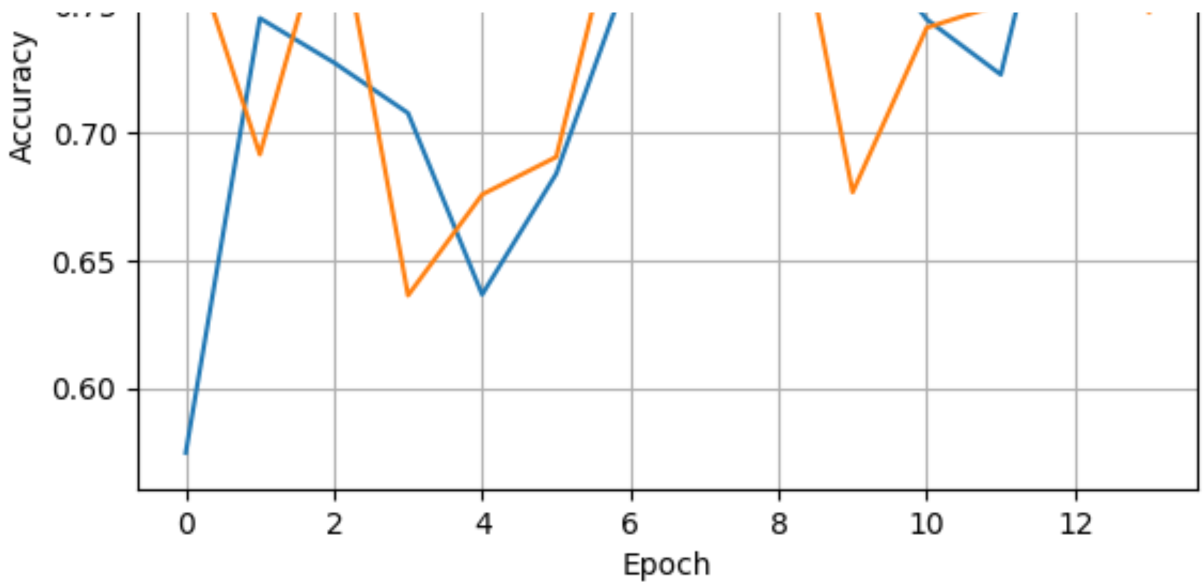
```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy:  {accuracy:.4f}")
print(f" Precision: {precision:.4f}")
print(f" Recall:    {recall:.4f}")
print(f" F1 Score:  {f1:.4f}")
```

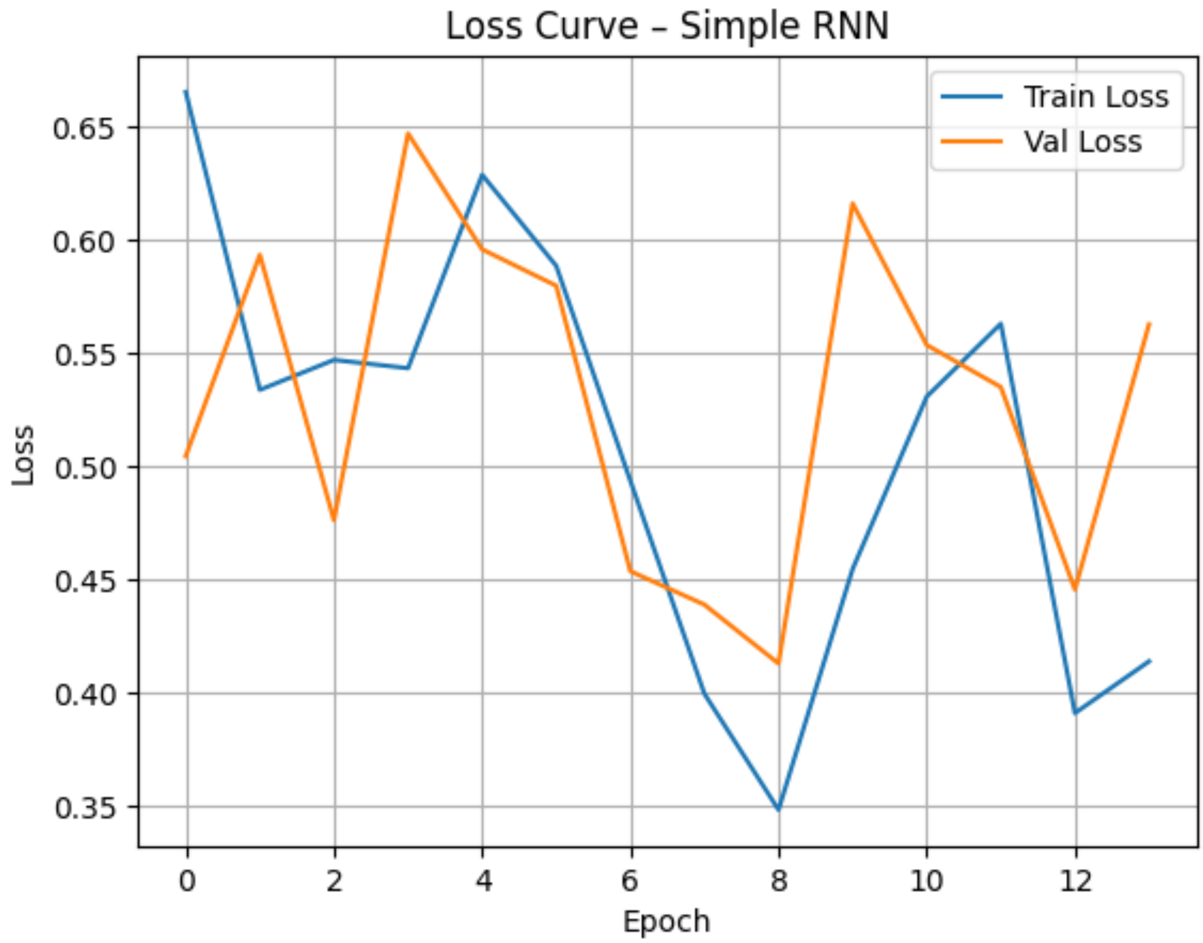
```
Accuracy:  0.8261
 Precision: 0.8227
 Recall:    0.8314
 F1 Score:  0.8270
```

Accuracy / Loss Curves

```
plt.plot(history_rnn.history['accuracy'], label='Train Accuracy')
plt.plot(history_rnn.history['val_accuracy'], label='Val Accuracy')
plt.title('Accuracy Curve – Simple RNN')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

plt.plot(history_rnn.history['loss'], label='Train Loss')
plt.plot(history_rnn.history['val_loss'], label='Val Loss')
plt.title('Loss Curve – Simple RNN')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



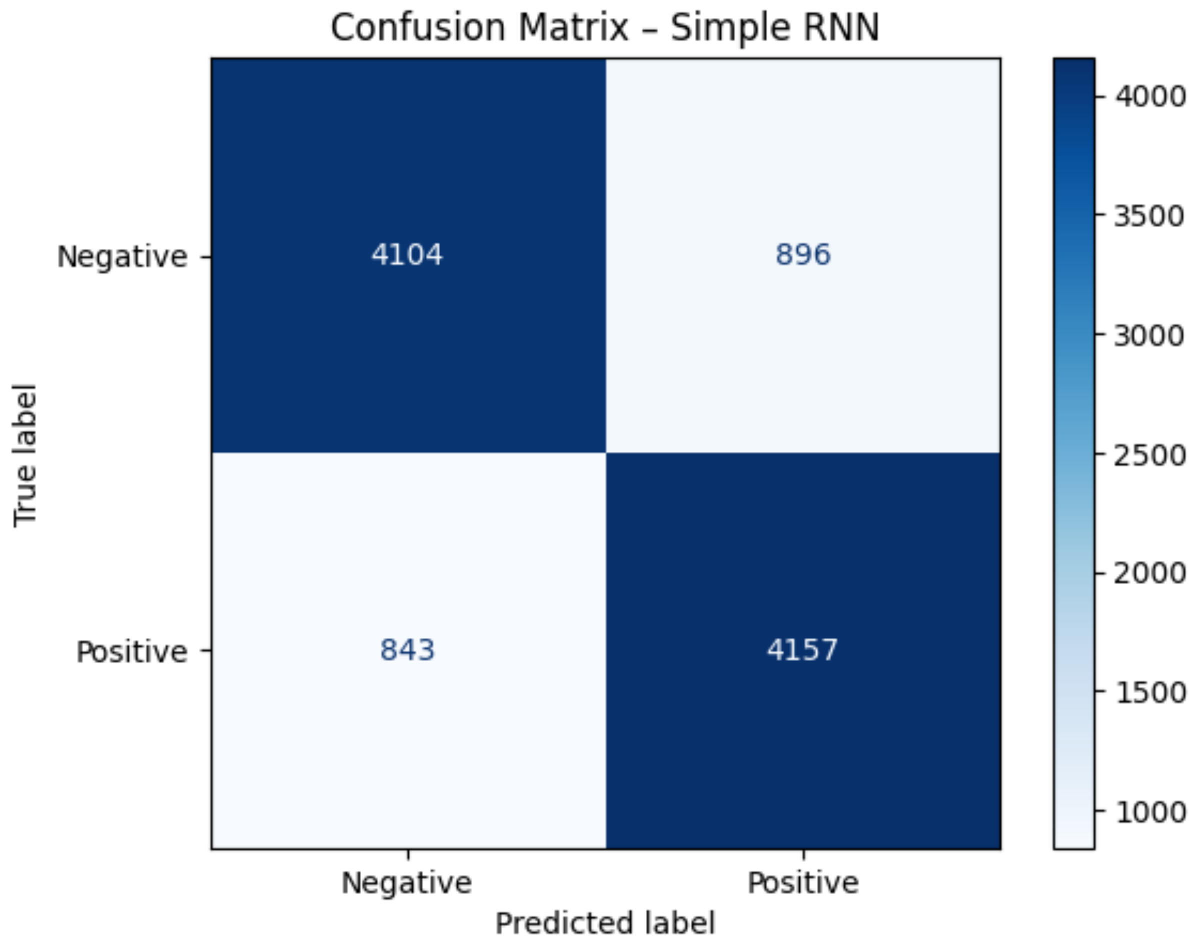




Confusion Matrix

```
y_pred_prob = model_rnn.predict(X_test_pad)
y_pred = (y_pred_prob > 0.5).astype("int32")

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Negative",
disp.plot(cmap='Blues')
plt.title("Confusion Matrix – Simple RNN")
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 22s 72ms/step
```



Classification Report

```
print(classification_report(y_test, y_pred, target_names=["Negative", "Positive
```

```
              precision    recall  f1-score   support

    Negative       0.83      0.82      0.83      5000
    Positive       0.82      0.83      0.83      5000

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.83     10000
weighted avg       0.83      0.83      0.83     10000
```

Qualitative Analysis

High-Confidence Correct Predictions

```
y_pred_prob = model_rnn.predict(X_test_pad).flatten()
y_pred = (y_pred_prob > 0.5).astype("int32")


correct_mask = (y_pred == y_test)
correct_probs = y_pred_prob[correct_mask]
correct_texts = X_test_texts[correct_mask]
correct_labels = y_test[correct_mask]


top_correct_indices = np.argsort(correct_probs)[-5:][::-1]


print(" Top High-Confidence Correct Predictions:\n")
for i in top_correct_indices:
    print(" Review:")
    print(correct_texts[i][:500])
    print(f"Actual Label: {correct_labels[i]}, Predicted Prob: {correct_probs[i
    print("-" * 60)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 63ms/step
 Top High-Confidence Correct Predictions:

 Review:
excellent blend music light comedy and drama with picture perfect performan
Actual Label: 1, Predicted Prob: 0.9768
------------------------------------------------------------
 Review:
excellent comedy star dudley moore support minnelli and good speaking john
Actual Label: 1, Predicted Prob: 0.9768
------------------------------------------------------------
 Review:
excellent story telling and cinematography poignant bite social commentary
Actual Label: 1, Predicted Prob: 0.9767
------------------------------------------------------------
 Review:
deathtrap give you twist every turn every single turn fact it big problem t
Actual Label: 1, Predicted Prob: 0.9767
------------------------------------------------------------
 Review:
refresh breath air when movie actually give you story line with begin middl
Actual Label: 1, Predicted Prob: 0.9766
------------------------------------------------------------
```

low-Confidence Predictions

```
wrong_preds = (y_test != y_pred.flatten())
wrong_reviews = X_test_texts[wrong_preds]

for i in range(3):
    print(" Review:")
    print(wrong_reviews[i])
    print(f"Actual: {y_test[wrong_preds][i]}, Predicted: {y_pred[wrong_preds][i
    print("-" * 60)
```

```
 Review:
have be pleasantly surprised sandra performance miss decide give murder num
Actual: 0, Predicted: 1
------------------------------------------------------------
 Review:
have never hear larry before but judge this effort into write and direct sh
Actual: 0, Predicted: 1
------------------------------------------------------------
 Review:
two star amanda plummer look like young version her father christopher plum
Actual: 0, Predicted: 1
------------------------------------------------------------
```

Long Short-Term Memory -LSTM-

```
model_lstm = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=300),
    LSTM(units=64, return_sequences=False),
    Dropout(0.4),
    Dense(32),
    LeakyReLU(alpha=0.01),
    Dense(1, activation='sigmoid')
])


model_lstm.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

```
history_lstm = model_lstm.fit(
    X_train_pad, y_train,
    validation_data=(X_test_pad, y_test),
    epochs=60,
    batch_size=128,
    callbacks=[EarlyStopping(patience=5, restore_best_weights=True)],
    verbose=1
)
```

```
Epoch 1/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 179s 559ms/step - accuracy: 0.4989 - loss: 0.6
Epoch 2/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 148s 472ms/step - accuracy: 0.5175 - loss: 0.6
Epoch 3/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 213s 507ms/step - accuracy: 0.5328 - loss: 0.6
Epoch 4/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 159s 508ms/step - accuracy: 0.5523 - loss: 0.6
Epoch 5/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 158s 505ms/step - accuracy: 0.5540 - loss: 0.6
Epoch 6/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 158s 506ms/step - accuracy: 0.5516 - loss: 0.6
Epoch 7/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 206s 517ms/step - accuracy: 0.5640 - loss: 0.6
Epoch 8/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 152s 486ms/step - accuracy: 0.7440 - loss: 0.5
Epoch 9/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 201s 483ms/step - accuracy: 0.8918 - loss: 0.2
Epoch 10/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 151s 483ms/step - accuracy: 0.9306 - loss: 0.1
Epoch 11/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 159s 508ms/step - accuracy: 0.9506 - loss: 0.1
Epoch 12/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 202s 508ms/step - accuracy: 0.9616 - loss: 0.1
Epoch 13/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 194s 484ms/step - accuracy: 0.9747 - loss: 0.0
Epoch 14/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 159s 509ms/step - accuracy: 0.9809 - loss: 0.0
```

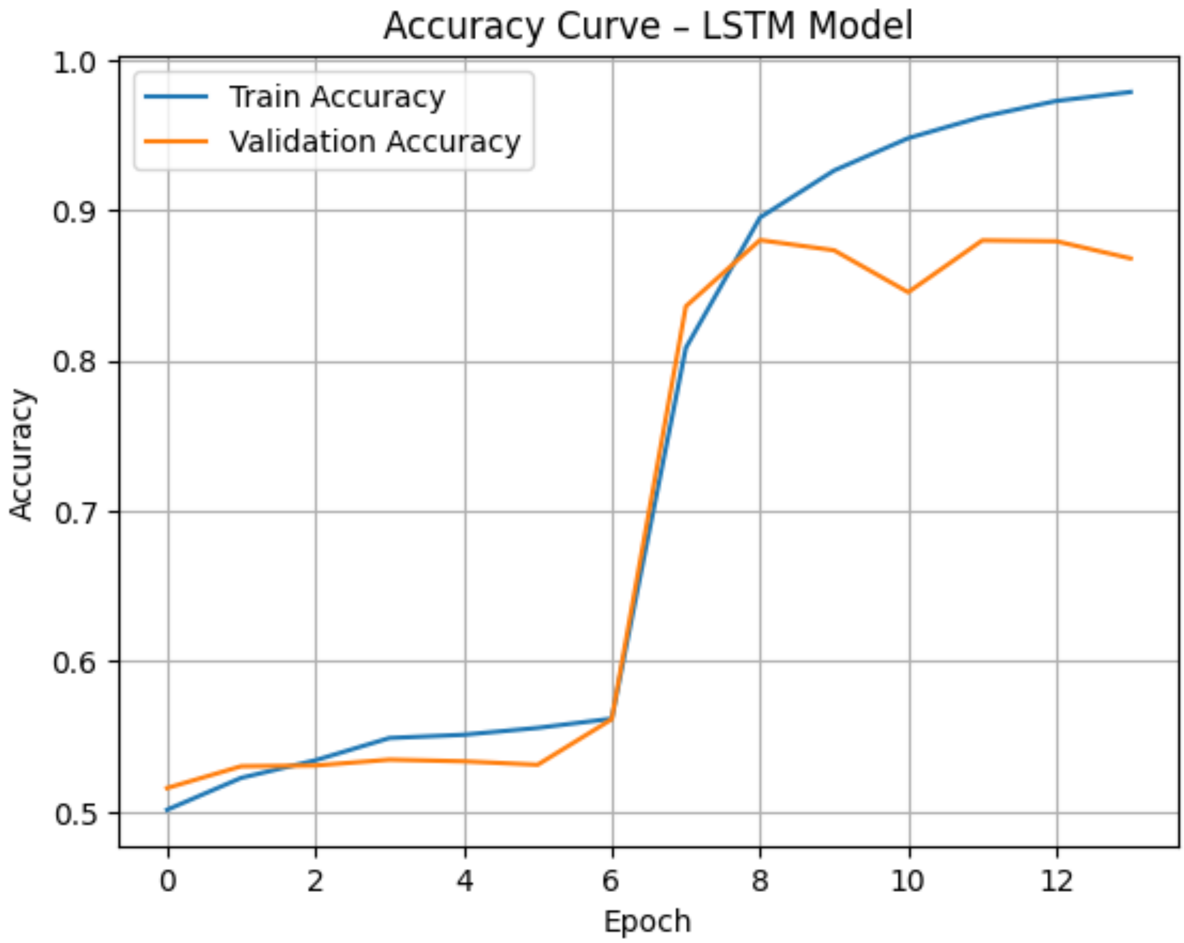
Evaluation of LSTM Model

```
y_pred_prob = model_lstm.predict(X_test_pad).flatten()
y_pred = (y_pred_prob > 0.5).astype("int32")

print(f" Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
print(f" Precision: {precision_score(y_test, y_pred):.4f}")
print(f" Recall:    {recall_score(y_test, y_pred):.4f}")
print(f" F1 Score:  {f1_score(y_test, y_pred):.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 24s 74ms/step
 Accuracy:  0.8804
 Precision: 0.8737
 Recall:    0.8894
 F1 Score:  0.8815
313/313 ━━━━━━━━━━━━━━━━━━━━ 12s 37ms/step
 Accuracy:  0.8804
 Precision: 0.8737
 Recall:    0.8894
 F1 Score:  0.8815
```

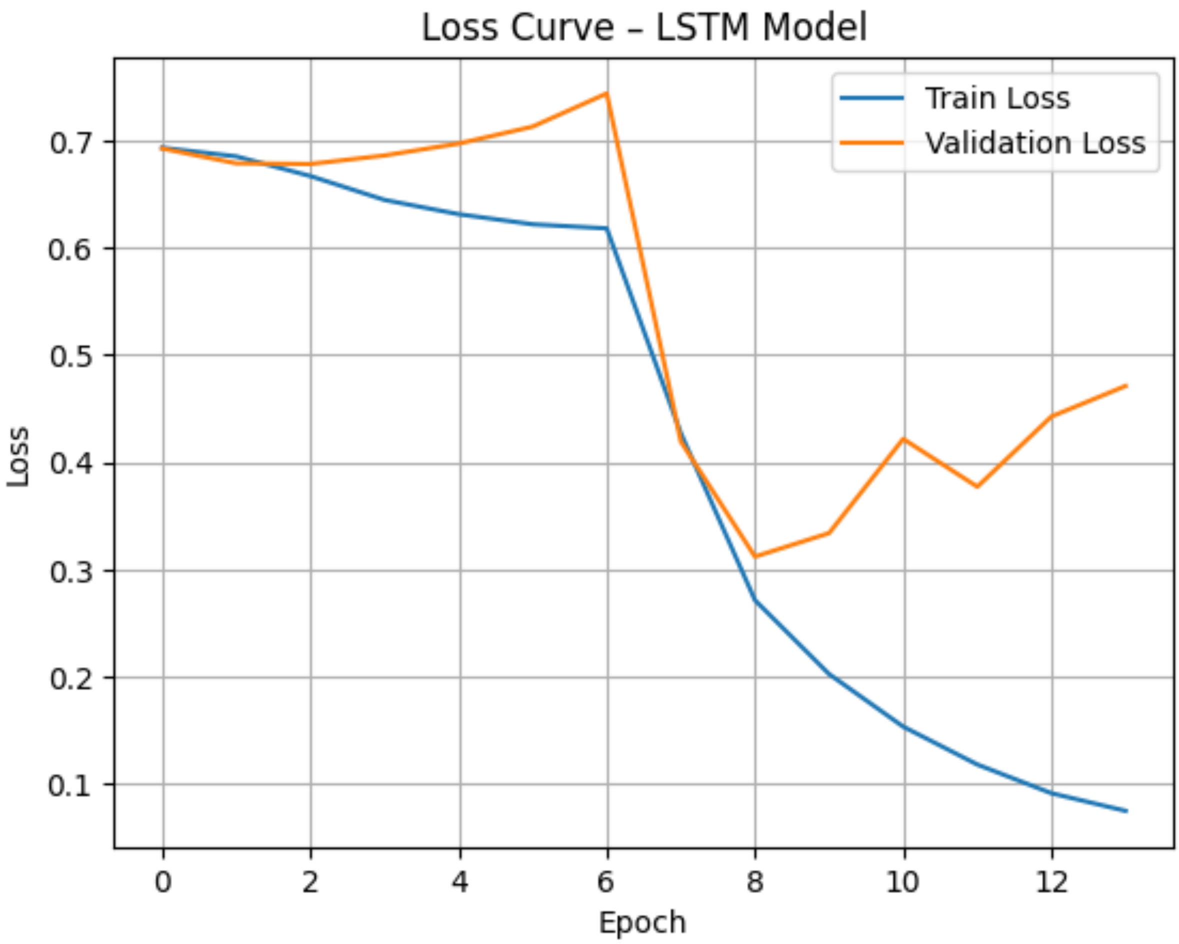
Plot Training & Validation Accuracy

```
plt.plot(history_lstm.history['accuracy'], label='Train Accuracy')
plt.plot(history_lstm.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Curve – LSTM Model')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



Plot Training & Validation Loss

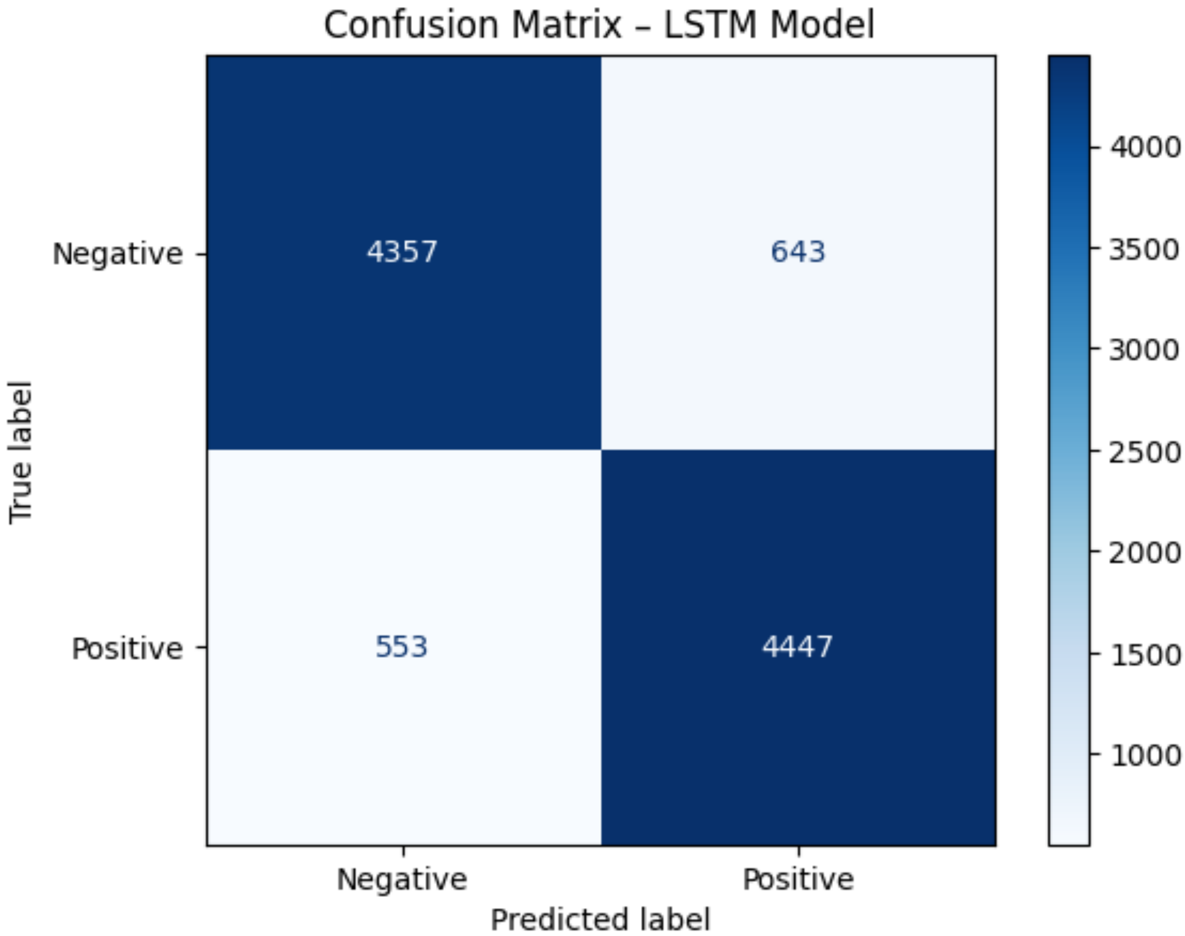
```
plt.plot(history_lstm.history['loss'], label='Train Loss')
plt.plot(history_lstm.history['val_loss'], label='Validation Loss')
plt.title('Loss Curve — LSTM Model')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



Confusion Matrix

```
y_pred_prob = model_lstm.predict(X_test_pad).flatten()
y_pred = (y_pred_prob > 0.5).astype("int32")

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Negative",
disp.plot(cmap='Blues')
plt.title("Confusion Matrix – LSTM Model")
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 11s 36ms/step
```



Classification Report

```
print(classification_report(y_test, y_pred, target_names=["Negative", "Positive
```

```
              precision    recall  f1-score   support

    Negative       0.89      0.87      0.88      5000
    Positive       0.87      0.89      0.88      5000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

High-Confidence Wrong Predictions (Qualitative Error Analysis)

```
wrong_mask = (y_pred != y_test)
wrong_conf = y_pred_prob[wrong_mask]
wrong_texts = X_test_texts[wrong_mask]

top_wrong = np.argsort(wrong_conf)[-3:][::-1]

print(" High-Confidence Wrong Predictions:\n")
for i in top_wrong:
    print("Review:")
    print(wrong_texts[i][:500]) # Use array indexing instead of .iloc
    print(f"Actual: {y_test[wrong_mask][i]}, Predicted Prob: {wrong_conf[i]:.4f
    print("-" * 60)
```

```
 High-Confidence Wrong Predictions:

Review:
firmly believe that the best oscar ceremony recent year be for two reason h
Actual: 0, Predicted Prob: 1.0000
------------------------------------------------------------
Review:
the story go something like this small town girl katie jessica simpson deci
Actual: 0, Predicted Prob: 1.0000
------------------------------------------------------------
Review:
saw this movie just now not when be release and best picture the year here
Actual: 0, Predicted Prob: 1.0000
------------------------------------------------------------
```

High-Confidence Correct Predictions

```
correct_mask = (y_pred == y_test)
correct_conf = y_pred_prob[correct_mask]
correct_texts = X_test_texts[correct_mask]

top_correct = np.argsort(correct_conf)[-3:][::-1]

print("High-Confidence Correct Predictions:\n")
for i in top_correct:
    print("Review:")
    print(correct_texts[i][:500]) # Access using array indexing
    print(f"Actual: {y_test[correct_mask][i]}, Predicted Prob: {correct_conf[i]
    print("-" * 60)
```

```
High-Confidence Correct Predictions:

Review:
the notorious bettie page gretchen mol taylor chris bauer jar harris sarah
Actual: 1, Predicted Prob: 1.0000
------------------------------------------------------------
Review:
one reason pixar have endure well and be successful that while their film r
Actual: 1, Predicted Prob: 1.0000
------------------------------------------------------------
Review:
be see the year the matrix with the release two sequel and computer game th
Actual: 1, Predicted Prob: 1.0000
------------------------------------------------------------
```

GRU Model

```
model_gru = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=300),
    GRU(units=64, return_sequences=False),
    Dropout(0.4),
    Dense(32),
    LeakyReLU(alpha=0.01),
```

```
    Dense(1, activation='sigmoid')
])

model_gru.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

history_gru = model_gru.fit(
    X_train_pad, y_train,
    validation_data=(X_test_pad, y_test),
    epochs=60,
    batch_size=128,
    callbacks=[EarlyStopping(patience=5, restore_best_weights=True)],
    verbose=1
)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/keras/src/layers/activations/leaky_
  warnings.warn(
Epoch 1/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 234s 723ms/step - accuracy: 0.5057 - loss: 0.6
Epoch 2/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 235s 639ms/step - accuracy: 0.5080 - loss: 0.6
Epoch 3/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 209s 668ms/step - accuracy: 0.5378 - loss: 0.6
Epoch 4/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 253s 639ms/step - accuracy: 0.7692 - loss: 0.4
Epoch 5/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 205s 654ms/step - accuracy: 0.9101 - loss: 0.2
Epoch 6/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 260s 649ms/step - accuracy: 0.9380 - loss: 0.1
Epoch 7/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 203s 648ms/step - accuracy: 0.9553 - loss: 0.1
Epoch 8/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 261s 647ms/step - accuracy: 0.9681 - loss: 0.1
Epoch 9/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 265s 659ms/step - accuracy: 0.9785 - loss: 0.0
Epoch 10/60
313/313 ━━━━━━━━━━━━━━━━━━━━ 260s 652ms/step - accuracy: 0.9836 - loss: 0.0
```

Generate Predictions

```
y_pred_prob = model_gru.predict(X_test_pad).flatten()
y_pred = (y_pred_prob > 0.5).astype("int32")
```

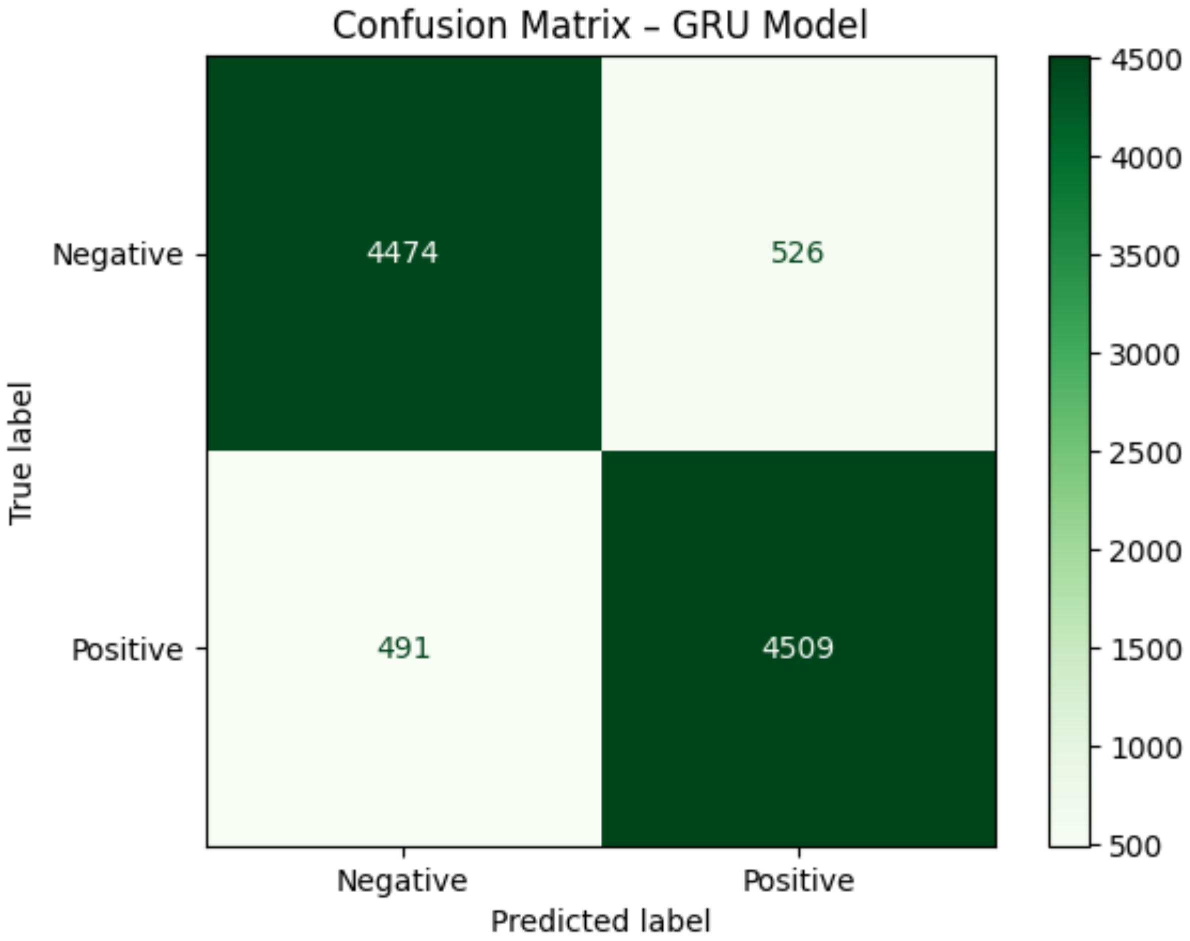
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 12s 39ms/step
```

```
print(f" Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
print(f" Precision: {precision_score(y_test, y_pred):.4f}")
print(f" Recall:    {recall_score(y_test, y_pred):.4f}")
print(f" F1 Score:  {f1_score(y_test, y_pred):.4f}")
```

```
 Accuracy:  0.8983
 Precision: 0.8955
 Recall:    0.9018
 F1 Score:  0.8987
```

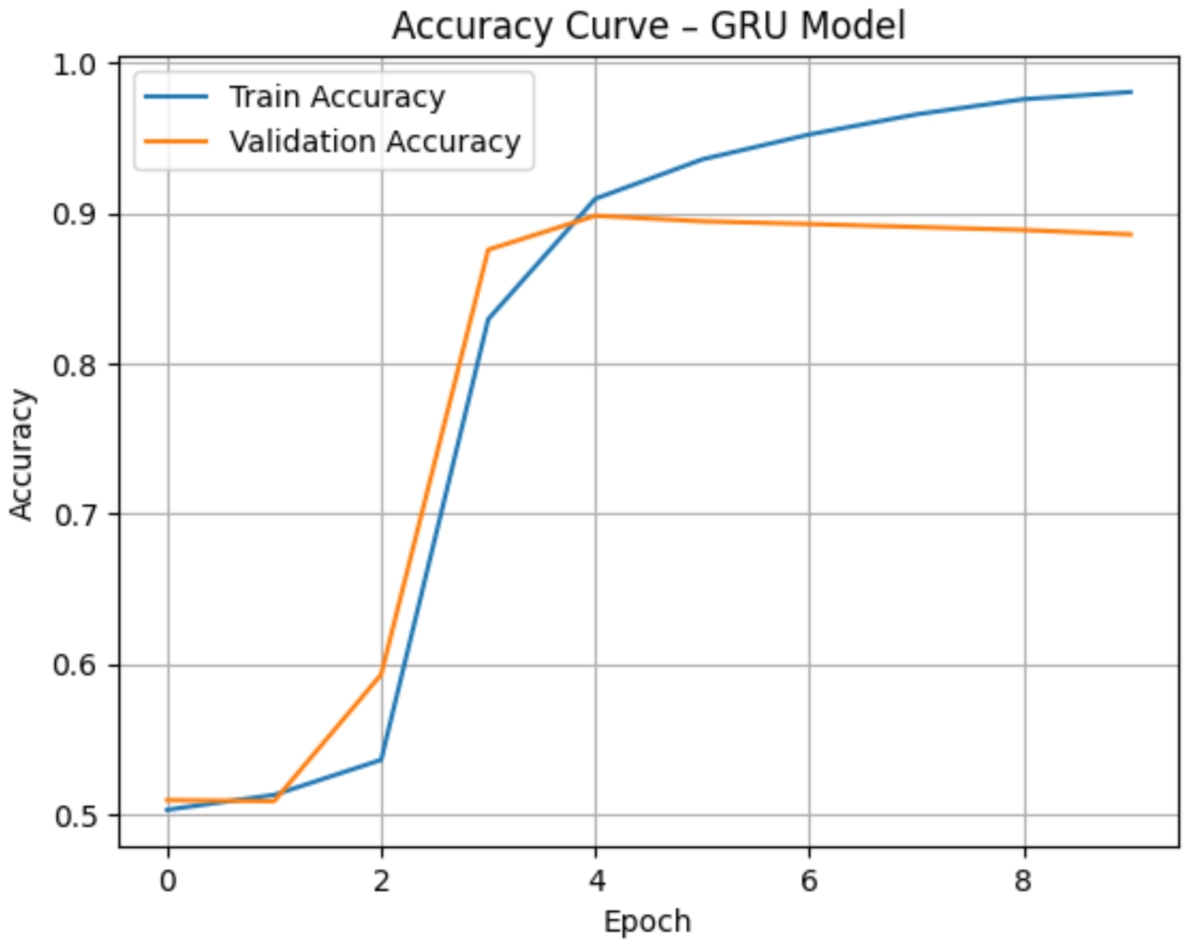
Plot Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Negative",
disp.plot(cmap='Greens')
plt.title("Confusion Matrix – GRU Model")
plt.show()
```



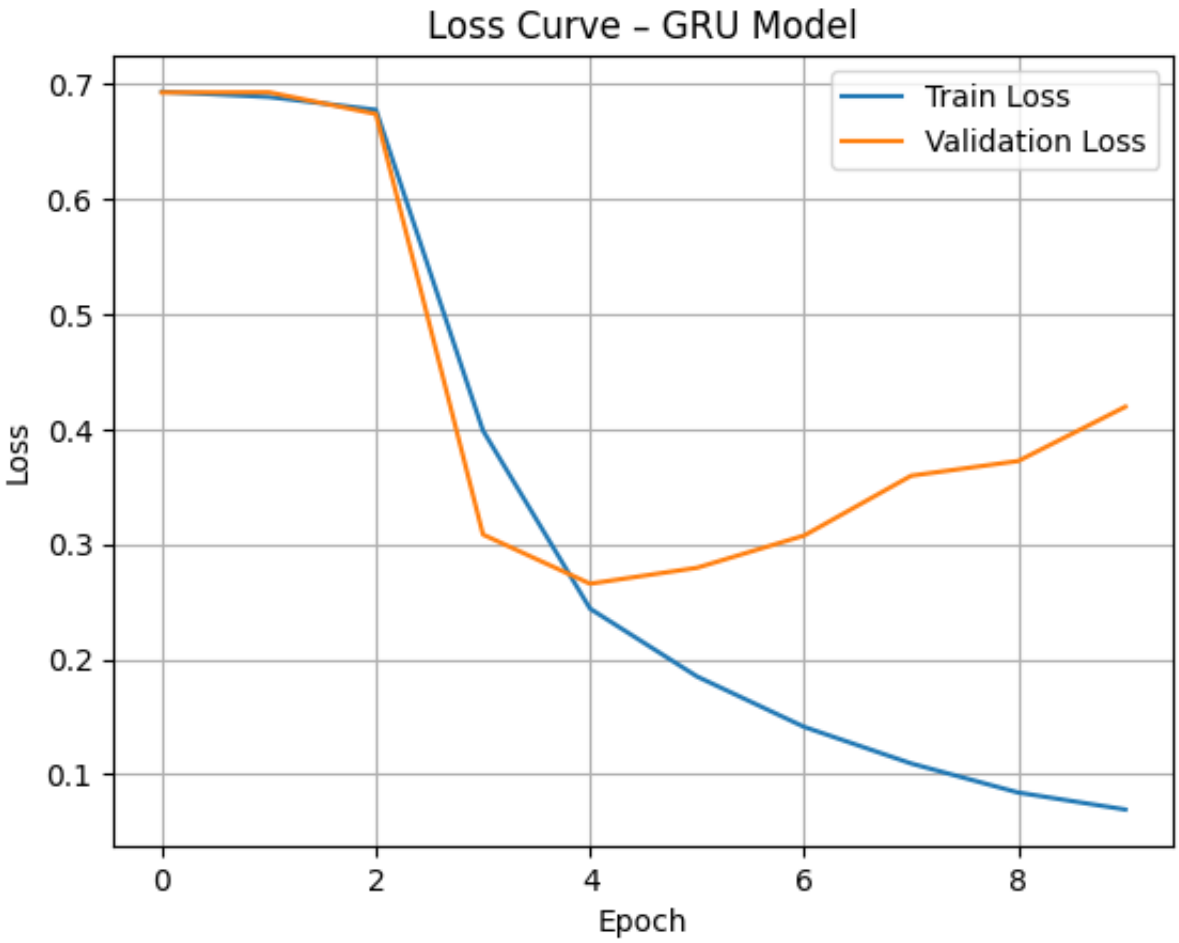
Accuracy Curve

```
plt.plot(history_gru.history['accuracy'], label='Train Accuracy')
plt.plot(history_gru.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Curve – GRU Model')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



Loss Curve

```
plt.plot(history_gru.history['loss'], label='Train Loss')
plt.plot(history_gru.history['val_loss'], label='Validation Loss')
plt.title('Loss Curve – GRU Model')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



Compare All Models: Accuracy & Loss Curves

Accuracy Comparison Plot

```
plt.figure(figsize=(10,6))
plt.plot(history_rnn.history['val_accuracy'], label='RNN - Validation Accuracy'
plt.plot(history_lstm.history['val_accuracy'], label='LSTM - Validation Accurac
plt.plot(history_gru.history['val_accuracy'], label='GRU - Validation Accuracy'

plt.title(' Validation Accuracy Comparison')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

Show hidden output

Loss Comparison Plot

```
plt.figure(figsize=(10,6))

plt.plot(history_rnn.history['val_loss'], label='RNN - Validation Loss')
plt.plot(history_lstm.history['val_loss'], label='LSTM - Validation Loss')
plt.plot(history_gru.history['val_loss'], label='GRU - Validation Loss')

plt.title('📉 Validation Loss Comparison')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```

models save

```
model_gru.save("gru_model.h5")
model_lstm.save("lstm_model.h5")
model_rnn.save("rnn_model.h5")
```

Hyperparameter Optimization

After evaluating all models, GRU achieved the highest validation performance with efficient training time. Therefore, it was selected for hyperparameter optimization using Optuna to further maximize its accuracy and generalization.

```
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-4.3.0-py3-none-any.whl.metadata (17 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.15.2-py3-none-any.whl.metadata (7.3 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: Mako in /usr/lib/python3/dist-packages (from
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/py
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.11/dis
Downloading optuna-4.3.0-py3-none-any.whl (386 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 386.6/386.6 kB 18.3 MB/s eta 0:
Downloading alembic-1.15.2-py3-none-any.whl (231 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 231.9/231.9 kB 21.6 MB/s eta 0:
Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Installing collected packages: colorlog, alembic, optuna
Successfully installed alembic-1.15.2 colorlog-6.9.0 optuna-4.3.0
```

```
def objective(trial):
    model = Sequential()


    embedding_dim = trial.suggest_categorical('embedding_dim', [32, 64, 128])
    model.add(Embedding(input_dim=10000, output_dim=embedding_dim, input_length


    gru_units = trial.suggest_int('gru_units', 32, 128, step=32)
    model.add(GRU(units=gru_units, return_sequences=False))


    dropout_rate = trial.suggest_float('dropout', 0.2, 0.5, step=0.1)
    model.add(Dropout(rate=dropout_rate))


    model.add(Dense(1, activation='sigmoid'))


    learning_rate = trial.suggest_float("lr", 1e-4, 1e-2, log=True)
    model.compile(
        loss='binary_crossentropy',
        optimizer=Adam(learning_rate=learning_rate),
        metrics=['accuracy']
    )


    batch_size = trial.suggest_categorical("batch_size", [64, 128, 256])
    history = model.fit(
        X_train_pad, y_train,
        validation_data=(X_test_pad, y_test),
        epochs=5,
        batch_size=batch_size,
        verbose=0
    )


    val_accuracy = history.history["val_accuracy"][-1]
    return val_accuracy
```

```
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=10)
```

```
[I 2025-04-18 13:34:34,230] A new study created in memory with name: no-nam
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:
  warnings.warn(
[I 2025-04-18 13:35:18,884] Trial 0 finished with value: 0.8898000121116638
[I 2025-04-18 13:35:40,368] Trial 1 finished with value: 0.890500009059906
[I 2025-04-18 13:36:22,636] Trial 2 finished with value: 0.6812999844551086
[I 2025-04-18 13:36:44,118] Trial 3 finished with value: 0.5289999842643738
[I 2025-04-18 13:36:59,823] Trial 4 finished with value: 0.5200999975204468
[I 2025-04-18 13:37:24,329] Trial 5 finished with value: 0.5073000192642212
[I 2025-04-18 13:38:03,015] Trial 6 finished with value: 0.8946999907493591
[I 2025-04-18 13:38:29,613] Trial 7 finished with value: 0.8883000016212463
[I 2025-04-18 13:39:08,501] Trial 8 finished with value: 0.8877000212669373
[I 2025-04-18 13:39:22,394] Trial 9 finished with value: 0.727400004863739
```

```
print(" Best Hyperparameters:")
for key, value in study.best_params.items():
    print(f"{key}: {value}")

print(f"\n Best Validation Accuracy: {study.best_value:.4f}")
```

```
 Best Hyperparameters:
embedding_dim: 32
gru_units: 32
dropout: 0.5
lr: 0.002925248561355503
batch_size: 64

 Best Validation Accuracy: 0.8947
```

Hyperparameter optimization was conducted using Optuna with a search space over embedding_dim, gru_units, dropout, batch_size, and learning_rate. The best configuration achieved a validation accuracy of {study.best_value:.4f}.

After automated hyperparameter tuning using Optuna, we rebuilt the GRU model using the best configuration and trained it for 10 full epochs. This yielded the final optimized performance reported."

Hyperparameter Optimization on GRU Model

```
model_gru_opt = Sequential([
    Embedding(input_dim=10000, output_dim=32, input_length=300),
    GRU(units=32, return_sequences=False),
    Dropout(0.5),
    Dense(32),
    LeakyReLU(alpha=0.01),
    Dense(1, activation='sigmoid')
])


optimizer = Adam(learning_rate=0.00293)

model_gru_opt.compile(
    loss='binary_crossentropy',
    optimizer=optimizer,
    metrics=['accuracy']
)


history_gru_opt = model_gru_opt.fit(
    X_train_pad, y_train,
    validation_data=(X_test_pad, y_test),
    epochs=60,
    batch_size=64,
    callbacks=[EarlyStopping(patience=10, restore_best_weights=True)],
    verbose=1
)
```

```
Epoch 1/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.4992 - loss: 0.693
Epoch 2/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.5103 - loss: 0.6916
Epoch 3/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 12ms/step - accuracy: 0.8040 - loss: 0.4379
Epoch 4/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.9206 - loss: 0.2159
Epoch 5/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.9465 - loss: 0.1572
Epoch 6/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.9628 - loss: 0.1140
Epoch 7/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.9705 - loss: 0.0952
Epoch 8/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.9795 - loss: 0.0633
Epoch 9/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 12ms/step - accuracy: 0.9864 - loss: 0.0465
Epoch 10/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.9880 - loss: 0.0434
Epoch 11/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.9901 - loss: 0.0339
Epoch 12/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 12ms/step - accuracy: 0.9917 - loss: 0.0272
Epoch 13/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - accuracy: 0.9946 - loss: 0.0199
```

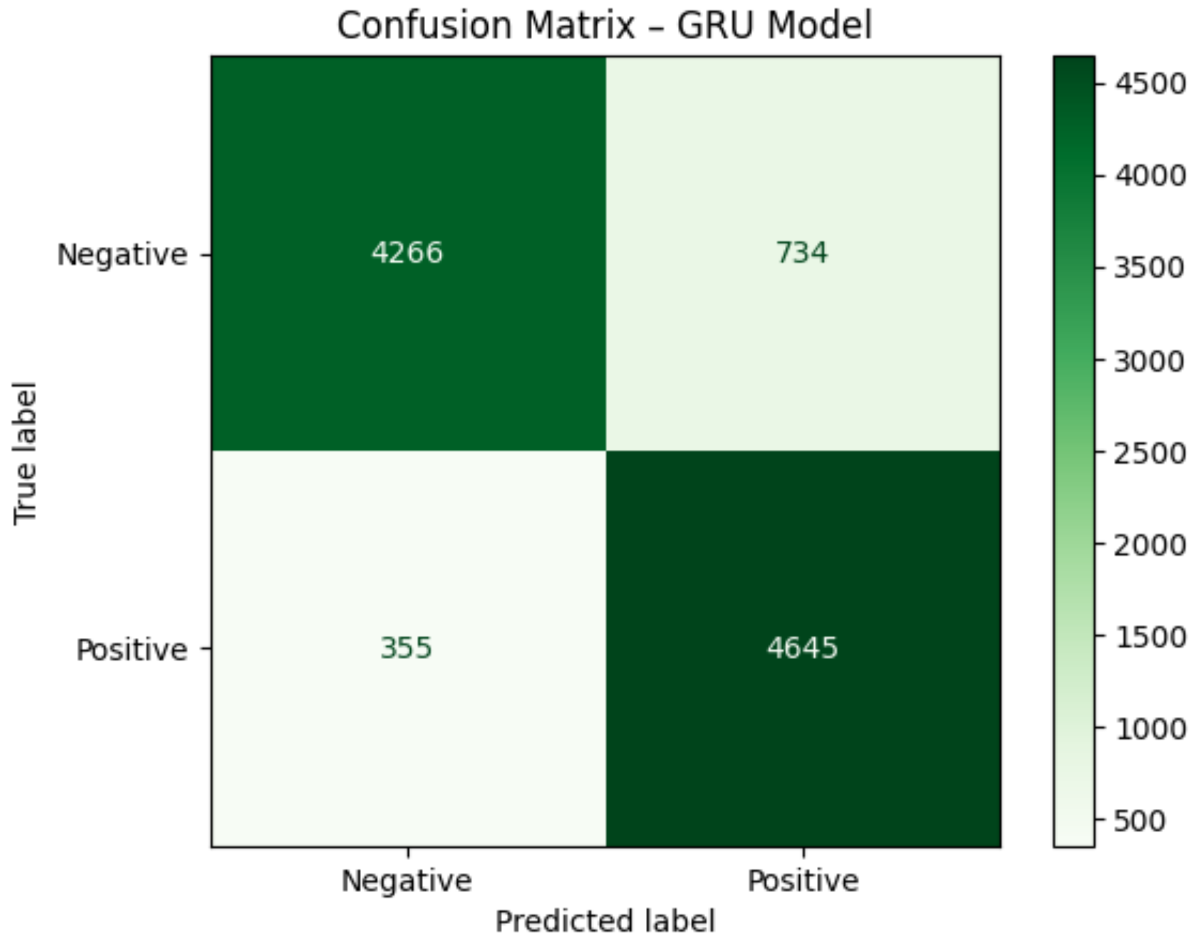
```
y_pred_prob = model_gru_opt.predict(X_test_pad).flatten()
y_pred = (y_pred_prob > 0.5).astype("int32")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```

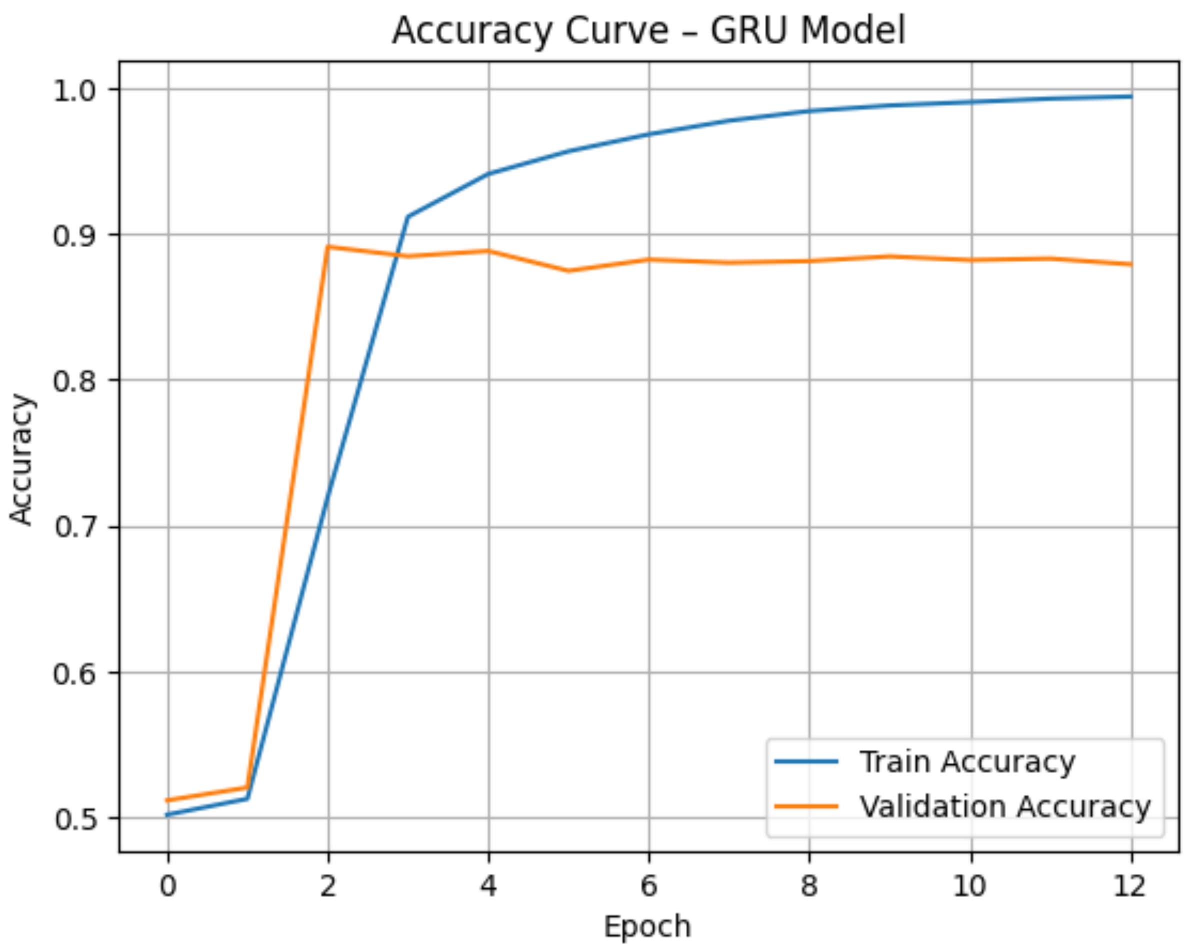
```
print(f" Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
print(f" Precision: {precision_score(y_test, y_pred):.4f}")
print(f" Recall:    {recall_score(y_test, y_pred):.4f}")
print(f" F1 Score:  {f1_score(y_test, y_pred):.4f}")
```

```
 Accuracy:  0.8711
 Precision: 0.8425
 Recall:    0.9128
 F1 Score:  0.8763
```

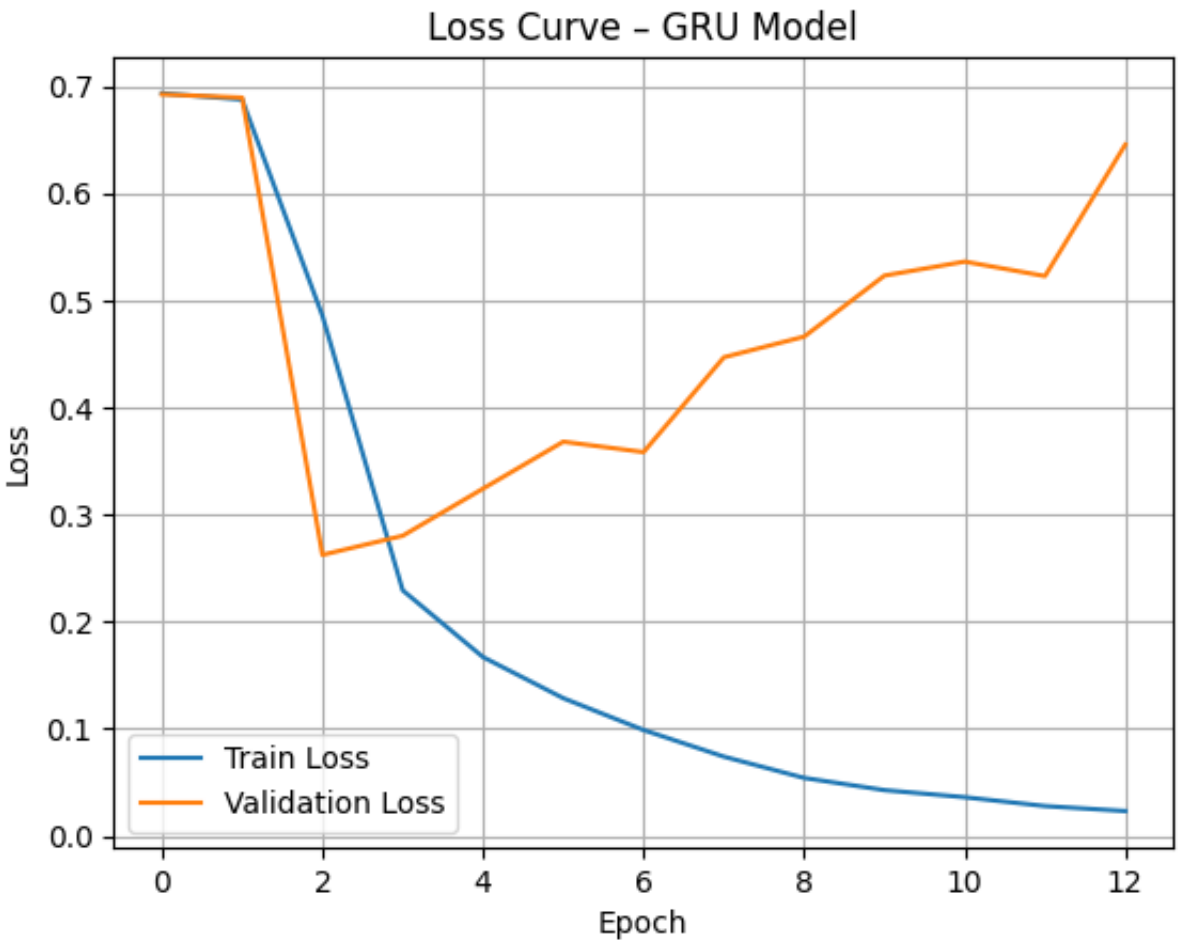
```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Negative",
disp.plot(cmap='Greens')
plt.title("Confusion Matrix – GRU Model")
plt.show()
```



```
plt.plot(history_gru_opt.history['accuracy'], label='Train Accuracy')
plt.plot(history_gru_opt.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Curve – GRU Model')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



```
plt.plot(history_gru_opt.history['loss'], label='Train Loss')
plt.plot(history_gru_opt.history['val_loss'], label='Validation Loss')
plt.title('Loss Curve – GRU Model')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



Bidirectional GRU

```
model_bi_gru = Sequential([
    Embedding(input_dim=10000, output_dim=32, input_length=300),
    Bidirectional(GRU(units=32, return_sequences=False)),
    Dropout(0.5),
    Dense(32),
    LeakyReLU(alpha=0.01),
    Dense(1, activation='sigmoid')
])


optimizer = Adam(learning_rate=0.00293)


model_bi_gru.compile(
    loss='binary_crossentropy',
    optimizer=optimizer,
    metrics=['accuracy']
)


history_bi_gru = model_bi_gru.fit(
    X_train_pad, y_train,
    validation_data=(X_test_pad, y_test),
    epochs=60,
    batch_size=64,
    callbacks=[EarlyStopping(patience=5, restore_best_weights=True)],
    verbose=1
)
```

```
Epoch 1/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 21ms/step - accuracy: 0.6878 - loss: 0.565
Epoch 2/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.8866 - loss: 0.282
Epoch 3/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9225 - loss: 0.207
Epoch 4/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9452 - loss: 0.153
Epoch 5/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9592 - loss: 0.118
Epoch 6/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9705 - loss: 0.089
Epoch 7/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9731 - loss: 0.080
```

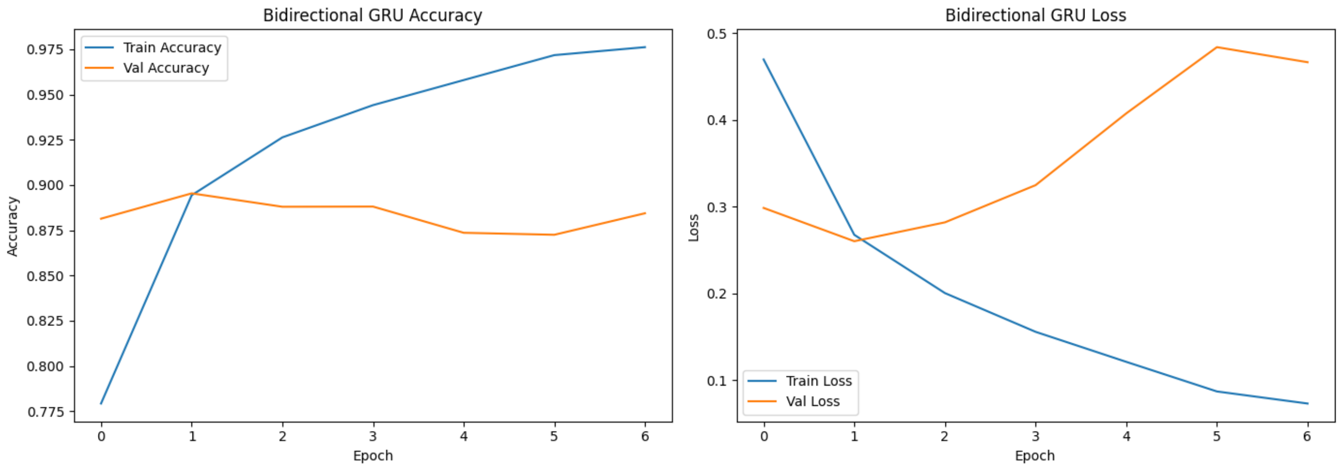
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s

y_pred_prob = model_bi_gru.predict(X_test_pad).flatten()
y_pred = (y_pred_prob > 0.5).astype("int32")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
```

```
print(f" Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
print(f" Precision: {precision_score(y_test, y_pred):.4f}")
print(f" Recall:    {recall_score(y_test, y_pred):.4f}")
print(f" F1 Score:  {f1_score(y_test, y_pred):.4f}")
```

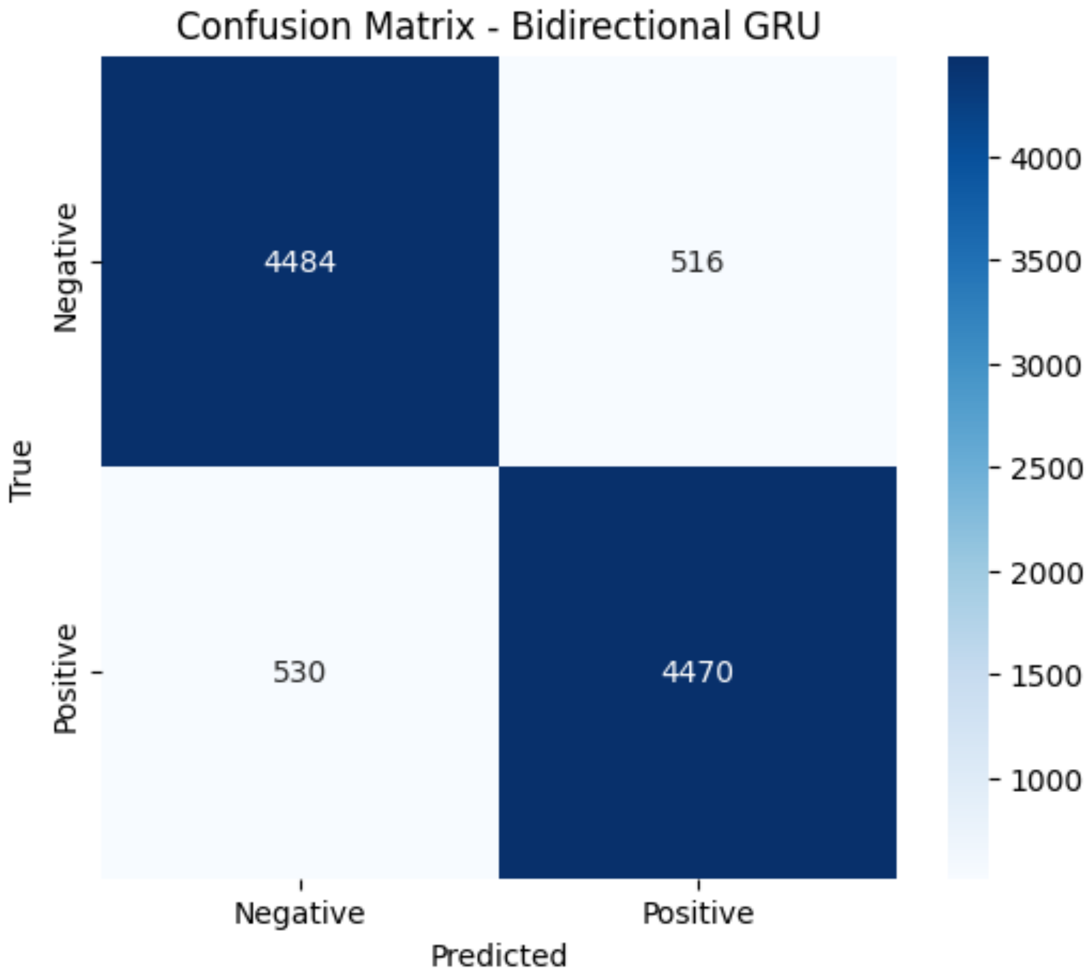
```
 Accuracy:  0.8954
 Precision: 0.8965
 Recall:    0.8940
 F1 Score:  0.8953
```

```
def plot_training_history(history, model_name="Model"):
    plt.figure(figsize=(14, 5))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(f'{model_name} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()


plot_training_history(history_bi_gru, model_name="Bidirectional GRU")
```




```
y_pred_prob = model_bi_gru.predict(X_test_pad)
y_pred = (y_pred_prob > 0.5).astype("int32")

print("Classification Report:")
print(classification_report(y_test, y_pred, digits=4))


cm = confusion_matrix(y_test, y_pred)


plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Pc
plt.title('Confusion Matrix - Bidirectional GRU')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step
Classification Report:
              precision    recall  f1-score   support

           0     0.8943    0.8968    0.8955      5000
           1     0.8965    0.8940    0.8953      5000

    accuracy                         0.8954     10000
   macro avg     0.8954    0.8954    0.8954     10000
weighted avg     0.8954    0.8954    0.8954     10000
```



Bidirectional LSTM

```
model_bi_lstm = Sequential([
    Embedding(input_dim=10000, output_dim=32, input_length=300),
    Bidirectional(LSTM(units=32, return_sequences=False)),
    Dropout(0.5),
    Dense(32),
    LeakyReLU(alpha=0.01),
    Dense(1, activation='sigmoid')
])

optimizer = Adam(learning_rate=0.00293)


model_bi_lstm.compile(
    loss='binary_crossentropy',
    optimizer=optimizer,
    metrics=['accuracy']
)


history_bi_lstm = model_bi_lstm.fit(
    X_train_pad, y_train,
    validation_data=(X_test_pad, y_test),
    epochs=60,
    batch_size=64,
    callbacks=[EarlyStopping(patience=5, restore_best_weights=True)],
    verbose=1
)
```

```
Epoch 1/60
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/keras/src/layers/activations/leaky_
  warnings.warn(
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 22ms/step - accuracy: 0.6776 - loss: 0.584
Epoch 2/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.8746 - loss: 0.324
Epoch 3/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.8830 - loss: 0.309
Epoch 4/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9160 - loss: 0.228
Epoch 5/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.8935 - loss: 0.269
Epoch 6/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9229 - loss: 0.208
Epoch 7/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9425 - loss: 0.168
Epoch 8/60
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 22ms/step - accuracy: 0.9503 - loss: 0.145
```

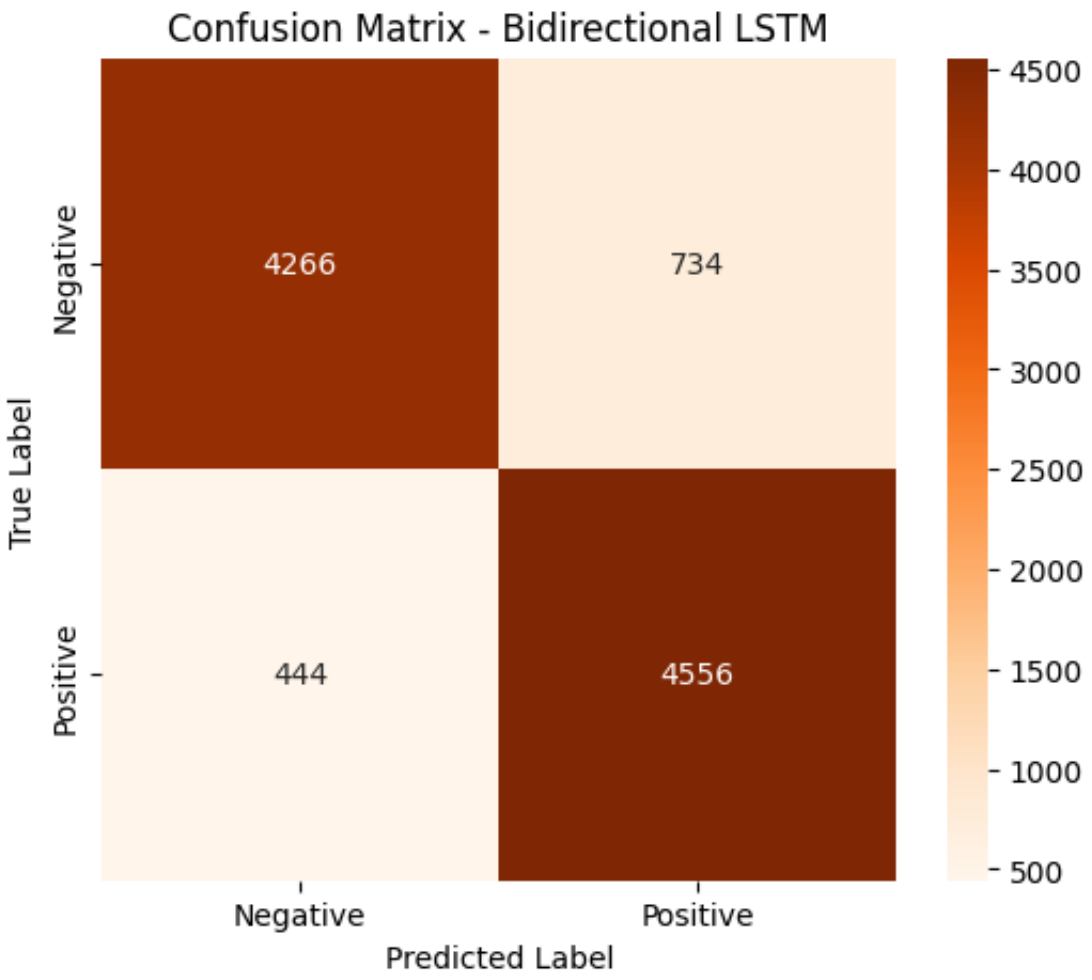
```
y_pred_prob = model_bi_lstm.predict(X_test_pad).flatten()
y_pred = (y_pred_prob > 0.5).astype("int32")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
```

```
print(f" Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
print(f" Precision: {precision_score(y_test, y_pred):.4f}")
print(f" Recall:    {recall_score(y_test, y_pred):.4f}")
print(f" F1 Score:  {f1_score(y_test, y_pred):.4f}")
```

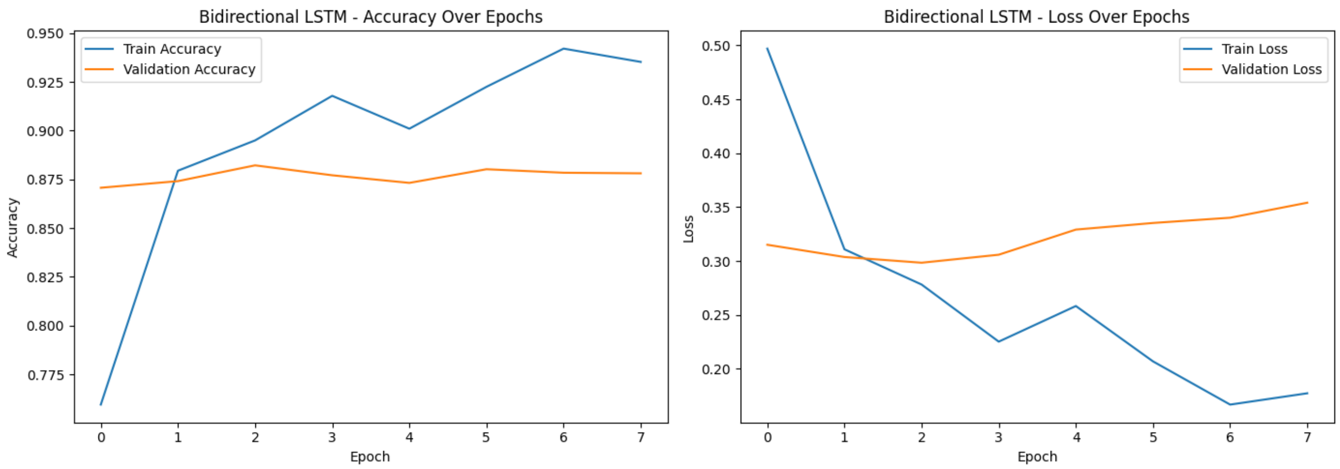
```
 Accuracy:  0.8822
 Precision: 0.8612
 Recall:    0.9112
 F1 Score:  0.8855
```

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges', xticklabels=['Negative', '
plt.title('Confusion Matrix - Bidirectional LSTM')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```




```
def plot_accuracy_loss(history, model_name="Model"):
    plt.figure(figsize=(14, 5))


    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{model_name} - Accuracy Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
```

```
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{model_name} - Loss Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()


plot_accuracy_loss(history_bi_lstm, model_name="Bidirectional LSTM")
```




```
Start coding or generate with AI.
```