In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
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, GRU, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from transformers import BertTokenizer, TFBertModel
import tensorflow as tf
from gensim.models import KeyedVectors
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.metrics import classification_report, accuracy_score, f1_score
import time

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline

import nltk
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/ytkuang/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/ytkuang/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[1]:
```
True
```

In [2]:
```
mbti_data = pd.read_csv("./MBTI 500.csv")
mbti_data
```

Out[2]:

	posts	type
0	know intj tool use interaction people excuse a...	INTJ
1	rap music ehh opp yeah know valid well know fa...	INTJ
2	preferably p hd low except wew lad video p min...	INTJ
3	drink like wish could drink red wine give head...	INTJ
4	space program ah bad deal meing freelance max ...	INTJ
...	...	...
106062	stay frustrate world life want take long nap w...	INFP
106063	fizzle around time mention sure mistake thing ...	INFP
106064	schedule modify hey w intp strong wing underst...	INFP
106065	enfj since january busy schedule able spend li...	INFP
106066	feel like men good problem tell parent want te...	INFP

106067 rows × 2 columns

In [3]:
```
mbti_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106067 entries, 0 to 106066
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   posts   106067 non-null  object
 1   type    106067 non-null  object
dtypes: object(2)
memory usage: 1.6+ MB
```

In [4]:
```
print("Duplicates: ",mbti_data.duplicated().sum())
```

```
Duplicates:  0
```

In [5]:
```
mbti_data.isna().count()
```

Out[5]:
```
posts    106067
type     106067
dtype: int64
```

Lemmatization

In [6]:
```
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Define All MBTI types
mbti_types = ['INTJ', 'INTP', 'ENTJ', 'ENTP', 'INFJ', 'INFP', 'ENFJ', 'ENFP',
              'ISTJ', 'ISFJ', 'ESTJ', 'ESFJ', 'ISTP', 'ISFP', 'ESTP', 'ESFP']

def clean_text(text):
    text = re.sub(r'http\S+', '', text)  # Remove URL
    text = re.sub(r'[^A-Za-z\s]', '', text)  # Remove non-alphabetic characters
    text = re.sub(r'@\w+', '', text) # Remove users ID
    for mbti in mbti_types:
        text = re.sub(mbti, '', text, flags=re.IGNORECASE) # Remove MBTI types (both upper- and lower-case)
    text = re.sub(r'[\(\){}<>:;]', '', text) # Remove emoticons
    text = text.lower()  # Convert to lowercase
    tokens = text.split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return " ".join(tokens)

mbti_data['clean_posts'] = mbti_data['posts'].apply(clean_text)
mbti_data
```

Out[6]:

	posts	type	clean_posts
0	know intj tool use interaction people excuse a...	INTJ	know tool use interaction people excuse antiso...
1	rap music ehh opp yeah know valid well know fa...	INTJ	rap music ehh opp yeah know valid well know fa...
2	preferably p hd low except wew lad video p min...	INTJ	preferably p hd low except wew lad video p min...
3	drink like wish could drink red wine give head...	INTJ	drink like wish could drink red wine give head...
4	space program ah bad deal meing freelance max ...	INTJ	space program ah bad deal meing freelance max ...
...	...	...	...
106062	stay frustrate world life want take long nap w...	INFP	stay frustrate world life want take long nap w...
106063	fizzle around time mention sure mistake thing ...	INFP	fizzle around time mention sure mistake thing ...
106064	schedule modify hey w intp strong wing underst...	INFP	schedule modify hey w strong wing understand p...
106065	enfj since january busy schedule able spend li...	INFP	since january busy schedule able spend little ...
106066	feel like men good problem tell parent want te...	INFP	feel like men good problem tell parent want te...

106067 rows × 3 columns

In [7]:
```
mbti_data[mbti_data['clean_posts']==''']
```

Out[7]:

	posts	type	clean_posts
22928	entp entp entp entp entp entp entp entp entp e...	INTP	

In [8]:
```
mbti_data.drop(index=22928, inplace=True)
```

In [9]:
```
mbti_data[mbti_data['clean_posts']==''']
```

Out[9]:

posts	type	clean_posts

In [10]:
```
mbti_data.to_csv('mbti_data_cleaned.csv', index=False)
```

In [11]:
```
# Encodeing
mbti_data['I/E'] = mbti_data['type'].apply(lambda x: 1 if x[0] == 'I' else 0)
mbti_data['S/N'] = mbti_data['type'].apply(lambda x: 1 if x[1] == 'S' else 0)
mbti_data['F/T'] = mbti_data['type'].apply(lambda x: 1 if x[2] == 'F' else 0)
mbti_data['J/P'] = mbti_data['type'].apply(lambda x: 1 if x[3] == 'J' else 0)
mbti_data
```

Out[11]:

	posts	type	clean_posts	I/E	S/N	F/T	J/P
0	know intj tool use interaction people excuse a...	INTJ	know tool use interaction people excuse antiso...	1	0	0	1
1	rap music ehh opp yeah know valid well know fa...	INTJ	rap music ehh opp yeah know valid well know fa...	1	0	0	1
2	preferably p hd low except wew lad video p min...	INTJ	preferably p hd low except wew lad video p min...	1	0	0	1
3	drink like wish could drink red wine give head...	INTJ	drink like wish could drink red wine give head...	1	0	0	1
4	space program ah bad deal meing freelance max ...	INTJ	space program ah bad deal meing freelance max ...	1	0	0	1
...	...	...	...	...	...	...	...
106062	stay frustrate world life want take long nap w...	INFP	stay frustrate world life want take long nap w...	1	0	1	0
106063	fizzle around time mention sure mistake thing ...	INFP	fizzle around time mention sure mistake thing ...	1	0	1	0
106064	schedule modify hey w intp strong wing underst...	INFP	schedule modify hey w strong wing understand p...	1	0	1	0
106065	enfj since january busy schedule able spend li...	INFP	since january busy schedule able spend little ...	1	0	1	0
106066	feel like men good problem tell parent want te...	INFP	feel like men good problem tell parent want te...	1	0	1	0

106066 rows × 7 columns

Text vectorization

In [13]:
```
word2vec_path = 'GoogleNews-vectors-negative300.bin'
word2vec_model = KeyedVectors.load_word2vec_format(word2vec_path, binary=True)

def text_to_vector(text, model, vector_size=300):
    tokens = word_tokenize(text)
```

```
    vectors = [model[word] for word in tokens if word in model]
    if len(vectors) == 0:  # If no words in the text exist in the word embedding model
        return np.zeros(vector_size)
    return np.mean(vectors, axis=0)

# Convert clean_posts to vector
mbti_data['w2v_vector'] = mbti_data['clean_posts'].apply(lambda x: text_to_vector(x, word2vec_model))
mbti_data
```

Out[13]:

	posts	type	clean_posts	I/E	S/N	F/T	J/P	w2v_vector
0	know intj tool use interaction people excuse a...	INTJ	know tool use interaction people excuse antiso...	1	0	0	1	[0.05413393, 0.016348278, 0.0030844838, 0.1083...
1	rap music ehh opp yeah know valid well know fa...	INTJ	rap music ehh opp yeah know valid well know fa...	1	0	0	1	[0.05711113, 0.0146960085, 0.0032905627, 0.112...
2	preferably p hd low except wew lad video p min...	INTJ	preferably p hd low except wew lad video p min...	1	0	0	1	[0.037168004, 0.016440965, -0.020269552, 0.075...
3	drink like wish could drink red wine give head...	INTJ	drink like wish could drink red wine give head...	1	0	0	1	[0.041060925, 0.020203583, 0.0071685174, 0.102...
4	space program ah bad deal meing freelance max ...	INTJ	space program ah bad deal meing freelance max ...	1	0	0	1	[0.026845422, 0.025289614, -0.005735043, 0.080...
...	...	...	...	...	...	...	...	...
106062	stay frustrate world life want take long nap w...	INFP	stay frustrate world life want take long nap w...	1	0	1	0	[0.037779823, 0.029206052, 0.008067803, 0.1268...
106063	fizzle around time mention sure mistake thing ...	INFP	fizzle around time mention sure mistake thing ...	1	0	1	0	[0.054155268, 0.0217389, -0.0034899262, 0.1055...
106064	schedule modify hey w intp strong wing underst...	INFP	schedule modify hey w strong wing understand p...	1	0	1	0	[0.046949226, 0.015541024, -0.0039697215, 0.11...
106065	enfj since january busy schedule able spend li...	INFP	since january busy schedule able spend little ...	1	0	1	0	[0.04922998, 0.00739275, -0.0145673575, 0.1157...
106066	feel like men good problem tell parent want te...	INFP	feel like men good problem tell parent want te...	1	0	1	0	[0.05153137, 0.012769314, 0.014004834, 0.09845...

106066 rows × 8 columns

In [14]:
```
mbti_data.to_csv('mbti_data_cleaned.csv', index=False)
```

SMOTE

In [15]:
```
X_smote = np.array(mbti_data['w2v_vector'].tolist())
y_smote = mbti_data['type']
```

In [16]:
```
# Define the desired number of samples for each class
desired_count = 1000

# Define the oversampling strategy for SMOTE
oversample_strategy = {label: desired_count for label, count in y_smote.value_counts().items() if count < desired_c

# Define the undersampling strategy for RandomUnderSampler
undersample_strategy = {label: desired_count for label, count in y_smote.value_counts().items() if count > desired_

# Create the SMOTE and RandomUnderSampler objects
smote = SMOTE(sampling_strategy=oversample_strategy, random_state=42)
undersampler = RandomUnderSampler(sampling_strategy=undersample_strategy, random_state=42)

# Combine SMOTE and RandomUnderSampler in a pipeline
pipeline = Pipeline(steps=[
    ('smote', smote),
    ('undersample', undersampler)
])
```

In [17]:
```
# Print class distribution before resampling
print("Before resampling:", y_smote.value_counts())
print()

# Apply the pipeline to resample the dataset
X_resampled, y_resampled = pipeline.fit_resample(X_smote, y_smote)

# Print class distribution after resampling
print("After resampling:", y_resampled.value_counts())
```

```
Before resampling: type
INTP    24960
INTJ    22427
INFJ    14963
INFP    12134
ENTP    11725
ENFP     6167
ISTP     3424
ENTJ     2955
ESTP     1986
ENFJ     1534
ISTJ     1243
ISFP      875
ISFJ      650
ESTJ      482
ESFP      360
ESFJ      181
Name: count, dtype: int64

After resampling: type
ENFJ    1000
ENFP    1000
ENTJ    1000
ENTP    1000
ESFJ    1000
ESFP    1000
ESTJ    1000
ESTP    1000
INFJ    1000
INFP    1000
INTJ    1000
INTP    1000
ISFJ    1000
ISFP    1000
ISTJ    1000
ISTP    1000
Name: count, dtype: int64
```

In [18]:

```
X_resampled_list = [list(row) for row in X_resampled]

# Combine X_resampled and y_resampled
```

```
resampled_df = pd.DataFrame({
    "type": y_resampled,
    "w2v_vector": X_resampled_list
})
resampled_df
```

Out[18]:

	type	w2v_vector
0	ENFJ	[0.039229408, 0.0059563955, 0.019520251, 0.130...
1	ENFJ	[0.04012091, 0.029049352, -0.0073196026, 0.084...
2	ENFJ	[0.0655946, 0.019188847, -0.00016400666, 0.117...
3	ENFJ	[0.04434076, 0.018996624, 0.0066809473, 0.1083...
4	ENFJ	[0.049488395, 0.01650631, 0.0027976301, 0.1107...
...	...	...
15995	ISTP	[0.053053286, 0.008966654, 0.0008905827, 0.116...
15996	ISTP	[0.055413235, 0.010032379, 0.007235305, 0.1241...
15997	ISTP	[0.048991177, 0.03053955, 0.0056444313, 0.1111...
15998	ISTP	[0.044655286, 0.021909144, 0.017070893, 0.1164...
15999	ISTP	[0.04858312, 0.025488254, 0.012184088, 0.11468...

16000 rows × 2 columns

In [19]:

```
# Final Encodeing
resampled_df['I/E'] = resampled_df['type'].apply(lambda x: 1 if x[0] == 'I' else 0)
resampled_df['S/N'] = resampled_df['type'].apply(lambda x: 1 if x[1] == 'S' else 0)
resampled_df['F/T'] = resampled_df['type'].apply(lambda x: 1 if x[2] == 'F' else 0)
resampled_df['J/P'] = resampled_df['type'].apply(lambda x: 1 if x[3] == 'J' else 0)
resampled_df
```

Out[19]:

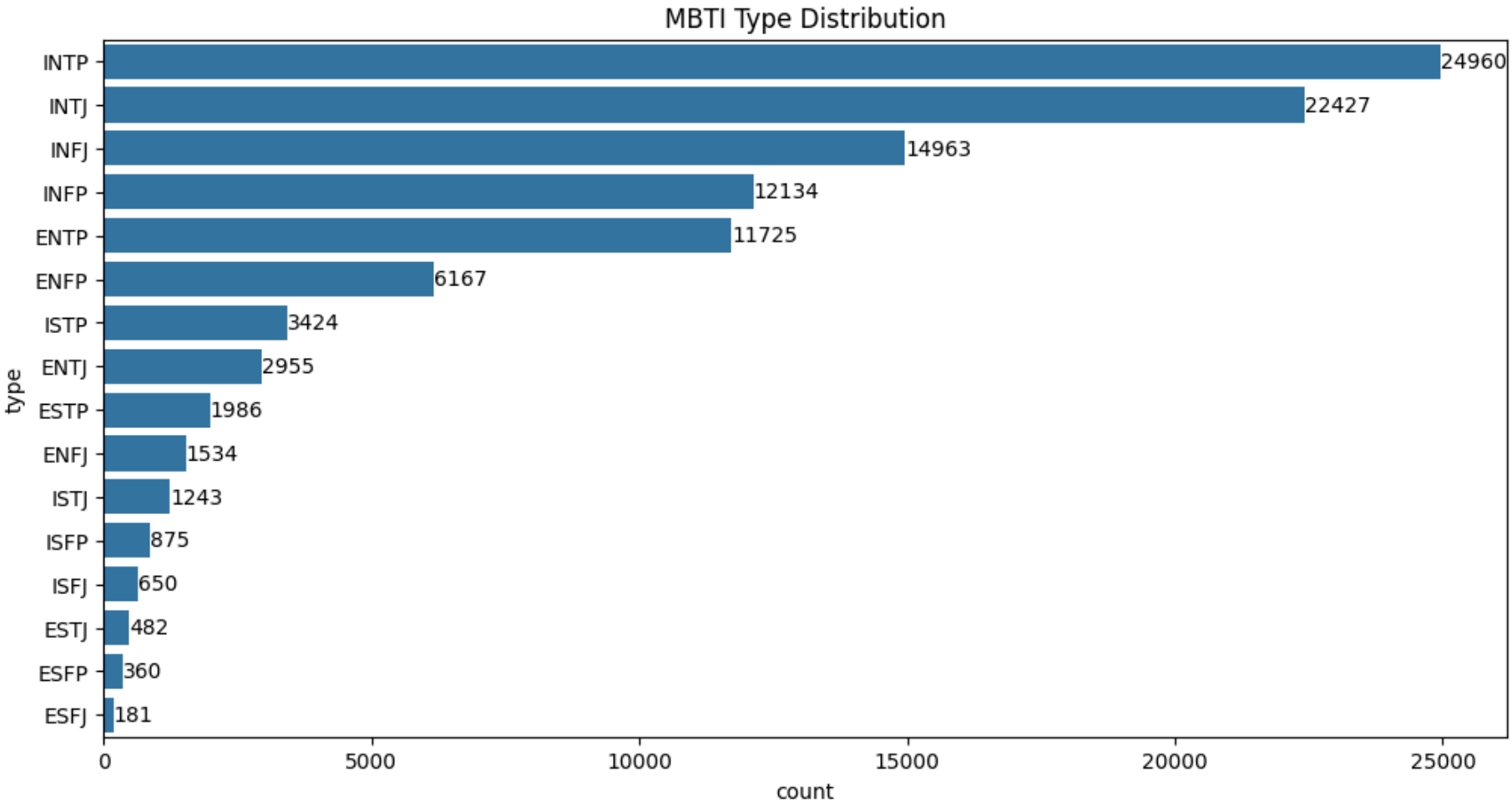
	type	w2v_vector	I/E	S/N	F/T	J/P
0	ENFJ	[0.039229408, 0.0059563955, 0.019520251, 0.130...	0	0	1	1
1	ENFJ	[0.04012091, 0.029049352, -0.0073196026, 0.084...	0	0	1	1
2	ENFJ	[0.0655946, 0.019188847, -0.00016400666, 0.117...	0	0	1	1
3	ENFJ	[0.04434076, 0.018996624, 0.0066809473, 0.1083...	0	0	1	1
4	ENFJ	[0.049488395, 0.01650631, 0.0027976301, 0.1107...	0	0	1	1
...	...	...	...	...	...	...
15995	ISTP	[0.053053286, 0.008966654, 0.0008905827, 0.116...	1	1	0	0
15996	ISTP	[0.055413235, 0.010032379, 0.007235305, 0.1241...	1	1	0	0
15997	ISTP	[0.048991177, 0.03053955, 0.0056444313, 0.1111...	1	1	0	0
15998	ISTP	[0.044655286, 0.021909144, 0.017070893, 0.1164...	1	1	0	0
15999	ISTP	[0.04858312, 0.025488254, 0.012184088, 0.11468...	1	1	0	0

16000 rows × 6 columns

EDA

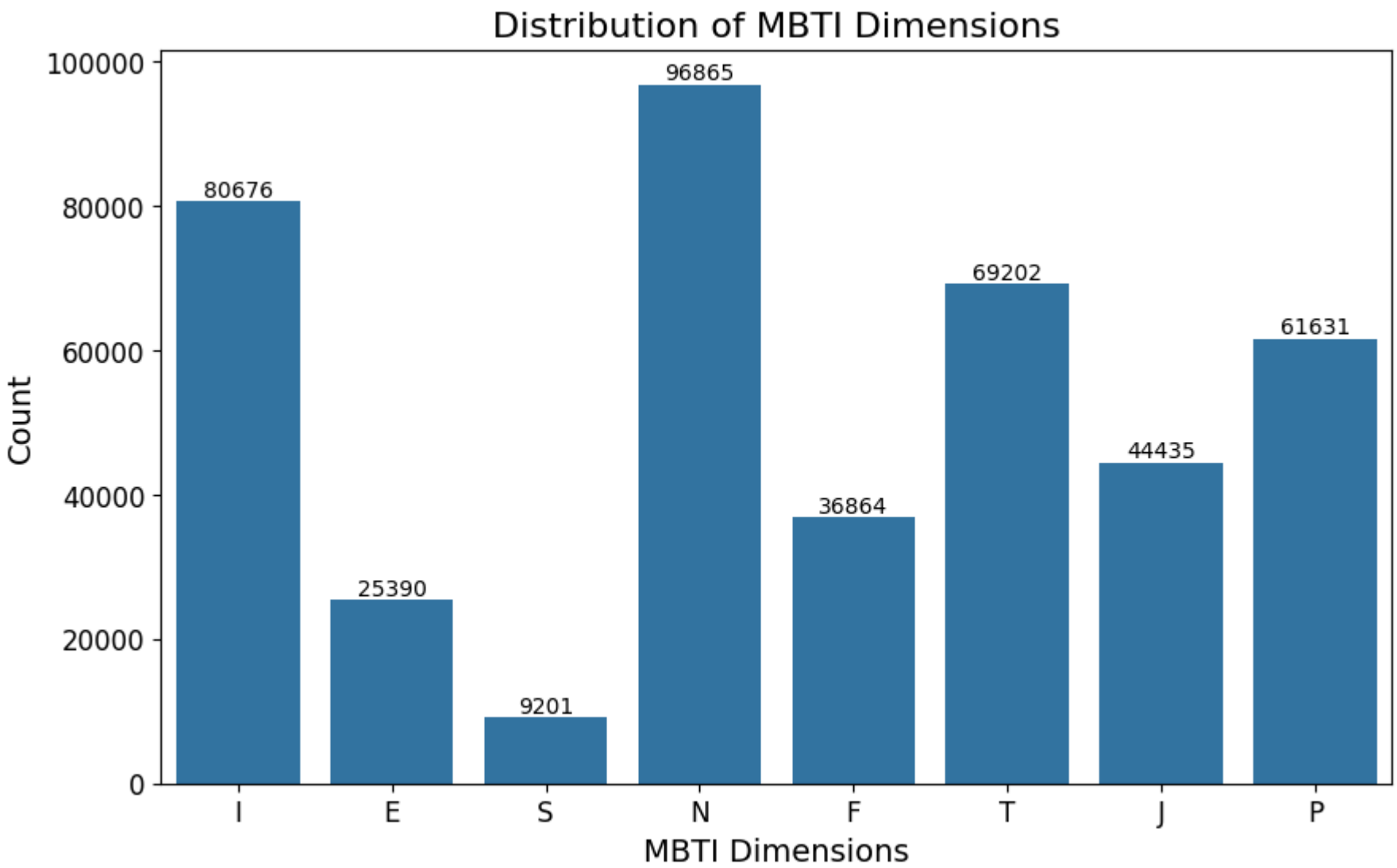
In [20]:

```
# Distribution for every MBTI types (Original)
plt.figure(figsize=(12, 6))
ax = sns.countplot(mbti_data['type'], order=mbti_data['type'].value_counts().index)
abs_values = mbti_data['type'].value_counts(ascending=False).values
ax.bar_label(container=ax.containers[0], labels=abs_values)
plt.title('MBTI Type Distribution')

# Save the plot to a file
plt.savefig('mbti_type_distribution.png', dpi=300, bbox_inches='tight')

plt.show()
```



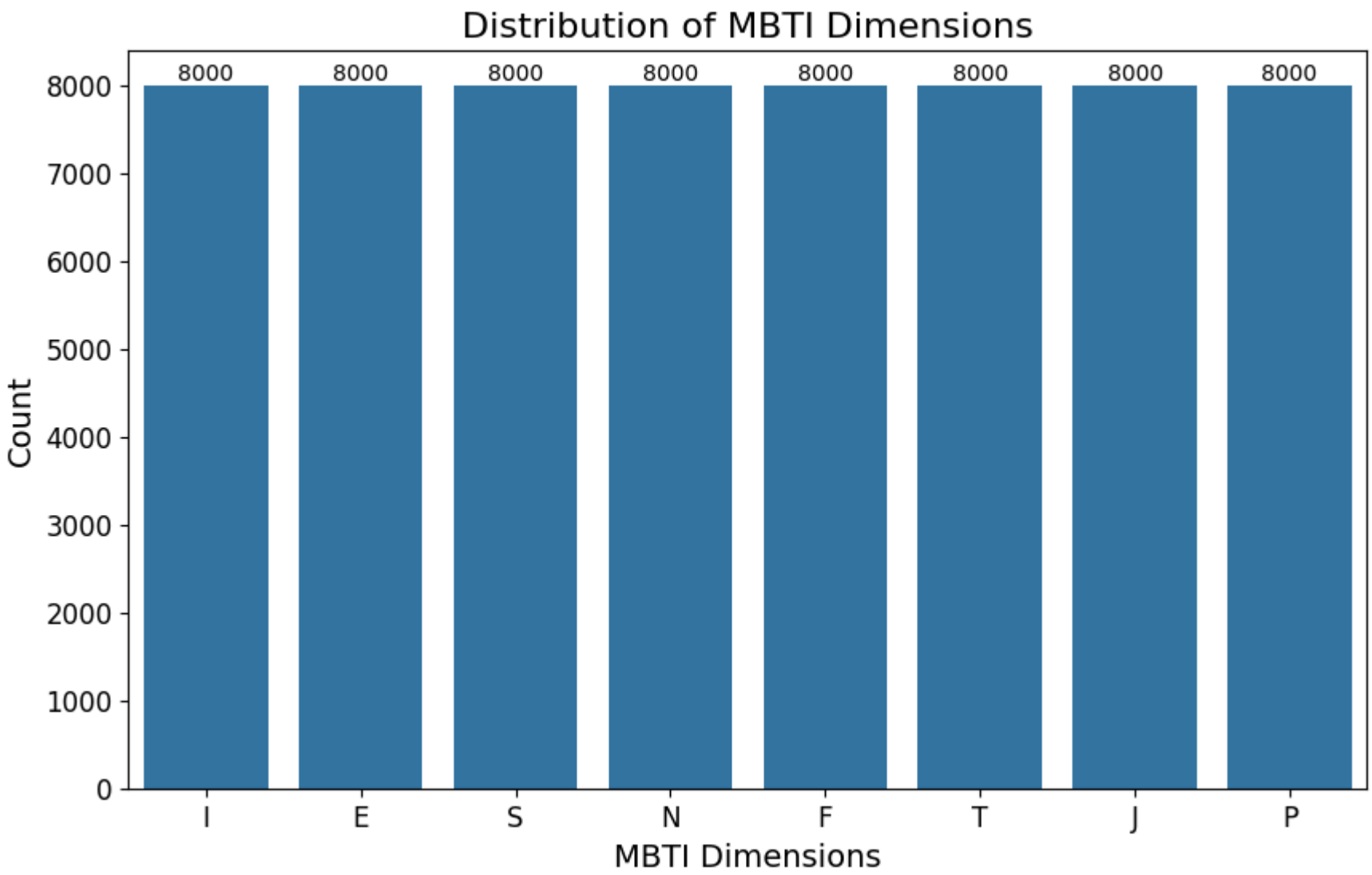
In [21]:
```
# Calaulate the sum of eight dimentions (Original)
dimension_counts = {
    'I': mbti_data['I/E'].sum(),
    'E': len(mbti_data) - mbti_data['I/E'].sum(),
    'S': mbti_data['S/N'].sum(),
    'N': len(mbti_data) - mbti_data['S/N'].sum(),
    'F': mbti_data['F/T'].sum(),
    'T': len(mbti_data) - mbti_data['F/T'].sum(),
    'J': mbti_data['J/P'].sum(),
    'P': len(mbti_data) - mbti_data['J/P'].sum()
```

```
}

# Create dimension dataframe
dimension_df = pd.DataFrame(list(dimension_counts.items()), columns=['Dimension', 'Count'])

# Plot the distribution of eight dimentions
plt.figure(figsize=(10, 6))
ax = sns.barplot(x='Dimension', y='Count', data=dimension_df)
ax.bar_label(container=ax.containers[0], labels=dimension_df['Count'])
plt.title('Distribution of MBTI Dimensions', fontsize=16)
plt.xlabel('MBTI Dimensions', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Save the plot to a file
plt.savefig('mbti_dimensions_distribution_before_smote.png', dpi=300, bbox_inches='tight')
plt.show()
```



In [22]:
```
# Calaulate the sum of eight dimentions (After resampling)
dimension_counts = {
    'I': resampled_df['I/E'].sum(),
    'E': len(resampled_df) - resampled_df['I/E'].sum(),
    'S': resampled_df['S/N'].sum(),
    'N': len(resampled_df) - resampled_df['S/N'].sum(),
    'F': resampled_df['F/T'].sum(),
```

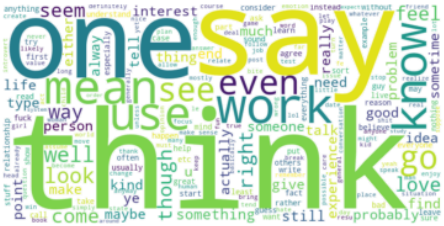
```
    'T': len(resampled_df) - resampled_df['F/T'].sum(),
    'J': resampled_df['J/P'].sum(),
    'P': len(resampled_df) - resampled_df['J/P'].sum()
}

# Create dimension dataframe
dimension_df = pd.DataFrame(list(dimension_counts.items()), columns=['Dimension', 'Count'])

# Plot the distribution of eight dimentions
plt.figure(figsize=(10, 6))
ax = sns.barplot(x='Dimension', y='Count', data=dimension_df)
ax.bar_label(container=ax.containers[0], labels=dimension_df['Count'])
plt.title('Distribution of MBTI Dimensions', fontsize=16)
plt.xlabel('MBTI Dimensions', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Save the plot to a file
plt.savefig('mbti_dimensions_distribution_after_smote.png', dpi=300, bbox_inches='tight')
plt.show()
```



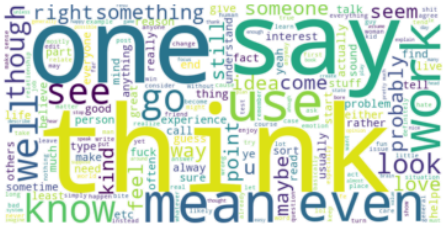
In [23]:
```
# Plot a word cloud for each MBTI type (Original)
def plot_wordcloud_for_mbti(df, mbti_type_column='type', post_column='clean_posts'):
    mbti_types = df[mbti_type_column].unique()

    plt.figure(figsize=(15, 15))

    for i, mbti_type in enumerate(mbti_types, 1):
```

```
        plt.subplot(4, 4, i)  # 16 types, arranged 4x4
        type_posts = df[df[mbti_type_column] == mbti_type][post_column].str.cat(sep=' ')  # Merge all posts of this
        wordcloud = WordCloud(width=800, height=400, background_color='white').generate(type_posts)  # Generate wor
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis('off')
        plt.title(mbti_type)

    plt.tight_layout()

    # Save the plot to a file
    plt.savefig('mbti_type_wordcloud.png', dpi=300, bbox_inches='tight')
    plt.show()

plot_wordcloud_for_mbti(mbti_data)
```

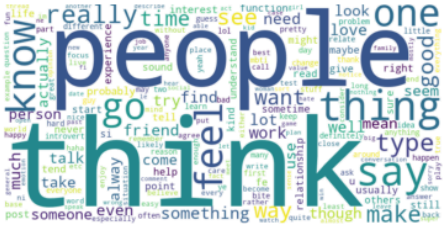
INTJ



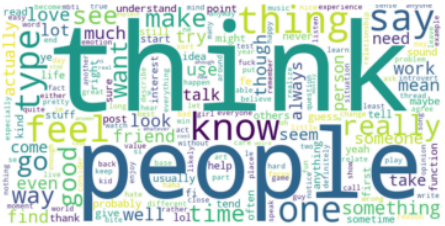
INTP



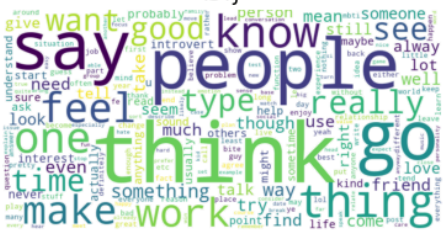
ISFJ



ISFP



ISTJ



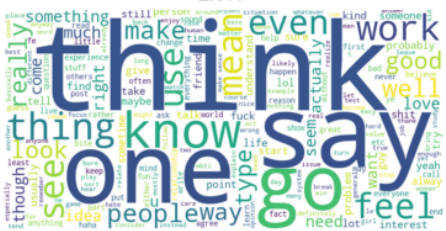
ISTP

ENFJ

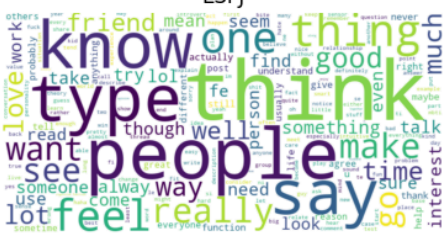
ENFP

ENTJ

ENTP



ESFJ



ESFP

ESTJ

ESTP

INFJ

INFP

Models

In [25]:
```
X = np.array(resampled_df['w2v_vector'].tolist())
y_columns = ['I/E', 'S/N', 'F/T', 'J/P']

results = {}

for y_col in y_columns:
    y = resampled_df[y_col].values

    # Split the data into training, validation, and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True)
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=42, shuffle=Tru

    X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
    X_val = X_val.reshape((X_val.shape[0], 1, X_val.shape[1]))
    X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

    # print(X_train.shape)
    # print(X_test.shape)
    # print(y_train.shape)
    # print(y_test.shape)
    # print(X_val.shape)
    # print(y_val.shape)

    # Define a function to create LSTM or GRU models
    def create_model(model_type):
        model = Sequential()
        if model_type == 'LSTM':
            model.add(Bidirectional(LSTM(128, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=Fa
        elif model_type == 'GRU':
            model.add(Bidirectional(GRU(128, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=Fal
        model.add(Dropout(0.5))
        model.add(Dense(64, activation='relu'))
```

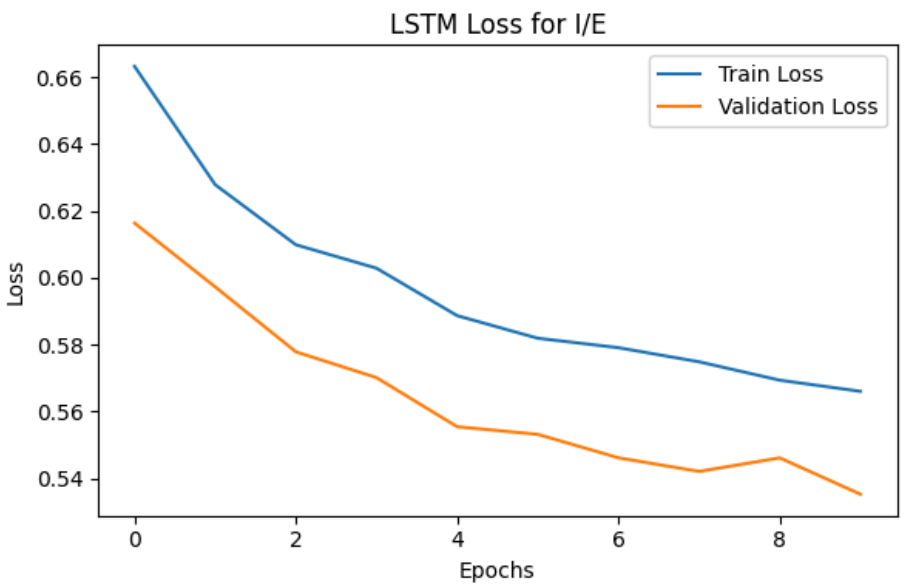
```
        model.add(Dropout(0.5))
        model.add(Dense(32, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))

        model.compile(optimizer=Adam(learning_rate=0.001),
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
        return model

    for model_type in ['LSTM', 'GRU']:
        # Create the model
        model = create_model(model_type)

        # Train the model
        start_time = time.time()
        history = model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=10,
            batch_size=32,
            shuffle=True,
            verbose=1
        )
        training_time = time.time() - start_time

        # Evaluate the model
        y_pred = (model.predict(X_test) > 0.5).astype("int32")
        test_accuracy = accuracy_score(y_test, y_pred)
        test_f1_score = f1_score(y_test, y_pred)
        classification_metrics = classification_report(y_test, y_pred, target_names=['Class 0', 'Class 1'], output_

        # Store results
        results[f"{y_col}_{model_type}"] = {
            "training_time": training_time,
            "test_accuracy": test_accuracy,
            "test_f1_score": test_f1_score,
            "classification_report": classification_metrics,
            "history": history.history
        }

        # Print evaluation results
```

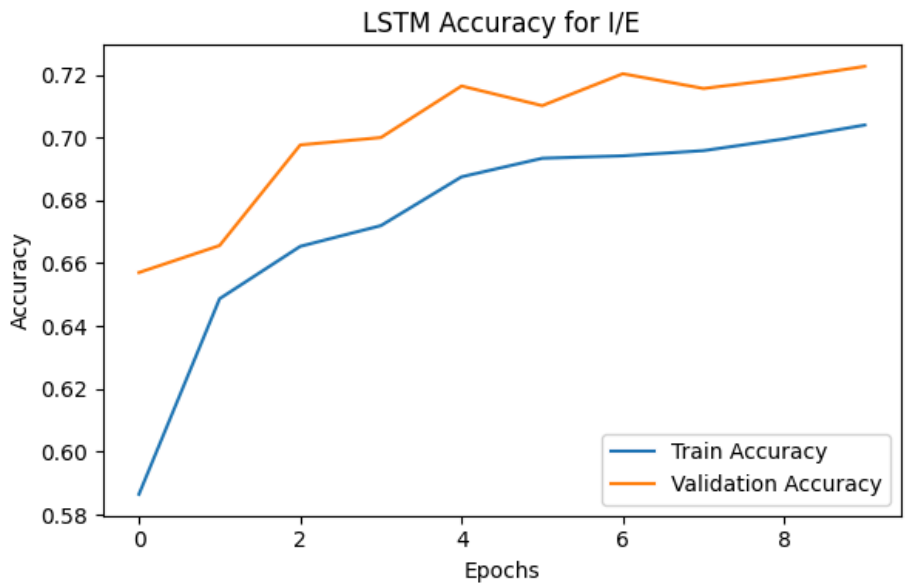
```python
        print(f"\nResults for Bidirectional {model_type} on {y_col}:")
        print(f"Training Time: {training_time:.2f} seconds")
        print(f"Test Accuracy: {test_accuracy:.4f}")
        print(f"Test F1 Score: {test_f1_score:.4f}")
        print("Classification Report:")
        print(classification_report(y_test, y_pred))

        # Plot loss and accuracy curves
        plt.figure(figsize=(12, 4))
        plt.subplot(1, 2, 1)
        plt.plot(history.history['loss'], label='Train Loss')
        plt.plot(history.history['val_loss'], label='Validation Loss')
        plt.title(f'{model_type} Loss for {y_col}')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()

        plt.subplot(1, 2, 2)
        plt.plot(history.history['accuracy'], label='Train Accuracy')
        plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
        plt.title(f'{model_type} Accuracy for {y_col}')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()

        plt.tight_layout()
        plt.show()

# Store final results
print("\nFinal Results Summary:")
for key, value in results.items():
    print(f"{key}: Test Accuracy: {value['test_accuracy']:.4f}, Training Time: {value['training_time']:.2f} seconds
```

```
Epoch 1/10
```

```
/Users/ytkuang/.venv/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

```
360/360 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5472 - loss: 0.6791 - val_accuracy: 0.6570 - val_loss: 0.6163
Epoch 2/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6432 - loss: 0.6309 - val_accuracy: 0.6656 - val_loss: 0.5973
Epoch 3/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6562 - loss: 0.6182 - val_accuracy: 0.6977 - val_loss: 0.5779
Epoch 4/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6718 - loss: 0.6015 - val_accuracy: 0.7000 - val_loss: 0.5702
Epoch 5/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.6899 - loss: 0.5844 - val_accuracy: 0.7164 - val_loss: 0.5555
Epoch 6/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6976 - loss: 0.5793 - val_accuracy: 0.7102 - val_loss: 0.5532
Epoch 7/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6951 - loss: 0.5798 - val_accuracy: 0.7203 - val_loss: 0.5462
Epoch 8/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7001 - loss: 0.5722 - val_accuracy: 0.7156 - val_loss: 0.5421
Epoch 9/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7009 - loss: 0.5655 - val_accuracy: 0.7188 - val_loss: 0.5462
Epoch 10/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7004 - loss: 0.5692 - val_accuracy: 0.7227 - val_loss: 0.5353
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 742us/step

Results for Bidirectional LSTM on I/E:
Training Time: 16.47 seconds
Test Accuracy: 0.7178
Test F1 Score: 0.7033
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.76      0.73      1604
           1       0.74      0.67      0.70      1596

    accuracy                           0.72      3200
   macro avg       0.72      0.72      0.72      3200
weighted avg       0.72      0.72      0.72      3200
```

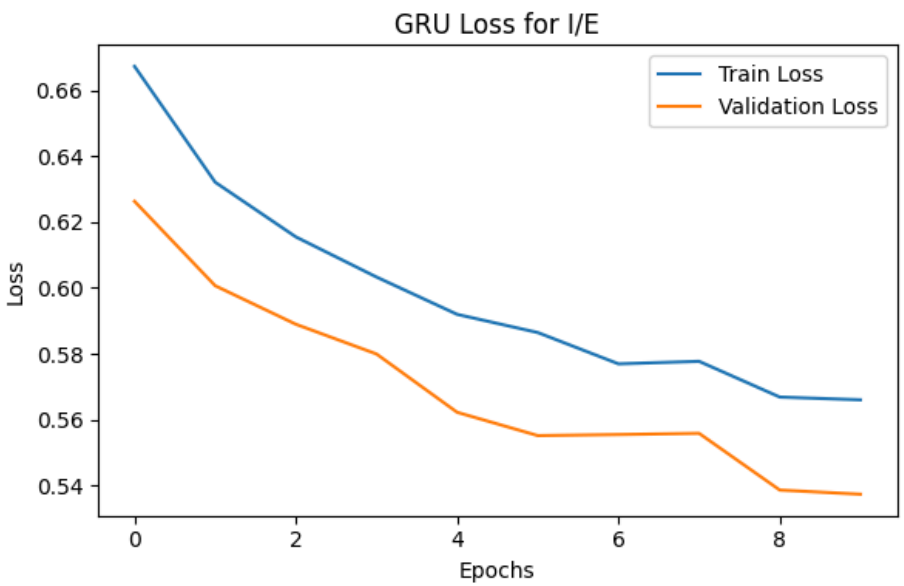


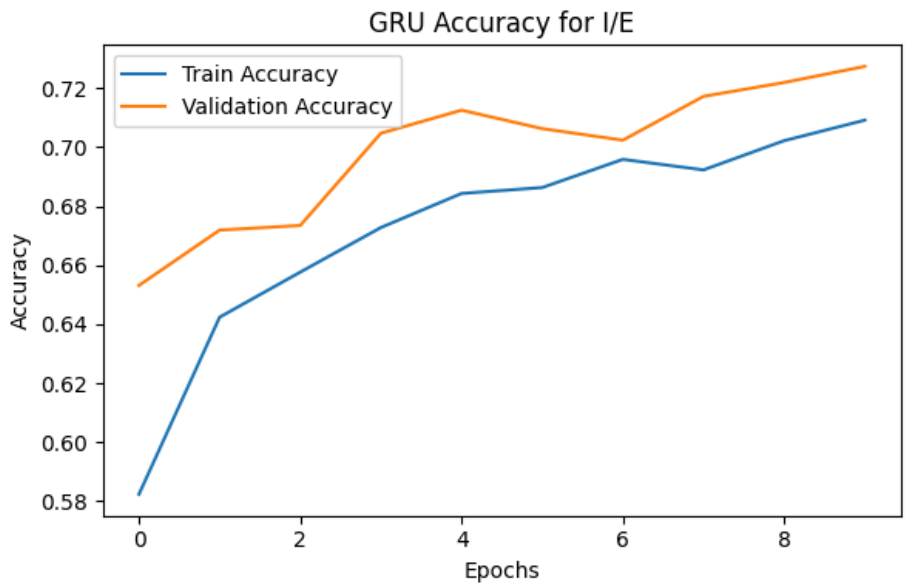



```
Epoch 1/10
```

```
/Users/ytkuang/.venv/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

```
360/360 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5476 - loss: 0.6828 - val_accuracy: 0.6531 - val_loss: 0.6263
Epoch 2/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6260 - loss: 0.6417 - val_accuracy: 0.6719 - val_loss: 0.6007
Epoch 3/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6472 - loss: 0.6237 - val_accuracy: 0.6734 - val_loss: 0.5890
Epoch 4/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6688 - loss: 0.6116 - val_accuracy: 0.7047 - val_loss: 0.5799
Epoch 5/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6823 - loss: 0.5924 - val_accuracy: 0.7125 - val_loss: 0.5623
Epoch 6/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6860 - loss: 0.5886 - val_accuracy: 0.7063 - val_loss: 0.5552
Epoch 7/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6977 - loss: 0.5766 - val_accuracy: 0.7023 - val_loss: 0.5555
Epoch 8/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6967 - loss: 0.5759 - val_accuracy: 0.7172 - val_loss: 0.5559
Epoch 9/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6936 - loss: 0.5708 - val_accuracy: 0.7219 - val_loss: 0.5387
Epoch 10/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.7059 - loss: 0.5716 - val_accuracy: 0.7273 - val_loss: 0.5374
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 894us/step

Results for Bidirectional GRU on I/E:
Training Time: 16.28 seconds
Test Accuracy: 0.7253
Test F1 Score: 0.7308
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.70      0.72      1604
           1       0.71      0.75      0.73      1596

    accuracy                           0.73      3200
   macro avg       0.73      0.73      0.73      3200
weighted avg       0.73      0.73      0.73      3200
```

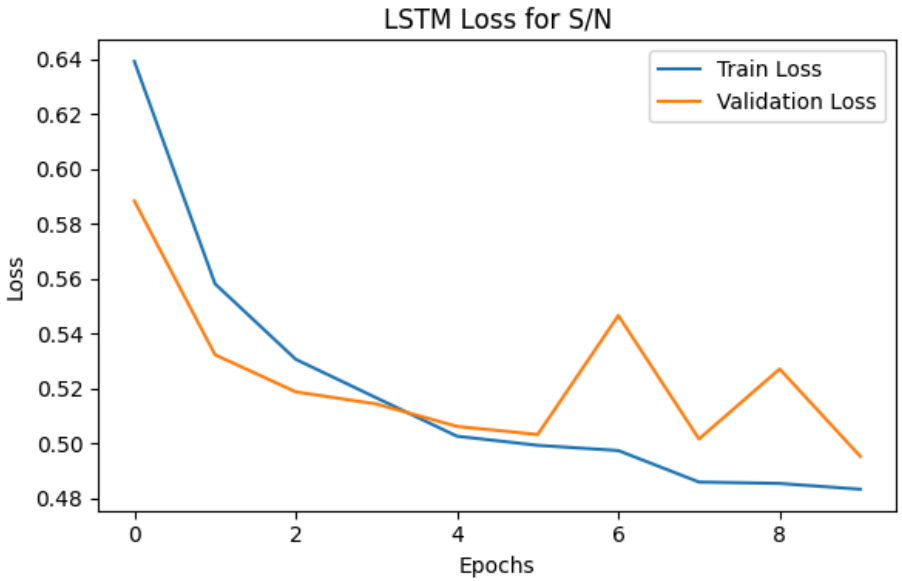


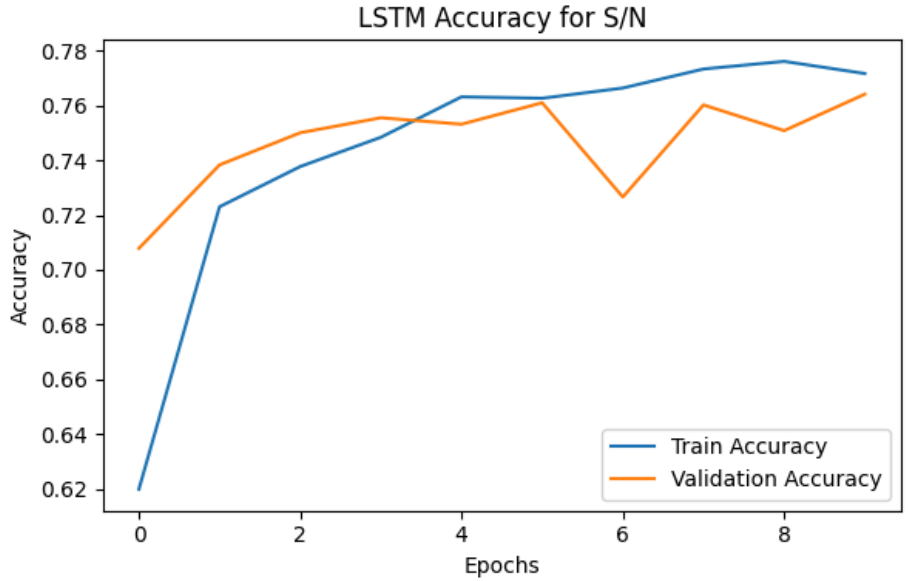



```
Epoch 1/10
```

```
/Users/ytkuang/.venv/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

```
360/360 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5622 - loss: 0.6705 - val_accuracy: 0.7078 - val_loss: 0.5884
Epoch 2/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7097 - loss: 0.5694 - val_accuracy: 0.7383 - val_loss: 0.5323
Epoch 3/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7324 - loss: 0.5351 - val_accuracy: 0.7500 - val_loss: 0.5188
Epoch 4/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7464 - loss: 0.5171 - val_accuracy: 0.7555 - val_loss: 0.5143
Epoch 5/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.7582 - loss: 0.5057 - val_accuracy: 0.7531 - val_loss: 0.5062
Epoch 6/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.7613 - loss: 0.4993 - val_accuracy: 0.7609 - val_loss: 0.5032
Epoch 7/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7683 - loss: 0.4957 - val_accuracy: 0.7266 - val_loss: 0.5466
Epoch 8/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7760 - loss: 0.4818 - val_accuracy: 0.7602 - val_loss: 0.5016
Epoch 9/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7769 - loss: 0.4811 - val_accuracy: 0.7508 - val_loss: 0.5271
Epoch 10/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.7623 - loss: 0.4944 - val_accuracy: 0.7641 - val_loss: 0.4952
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 713us/step

Results for Bidirectional LSTM on S/N:
Training Time: 20.69 seconds
Test Accuracy: 0.7788
Test F1 Score: 0.7759
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.78      0.78      1626
           1       0.77      0.78      0.78      1574

    accuracy                           0.78      3200
   macro avg       0.78      0.78      0.78      3200
weighted avg       0.78      0.78      0.78      3200
```

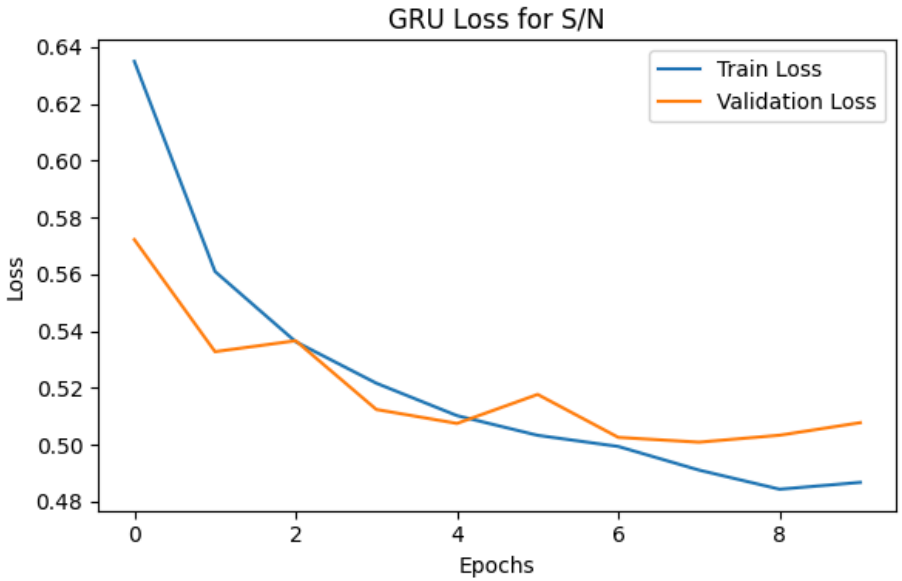


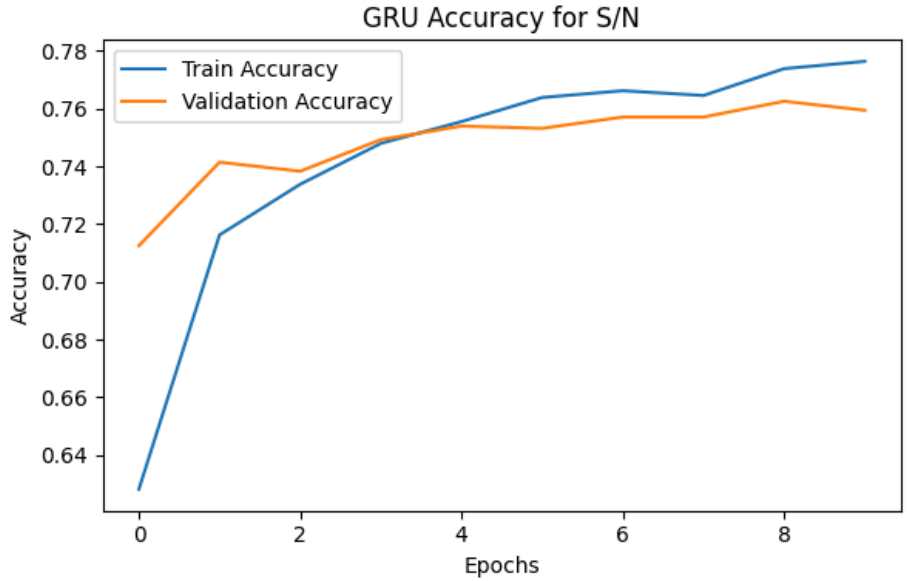



```
Epoch 1/10
```

```
/Users/ytkuang/.venv/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

```
360/360 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.5726 - loss: 0.6662 - val_accuracy: 0.7125 - val_loss: 0.5722
Epoch 2/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.7044 - loss: 0.5734 - val_accuracy: 0.7414 - val_loss: 0.5327
Epoch 3/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7274 - loss: 0.5463 - val_accuracy: 0.7383 - val_loss: 0.5366
Epoch 4/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7485 - loss: 0.5225 - val_accuracy: 0.7492 - val_loss: 0.5124
Epoch 5/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7589 - loss: 0.5057 - val_accuracy: 0.7539 - val_loss: 0.5075
Epoch 6/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7655 - loss: 0.5040 - val_accuracy: 0.7531 - val_loss: 0.5177
Epoch 7/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7707 - loss: 0.4959 - val_accuracy: 0.7570 - val_loss: 0.5026
Epoch 8/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7571 - loss: 0.4997 - val_accuracy: 0.7570 - val_loss: 0.5009
Epoch 9/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7709 - loss: 0.4878 - val_accuracy: 0.7625 - val_loss: 0.5033
Epoch 10/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7744 - loss: 0.4926 - val_accuracy: 0.7594 - val_loss: 0.5077
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 716us/step

Results for Bidirectional GRU on S/N:
Training Time: 18.00 seconds
Test Accuracy: 0.7728
Test F1 Score: 0.7494
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.85      0.79      1626
           1       0.82      0.69      0.75      1574

    accuracy                           0.77      3200
   macro avg       0.78      0.77      0.77      3200
weighted avg       0.78      0.77      0.77      3200
```

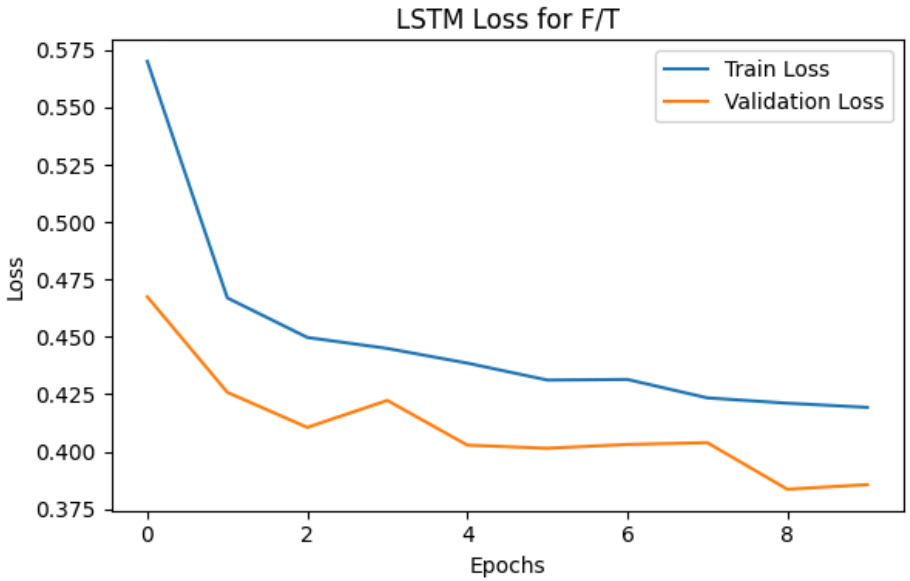


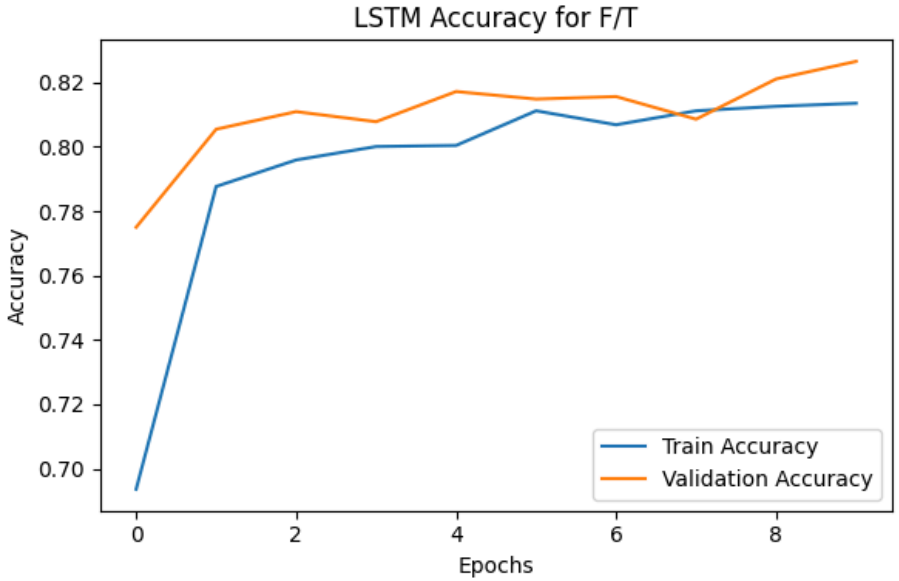



```
Epoch 1/10
```

```
/Users/ytkuang/.venv/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

```
360/360 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6147 - loss: 0.6366 - val_accuracy: 0.7750 - val_loss: 0.4675
Epoch 2/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.7876 - loss: 0.4649 - val_accuracy: 0.8055 - val_loss: 0.4258
Epoch 3/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7977 - loss: 0.4524 - val_accuracy: 0.8109 - val_loss: 0.4105
Epoch 4/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.7943 - loss: 0.4481 - val_accuracy: 0.8078 - val_loss: 0.4223
Epoch 5/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7985 - loss: 0.4411 - val_accuracy: 0.8172 - val_loss: 0.4028
Epoch 6/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8076 - loss: 0.4351 - val_accuracy: 0.8148 - val_loss: 0.4015
Epoch 7/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8068 - loss: 0.4313 - val_accuracy: 0.8156 - val_loss: 0.4031
Epoch 8/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8183 - loss: 0.4108 - val_accuracy: 0.8086 - val_loss: 0.4039
Epoch 9/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8159 - loss: 0.4120 - val_accuracy: 0.8211 - val_loss: 0.3836
Epoch 10/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8129 - loss: 0.4210 - val_accuracy: 0.8266 - val_loss: 0.3856
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 767us/step

Results for Bidirectional LSTM on F/T:
Training Time: 20.53 seconds
Test Accuracy: 0.8281
Test F1 Score: 0.8381
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.77      0.82      1584
           1       0.80      0.88      0.84      1616

    accuracy                           0.83      3200
   macro avg       0.83      0.83      0.83      3200
weighted avg       0.83      0.83      0.83      3200
```

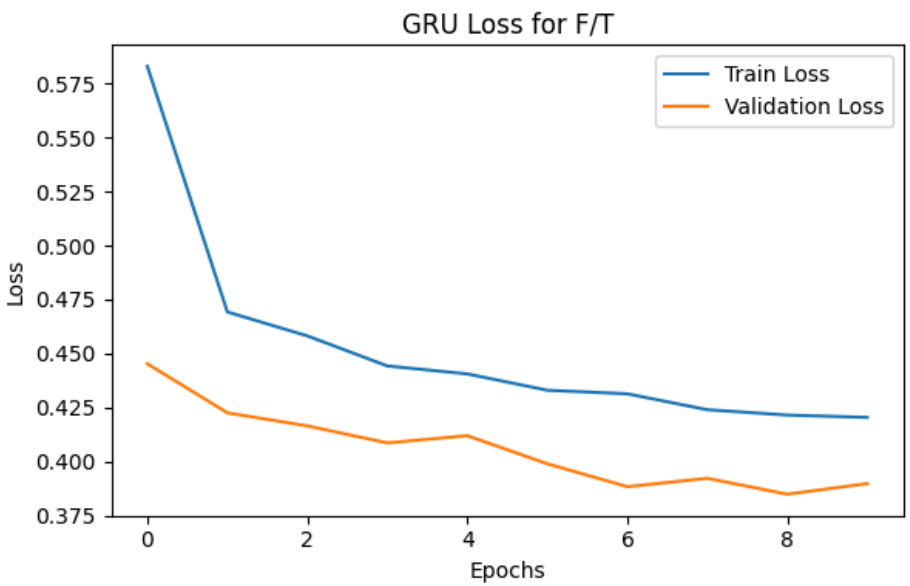


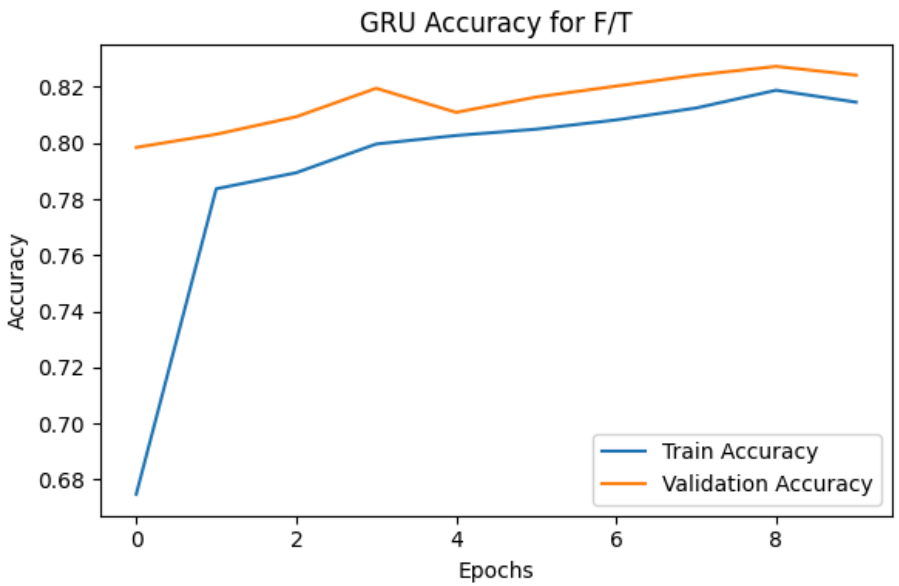



```
Epoch 1/10
```

```
/Users/ytkuang/.venv/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

```
360/360 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5969 - loss: 0.6495 - val_accuracy: 0.7984 - val_loss: 0.4452
Epoch 2/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7770 - loss: 0.4747 - val_accuracy: 0.8031 - val_loss: 0.4225
Epoch 3/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7859 - loss: 0.4627 - val_accuracy: 0.8094 - val_loss: 0.4165
Epoch 4/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7942 - loss: 0.4547 - val_accuracy: 0.8195 - val_loss: 0.4086
Epoch 5/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7964 - loss: 0.4501 - val_accuracy: 0.8109 - val_loss: 0.4120
Epoch 6/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8038 - loss: 0.4357 - val_accuracy: 0.8164 - val_loss: 0.3989
Epoch 7/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8135 - loss: 0.4238 - val_accuracy: 0.8203 - val_loss: 0.3883
Epoch 8/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8166 - loss: 0.4214 - val_accuracy: 0.8242 - val_loss: 0.3922
Epoch 9/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8246 - loss: 0.4147 - val_accuracy: 0.8273 - val_loss: 0.3849
Epoch 10/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8175 - loss: 0.4172 - val_accuracy: 0.8242 - val_loss: 0.3898
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step

Results for Bidirectional GRU on F/T:
Training Time: 15.44 seconds
Test Accuracy: 0.8250
Test F1 Score: 0.8235
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.84      0.83      1584
           1       0.84      0.81      0.82      1616

    accuracy                           0.82      3200
   macro avg       0.83      0.83      0.82      3200
weighted avg       0.83      0.82      0.82      3200
```

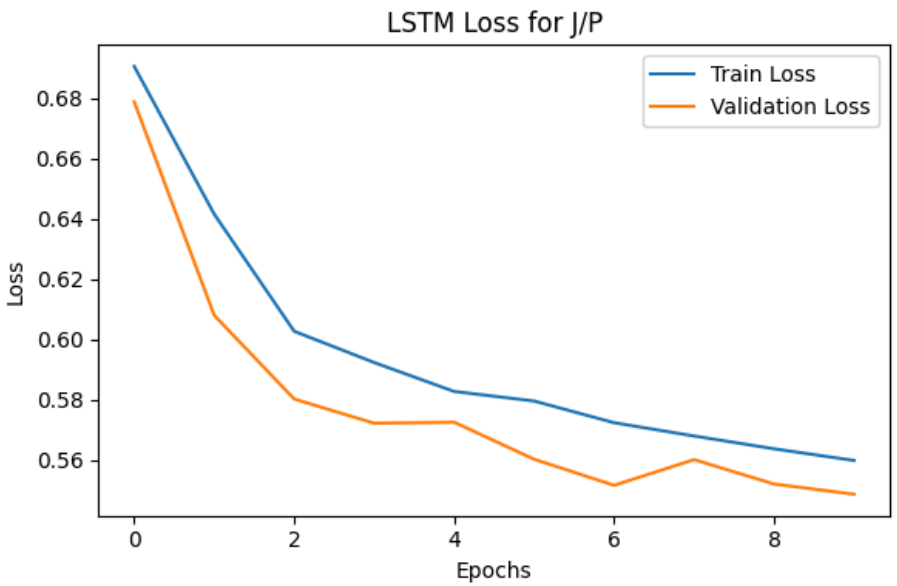


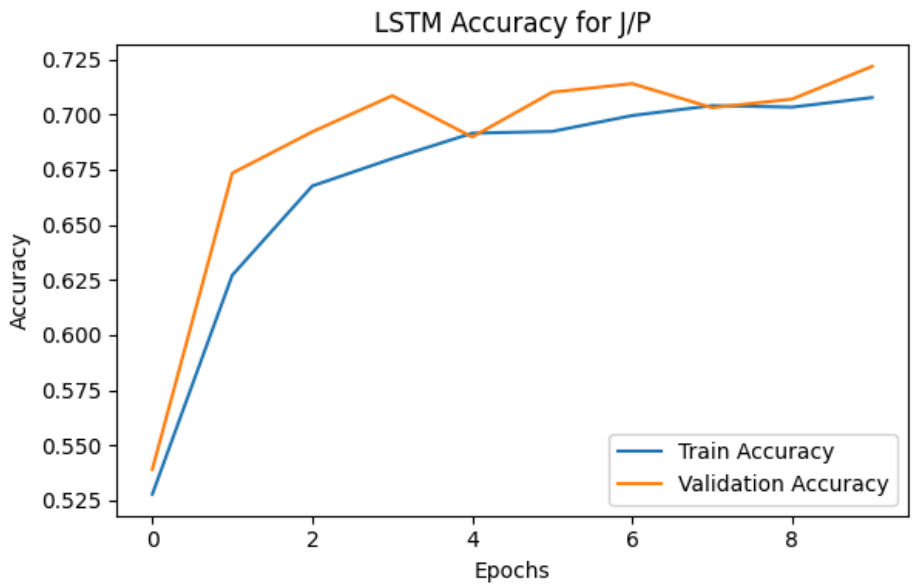



```
Epoch 1/10
```

```
/Users/ytkuang/.venv/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

```
360/360 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.5151 - loss: 0.6925 - val_accuracy: 0.5391 - val_loss: 0.6788
Epoch 2/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6020 - loss: 0.6590 - val_accuracy: 0.6734 - val_loss: 0.6079
Epoch 3/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6601 - loss: 0.6063 - val_accuracy: 0.6922 - val_loss: 0.5801
Epoch 4/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6771 - loss: 0.5916 - val_accuracy: 0.7086 - val_loss: 0.5721
Epoch 5/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6922 - loss: 0.5767 - val_accuracy: 0.6898 - val_loss: 0.5724
Epoch 6/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6889 - loss: 0.5861 - val_accuracy: 0.7102 - val_loss: 0.5601
Epoch 7/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6993 - loss: 0.5752 - val_accuracy: 0.7141 - val_loss: 0.5515
Epoch 8/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6987 - loss: 0.5682 - val_accuracy: 0.7031 - val_loss: 0.5600
Epoch 9/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7078 - loss: 0.5590 - val_accuracy: 0.7070 - val_loss: 0.5519
Epoch 10/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.7089 - loss: 0.5567 - val_accuracy: 0.7219 - val_loss: 0.5485
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step

Results for Bidirectional LSTM on J/P:
Training Time: 18.57 seconds
Test Accuracy: 0.7041
Test F1 Score: 0.7326
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.61      0.67      1578
           1       0.68      0.80      0.73      1622

    accuracy                           0.70      3200
   macro avg       0.71      0.70      0.70      3200
weighted avg       0.71      0.70      0.70      3200
```

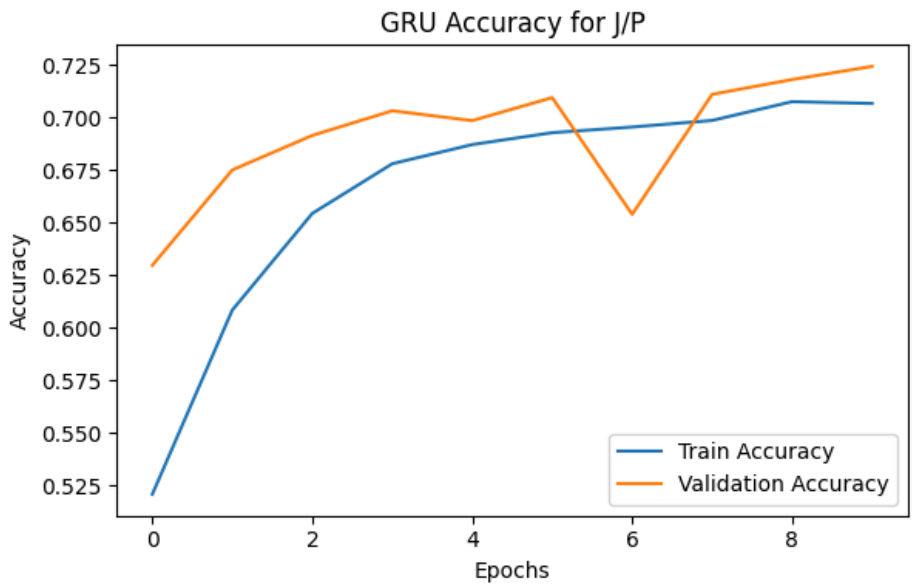





```
Epoch 1/10
```

```
/Users/ytkuang/.venv/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

```
360/360 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5029 - loss: 0.6934 - val_accuracy: 0.6297 - val_loss: 0.6716
Epoch 2/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5903 - loss: 0.6655 - val_accuracy: 0.6750 - val_loss: 0.6089
Epoch 3/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6505 - loss: 0.6183 - val_accuracy: 0.6914 - val_loss: 0.5852
Epoch 4/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6753 - loss: 0.5959 - val_accuracy: 0.7031 - val_loss: 0.5688
Epoch 5/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6895 - loss: 0.5833 - val_accuracy: 0.6984 - val_loss: 0.5669
Epoch 6/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6918 - loss: 0.5746 - val_accuracy: 0.7094 - val_loss: 0.5623
Epoch 7/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6958 - loss: 0.5759 - val_accuracy: 0.6539 - val_loss: 0.6018
Epoch 8/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6984 - loss: 0.5737 - val_accuracy: 0.7109 - val_loss: 0.5662
Epoch 9/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7100 - loss: 0.5613 - val_accuracy: 0.7180 - val_loss: 0.5539
Epoch 10/10
360/360 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7087 - loss: 0.5569 - val_accuracy: 0.7242 - val_loss: 0.5414
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 845us/step

Results for Bidirectional GRU on J/P:
Training Time: 14.29 seconds
Test Accuracy: 0.7072
Test F1 Score: 0.7002
Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.74      0.71      1578
           1       0.73      0.67      0.70      1622

    accuracy                           0.71      3200
   macro avg       0.71      0.71      0.71      3200
weighted avg       0.71      0.71      0.71      3200
```




```
Final Results Summary:
I/E_LSTM: Test Accuracy: 0.7178, Training Time: 16.47 seconds
I/E_GRU: Test Accuracy: 0.7253, Training Time: 16.28 seconds
S/N_LSTM: Test Accuracy: 0.7788, Training Time: 20.69 seconds
S/N_GRU: Test Accuracy: 0.7728, Training Time: 18.00 seconds
F/T_LSTM: Test Accuracy: 0.8281, Training Time: 20.53 seconds
F/T_GRU: Test Accuracy: 0.8250, Training Time: 15.44 seconds
J/P_LSTM: Test Accuracy: 0.7041, Training Time: 18.57 seconds
J/P_GRU: Test Accuracy: 0.7072, Training Time: 14.29 seconds
```