True vs Fake News Text Classification

3.1 Text Preprocessing, Tokenization, and Sequence Padding:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Import Required Libraries:

```
# Import libraries for data manipulation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Word cloud for data visualization
from wordcloud import WordCloud


# Regular expressions and string handling
import re
import string

# Natural Language Toolkit (NLTK) for text processing
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download required NLTK resources
nltk.download('punkt')           # Tokenizer models
nltk.download('punkt_tab')
nltk.download('stopwords')       # Stopword list
nltk.download('wordnet')         # Lemmatizer dictionary
nltk.download('omw-1.4')         # Lemmatizer wordnet data
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
True
```

```
!pip install -U numpy gensim
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Collecting numpy
  Downloading numpy-2.2.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 4.2 MB/s eta 0:00:00
Collecting gensim
  Downloading gensim-4.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.1 kB)
Collecting numpy
  Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 5.6 MB/s eta 0:00:00
Collecting scipy<1.14.0,>=1.7.0 (from gensim)
  Downloading scipy-1.13.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.6/60.6 kB 5.8 MB/s eta 0:00:00
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.17.2)
Downloading gensim-4.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (26.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.7/26.7 MB 84.1 MB/s eta 0:00:00
Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 95.5 MB/s eta 0:00:00
Downloading scipy-1.13.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (38.6 MB)
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.6/38.6 MB 18.5 MB/s eta 0:00:00
Installing collected packages: numpy, scipy, gensim
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: scipy
    Found existing installation: scipy 1.15.2
    Uninstalling scipy-1.15.2:
      Successfully uninstalled scipy-1.15.2
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is incompatible.
tsfresh 0.21.0 requires scipy>=1.14.0; python_version >= "3.10", but you have scipy 1.13.1 which is incompatible.
Successfully installed gensim-4.3.3 numpy-1.26.4 scipy-1.13.1
```

Load the dataset:

```
# Load the dataset
file_path = '/content/drive/MyDrive/AI ML/10.True vs. Fake News Dataset/truevsfakenews.csv'
data = pd.read_csv(file_path)
data.head()
```

	text	label
0	WASHINGTON (Reuters) - The Republican and Demo...	true
1	Women should get as far away from Oklahoma as ...	fake
2	Another huge crowd of Americans tuned in last ...	fake
3	Donald Trump is desperate to stop the investig...	fake
4	(Reuters) - Planned Parenthood, the U.S. medic...	true

Clean the text:

```
# Drop missing values
data = data[['text', 'label']].dropna()

# Normalize labels to lowercase
data['label'] = data['label'].str.lower()
```

Define Preprocessing Function

```
# Initialize lemmatizer and stopwords
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Contraction mapping
contractions_dict = {
    "don't": "do not", "doesn't": "does not", "didn't": "did not",
    "can't": "cannot", "won't": "will not", "shouldn't": "should not",
    "isn't": "is not", "aren't": "are not", "wasn't": "was not",
    "weren't": "were not", "hasn't": "has not", "haven't": "have not",
    "hadn't": "had not", "mightn't": "might not", "mustn't": "must not",
    "i'm": "i am", "you're": "you are", "he's": "he is", "she's": "she is",
    "it's": "it is", "we're": "we are", "they're": "they are",
    "i've": "i have", "you've": "you have", "we've": "we have",
    "they've": "they have", "i'll": "i will", "you'll": "you will",
    "he'll": "he will", "she'll": "she will", "we'll": "we will",
    "they'll": "they will"
}

# Compile regex pattern for contractions
contraction_pattern = re.compile(
    r'\b({})\b'.format('|'.join(re.escape(k) for k in contractions_dict.keys())),
    flags=re.IGNORECASE
)

# Function to fix Unicode apostrophe replacements like â€™
def fix_unicode_artifacts(text):
    # Replace common Unicode misinterpretations
    text = text.replace("â€™", "'") \
               .replace("â€œ", "“") \
               .replace("â€", "”") \
               .replace("â€˜", "‘") \
               .replace("â€“", "–") \
               .replace("â€”", "—") \
               .replace("â‚¬", "€") \
               .replace("Â", "")  # Removes stray Â characters
    return text

# Expand contractions
def expand_contractions(text):
    def replace(match):
        word = match.group(0)
        return contractions_dict.get(word.lower(), word)
    return contraction_pattern.sub(replace, text)

# Full preprocessing function
def preprocess_text(text):
    # Step 1: Fix Unicode artifacts
    text = fix_unicode_artifacts(text)

    # Step 2: Lowercase
    text = text.lower()

    # Step 3: Expand contractions
    text = expand_contractions(text)

    # Step 4: Remove URLs
    text = re.sub(r'https?://\S+|www\.\S+', '', text)

    # Step 5: Remove @mentions and #hashtags
    text = re.sub(r'@\w+|#', '', text)

    # Step 6: Remove punctuation (except apostrophes inside words)
    text = re.sub(r'[^\w\s\']', '', text)

    # Step 7: Remove numbers
    text = re.sub(r'\d+', '', text)

    # Step 8: Tokenize and remove stopwords
    tokens = text.split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]

    return ' '.join(tokens)


# Sample raw text
sample_text = [
    "TBILISI (Reuters) - The Trump Organization pulled out of a $250-million real estate project in ex-Soviet Georgia to avoid a potential c
]

# Apply cleaning
for i, text in enumerate(sample_text):
    cleaned = preprocess_text(text)
    print("Original :", text)
    print("Cleaned  :", cleaned)
    print()
```

```
Original : TBILISI (Reuters) - The Trump Organization pulled out of a $250-million real estate project in ex-Soviet Georgia to avoid a p
Cleaned  : tbilisi reuters trump organization pulled million real estate project exsoviet georgia avoid potential conflict donald trump'
```

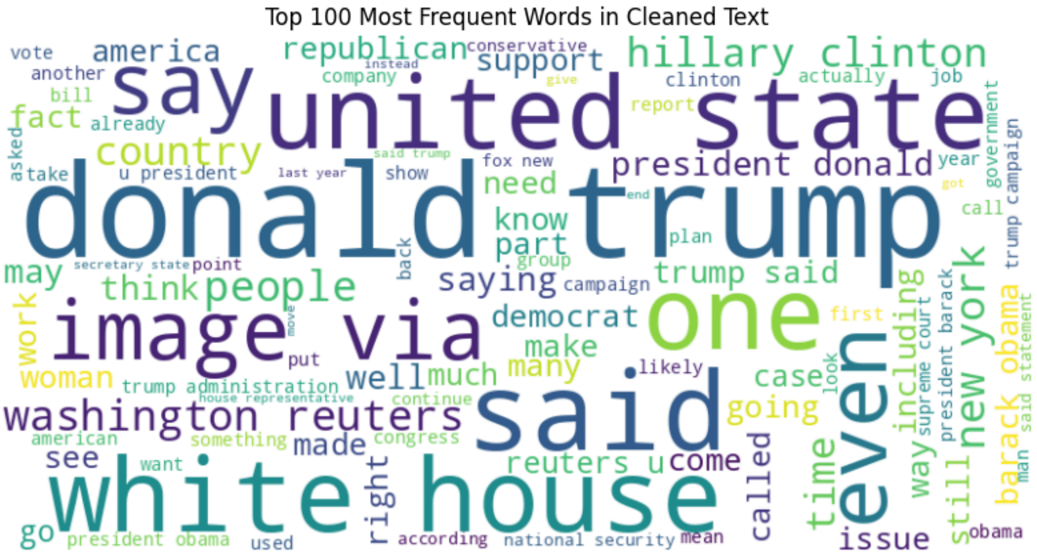
Apply Preprocessing to Dataset

```
# Apply preprocessing
data['cleaned_text'] = data['text'].apply(preprocess_text)
```

Visualize Cleaned Data Using Word Cloud

```
# Generate word cloud from cleaned text
all_words = ' '.join(data['cleaned_text'])
```

```
wordcloud = WordCloud(width=800, height=400, background_color='white', max_words=100).generate(all_words)

plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Top 100 Most Frequent Words in Cleaned Text')
plt.show()
```



Train/Test Split

```
from sklearn.model_selection import train_test_split

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    data['cleaned_text'],
    data['label'],
    test_size=0.2,
    random_state=42,
    stratify=data['label']
)
```

Tokenization

```
from tensorflow.keras.preprocessing.text import Tokenizer
import numpy as np


# Initialize tokenizer
tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')
tokenizer.fit_on_texts(X_train)  # Only fit on training data

# Convert texts to sequences
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)
```

Padding Sequences Based on 95th Percentile

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Calculate sequence lengths
seq_lengths = [len(seq) for seq in X_train_seq]

# Get 95th percentile of sequence lengths
max_len = int(np.percentile(seq_lengths, 95))
print(f"Padding sequences to maximum length: {max_len}")

# Pad sequences
```

```
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len, padding='post', truncating='post')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post', truncating='post')

# Check shapes
print(f"\nTraining data shape after padding: {X_train_pad.shape}")
print(f"Testing data shape after padding: {X_test_pad.shape}")
```

```
Padding sequences to maximum length: 494

Training data shape after padding: (16000, 494)
Testing data shape after padding: (4000, 494)
```

Label Encoding

```
from sklearn.preprocessing import LabelEncoder

# Encode string labels to integers
le = LabelEncoder()
y_train_enc = le.fit_transform(y_train)
y_test_enc = le.transform(y_test)

# Get number of classes (should be 2 for binary classification)
num_classes = len(le.classes_)  # Output: 2
print(f"Number of classes: {num_classes}")
print(f"Class names: {list(le.classes_)}")
```

```
Number of classes: 2
Class names: ['fake', 'true']
```

3.2 Model Building and Training

Import Required Libraries

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, Dense
from tensorflow.keras.initializers import Constant
import gensim.downloader as api
import numpy as np
```

MODEL 1: Simple RNN with Trainable Embedding

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

model_rnn = Sequential()

# Embedding Layer
model_rnn.add(Embedding(
    input_dim=10000,
    output_dim=128,
    input_length=max_len
))

# Recurrent Layer
model_rnn.add(SimpleRNN(units=64))

# Output Layer
model_rnn.add(Dense(1, activation='sigmoid'))

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

# Build the model manually by specifying the input shape
model_rnn.build(input_shape=(None, X_train_pad.shape[1]))
```

```
# Print Model Summary
model_rnn.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just
  warnings.warn(
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding (Embedding)           │ (None, 494, 128)       │     1,280,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ simple_rnn (SimpleRNN)          │ (None, 64)             │        12,352 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 1)              │            65 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 1,292,417 (4.93 MB)
 Trainable params: 1,292,417 (4.93 MB)
```

MODEL 2: LSTM with Trainable Embedding

```
from tensorflow.keras.layers import LSTM

model_lstm = Sequential()

# Embedding Layer
model_lstm.add(Embedding(
    input_dim=10000,
    output_dim=128,
    input_length=max_len
))

# LSTM Layer
model_lstm.add(LSTM(units=64))

# Output Layer
model_lstm.add(Dense(1, activation='sigmoid'))

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

# Build the model
model_lstm.build(input_shape=(None, X_train_pad.shape[1]))

model_lstm.summary()
```

```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding_1 (Embedding)         │ (None, 494, 128)       │     1,280,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm (LSTM)                     │ (None, 64)             │        49,408 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)              │            65 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 1,329,473 (5.07 MB)
 Trainable params: 1,329,473 (5.07 MB)
 Non-trainable params: 0 (0.00 B)
```

MODEL 3: LSTM with Pretrained Word2Vec Embeddings

Download and load a pretrained Word2Vec model

```
embedding_model = api.load("glove-wiki-gigaword-50")  # 50-dimensional vectors
embedding_dim = 50
```

```
[==================================================] 100.0% 66.0/66.0MB downloaded
```

Create Embedding Matrix

```
vocab_size = min(len(tokenizer.word_index) + 1, 11000)  # Cap at 10,000 words
embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in tokenizer.word_index.items():
    if i >= 10000:
        continue
    if word in embedding_model:
        embedding_vector = embedding_model[word]
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

Define LSTM Model Using Word2Vec Embeddings

```
from tensorflow.keras.layers import Embedding
from tensorflow.keras.initializers import Constant

model_lstm_word2vec = Sequential()

# Embedding Layer with Pretrained Weights
model_lstm_word2vec.add(Embedding(
    input_dim=vocab_size,
    output_dim=embedding_dim,
    input_length=max_len,
    embeddings_initializer=Constant(embedding_matrix),  # Use pretrained weights
    trainable=False  # Freeze embeddings unless fine-tuning
))

# LSTM Layer
model_lstm_word2vec.add(LSTM(units=64))

# Output Layer
model_lstm_word2vec.add(Dense(1, activation='sigmoid'))

# Compile
model_lstm_word2vec.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

# Build the model manually to initialize layers
model_lstm_word2vec.build(input_shape=(None, X_train_pad.shape[1]))

model_lstm_word2vec.summary()
```

```
Model: "sequential_6"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding_6 (Embedding)         │ (None, 494, 50)        │       550,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_5 (LSTM)                   │ (None, 64)             │        29,440 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_6 (Dense)                 │ (None, 1)              │            65 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 579,505 (2.21 MB)
 Trainable params: 29,505 (115.25 KB)
 Non-trainable params: 550,000 (2.10 MB)
```

3.3 Model Training and Evaluation:

Define Callbacks

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

Train Simple RNN

```
history_rnn = model_rnn.fit(
    X_train_pad,
    y_train_enc,
    validation_split=0.2,
    epochs=20,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 15s 37ms/step - accuracy: 0.5266 - loss: 0.6704 - val_accuracy: 0.4956 - val_loss: 0.6858
Epoch 2/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 15s 37ms/step - accuracy: 0.5368 - loss: 0.6604 - val_accuracy: 0.4959 - val_loss: 0.6859
Epoch 3/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 15s 38ms/step - accuracy: 0.5373 - loss: 0.6603 - val_accuracy: 0.5344 - val_loss: 0.6848
Epoch 4/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 21s 38ms/step - accuracy: 0.5311 - loss: 0.6590 - val_accuracy: 0.5312 - val_loss: 0.6909
Epoch 5/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 20s 38ms/step - accuracy: 0.5356 - loss: 0.6598 - val_accuracy: 0.5316 - val_loss: 0.6865
Epoch 6/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 15s 38ms/step - accuracy: 0.5370 - loss: 0.6603 - val_accuracy: 0.4906 - val_loss: 0.6884
Epoch 7/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 20s 37ms/step - accuracy: 0.5371 - loss: 0.6605 - val_accuracy: 0.5319 - val_loss: 0.6877
Epoch 8/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 21s 38ms/step - accuracy: 0.5405 - loss: 0.6593 - val_accuracy: 0.5312 - val_loss: 0.6877
```

Train LSTM

```
history_lstm = model_lstm.fit(
    X_train_pad,
    y_train_enc,
    validation_split=0.2,
    epochs=20,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 11s 20ms/step - accuracy: 0.5334 - loss: 0.6899 - val_accuracy: 0.5456 - val_loss: 0.6722
Epoch 2/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - accuracy: 0.5387 - loss: 0.6716 - val_accuracy: 0.5000 - val_loss: 0.6690
Epoch 3/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 21ms/step - accuracy: 0.5617 - loss: 0.6618 - val_accuracy: 0.7881 - val_loss: 0.4676
Epoch 4/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.8980 - loss: 0.3155 - val_accuracy: 0.9684 - val_loss: 0.1089
Epoch 5/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.9870 - loss: 0.0569 - val_accuracy: 0.9669 - val_loss: 0.1048
Epoch 6/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9923 - loss: 0.0377 - val_accuracy: 0.9787 - val_loss: 0.0870
Epoch 7/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 21ms/step - accuracy: 0.9956 - loss: 0.0211 - val_accuracy: 0.9816 - val_loss: 0.0630
Epoch 8/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - accuracy: 0.9985 - loss: 0.0067 - val_accuracy: 0.9825 - val_loss: 0.0655
Epoch 9/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.9995 - loss: 0.0034 - val_accuracy: 0.9841 - val_loss: 0.0692
Epoch 10/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 21ms/step - accuracy: 0.9998 - loss: 0.0018 - val_accuracy: 0.9850 - val_loss: 0.0589
Epoch 11/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.9995 - loss: 0.0034 - val_accuracy: 0.9881 - val_loss: 0.0618
Epoch 12/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - accuracy: 0.9997 - loss: 0.0020 - val_accuracy: 0.9866 - val_loss: 0.0731
Epoch 13/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 1.0000 - loss: 3.1964e-04 - val_accuracy: 0.9819 - val_loss: 0.1102
Epoch 14/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 21ms/step - accuracy: 0.9980 - loss: 0.0073 - val_accuracy: 0.9787 - val_loss: 0.0948
Epoch 15/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.9990 - loss: 0.0028 - val_accuracy: 0.9831 - val_loss: 0.0787
```

Train LSTM with Word2Vec

```
history_word2vec = model_lstm_word2vec.fit(
    X_train_pad
    X_train_pad,
    y_train_enc,
    validation_split=0.2,
    epochs=20,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

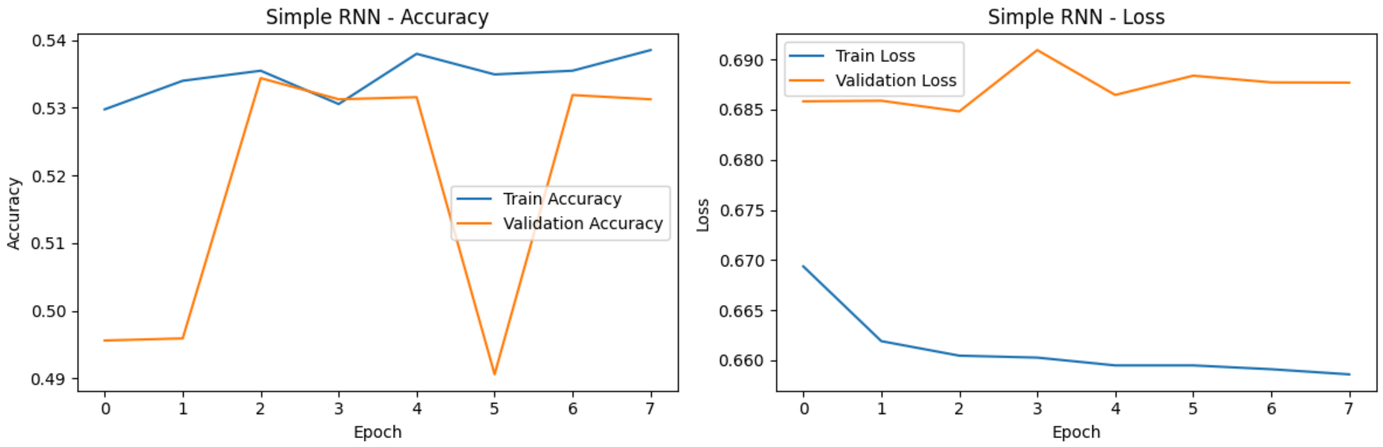
```
Epoch 1/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - accuracy: 0.5689 - loss: 0.6584 - val_accuracy: 0.6194 - val_loss: 0.5873
Epoch 2/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.7218 - loss: 0.4937 - val_accuracy: 0.9328 - val_loss: 0.2881
Epoch 3/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - accuracy: 0.7648 - loss: 0.4771 - val_accuracy: 0.5534 - val_loss: 0.6689
Epoch 4/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.6393 - loss: 0.5924 - val_accuracy: 0.9137 - val_loss: 0.2593
Epoch 5/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 12s 22ms/step - accuracy: 0.9129 - loss: 0.2562 - val_accuracy: 0.9331 - val_loss: 0.2244
Epoch 6/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - accuracy: 0.9375 - loss: 0.2140 - val_accuracy: 0.9344 - val_loss: 0.2367
Epoch 7/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.9227 - loss: 0.2494 - val_accuracy: 0.9247 - val_loss: 0.2353
Epoch 8/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.9223 - loss: 0.2388 - val_accuracy: 0.9337 - val_loss: 0.2214
Epoch 9/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9428 - loss: 0.2078 - val_accuracy: 0.9481 - val_loss: 0.1996
Epoch 10/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9485 - loss: 0.1939 - val_accuracy: 0.9319 - val_loss: 0.2269
Epoch 11/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.9393 - loss: 0.2082 - val_accuracy: 0.9294 - val_loss: 0.2268
Epoch 12/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 11s 19ms/step - accuracy: 0.9353 - loss: 0.2155 - val_accuracy: 0.9500 - val_loss: 0.1950
Epoch 13/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9247 - loss: 0.2452 - val_accuracy: 0.9475 - val_loss: 0.1927
Epoch 14/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9526 - loss: 0.1810 - val_accuracy: 0.9569 - val_loss: 0.1791
Epoch 15/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.9543 - loss: 0.1750 - val_accuracy: 0.9575 - val_loss: 0.1715
Epoch 16/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9549 - loss: 0.1734 - val_accuracy: 0.9569 - val_loss: 0.1669
Epoch 17/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 11s 21ms/step - accuracy: 0.8995 - loss: 0.3141 - val_accuracy: 0.6031 - val_loss: 0.6326
Epoch 18/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 9s 19ms/step - accuracy: 0.7323 - loss: 0.5014 - val_accuracy: 0.9228 - val_loss: 0.2373
Epoch 19/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - accuracy: 0.8281 - loss: 0.3568 - val_accuracy: 0.9397 - val_loss: 0.2018
Epoch 20/20
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9401 - loss: 0.1987 - val_accuracy: 0.9463 - val_loss: 0.1891
```

Plot Training vs Validation Accuracy & Loss

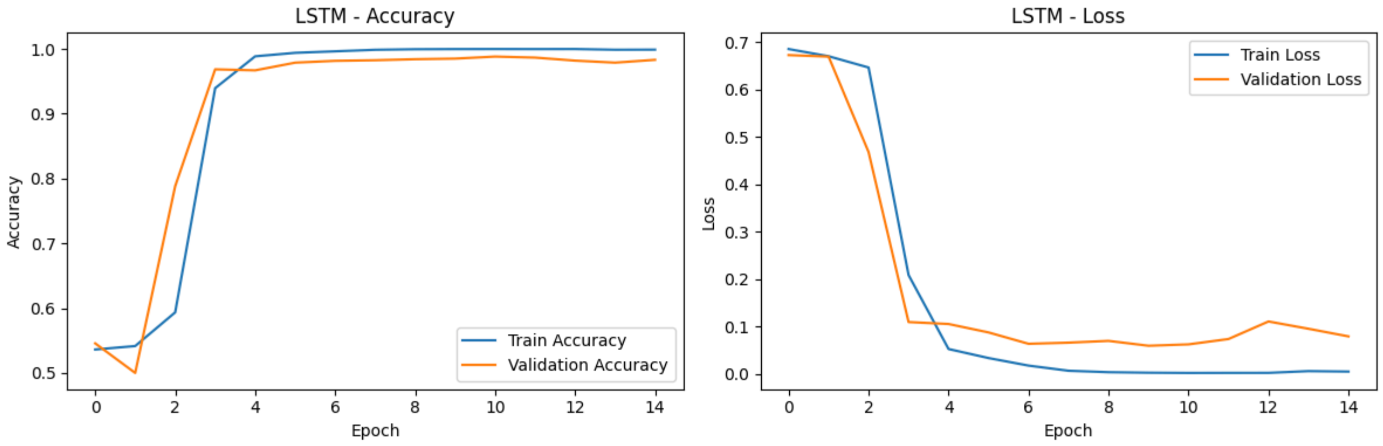
```
import matplotlib.pyplot as plt

def plot_history(history, model_name):
    plt.figure(figsize=(12, 4))

    # Accuracy
    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')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend()

    # Loss
    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')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()

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

Plot Each Model's History

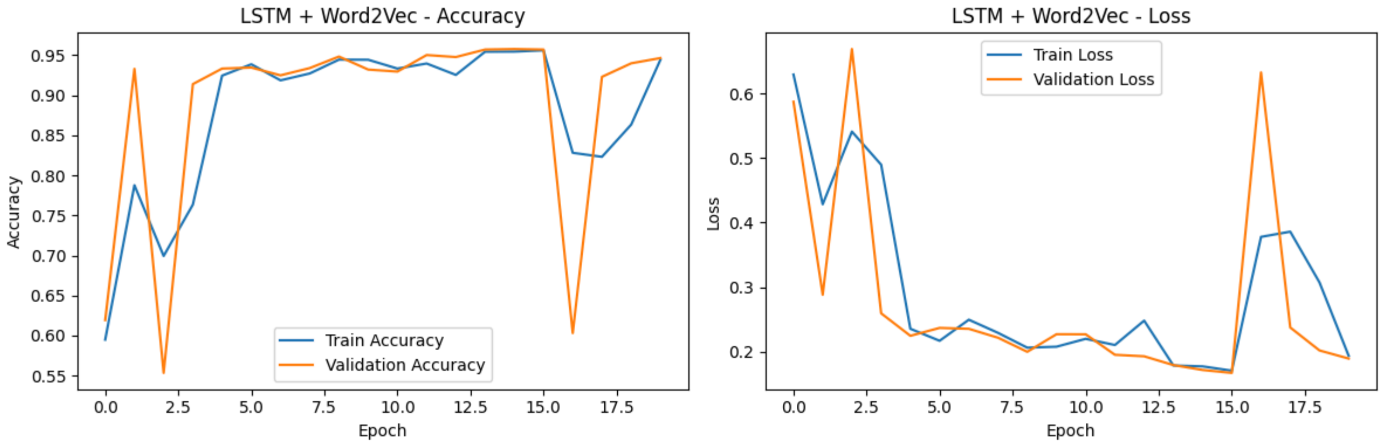
```
plot_history(history_rnn, "Simple RNN")
```



```
plot_history(history_lstm, "LSTM")
```



```
plot_history(history_word2vec, "LSTM + Word2Vec")
```



Evaluate Models on Test Set

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import numpy as np

def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)

    # Binary classification: threshold at 0.5
    y_pred_classes = (y_pred > 0.5).astype(int).flatten()

    acc = accuracy_score(y_test, y_pred_classes)
    print(f"\n{model_name} - Test Accuracy: {acc:.4f}")

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred_classes)
    print(f"\nConfusion Matrix:\n{cm}")

    # Classification Report
    cr = classification_report(y_test, y_pred_classes, target_names=le.classes_)
    print(f"\nClassification Report:\n{cr}")
```

Run Evaluation for All Models

```
evaluate_model(model_rnn, X_test_pad, y_test_enc, "Simple RNN")
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step

Simple RNN - Test Accuracy: 0.5333

Confusion Matrix:
[[1972   28]
 [1839  161]]

Classification Report:
              precision    recall  f1-score   support

        fake       0.52      0.99      0.68      2000
        true       0.85      0.08      0.15      2000

    accuracy                           0.53      4000
   macro avg       0.68      0.53      0.41      4000
weighted avg       0.68      0.53      0.41      4000
```

```
evaluate_model(model_lstm, X_test_pad, y_test_enc, "LSTM")
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step

LSTM - Test Accuracy: 0.9815

Confusion Matrix:
[[1957   43]
 [  31 1969]]

Classification Report:
              precision    recall  f1-score   support

        fake       0.98      0.98      0.98      2000
        true       0.98      0.98      0.98      2000

    accuracy                           0.98      4000
   macro avg       0.98      0.98      0.98      4000
weighted avg       0.98      0.98      0.98      4000
```

```
evaluate_model(model_lstm_word2vec, X_test_pad, y_test_enc, "LSTM + Word2Vec")
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step

LSTM + Word2Vec - Test Accuracy: 0.9483

Confusion Matrix:
[[1821  179]
 [  28 1972]]
```

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

        fake       0.98      0.91      0.95      2000
        true       0.92      0.99      0.95      2000

    accuracy                           0.95      4000
   macro avg       0.95      0.95      0.95      4000
weighted avg       0.95      0.95      0.95      4000
```

Compare Model Performances

```
from prettytable import PrettyTable
from sklearn.metrics import accuracy_score

table = PrettyTable()
table.field_names = ["Model", "Test Accuracy"]

for model, name in zip(
    [model_rnn, model_lstm, model_lstm_word2vec],
    ["Simple RNN", "LSTM", "LSTM + Word2Vec"]
):
    y_pred = model.predict(X_test_pad)

    # For binary classification using sigmoid output
    y_pred_classes = (y_pred > 0.5).astype(int).flatten()

    acc = accuracy_score(y_test_enc, y_pred_classes)
    table.add_row([name, f"{acc:.4f}"])

print("\nModel Comparison:")
print(table)
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step

Model Comparison:
+-----------------+---------------+
|      Model      | Test Accuracy |
+-----------------+---------------+
|    Simple RNN   |     0.5333    |
|       LSTM      |     0.9815    |
| LSTM + Word2Vec |     0.9483    |
+-----------------+---------------+
```

Save Models

```
# Save Simple RNN Model
model_rnn.save("/content/drive/MyDrive/AI ML/simple_rnn_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

```
# Save LSTM Model
model_lstm.save("/content/drive/MyDrive/AI ML/lstm_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

```
# Save LSTM + Word2Vec Model
model_lstm_word2vec.save("/content/drive/MyDrive/AI ML/lstm_word2vec_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

Load Required Libraries again

```
!pip install gradio
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.10.0 (from gradio)
  Downloading gradio_client-1.10.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.31.1)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.26.4)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.18)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.4)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.11.9-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.16
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.67.1
Requirement already satisfied: hf-xet<2.0.0,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (
```

```
import gradio as gr
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

# Ensure NLTK resources are available
nltk.download('stopwords')
nltk.download('wordnet')
```

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

Redefine Preprocessing Function

```
# Initialize lemmatizer and stopwords
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Improved contraction mapping
contractions_dict = {
    "don't": "do not", "doesn't": "does not", "didn't": "did not",
    "can't": "cannot", "won't": "will not", "shouldn't": "should not",
    "isn't": "is not", "aren't": "are not", "wasn't": "was not",
    "weren't": "were not", "hasn't": "has not", "haven't": "have not",
    "hadn't": "had not", "mightn't": "might not", "mustn't": "must not",
    "i'm": "i am", "you're": "you are", "he's": "he is", "she's": "she is",
    "it's": "it is", "we're": "we are", "they're": "they are",
    "i've": "i have", "you've": "you have", "we've": "we have",
    "they've": "they have", "i'll": "i will", "you'll": "you will",
    "he'll": "he will", "she'll": "she will", "we'll": "we will",
    "they'll": "they will"
}

# Compile regex pattern for contractions
contraction_pattern = re.compile(
    r'\b({})\b'.format('|'.join(re.escape(k) for k in contractions_dict.keys())),
    flags=re.IGNORECASE
)

# Fix Unicode artifacts
def fix_unicode_artifacts(text):
    return text.replace("â€™", "'").replace("â€œ", "“").replace("â€⍰", "”").replace("â€“", "–").replace("Â", "")

# Expand contractions
def expand_contractions(text):
    def replace(match):
        word = match.group(0).lower()
        return contractions_dict.get(word, word)
    return contraction_pattern.sub(replace, text)

# Full preprocessing function
def preprocess_input(text):
    text = fix_unicode_artifacts(text)
    text = text.lower()
    text = expand_contractions(text)
    text = re.sub(r'https?://\S+|www\.\S+', '', text)
    text = re.sub(r'@\w+|#', '', text)
    text = re.sub(r'[^\w\s\']', '', text)
    text = re.sub(r'\d+', '', text)
    tokens = text.split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return ' '.join(tokens)
```

Load Saved Models

```
from tensorflow.keras.models import load_model

# Ask user for the directory where models are saved
save_path = input("Enter the directory path where your models are saved (e.g., /content/drive/MyDrive/AI ML): ")

# Ensure the path doesn't end with a slash for consistency
save_path = save_path.rstrip('/')

# Load models from user-specified path
loaded_rnn = load_model(f"{save_path}/simple_rnn_model.h5")
loaded_lstm = load_model(f"{save_path}/lstm_model.h5")
loaded_lstm_word2vec = load_model(f"{save_path}/lstm_word2vec_model.h5")

print("\n Models loaded successfully!")
```

```
Enter the directory path where your models are saved (e.g., /content/drive/MyDrive/AI ML): /content/drive/MyDrive/AI ML
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t

 Models loaded successfully!
```

Define Prediction Function

```
# Global tokenizer and max_len used during training
global_tokenizer = tokenizer  # Use the one from earlier
global_max_len = X_train_pad.shape[1]

# Label encoder
le_classes = le.classes_  # e.g., ['fake', 'true']

def predict_news(news_text, model_choice):
    # Preprocess input
    cleaned_text = preprocess_input(news_text)

    # Tokenize and pad
    sequence = global_tokenizer.texts_to_sequences([cleaned_text])
    padded_seq = pad_sequences(sequence, maxlen=global_max_len, padding='post', truncating='post')

    # Choose model
    if model_choice == "Simple RNN":
        model = loaded_rnn
    elif model_choice == "LSTM":
        model = loaded_lstm
    else:
        model = loaded_lstm_word2vec

    # Predict
    prediction = model.predict(padded_seq, verbose=0)
    predicted_label = le_classes[np.argmax(prediction)]

    # Return styled HTML output
    if predicted_label == 'true':
        return "<div style='padding:20px; background-color:#d4edda; color:#155724; font-size:20px; border-radius:8px; text-align:center;'>✅
    else:
        return "<div style='padding:20px; background-color:#f8d7da; color:#721c24; font-size:20px; border-radius:8px; text-align:center;'>❌
```

Create Gradio Interface

```
# Dropdown for model selection
model_options = ["Simple RNN", "LSTM", "LSTM + Word2Vec"]

# Custom CSS for better styling
custom_css = """
.gradio-container {
    background-color: #f9f9f9;
    font-family: 'Arial';
}
button {
    background-color: #1e90ff;
    color: white;
    border-radius: 8px;
    padding: 10px 20px;
}
"""

# Gradio Interface
demo = gr.Interface(
    fn=predict_news,
    inputs=[
        gr.Textbox(lines=5, placeholder="Enter news article or tweet here...", label="Input Text"),
        gr.Dropdown(choices=model_options, value="LSTM + Word2Vec", label="Select Model")
    ],
    outputs=gr.HTML(label="Prediction Result"),  # <-- Changed from Markdown to HTML
    title="📰 True vs Fake News Classifier",
    description="Classify whether a piece of news is true or fake using RNN, LSTM, or LSTM with Word2Vec embeddings.",
    theme="soft",
    css=custom_css,
    allow_flagging="never"
)

# Launch Gradio App
demo.launch()
```

```
/usr/local/lib/python3.11/dist-packages/gradio/interface.py:415: UserWarning: The `allow_flagging` parameter in `Interface` is deprecate
  warnings.warn(
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://8b1163f7202e4f29cb.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working dir
```

gradio

No interface is running right now