Final Project: Language Translator

Introduction

Language translation plays a crucial role in enabling communication across diverse linguistic and cultural boundaries. With the rise of globalization and digital communication, the demand for efficient and accurate translation systems has grown significantly. Traditional approaches to translation often relied on rule-based methods, which were labor-intensive and lacked flexibility. The advent of artificial intelligence, particularly deep learning, has transformed the field of natural language processing (NLP). Sequence-to-sequence (seq2seq) models, a type of deep learning architecture, have proven effective for translation tasks by modeling the relationship between input and output text sequences.

This project implements a seq2seq translator from English to Chinese using an encoder-decoder architecture with an integrated attention mechanism. The encoder utilizes an LSTM to capture sequential patterns in the input language, while the decoder generates the translated sequence with contextual focus. The attention mechanism ensures the model effectively aligns the source and target languages during translation. By exploring this approach, the project aims to provide insights into the mechanics of neural machine translation and its potential applications.

Exploratory Data Analysis (EDA) and Data Preprocessing

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense,
Attention, Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import pickle
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from colorama import Fore, Style

# Load data
file_path = "/kaggle/input/jsaaxs/cmn.txt"

def load_data(file_path):
    questions, answers = [], []
    with open(file_path, 'r', encoding='utf-8') as file:
        for line in file:
            parts = line.split('\t')
            if len(parts) >= 2:
```

```python
                question = parts[0].strip()
                answer = parts[1].strip()
                questions.append(question)
                answers.append(' '.join(list(answer)))
    return questions, answers

def add_start_end_tokens(target_texts):
    start_token = 'startseq'
    end_token = 'endseq'
    updated_target_texts = []
    for text in target_texts:
        updated_text = f"{start_token} {text} {end_token}"
        updated_target_texts.append(updated_text)
    return updated_target_texts

input_texts, target_texts = load_data(file_path)

# Add startseq and endseq to the target (answers)
target_texts = add_start_end_tokens(target_texts)

data = pd.DataFrame({"English":input_texts,
                     "Chinese":target_texts})
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29909 entries, 0 to 29908
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   English  29909 non-null  object
 1   Chinese  29909 non-null  object
dtypes: object(2)
memory usage: 467.5+ KB
None
```

```python
# Count the number of duplicate rows
e_num_duplicates = data["English"].duplicated().sum()
print(f"Number of English duplicate rows: {e_num_duplicates}")

c_num_duplicates = data["Chinese"].duplicated().sum()
print(f"Number of Chinese duplicate rows: {c_num_duplicates}")

# Remove rows where the 'English' column has duplicate values
data = data[~data["English"].duplicated()]

print("-"*50)
# Verify the new dataset information
print(data.info())
```

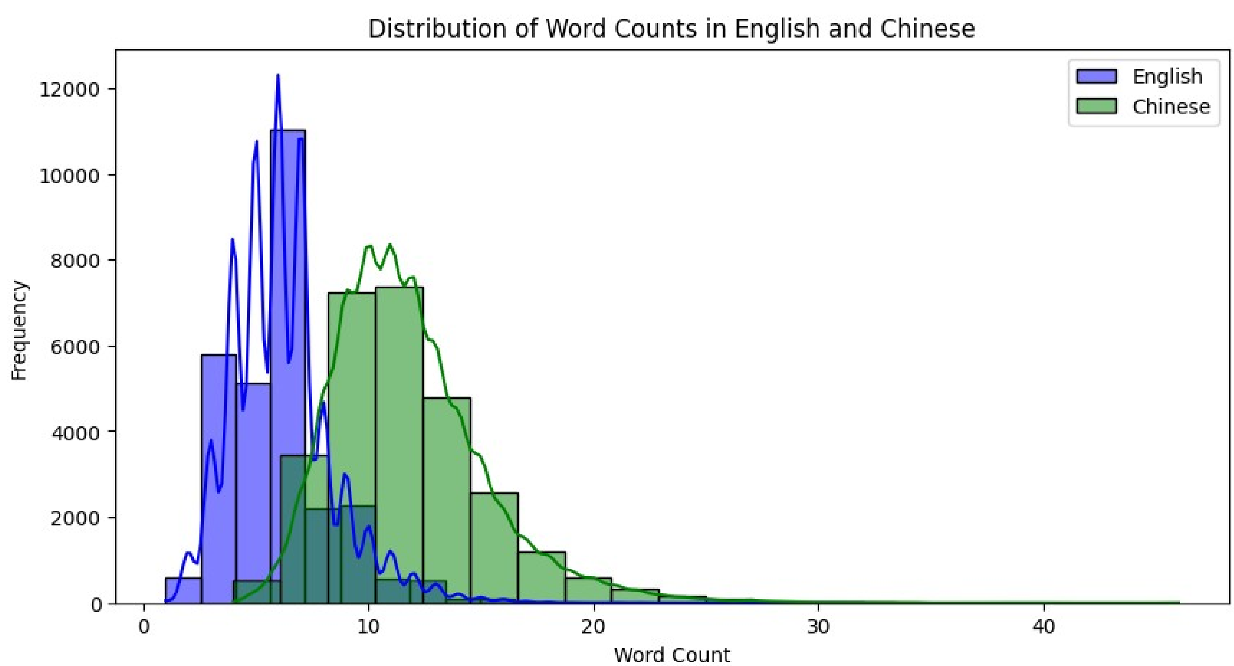
```
Number of English duplicate rows: 1580
Number of Chinese duplicate rows: 3770
```

```
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
Index: 28329 entries, 0 to 29908
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   English  28329 non-null  object
 1   Chinese  28329 non-null  object
dtypes: object(2)
memory usage: 664.0+ KB
None

# Display 5 paired English and Chinese texts randomly in the dataset
random_sample = data.sample(5)
print("Randomly Selected Text Pairs:")
for index, row in random_sample.iterrows():
    print(f"English: {row['English']}")
    print(f"Chinese: {row['Chinese']}")
    print("-" * 50)

Randomly Selected Text Pairs:
English: I should've chosen a shorter username.
Chinese: startseq 我该用短一点的用户名的。 endseq
--------------------------------------------------
English: You will fail.
Chinese: startseq 你會失敗。 endseq
--------------------------------------------------
English: Is that a coyote?
Chinese: startseq 那是丛林狼吗？ endseq
--------------------------------------------------
English: Do you want a drink?
Chinese: startseq 你想喝點什麼嗎？ endseq
--------------------------------------------------
English: Then what?
Chinese: startseq 那又怎樣？ endseq
--------------------------------------------------

# The minimum length and maximum length of each texts
english_length = data["English"].apply(lambda x: len(x.split()))
chinese_length = data["Chinese"].apply(lambda x: len(x.split()))

max_english_length = english_length.max()
max_chinese_length = chinese_length.max()
print(f"Maximum English Text Length: {max_english_length}")
print(f"Maximum Chinese Text Length: {max_chinese_length}")

min_english_length = english_length.min()
min_chinese_length = chinese_length.min()
print(f"Minimum English Text Length: {min_english_length}")
print(f"Minimum Chinese Text Length: {min_chinese_length}")
```

```
Maximum English Text Length: 32
Maximum Chinese Text Length: 46
Minimum English Text Length: 1
Minimum Chinese Text Length: 4
```

```python
# Distribution of English and Chinese Texts lengths
plt.figure(figsize=(10, 5))
sns.histplot(english_length, kde=True, bins=20, color='blue',
label='English')
sns.histplot(chinese_length, kde=True, bins=20, color='green',
label='Chinese')
plt.legend()
plt.title("Distribution of Word Counts in English and Chinese")
plt.xlabel("Word Count")
plt.ylabel("Frequency")
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



```
# Average word length
avg_english_length = english_length.mean()
avg_chinese_length = chinese_length.mean()

print(f"Average words in a English: {avg_english_length:.0f}")
print(f"Average words in an Chinese: {avg_chinese_length:.0f}")
```

```
Average words in a English: 6
Average words in an Chinese: 12
```

```
# Prepare data for model

input_texts = data["English"].tolist()
target_texts = data["Chinese"].tolist()

# Tokenize the data
tokenizer = Tokenizer(filters='')
tokenizer.fit_on_texts(input_texts + target_texts)
input_sequences = tokenizer.texts_to_sequences(input_texts)
target_sequences = tokenizer.texts_to_sequences(target_texts)

# Use the maximum length of input and output sequences
MAX_NUMBER = 50
max_input_len = MAX_NUMBER
max_target_len = MAX_NUMBER

# Padding
input_sequences = pad_sequences(input_sequences, maxlen=max_input_len,
padding='post')
target_sequences = pad_sequences(target_sequences,
maxlen=max_target_len, padding='post')

# Vocabulary size
vocab_size = len(tokenizer.word_index) + 1
print(f"Vocabulary Size: {vocab_size}, Max Input Length:
{max_input_len}, Max Target Length: {max_target_len}")

# Split data into training and validation sets (80-20 split)
input_train, input_val, target_train, target_val =
train_test_split(input_sequences, target_sequences, test_size=0.2)

# shift target sequences
target_train_input = target_train[:, :-1]
target_train_output = target_train[:, 1:]
target_val_input = target_val[:, :-1]
target_val_output = target_val[:, 1:]

# Ensure the target sequences are padded to the correct length of
max_target_len
target_train_input = pad_sequences(target_train_input,
maxlen=max_target_len, padding='post')
```

```
target_train_output = pad_sequences(target_train_output,
maxlen=max_target_len, padding='post')
target_val_input = pad_sequences(target_val_input,
maxlen=max_target_len, padding='post')
target_val_output = pad_sequences(target_val_output,
maxlen=max_target_len, padding='post')

# Check the shapes
print(f"Input train shape: {input_train.shape}")
print(f"Target train input shape: {target_train_input.shape}")
print(f"Target train output shape: {target_train_output.shape}")
```

```
Vocabulary Size: 15489, Max Input Length: 50, Max Target Length: 50
Input train shape: (22663, 50)
Target train input shape: (22663, 50)
Target train output shape: (22663, 50)
```

Build the Model

```
# Encoder
encoder_inputs = Input(shape=(max_input_len,))
encoder_embedding = Embedding(vocab_size, 128)(encoder_inputs)
encoder_lstm = LSTM(256, return_sequences=True, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)
encoder_states = [state_h, state_c]

# Decoder
decoder_inputs = Input(shape=(max_target_len,))
decoder_embedding = Embedding(vocab_size, 128)(decoder_inputs)
decoder_lstm = LSTM(256, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding,
initial_state=encoder_states)

# Attention Mechanism
attention_layer = Attention(use_scale=True)
attention_outputs = attention_layer([decoder_outputs,
encoder_outputs])

# Combine Attention and Decoder Output
decoder_combined_context = Concatenate(axis=-1)([attention_outputs,
decoder_outputs])

# Output Dense Layer
decoder_dense = Dense(vocab_size, activation='softmax')
decoder_final_output = decoder_dense(decoder_combined_context)

# Define Seq2Seq Model
seq2seq_model = Model([encoder_inputs, decoder_inputs],
decoder_final_output)
seq2seq_model.compile(optimizer='adam',
```

```
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

seq2seq_model.summary()

Model: "functional_1"

┏━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━
━━━━━┓
┃ Layer (type)         ┃ Output Shape       ┃    Param # ┃ Connected to
┃
┡━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━
━━━━━┩
│ input_layer          │ (None, 50)         │          0 │ -
│ (InputLayer)         │                    │            │
│
├──────────────────────┼────────────────────┼────────────┼────────────────
─────┤
│ input_layer_1        │ (None, 50)         │          0 │ -
│ (InputLayer)         │                    │            │
│
├──────────────────────┼────────────────────┼────────────┼────────────────
─────┤
│ embedding            │ (None, 50, 128)    │  1,982,592 │
input_layer[0][0] │
│ (Embedding)          │                    │            │
│
├──────────────────────┼────────────────────┼────────────┼────────────────
─────┤
│ embedding_1          │ (None, 50, 128)    │  1,982,592 │
input_layer_1[0]… │
│ (Embedding)          │                    │            │
│
├──────────────────────┼────────────────────┼────────────┼────────────────
─────┤
│ lstm (LSTM)          │ [(None, 50, 256),  │    394,240 │ embedding[0]
[0]   │
│                      │ (None, 256),       │            │
│                      │ (None, 256)]       │            │
│
├──────────────────────┼────────────────────┼────────────┼────────────────
─────┤
│ lstm_1 (LSTM)        │ [(None, 50, 256),  │    394,240 │
embedding_1[0][0… │
│                      │ (None, 256),       │            │ lstm[0][1],
│                      │ (None, 256)]       │            │ lstm[0][2]
│
```

```
├─────────────────────────────┼─────────────────────────┼─────────────────┼────────────────────
───────┤
│ attention                   │ (None, 50, 256)         │               1 │ lstm_1[0][0],
│
│ (Attention)                 │                         │                 │ lstm[0][0]
│
├─────────────────────────────┼─────────────────────────┼─────────────────┼────────────────────
───────┤
│ concatenate                 │ (None, 50, 512)         │               0 │ attention[0]
[0],   │
│ (Concatenate)               │                         │                 │ lstm_1[0][0]
│
├─────────────────────────────┼─────────────────────────┼─────────────────┼────────────────────
───────┤
│ dense (Dense)               │ (None, 50, 15489)       │   7,945,857     │
concatenate[0][0] │
└─────────────────────────────┴─────────────────────────┴─────────────────┴────────────────────
───────┘

 Total params: 12,699,522 (48.44 MB)

 Trainable params: 12,699,522 (48.44 MB)

 Non-trainable params: 0 (0.00 B)
```

Model Training

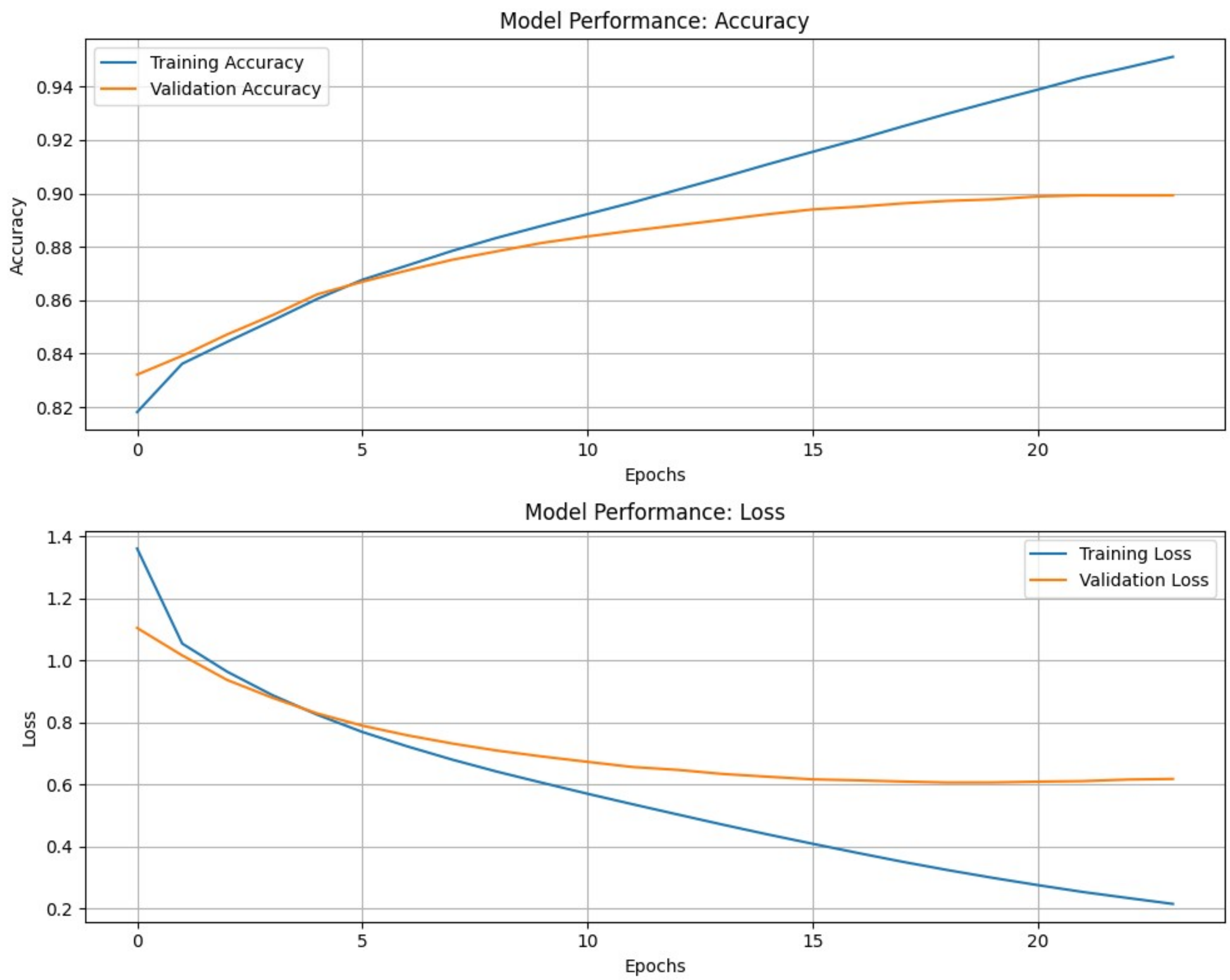
```python
# Add early stopping for the model to avoid overfitting
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=2,
    restore_best_weights=True
)

# Train the model
history = seq2seq_model.fit(
    [input_train, target_train_input],
    target_train_output,
    batch_size=64,
    epochs=50,
    validation_data=([input_val, target_val_input],
target_val_output),
    callbacks=[early_stopping]
)

# Save the trained model
seq2seq_model.save('translator_model.h5')
```

```
Epoch 1/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 43s 112ms/step - accuracy: 0.7915 - loss:
```

```
2.1133 - val_accuracy: 0.8322 - val_loss: 1.1041
Epoch 2/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.8341 - loss:
1.0763 - val_accuracy: 0.8392 - val_loss: 1.0156
Epoch 3/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.8426 - loss:
0.9788 - val_accuracy: 0.8472 - val_loss: 0.9364
Epoch 4/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.8506 - loss:
0.9008 - val_accuracy: 0.8544 - val_loss: 0.8791
Epoch 5/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.8584 - loss:
0.8350 - val_accuracy: 0.8622 - val_loss: 0.8284
Epoch 6/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 110ms/step - accuracy: 0.8670 - loss:
0.7726 - val_accuracy: 0.8669 - val_loss: 0.7890
Epoch 7/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.8718 - loss:
0.7292 - val_accuracy: 0.8712 - val_loss: 0.7578
Epoch 8/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 110ms/step - accuracy: 0.8778 - loss:
0.6814 - val_accuracy: 0.8752 - val_loss: 0.7313
Epoch 9/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.8825 - loss:
0.6440 - val_accuracy: 0.8784 - val_loss: 0.7084
Epoch 10/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.8876 - loss:
0.6056 - val_accuracy: 0.8815 - val_loss: 0.6894
Epoch 11/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.8920 - loss:
0.5703 - val_accuracy: 0.8839 - val_loss: 0.6723
Epoch 12/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.8968 - loss:
0.5340 - val_accuracy: 0.8861 - val_loss: 0.6556
Epoch 13/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.9017 - loss:
0.4994 - val_accuracy: 0.8881 - val_loss: 0.6464
Epoch 14/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 110ms/step - accuracy: 0.9069 - loss:
0.4651 - val_accuracy: 0.8901 - val_loss: 0.6332
Epoch 15/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 110ms/step - accuracy: 0.9117 - loss:
0.4336 - val_accuracy: 0.8922 - val_loss: 0.6244
Epoch 16/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.9161 - loss:
0.4056 - val_accuracy: 0.8940 - val_loss: 0.6159
Epoch 17/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.9208 - loss:
0.3748 - val_accuracy: 0.8950 - val_loss: 0.6127
```

```
Epoch 18/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.9261 - loss:
0.3457 - val_accuracy: 0.8963 - val_loss: 0.6087
Epoch 19/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.9308 - loss:
0.3194 - val_accuracy: 0.8972 - val_loss: 0.6056
Epoch 20/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.9358 - loss:
0.2914 - val_accuracy: 0.8978 - val_loss: 0.6058
Epoch 21/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.9391 - loss:
0.2745 - val_accuracy: 0.8988 - val_loss: 0.6084
Epoch 22/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 111ms/step - accuracy: 0.9450 - loss:
0.2461 - val_accuracy: 0.8993 - val_loss: 0.6100
Epoch 23/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 110ms/step - accuracy: 0.9480 - loss:
0.2306 - val_accuracy: 0.8992 - val_loss: 0.6153
Epoch 24/50
355/355 ━━━━━━━━━━━━━━━━━━━━ 39s 110ms/step - accuracy: 0.9524 - loss:
0.2097 - val_accuracy: 0.8993 - val_loss: 0.6172

# Save the tokenizer
with open('tokenizer.pkl', 'wb') as file:
    pickle.dump(tokenizer, file)

# Plot the performance for the model
fig, ax = plt.subplots(2, 1, figsize=(10, 8))

# Plot Accuracy
ax[0].plot(history.history['accuracy'], label='Training Accuracy')
ax[0].plot(history.history['val_accuracy'], label='Validation
Accuracy')
ax[0].set_title('Model Performance: Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend()
ax[0].grid()

# Plot Loss
ax[1].plot(history.history['loss'], label='Training Loss')
ax[1].plot(history.history['val_loss'], label='Validation Loss')
ax[1].set_title('Model Performance: Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend()
ax[1].grid()

# Adjust layout and display the plot
```

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



Build the Interative Translator

```
# Function to decode sequence into words
def decode_output(sequence, tokenizer):
    reverse_word_index = {index: word for word, index in
tokenizer.word_index.items()}
    decoded_sentence = ' '.join([reverse_word_index.get(i, '') for i
in sequence if i != 0])
    return decoded_sentence

# Function to chat with the chatbot
def translator():
    # Load the trained model
    seq2seq_model = tf.keras.models.load_model('translator_model.h5')

    # Load the tokenizer
```

```
    with open('tokenizer.pkl', 'rb') as file:
        tokenizer = pickle.load(file)

    print(f"{Fore.GREEN}English to Chinese Translator:
{Style.RESET_ALL} Hi! How can I help you today? (type 'exit' to
quit)")

    while True:
        user_input = input(f"{Fore.CYAN}You:{Style.RESET_ALL} ")
        if user_input.lower() == 'exit':
            print(f"{Fore.RED}Chatbot: Goodbye!{Style.RESET_ALL}")
            break

        # Preprocess the user input (convert to sequence)
        user_input_seq = tokenizer.texts_to_sequences([user_input])
        user_input_seq = pad_sequences(user_input_seq,
maxlen=max_input_len, padding='post')

        # Create an initial decoder input (startseq)
        decoder_input_seq = np.zeros((1, max_target_len))
        decoder_input_seq[0, 0] = tokenizer.word_index['startseq']

        # Predict the response
        for t in range(1, max_target_len):
            # Predict next word in the sequence
            predictions = seq2seq_model.predict([user_input_seq,
decoder_input_seq], verbose=0)
            predicted_index = np.argmax(predictions[0, t-1, :])

            # If we predict endseq, stop the loop
            if predicted_index == tokenizer.word_index['endseq']:
                break

            # Add predicted word to decoder input sequence for the
next prediction
            decoder_input_seq[0, t] = predicted_index

        # Decode the predicted sequence
        predicted_sentence = decode_output(decoder_input_seq[0],
tokenizer)
        predicted_sentence = predicted_sentence.replace('startseq',
'').replace('endseq', '').strip()
        predicted_sentence = predicted_sentence.replace(' ', '')

        print(f"{Fore.YELLOW}Translator:{Style.RESET_ALL}
{predicted_sentence}")
```

Interactive Tranlator

```
translator()
```

```
English to Chinese Translator: Hi! How can I help you today? (type
'exit' to quit)

You:  I want you to help me.

Translator: 我想让你帮我。

You:  where are you?

Translator: 你在哪儿？

You:  She is my sister.

Translator: 她是我的姐妹。

You:  exit

Chatbot: Goodbye!
```

Conclusion

This project successfully implemented a sequence-to-sequence (Seq2Seq) model with attention for English-to-Chinese translation. The model architecture includes an encoder-decoder framework with an attention mechanism, enabling the model to focus on relevant parts of the input sequence during translation. The training process leveraged a dataset of English-Chinese sentence pairs, and performance was evaluated using accuracy and loss metrics. The results demonstrate the model's ability to generate reasonably accurate translations.

Limitations:

- **Dataset Quality and Size**:
  The quality and size of the dataset were limited, which impacts the model's generalization accuracy.

- **Computational Intensity**:
  Training was computationally intensive. Padding was applied to the texts before feeding them into the model to maintain consistent sequence lengths.

- **Vocabulary Coverage**:
  The tokenizer's vocabulary size was limited, affecting the model's ability to handle out-of-vocabulary words.

Future Improvements:

- **Dataset Expansion**:
  Using a larger and more diverse dataset would improve the model's robustness and ability to generalize better to unseen data.

- **Model Optimization**:
  Experimenting with more advanced architectures, such as transformer-based models (e.g., BERT, GPT, T5), could significantly enhance translation performance.
- **Pretrained Models and Transfer Learning**:
  Future work could involve leveraging pretrained models or fine-tuning them on specific translation tasks to improve accuracy and reduce the need for large-scale training.