```python
# ==============================
# importing libraries
# ==============================

import tensorflow as tf
from tensorflow.keras.layers import Input,Dense,Embedding,GRU
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam
import numpy as np
import unicodedata
import re
import matplotlib.pyplot as plt
%matplotlib inline

# ==============================
# problem_set_1
# ==============================

def scaled_dot_product_attention(query, key, value):
    """
    Compute the scaled dot product attention.
    Arguments:
    query: tensor with shape (batch_size, input_sequence_length, query_dim)
    key: tensor with shape (batch_size, input_sequence_length, key_dim)
    value: tensor with shape (batch_size, input_sequence_length, value_dim)

    Returns:
    output: tensor with shape (batch_size, input_sequence_length, value_dim)
    """
    if query.ndim > 3:
        query = np.reshape(query, (query.shape[0], query.shape[1], -1))
    if key.ndim > 3:
        key = np.reshape(key, (key.shape[0], key.shape[1], -1))
    if value.ndim > 3:
        value = np.reshape(value, (value.shape[0], value.shape[1], -1))

    dot_product = np.matmul(query, key.transpose(0, 2, 1))
    scaled_dot_product = dot_product / np.sqrt(query.shape[-1])
    attention_scores = np.exp(scaled_dot_product)
    attention_weights = attention_scores / np.sum(attention_scores, axis=-1, keepdims=True)
    output = np.matmul(attention_weights, value)
    return output

def split_heads(x, num_heads):
    """
    Compute Split Heads
    Arguments:
    x: tensor of shape (batch_size, input_sequence_length, num_heads * query_dim/key_dim/value_dim)
    num_heads: integer

    Returns:
    output: tensor with shape (batch_size, input_sequence_length, num_heads, -1)
    """
    batch_size, input_sequence_length, concatenated_dim = x.shape
    head_dim = concatenated_dim // num_heads
    reshaped_tensor = np.reshape(x, (batch_size, input_sequence_length, num_heads, head_dim))
```

```
    return reshaped_tensor

def multi_head_scaled_attention(query, key, value, num_heads, W_q, W_k, W_v):
    """
    Compute the multi-head attention.
    Arguments:
    query: tensor with shape (batch_size, input_sequence_length, query_dim)
    key: tensor with shape (batch_size, input_sequence_length, key_dim)
    value: tensor with shape (batch_size, input_sequence_length, value_dim)
    num_heads: integer
    W_q: matrix with shape (query_dim, num_heads * query_dim)
    W_k: matrix with shape (key_dim, num_heads * key_dim)
    W_v: matrix with shape (value_dim, num_heads * value_dim)

    Returns:
    output: tensor with shape (batch_size, input_sequence_length, num_heads * value_dim)
    """
    projected_query = np.matmul(query, W_q)
    projected_key = np.matmul(key, W_k)
    projected_value = np.matmul(value, W_v)
    query_heads = split_heads(projected_query, num_heads)
    key_heads = split_heads(projected_key, num_heads)
    value_heads = split_heads(projected_value, num_heads)
    attention_heads = scaled_dot_product_attention(query_heads, key_heads, value_heads)
    concatenated_attention = np.reshape(attention_heads, (query.shape[0], query.shape[1], -1))
    return concatenated_attention

# Testing out with following input values
input_seq_len=5 # Maximum length of the input sequence
d_q=8           # Dimensionality of the linearly projected queries
d_k=8           # Dimensionality of the linearly projected keys
d_v=8           # Dimensionality of the linearly projected values
batch_size=64   # Batch size from the training process
num_heads=8     # Number of self-attention heads
query = np.random.randn(batch_size, input_seq_len, d_q) # generating input query matrix
key = np.random.randn(batch_size, input_seq_len, d_k)   # generating input key matrix
value = np.random.randn(batch_size, input_seq_len, d_v) # generating input value matrix
W_q = np.random.randn(d_q, num_heads*d_q) # for generating num head projection matrices for queries
W_k = np.random.randn(d_k, num_heads*d_k) # for generating num head projection matrices for keys
W_v = np.random.randn(d_v, num_heads*d_v) # for generating num head projection matrices for values

# Testing code of scaled dot product attention
attention=scaled_dot_product_attention(query, key, value)
print("Scaled Dot Product Attention:", attention)
print("Scaled Dot Product Attention Shape:", attention.shape)

# Testing code of multi head scaled attention
multi_head_attention=multi_head_scaled_attention(query, key, value, num_heads, W_q, W_k, W_v)
print("Multi Head Scaled Attention", multi_head_attention)
print("Multi Head Scaled Attention Shape:", multi_head_attention.shape)

# ============================
# problem_set_2
# ============================

start_token = 'sos'
end_token = 'eos'
oov_token = 'unk'
BATCH_SIZE = 32
EPOCHS = 31
GRU_UNITS = 256
```

```python

def txt_pre_processing(txt:str)->str:
  txt = txt.lower().strip()
  txt = unicodedata.normalize('NFKD',txt).encode('ascii','ignore').decode('utf-8')
  txt = re.sub(pattern=r'[^\sa-z\d\.\?\!\,]',repl='',string=str(txt))
  txt = re.sub(pattern=r'([\.\?\!\,])',repl=r' \1 ',string=str(txt))
  txt = re.sub(pattern=r'\s+',repl=r' ',string=str(txt)).strip()
  txt = start_token + ' ' + txt + ' ' + end_token
  return txt

def load_data() -> tuple:
  context : list = list()
  target : list = list()
  with open(file='./eng-fra.txt',mode='r',encoding='utf-8') as inputstream:
    for text in inputstream:
      lines = text.replace('\n','').replace('\r','').split('\t')
      eng_txt = lines[0]
      fr_txt = lines[1]
      eng_txt = txt_pre_processing(txt=eng_txt)
      fr_txt = txt_pre_processing(txt=fr_txt)
      context.append(eng_txt)
      target.append(fr_txt)
  context = np.array(context)
  target = np.array(target)
  return context,target

eng_sentences,fr_sentences = load_data()
shuffling_indices = np.arange(len(eng_sentences))
np.random.shuffle(shuffling_indices)
eng_sentences = eng_sentences[shuffling_indices]
fr_sentences = fr_sentences[shuffling_indices]

max_seq_length = max([len(x.split(' ')) for x in eng_sentences])

eng_tokenizer = Tokenizer()
eng_tokenizer.fit_on_texts(eng_sentences)
eng_vocab_words = eng_tokenizer.word_index.keys()
eng_tokenizer.word_index[oov_token] = len(eng_tokenizer.word_index) + 1
eng_vocab_size = len(eng_tokenizer.word_index) + 1

fr_tokenizer = Tokenizer()
fr_tokenizer.fit_on_texts(fr_sentences)
fr_vocab_size = len(fr_tokenizer.word_index) + 1

eng_sequences = eng_tokenizer.texts_to_sequences(eng_sentences)
fr_sequences = fr_tokenizer.texts_to_sequences(fr_sentences)

eng_sequences = pad_sequences(eng_sequences,maxlen=max_seq_length,padding='post')
fr_sequences = pad_sequences(fr_sequences,maxlen=max_seq_length,padding='post')

split_80_20: int = int(eng_sequences.shape[0]*0.8)
X_train,y_train = eng_sequences[:split_80_20,:],fr_sequences[:split_80_20]
X_test,y_test = eng_sequences[split_80_20:,:],fr_sequences[split_80_20:]
y_train = to_categorical(y_train,num_classes=fr_vocab_size)
y_test = to_categorical(y_test,num_classes=fr_vocab_size)

# =============================
# Load Glove Embedding
# =============================
glove_embeddings_mapping : dict = dict()
glove_embeddings_size = 50
with open(file='./glove.6B.50d.txt',mode='r',encoding='utf-8') as inputstream:
```

```python
    for text in inputstream:
        text = text.split()
        glove_embeddings_mapping[text[0]] = np.asarray(text[1:],dtype='float32')

# ============================
# Glove Matrix
# ============================
glove_embedding_matrix = np.zeros(shape=(eng_vocab_size,glove_embeddings_size))
for txt,idx in eng_tokenizer.word_index.items():
    if txt in glove_embeddings_mapping:
        glove_embedding_matrix[idx] = glove_embeddings_mapping[txt]

# ============================
# enc + dec
# ============================
encoder_inputs = Input(shape=(None,))
encoder_embedding = Embedding(input_dim=eng_vocab_size,output_dim=glove_embeddings_size,weights=[glove_embedding_matrix],trainable=False)(encoder_inputs)
encoder_gru = GRU(GRU_UNITS,return_state=True)
encoder_outputs,encoder_state = encoder_gru(encoder_embedding)
decoder_inputs = Input(shape=(None,))
decoder_embedding = Embedding(input_dim=fr_vocab_size,output_dim=glove_embeddings_size)(decoder_inputs)
decoder_gru = GRU(GRU_UNITS,return_sequences=True,return_state=True)
decoder_outputs,_ = decoder_gru(decoder_embedding,initial_state=encoder_state)
decoder_dense = Dense(fr_vocab_size,activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
model = Model([encoder_inputs,decoder_inputs],decoder_outputs)
model.compile(optimizer=Adam(learning_rate=3e-5,epsilon=1e-07,),loss='categorical_crossentropy',metrics=['accuracy'])

# ============================
# Testing Translation
# ============================
test_sentence = X_test[-1].reshape(1,-1)
translations_tracking : dict = dict()
history_translations_tracking : dict = {
    'loss' : list(),
    'val_loss' : list(),
    'accuracy' : list(),
    'val_accuracy' : list(),
}
history_tracking : dict = {
    'loss' : list(),
    'val_loss' : list(),
    'accuracy' : list(),
    'val_accuracy' : list(),
}

for epoch in range(EPOCHS):
    history = model.fit([X_train,X_train],y_train,epochs=1,batch_size=BATCH_SIZE,validation_data=([X_test,X_test],y_test))
    history_tracking['loss'].append(history.history['loss'])
    history_tracking['val_loss'].append(history.history['val_loss'])
    history_tracking['accuracy'].append(history.history['accuracy'])
    history_tracking['val_accuracy'].append(history.history['val_accuracy'])
    if epoch == 0 or epoch % 5 == 0:
        curr_trans = model.predict([test_sentence,test_sentence],batch_size=1)
        translations_tracking[epoch] = {
                                        'correct' : eng_tokenizer.sequences_to_texts([test_sentence[0]]),
                                        'translated' : fr_tokenizer.sequences_to_texts([np.argmax(curr_trans,axis=-1)[0]]),
                                        }
        history_translations_tracking['loss'].append(history.history['loss'])
        history_translations_tracking['val_loss'].append(history.history['val_loss'])
        history_translations_tracking['accuracy'].append(history.history['accuracy'])
```

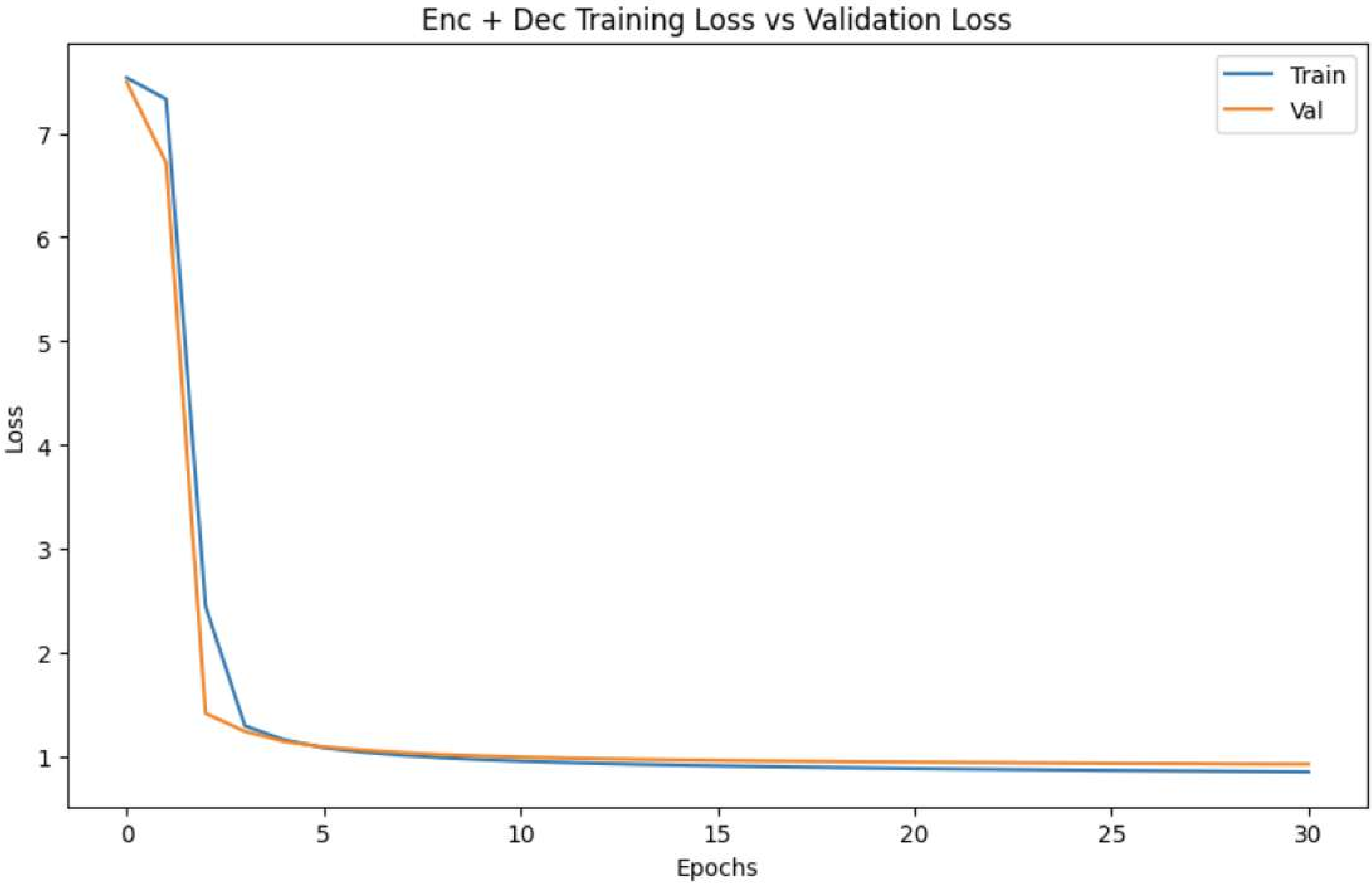
```
        history_translations_tracking['val_accuracy'].append(history.history['val_accuracy'])
    else:
        continue

# ==============================
# Plotting training and the testing loss for 0th and multiple of 5 epoch.
# ==============================
fig,axs = plt.subplots(2,1,figsize=(10,13))
axs[0].plot(history_translations_tracking['loss'])
axs[0].plot(history_translations_tracking['val_loss'])
axs[0].title.set_text('Enc + Dec Training Loss vs Validation Loss')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Loss')
axs[0].legend(['Train','Val'])
axs[1].plot(history_translations_tracking['accuracy'])
axs[1].plot(history_translations_tracking['val_accuracy'])
axs[1].title.set_text('Enc + Dec Training Accuracy vs Validation Accuracy')
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Accuracy')
axs[1].legend(['Train','Val'])

# ==============================
# Plotting training and the testing loss for each epoch.
# ==============================
fig,axs = plt.subplots(2,1,figsize=(10,13))
axs[0].plot(history_tracking['loss'])
axs[0].plot(history_tracking['val_loss'])
axs[0].title.set_text('Enc + Dec Training Loss vs Validation Loss')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Loss')
axs[0].legend(['Train','Val'])
axs[1].plot(history_tracking['accuracy'])
axs[1].plot(history_tracking['val_accuracy'])
axs[1].title.set_text('Enc + Dec Training Accuracy vs Validation Accuracy')
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Accuracy')
axs[1].legend(['Train','Val'])
```

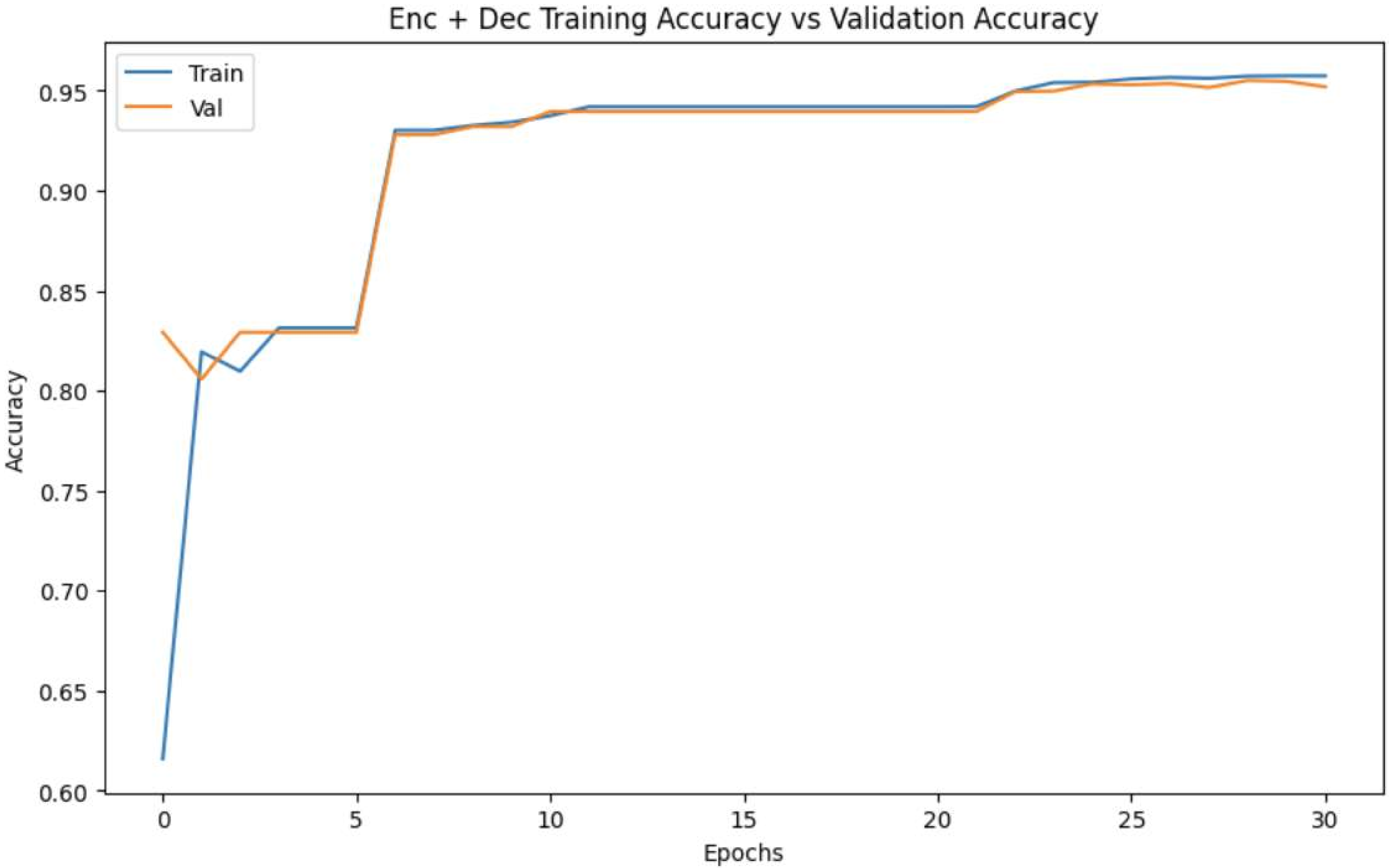
```
scaled_dot_product_attention -> query.shape = (64, 5, 8)
scaled_dot_product_attention -> key.shape = (64, 5, 8)
scaled_dot_product_attention -> value.shape = (64, 5, 8)
scaled_dot_product_attention -> dot_product.shape = (64, 5, 5)
scaled_dot_product_attention -> scaled_dot_product.shape = (64, 5, 5)
scaled_dot_product_attention -> attention_scores.shape = (64, 5, 5)
scaled_dot_product_attention -> attention_weights.shape = (64, 5, 5)
scaled_dot_product_attention -> output.shape = (64, 5, 8)
Scaled Dot Product Attention: [[[ 0.32049449 -0.26037721  0.57715652 ...  0.67257267  0.57302852
    0.42918147]
  [ 0.18361762 -0.1615577   0.74465832 ...  0.67088495  0.83321277
    0.41839623]
  [ 0.25670707 -0.6283299   1.1702535  ...  0.57414191  0.20628595
    0.47909316]
  [ 0.48596639  0.07771385 -0.16564297 ...  0.78107628  1.00761391
    0.33899313]
  [ 0.34698335  0.26430095 -0.43351202 ...  0.86874091  1.03004984
    0.3646912 ]]

 [[ 0.09733811  0.29871887 -0.04301486 ... -0.49032091  0.20637658
   -0.06015095]
  [ 0.21399615  0.27831926  0.10238223 ... -0.40469542  0.24720898
    0.27278054]
  [-0.03270254  0.40088035  0.07822751 ... -0.6222459   0.50397853
    0.1627915 ]
  [-0.09250097  0.37620038 -0.12072245 ... -0.64427094  0.31387475
   -0.24786255]
  [ 0.10300623  0.21843467 -0.07129473 ... -0.4805931   0.25780556
   -0.09008697]]

 [[ 0.7587503  -1.22443682  0.6396305  ...  0.1048958   0.7339676
   -0.41744447]
  [ 0.37518944 -0.4003244   0.29736823 ...  0.0821836   0.39091867
   -0.3322847 ]
  [ 0.63708029  0.02857291 -0.17430152 ... -0.12008221  0.69168012
   -0.51092033]
  [ 0.63177876 -0.2736909  -0.25066728 ...  0.12965289  1.15408735
   -0.82825644]
  [ 0.27850985 -0.45713751 -0.33024877 ...  0.18237319  0.75179257
   -0.59662702]]

 ...

 [[ 0.86287874  0.87762199 -0.31237313 ...  0.47403448 -0.52667169
    0.22082798]
  [ 0.47433115 -0.07577435  0.0274051  ... -0.34456309  0.14853777
    0.38885748]
  [ 0.29648921  0.24600455  0.19818333 ... -0.01334004  0.22414259
    0.57022444]
  [ 0.15305594 -0.08036415  0.11077052 ... -0.22036802 -0.18909445
    0.49740526]
  [ 0.23558292  0.16375332  0.17607076 ... -0.04813029  0.0650367
    0.55424271]]

 [[ 0.43266217  0.45181045 -0.53921716 ...  0.82770425 -0.03493847
   -0.54090625]
  [ 0.48486403  0.17985689 -0.11505856 ...  0.28442258  0.11320648
   -0.28030211]
```

```
   0.55424271]]

 [[ 0.43266217  0.45181045 -0.53921716 ...  0.82770425 -0.03493847
   -0.54090625]
  [ 0.48486403  0.17985689 -0.11505856 ...  0.28442258  0.11320648
   -0.28030211]
  [ 0.3152738   0.29972283 -0.48899666 ...  0.65538579 -0.0623106
    0.28312533]
  [ 0.03627946  0.23007976  0.17741296 ...  1.18159433 -0.1205346
    0.11506042]
  [ 0.71147901  0.5041412  -0.70295973 ...  0.46050544  0.09043237
   -1.04158201]]

 [[-0.74940332  0.36280141  0.26116175 ... -0.57908213 -0.21421493
    0.25120602]
  [-0.39935223  0.46270689 -0.05223337 ... -0.3855749  -0.31565068
   -0.06946156]
  [-0.15002964  0.7667206  -0.49081544 ... -1.1160412   0.05242817
   -0.11869961]
  [-0.24597695  0.38134416 -0.43823219 ... -0.56154394 -0.35290235
   -0.18280336]
  [-0.37389065  0.85254153 -0.22990709 ... -1.00939112 -0.01977578
   -0.0021528 ]]]
Scaled Dot Product Attention Shape: (64, 5, 8)
==========
multi_head_scaled_attention -> projected_query.shape = (64, 5, 64)
multi_head_scaled_attention -> projected_key.shape = (64, 5, 64)
multi_head_scaled_attention -> projected_value.shape = (64, 5, 64)
multi_head_scaled_attention -> query_heads.shape = (64, 5, 8, 8)
multi_head_scaled_attention -> key_heads.shape = (64, 5, 8, 8)
multi_head_scaled_attention -> value_heads.shape = (64, 5, 8, 8)
scaled_dot_product_attention -> query.shape = (64, 5, 64)
scaled_dot_product_attention -> key.shape = (64, 5, 64)
scaled_dot_product_attention -> value.shape = (64, 5, 64)
scaled_dot_product_attention -> dot_product.shape = (64, 5, 5)
scaled_dot_product_attention -> scaled_dot_product.shape = (64, 5, 5)
scaled_dot_product_attention -> attention_scores.shape = (64, 5, 5)
scaled_dot_product_attention -> attention_weights.shape = (64, 5, 5)
scaled_dot_product_attention -> output.shape = (64, 5, 64)
Multi Head Scaled Attention [[[-1.10576549  3.34213618 -1.09227605 ... -2.68362801  6.71951716
   -5.0379075 ]
  [ 3.22571549  2.00964825  0.1544249  ...  0.82026753  0.87772739
   -0.92102584]
  [-0.87755784  3.38009532 -1.03371098 ... -2.5147299   6.48268136
   -4.89333112]
  [ 0.27984611  0.02480872  0.18258911 ...  1.84044702  2.14403798
    1.59627869]
  [ 0.54358086  0.57503965 -0.25579966 ...  2.9502067  -1.85284197
    1.1064349 ]]

 [[-1.62861208  2.4619379  -0.34590429 ...  0.69778489 -1.30499806
   -1.25729776]
  [-1.57034696  2.40017486 -0.32376202 ...  0.68566511 -1.31363848
   -1.1969879 ]
  [ 1.22332422 -0.65744974  0.73922393 ...  0.08230363 -1.75732589
    1.74940991]
  [-1.40826917  2.15407997 -0.25886037 ...  0.65065985 -1.33491208
   -0.96892455]
  [-0.7867873  -0.87297427  0.14240872 ...  0.91474751 -0.61873359
```

```
 [[-1.62861208  2.4619379  -0.34590429 ...  0.69778489 -1.30499806
   -1.25729776]
  [-1.57034696  2.40017486 -0.32376202 ...  0.68566511 -1.31363848
   -1.1969879 ]
  [ 1.22332422 -0.65744974  0.73922393 ...  0.08230363 -1.75732589
    1.74940991]
  [-1.40826917  2.15407997 -0.25886037 ...  0.65065985 -1.33491208
   -0.96892455]
  [-0.7867873  -0.87297427  0.14240872 ...  0.91474751 -0.61873359
    2.09101434]]

 [[-1.55576017 -1.48412891  1.33913957 ... -2.67502232  3.59819332
   -0.08224043]
  [-1.32251771 -2.57036847  0.92133628 ... -1.97955463  2.12090305
    0.83482006]
  [ 3.09096162  6.41962885  0.3837507  ...  1.30014462  0.14192193
   -3.75912086]
  [ 3.15971335  6.48500103  0.40526569 ...  1.25974027  0.20453632
   -3.86178132]
  [ 0.63945916 -1.1304109  -0.74444328 ...  1.86229298 -2.94315723
    1.91299072]]

 ...

 [[ 0.24413171 -1.77120864 -0.81233591 ...  3.21259207  0.43097956
    3.25990132]
  [ 0.44533161 -1.31569924 -0.64796026 ...  3.1800236   0.6045574
    3.15987864]
  [ 3.36655761  5.2973567   1.75064552 ...  2.6728601   3.14037056
    1.69081288]
  [ 0.44809837  2.81245393  1.1168635  ...  0.36076492  2.42829345
    0.52082055]
  [ 0.31718308 -1.56516213 -0.7300772  ...  3.15089861  0.52540473
    3.18531628]]

 [[-1.24195709 -2.40513444 -1.60342133 ...  1.48904369 -1.28392825
   -0.10417113]
  [-0.81987015  0.92844469 -1.38007812 ...  3.57147195 -2.95665296
    0.46726103]
  [ 0.57122004  3.08912381  1.53377398 ... -0.19168696  1.3939471
    2.0800465 ]
  [ 0.43263325 -1.89560183 -0.88595476 ... -1.41954168  0.23889672
   -2.32849922]
  [ 0.06658936  1.62020413  0.57646473 ...  0.43944132  0.44043603
    1.38041953]]

 [[-2.50422211 -6.10252394  2.15616899 ... -0.12749075 -3.05709315
    7.06989817]
  [ 1.69929339 -0.96767492  0.82998026 ... -0.24005252 -3.79681386
    3.28837821]
  [ 1.69927007 -0.96770342  0.82998759 ... -0.24005192 -3.79680972
    3.28839911]
  [-2.02190098 -2.43940498 -1.85465602 ... -1.69335101 -0.39778574
   -2.53800194]
  [ 1.04684875 -1.76242162  1.03499092 ... -0.22269616 -3.68103031
    3.87253629]]]
Multi Head Scaled Attention Shape: (64, 5, 64)
```

```
Epoch 0: {'correct': ['sos how are you eos'], 'translated': ['sos']}
Epoch 5: {'correct': ['sos how are you eos'], 'translated': ['sos ca eos']}
Epoch 10: {'correct': ['sos how are you eos'], 'translated': ['sos comment il eos']}
Epoch 15: {'correct': ['sos how are you eos'], 'translated': ['sos comment allez eos']}
Epoch 20: {'correct': ['sos how are you eos'], 'translated': ['sos comment allez eos']}
Epoch 25: {'correct': ['sos how are you eos'], 'translated': ['sos comment allez vous eos']}
Epoch 30: {'correct': ['sos how are you eos'], 'translated': ['sos comment allez vous eos']}
```


Enc + Dec Training Loss vs Validation Loss
Train
Val
Loss
Epochs
0
5
10
15
20
25
30
1
2
3
4
5
6
7



Enc + Dec Training Accuracy vs Validation Accuracy
Train
Val
Accuracy
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
0
5
10
15
20
25
30
Epochs