Task 4: Encoder-Decoder with LSTM in Keras

```
# Step 1: Set Parameters
BUFFER_SIZE = len(input_tensor)
BATCH_SIZE = 64
steps_per_epoch = BUFFER_SIZE // BATCH_SIZE
embedding_dim = 256
units = 512
vocab_inp_size = len(inp_tokenizer.word_index) + 1
vocab_tar_size = len(targ_tokenizer.word_index) + 1

dataset                =                tf.data.Dataset.from_tensor_slices((input_tensor,
target_tensor)).shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)

# Step 2: Build Encoder
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units):
        super(Encoder, self).__init__()
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.lstm = tf.keras.layers.LSTM(self.enc_units,
                                         return_sequences=True,
                                         return_state=True)

    def call(self, x, hidden):
        x = self.embedding(x)
        output, state_h, state_c = self.lstm(x, initial_state=hidden)
        return output, state_h, state_c

    def initialize_hidden_state(self, batch_sz):
                    return   [tf.zeros((batch_sz,   self.enc_units)),   tf.zeros((batch_sz,
self.enc_units))]

# Step 3: Build Decoder
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units):
        super(Decoder, self).__init__()
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.lstm = tf.keras.layers.LSTM(self.dec_units,
                                         return_sequences=True,
                                         return_state=True)
        self.fc = tf.keras.layers.Dense(vocab_size)

    def call(self, x, hidden):
        x = self.embedding(x)
        output, _, _ = self.lstm(x, initial_state=hidden)
        x = self.fc(output)
        return x

# Step 4: Compile and Loss Function
encoder = Encoder(vocab_inp_size, embedding_dim, units)
```

```
decoder = Decoder(vocab_tar_size, embedding_dim, units)

optimizer = tf.keras.optimizers.Adam()
loss_object       =       tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True,
reduction='none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)
    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask
    return tf.reduce_mean(loss_)

# Step 5: Train the Model
EPOCHS = 10

for epoch in range(EPOCHS):
    total_loss = 0
    for (batch, (inp, targ)) in enumerate(dataset):
        enc_hidden = encoder.initialize_hidden_state(BATCH_SIZE)
        with tf.GradientTape() as tape:
            enc_output, enc_hidden_h, enc_hidden_c = encoder(inp, enc_hidden)
            dec_hidden = [enc_hidden_h, enc_hidden_c]
                    dec_input  =  tf.expand_dims([targ_tokenizer.word_index['<start>']]  *
BATCH_SIZE, 1)

            predictions = decoder(dec_input, dec_hidden)
            loss = loss_function(targ, predictions)

        total_loss += loss

        variables = encoder.trainable_variables + decoder.trainable_variables
        gradients = tape.gradient(loss, variables)
        optimizer.apply_gradients(zip(gradients, variables))

    print(f'Epoch {epoch+1}, Loss: {total_loss/steps_per_epoch:.4f}')
```