Multilingual Code Switching ASR (Automatic Speech Recognition)

Source Code:

```
import pandas as pd

import tensorflow as tf

import numpy as np


# Load the dataset from CSV

df = pd.read_excel('/content/dataset1.xlsx')


# Tokenize code-switched Tamil-English sentences

tamil_english_code_switch = df['Tamil'].tolist()

tokenizer_cs = tf.keras.preprocessing.text.Tokenizer(filters='')

tokenizer_cs.fit_on_texts(tamil_english_code_switch)

tokenized_cs = tokenizer_cs.texts_to_sequences(tamil_english_code_switch)


# Tokenize pure English translations

pure_english_translation = df['English'].tolist()

tokenizer_en = tf.keras.preprocessing.text.Tokenizer(filters='')

tokenizer_en.fit_on_texts(pure_english_translation)

tokenized_en = tokenizer_en.texts_to_sequences(pure_english_translation)


# Pad sequences

max_length = 100  # Set your desired maximum sequence length

padded_cs = tf.keras.preprocessing.sequence.pad_sequences(tokenized_cs,
maxlen=max_length, padding='post')

padded_en = tf.keras.preprocessing.sequence.pad_sequences(tokenized_en,
maxlen=max_length, padding='post')


# Split the dataset into training and validation sets

train_size = int(0.8 * len(padded_cs))

train_cs, val_cs = padded_cs[:train_size], padded_cs[train_size:]
```

```
train_en, val_en = padded_en[:train_size], padded_en[train_size:]


# Define the vocabulary sizes
vocab_size_cs = len(tokenizer_cs.word_index) + 1
vocab_size_en = len(tokenizer_en.word_index) + 1
import pandas as pd
import tensorflow as tf


# Load the dataset from CSV
df = pd.read_excel('/content/dataset1.xlsx')


tamil_english_code_switch = df['Tamil'].tolist()
tokenizer_cs = tf.keras.preprocessing.text.Tokenizer(filters='', oov_token='<unk>')
  # Add this line to fit the special tokens
tokenizer_cs.fit_on_texts(tamil_english_code_switch)
tokenized_cs = tokenizer_cs.texts_to_sequences(tamil_english_code_switch)


# Tokenize pure English translations
pure_english_translation = df['English'].tolist()
tokenizer_en = tf.keras.preprocessing.text.Tokenizer(filters='', oov_token='<unk>')
tokenizer_en.fit_on_texts(pure_english_translation)
tokenized_en = tokenizer_en.texts_to_sequences(pure_english_translation)


# Pad sequences if necessary
max_length = 100  # Set your desired maximum sequence length
padded_cs = tf.keras.preprocessing.sequence.pad_sequences(tokenized_cs,
maxlen=max_length, padding='post')
padded_en = tf.keras.preprocessing.sequence.pad_sequences(tokenized_en,
maxlen=max_length, padding='post')


# Split the dataset into training and validation sets
```

```
train_size = int(0.8 * len(padded_cs))
train_cs, val_cs = padded_cs[:train_size], padded_cs[train_size:]
train_en, val_en = padded_en[:train_size], padded_en[train_size:]

# Define the vocabulary sizes
vocab_size_cs = len(tokenizer_cs.word_index)+1
vocab_size_en = len(tokenizer_en.word_index)+1
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, Attention, Concatenate

# Define embedding dimensions and encoder units
embedding_dim = 256
encoder_units = 512
decoder_units = 512

# Define the encoder
# Define the encoder input layer with the correct shape
encoder_input = Input(shape=(padded_cs.shape[1],))
encoder_embedding = Embedding(vocab_size_cs, embedding_dim, input_length=max_length)(encoder_input)
encoder_lstm = LSTM(encoder_units, return_sequences=True, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)
encoder_states = [state_h, state_c]

# Define the decoder
decoder_input = Input(shape=(None,))
decoder_embedding = Embedding(vocab_size_en, embedding_dim)
decoder_embedded = decoder_embedding(decoder_input)
decoder_lstm = LSTM(decoder_units, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedded, initial_state=encoder_states)
```

 # Define attention mechanism

attention = Attention()

context_vector = attention([decoder_outputs, encoder_outputs])

 # Concatenate context vector and decoder output

decoder_combined_context = Concatenate(axis=-1)([context_vector, decoder_outputs])

 # Dense layer to output probabilities over the target vocabulary

decoder_dense = Dense(vocab_size_en, activation='softmax')

decoder_outputs = decoder_dense(decoder_combined_context)

 # Define the model

model = Model([encoder_input, decoder_input], decoder_outputs)

 # Compile the model

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

 # Print model summary

model.summary()

 # Train the model

batch_size = 4

epochs = 10

model.fit([train_cs, train_en[:, :-1]], train_en[:, 1:], validation_data=([val_cs, val_en[:, :-1]], val_en[:, 1:]), batch_size=batch_size, epochs=epochs)

 # Define the input sentence

input_sentence = "Sometimes makkal parunga"

tokenized_sequence = tokenizer_cs.texts_to_sequences([input_sentence])[0]

```
for i, token_id in enumerate(tokenized_sequence):
    if token_id == tokenizer_cs.word_index['<unk>']:
        original_token = input_sentence.split()[i]
        tokenized_sequence[i] = tokenizer_cs.word_index.get(original_token, token_id)
words = [tokenizer_en.index_word.get(token_id, '<unk>') for token_id in tokenized_sequence]
translated_sentence = ' '.join(words)
translated_sentence = translated_sentence.replace('<unk>', 'sometimes')

print("Tokenized Sequence:", tokenized_sequence)
print("Translated Sentence:", translated_sentence)
```