```
import json
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder

# 1. Load Data
with open('finger_sequences.json', 'r') as f:
    data = json.load(f)

finger_sequences = []
time_sequences = []
labels = []

# Each key in the JSON is a gesture label (e.g. "CallHarisVai", "Noise", etc.)
for gesture, sequences in data.items():
    for seq in sequences:
        f_seq = []
        t_seq = []
        for pair in seq:
            f_seq.append(int(pair[0]))       # Finger ID as int
            t_seq.append(float(pair[1]))     # Time as float
        finger_sequences.append(f_seq)
        time_sequences.append(t_seq)
        labels.append(gesture)

# 2. Compute global mean duration for padding
all_times = [t for seq in time_sequences for t in seq]
mean_duration = np.mean(all_times)

# 3. Define a finger map for label-specific padding
finger_map = {
    "CallHarisVai": 2,
    "Noise": 1,
    "LikeMe": 3,
    "ComeClose": 8  # Ensure the keys match your actual labels!
}

# 4. Custom padding function returning separate arrays for fingers and times
def custom_pad_sequences(finger_seqs, time_seqs, labels, maxlen, padding='post'):
    padded_fingers = []
    padded_times = []
    for i, (f_seq, t_seq) in enumerate(zip(finger_seqs, time_seqs)):
        label = labels[i]
        # Use label-specific pad value; default is -1
        pad_finger = finger_map.get(label, -1)
        pad_time = mean_duration
        if len(f_seq) < maxlen:
            pad_count = maxlen - len(f_seq)
            if padding == 'pre':
                padded_f_seq = [pad_finger] * pad_count + f_seq
                padded_t_seq = [pad_time] * pad_count + t_seq
            else:  # 'post'
                padded_f_seq = f_seq + [pad_finger] * pad_count
                padded_t_seq = t_seq + [pad_time] * pad_count
        else:
            padded_f_seq = f_seq[:maxlen]
            padded_t_seq = t_seq[:maxlen]
        padded_fingers.append(padded_f_seq)
        padded_times.append(padded_t_seq)
    return np.array(padded_fingers, dtype=np.int32), np.array(padded_times, dtype=np.float32)

# 5. Determine maximum sequence length and apply custom padding
max_len = max(len(seq) for seq in finger_sequences)
padded_fingers, padded_times = custom_pad_sequences(finger_sequences, time_sequences, labels, maxlen=max_len)

# 6. Encode labels to one-hot vectors
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)
num_classes = len(label_encoder.classes_)
onehot_labels = tf.keras.utils.to_categorical(encoded_labels, num_classes=num_classes)

# 7. Build the TensorFlow Model
embedding_dim = 8
# Calculate vocab_size based on your original data (assuming padded values are handled separately)
max_finger = max(max(seq) for seq in finger_sequences)
```

```
vocab_size = max_finger + 1  # Since we assume indices 0..max_finger

# Define inputs
finger_input = tf.keras.Input(shape=(max_len,), name='finger_input')
time_input = tf.keras.Input(shape=(max_len,), name='time_input')

# Note: Set mask_zero=False because our custom padding tokens are not zero.
x1 = tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=embedding_dim, mask_zero=False)(finger_input)
# Reshape time input to have a channel dimension
x2 = tf.keras.layers.Reshape((max_len, 1))(time_input)
# Concatenate the embedded finger data with the time data
x = tf.keras.layers.Concatenate(axis=-1)([x1, x2])

# Process the sequence with an LSTM
x = tf.keras.layers.LSTM(64)(x)
x = tf.keras.layers.Dense(32, activation='relu')(x)
output = tf.keras.layers.Dense(num_classes, activation='softmax')(x)

model = tf.keras.Model(inputs=[finger_input, time_input], outputs=output)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# 8. Train the Model
history = model.fit(
    [padded_fingers, padded_times],
    onehot_labels,
    epochs=20,
    batch_size=32,
    validation_split=0.2
)
```

```
Model: "functional_3"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)              ┃ Output Shape           ┃        Param # ┃ Connected to           ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━┩
│ finger_input (InputLayer) │ (None, 33)             │              0 │ -                      │
├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤
│ time_input (InputLayer)   │ (None, 33)             │              0 │ -                      │
├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤
│ embedding_3 (Embedding)   │ (None, 33, 8)          │             72 │ finger_input[0][0]     │
├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤
│ reshape_3 (Reshape)       │ (None, 33, 1)          │              0 │ time_input[0][0]       │
├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤
│ concatenate_6             │ (None, 33, 9)          │              0 │ embedding_3[0][0],     │
│ (Concatenate)             │                        │                │ reshape_3[0][0]        │
└───────────────────────────┴────────────────────────┴────────────────┴────────────────────────┘
```

```python
# 9. Prediction Example
# Suppose we have a new gesture sequence
new_sequence = [
    ["1", 0.5],
    ["2", -0.3],
    ["3", 1.2]
]
new_finger_seq = [int(pair[0]) for pair in new_sequence]
new_time_seq = [float(pair[1]) for pair in new_sequence]

# Use the same custom padding for the new sample; here, assume label "CallHarisVai" for padding purposes
new_padded_fingers, new_padded_times = custom_pad_sequences([new_finger_seq], [new_time_seq], ["CallHarisVai"], maxlen=max_len)

prediction = model.predict([new_padded_fingers, new_padded_times])
predicted_class = label_encoder.inverse_transform([np.argmax(prediction)])
print("Predicted Gesture:", predicted_class[0])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 659ms/step
Predicted Gesture: CallHarisVai
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - accuracy: 0.9062 - loss: 1.2609 - val_accuracy: 0.0000e+00 - val_loss: 1.3163
Epoch 8/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step - accuracy: 0.8854 - loss: 1.2067 - val_accuracy: 0.0000e+00 - val_loss: 1.2874
Epoch 9/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step - accuracy: 0.9167 - loss: 1.1437 - val_accuracy: 0.0000e+00 - val_loss: 1.2585
Epoch 10/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.8958 - loss: 1.0432 - val_accuracy: 0.0000e+00 - val_loss: 1.2326
Epoch 11/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 0.8854 - loss: 0.9151 - val_accuracy: 0.0000e+00 - val_loss: 1.2305
Epoch 12/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.8750 - loss: 0.7632 - val_accuracy: 0.0000e+00 - val_loss: 1.2820
Epoch 13/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - accuracy: 0.8021 - loss: 0.6845 - val_accuracy: 0.0000e+00 - val_loss: 1.2209
Epoch 14/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - accuracy: 0.8333 - loss: 0.5697 - val_accuracy: 0.5000 - val_loss: 1.0535
Epoch 15/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.9167 - loss: 0.4296 - val_accuracy: 0.8125 - val_loss: 0.8503
Epoch 16/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.9583 - loss: 0.3693 - val_accuracy: 0.8125 - val_loss: 0.7562
Epoch 17/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 145ms/step - accuracy: 0.9896 - loss: 0.2946 - val_accuracy: 0.8750 - val_loss: 0.6655
Epoch 18/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step - accuracy: 0.9896 - loss: 0.2393 - val_accuracy: 0.8750 - val_loss: 0.6467
Epoch 19/20
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step - accuracy: 1.0000 - loss: 0.1961 - val_accuracy: 0.7500 - val_loss: 0.7372
```