B0: Thiết lập Môi trường và Tải Dữ liệu

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Correcting the file paths based on the extraction location
df_train = pd.read_csv('/content/data_extracted/hwu/train.csv', sep=',', header=None, names=['text', 'intent'])
df_val = pd.read_csv('/content/data_extracted/hwu/val.csv', sep=',', header=None, names=['text', 'intent'])
df_test = pd.read_csv('/content/data_extracted/hwu/test.csv', sep=',', header=None, names=['text', 'intent'])

print("Train shape:", df_train.shape)
print("Validation shape:", df_val.shape)
print("Test shape:", df_test.shape)
```

```
Train shape: (8955, 2)
Validation shape: (1077, 2)
Test shape: (1077, 2)
```

```
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit LabelEncoder on the combined intents to ensure all possible intents are covered
all_intents = pd.concat([df_train['intent'], df_val['intent'], df_test['intent']], axis=0)
label_encoder.fit(all_intents)

# Transform the intent columns in train, validation, and test dataframes
df_train['intent_encoded'] = label_encoder.transform(df_train['intent'])
df_val['intent_encoded'] = label_encoder.transform(df_val['intent'])
df_test['intent_encoded'] = label_encoder.transform(df_test['intent'])

print("First 5 rows of df_train with encoded intents:")
display(df_train.head())

print("\nClasses (original intent labels) and their numerical encoding:")
display(pd.Series(label_encoder.classes_))
```

```
First 5 rows of df_train with encoded intents:
```

	text	intent	intent_encoded
0	text	category	9
1	what alarms do i have set right now	alarm_query	0
2	checkout today alarm of meeting	alarm_query	0
3	report alarm settings	alarm_query	0
4	see see for me the alarms that you have set to...	alarm_query	0

```
Classes (original intent labels) and their numerical encoding:
```

	0
0	alarm_query
1	alarm_remove
2	alarm_set
3	audio_volume_down
4	audio_volume_mute
...	...
60	transport_query
61	transport_taxi
62	transport_ticket
63	transport_traffic
64	weather_query

65 rows × 1 columns

dtype: object

B1: Pipeline TF-IDF + Logistic Regression

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report

# 1. Tạo pipeline gồm TF-IDF và Logistic Regression
tfidf_lr_pipeline = make_pipeline(
    TfidfVectorizer(max_features=5000),
    LogisticRegression(max_iter=1000, solver='lbfgs', n_jobs=-1)
)

# 2. Huấn luyện pipeline trên tập train
tfidf_lr_pipeline.fit(df_train['text'], df_train['intent_encoded'])

# 3. Dự đoán trên tập test
y_pred = tfidf_lr_pipeline.predict(df_test['text'])

# 4. Đánh giá mô hình
print("=== Classification Report (TF-IDF + Logistic Regression) ===")
print(classification_report(
    df_test['intent_encoded'],
    y_pred,
    target_names=label_encoder.classes_
))
```

```
        email_sendemail       0.77      0.89      0.83        19
         general_affirm       1.00      1.00      1.00        19
    general_commandstop       1.00      1.00      1.00        19
        general_confirm       1.00      1.00      1.00        19
       general_dontcare       0.90      1.00      0.95        19
        general_explain       1.00      0.95      0.97        19
           general_joke       1.00      1.00      1.00        12
         general_negate       0.95      1.00      0.97        19
         general_praise       0.95      1.00      0.97        19
         general_quirky       0.36      0.26      0.30        19
         general_repeat       0.90      1.00      0.95        19
           iot_cleaning       1.00      1.00      1.00        16
             iot_coffee       1.00      0.95      0.97        19
    iot_hue_lightchange       0.75      0.79      0.77        19
       iot_hue_lightdim       0.91      0.83      0.87        12
       iot_hue_lightoff       0.89      0.89      0.89        19
        iot_hue_lighton       0.67      0.67      0.67         3
        iot_hue_lightup       0.92      0.86      0.89        14
           iot_wemo_off       0.80      0.89      0.84         9
            iot_wemo_on       0.78      1.00      0.88         7
      lists_createoradd       0.68      0.79      0.73        19
            lists_query       0.75      0.79      0.77        19
           lists_remove       0.85      0.89      0.87        19
         music_likeness       0.65      0.61      0.63        18
            music_query       0.71      0.53      0.61        19
         music_settings       1.00      0.57      0.73         7
             news_query       0.75      0.63      0.69        19
         play_audiobook       0.95      0.95      0.95        19
              play_game       0.81      0.68      0.74        19
             play_music       0.58      0.74      0.65        19
          play_podcasts       1.00      0.84      0.91        19











               accuracy                           0.84      1077
              macro avg       0.83      0.82      0.82      1077
           weighted avg       0.84      0.84      0.83      1077

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defi
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defi
```

B2: Pipeline Word2Vec (Trung bình) + Dense Layer

```python
import numpy as np
from gensim.models import Word2Vec
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical

# 1. Huấn luyện mô hình Word2Vec trên dữ liệu văn bản
sentences = [text.split() for text in df_train['text']]
w2v_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# 2. Hàm chuyển câu thành vector trung bình
def sentence_to_avg_vector(text, model):
    words = text.split()
    valid_words = [w for w in words if w in model.wv]
    if not valid_words:
        return np.zeros(model.vector_size)
    return np.mean([model.wv[w] for w in valid_words], axis=0)

# 3. Tạo dữ liệu train/val/test
X_train_avg = np.array([sentence_to_avg_vector(t, w2v_model) for t in df_train['text']])
X_val_avg = np.array([sentence_to_avg_vector(t, w2v_model) for t in df_val['text']])
X_test_avg = np.array([sentence_to_avg_vector(t, w2v_model) for t in df_test['text']])

y_train = to_categorical(df_train['intent_encoded'])
y_val = to_categorical(df_val['intent_encoded'])
y_test = to_categorical(df_test['intent_encoded'])

num_classes = y_train.shape[1]

print("X_train_avg shape:", X_train_avg.shape)
print("y_train shape:", y_train.shape)

# 4. Xây dựng mô hình Dense đơn giản
model = Sequential([
    Dense(128, activation='relu', input_shape=(w2v_model.vector_size,)),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

# 5. Compile mô hình
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# 6. Huấn luyện mô hình
history = model.fit(
    X_train_avg, y_train,
    validation_data=(X_val_avg, y_val),
    epochs=10,
    batch_size=32,
    verbose=1
)

# 7. Đánh giá mô hình
test_loss, test_acc = model.evaluate(X_test_avg, y_test, verbose=0)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
X_train_avg shape: (8955, 100)
y_train shape: (8955, 65)
Epoch 1/10
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
280/280 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.0195 - loss: 4.1626 - val_accuracy: 0.0529 - val_loss: 4.1031
Epoch 2/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.0307 - loss: 4.1047 - val_accuracy: 0.0650 - val_loss: 4.0321
Epoch 3/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.0511 - loss: 4.0250 - val_accuracy: 0.0734 - val_loss: 3.9172
Epoch 4/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.0602 - loss: 3.9122 - val_accuracy: 0.0780 - val_loss: 3.7949
Epoch 5/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.0695 - loss: 3.8046 - val_accuracy: 0.0929 - val_loss: 3.6901
Epoch 6/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.0829 - loss: 3.7334 - val_accuracy: 0.0826 - val_loss: 3.6197
Epoch 7/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.0881 - loss: 3.6415 - val_accuracy: 0.1068 - val_loss: 3.5633
Epoch 8/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.0937 - loss: 3.6000 - val_accuracy: 0.1439 - val_loss: 3.4897
Epoch 9/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.1038 - loss: 3.5505 - val_accuracy: 0.1699 - val_loss: 3.4485
Epoch 10/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.1033 - loss: 3.4985 - val_accuracy: 0.1662 - val_loss: 3.4114
Test Accuracy: 0.1448
```

B3: Mô hình Nâng cao (Embedding Pre-trained + LSTM)

```python
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical

# 1. Tokenizer và chuyển văn bản thành chuỗi chỉ số
tokenizer = Tokenizer(oov_token="<UNK>")
tokenizer.fit_on_texts(df_train['text'])

# Chuyển text thành chuỗi số
train_sequences = tokenizer.texts_to_sequences(df_train['text'])
val_sequences = tokenizer.texts_to_sequences(df_val['text'])
test_sequences = tokenizer.texts_to_sequences(df_test['text'])

# Padding để các chuỗi có cùng độ dài
max_len = 50
X_train_pad = pad_sequences(train_sequences, maxlen=max_len, padding='post')
X_val_pad = pad_sequences(val_sequences, maxlen=max_len, padding='post')
X_test_pad = pad_sequences(test_sequences, maxlen=max_len, padding='post')

y_train = to_categorical(df_train['intent_encoded'])
y_val = to_categorical(df_val['intent_encoded'])
y_test = to_categorical(df_test['intent_encoded'])

num_classes = y_train.shape[1]

# 2. Tạo ma trận trọng số Embedding từ mô hình Word2Vec đã huấn luyện
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = w2v_model.vector_size

embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    if word in w2v_model.wv:
        embedding_matrix[i] = w2v_model.wv[word]

print("Embedding matrix shape:", embedding_matrix.shape)

# 3. Xây dựng mô hình LSTM với Embedding pre-trained
lstm_model_pretrained = Sequential([
    Embedding(
        input_dim=vocab_size,
        output_dim=embedding_dim,
        weights=[embedding_matrix],
        input_length=max_len,
        trainable=False  # đóng băng lớp embedding
    ),
    LSTM(128, dropout=0.2, recurrent_dropout=0.2),
    Dense(num_classes, activation='softmax')
])
```

```
# 4. Compile mô hình
lstm_model_pretrained.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# 5. Huấn luyện mô hình (dùng EarlyStopping để tránh overfitting)
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = lstm_model_pretrained.fit(
    X_train_pad, y_train,
    validation_data=(X_val_pad, y_val),
    epochs=15,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)

# 6. Đánh giá mô hình
test_loss, test_acc = lstm_model_pretrained.evaluate(X_test_pad, y_test, verbose=0)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
Embedding matrix shape: (4265, 100)
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecate
  warnings.warn(
Epoch 1/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 42s 121ms/step - accuracy: 0.0184 - loss: 4.1600 - val_accuracy: 0.0176 - val_loss: 4.1322
Epoch 2/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 32s 115ms/step - accuracy: 0.0167 - loss: 4.1424 - val_accuracy: 0.0176 - val_loss: 4.1288
Epoch 3/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 34s 123ms/step - accuracy: 0.0210 - loss: 4.1261 - val_accuracy: 0.0223 - val_loss: 4.0819
Epoch 4/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 32s 115ms/step - accuracy: 0.0279 - loss: 4.0411 - val_accuracy: 0.0474 - val_loss: 3.9173
Epoch 5/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 34s 120ms/step - accuracy: 0.0492 - loss: 3.8962 - val_accuracy: 0.0464 - val_loss: 3.8574
Epoch 6/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 32s 116ms/step - accuracy: 0.0487 - loss: 3.8538 - val_accuracy: 0.0641 - val_loss: 3.7920
Epoch 7/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 41s 117ms/step - accuracy: 0.0550 - loss: 3.8115 - val_accuracy: 0.0752 - val_loss: 3.6674
Epoch 8/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 32s 113ms/step - accuracy: 0.0606 - loss: 3.7396 - val_accuracy: 0.0761 - val_loss: 3.6095
Epoch 9/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 41s 113ms/step - accuracy: 0.0724 - loss: 3.6792 - val_accuracy: 0.0882 - val_loss: 3.5549
Epoch 10/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 39s 138ms/step - accuracy: 0.0695 - loss: 3.6170 - val_accuracy: 0.0994 - val_loss: 3.5039
Epoch 11/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 39s 132ms/step - accuracy: 0.0763 - loss: 3.5861 - val_accuracy: 0.0891 - val_loss: 3.4960
Epoch 12/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 35s 124ms/step - accuracy: 0.0775 - loss: 3.5384 - val_accuracy: 0.0919 - val_loss: 3.4493
Epoch 13/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 39s 118ms/step - accuracy: 0.0827 - loss: 3.5146 - val_accuracy: 0.0808 - val_loss: 3.5263
Epoch 14/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 32s 113ms/step - accuracy: 0.0768 - loss: 3.5660 - val_accuracy: 0.0901 - val_loss: 3.4687
Epoch 15/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 34s 121ms/step - accuracy: 0.0893 - loss: 3.4767 - val_accuracy: 0.1086 - val_loss: 3.3615
Test Accuracy: 0.1077
```

B4: Mô hình Nâng cao (Embedding học từ đầu + LSTM)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical

# 1. Xây dựng mô hình LSTM (Embedding học từ đầu)
lstm_model_scratch = Sequential([
    Embedding(
        input_dim=vocab_size,
        output_dim=100,      # embedding_dim tùy chọn (có thể thử 100 hoặc 300)
        input_length=max_len # phải khớp với padding ở nhiệm vụ 3
    ),
    LSTM(128, dropout=0.2, recurrent_dropout=0.2),
    Dense(num_classes, activation='softmax')
])
```

```
# 2. Compile mô hình
lstm_model_scratch.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# 3. Sử dụng EarlyStopping để ngăn overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# 4. Huấn luyện mô hình
history_scratch = lstm_model_scratch.fit(
    X_train_pad, y_train,
    validation_data=(X_val_pad, y_val),
    epochs=15,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)

# 5. Đánh giá mô hình trên tập test
test_loss, test_acc = lstm_model_scratch.evaluate(X_test_pad, y_test, verbose=0)
print(f"Test Accuracy (Embedding học từ đầu + LSTM): {test_acc:.4f}")
```

```
Epoch 1/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 42s 132ms/step - accuracy: 0.0183 - loss: 4.1584 - val_accuracy: 0.0176 - val_loss: 4.1305
Epoch 2/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 50s 177ms/step - accuracy: 0.0159 - loss: 4.1368 - val_accuracy: 0.0176 - val_loss: 4.1317
Epoch 3/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 43s 152ms/step - accuracy: 0.0151 - loss: 4.1360 - val_accuracy: 0.0176 - val_loss: 4.1293
Epoch 4/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 80s 147ms/step - accuracy: 0.0171 - loss: 4.1337 - val_accuracy: 0.0176 - val_loss: 4.1300
Epoch 5/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 79s 137ms/step - accuracy: 0.0133 - loss: 4.1324 - val_accuracy: 0.0176 - val_loss: 4.1288
Epoch 6/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 40s 132ms/step - accuracy: 0.0174 - loss: 4.1345 - val_accuracy: 0.0176 - val_loss: 4.1287
Epoch 7/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 38s 135ms/step - accuracy: 0.0193 - loss: 4.1320 - val_accuracy: 0.0176 - val_loss: 4.1284
Epoch 8/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 38s 136ms/step - accuracy: 0.0168 - loss: 4.1308 - val_accuracy: 0.0176 - val_loss: 4.1285
Epoch 9/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 38s 136ms/step - accuracy: 0.0154 - loss: 4.1322 - val_accuracy: 0.0176 - val_loss: 4.1295
Epoch 10/15
280/280 ━━━━━━━━━━━━━━━━━━━━ 38s 134ms/step - accuracy: 0.0140 - loss: 4.1330 - val_accuracy: 0.0176 - val_loss: 4.1286
Test Accuracy (Embedding học từ đầu + LSTM): 0.0176
```

B5: Đánh giá, So sánh và Phân tích

```
from sklearn.metrics import f1_score, log_loss

# TF-IDF + Logistic Regression
y_test_pred_tf = tfidf_lr_pipeline.predict(df_test['text'])
f1_tf = f1_score(df_test['intent_encoded'], y_test_pred_tf, average='macro')

# Word2Vec trung bình + Dense Layer
y_test_pred_w2v = np.argmax(lstm_model_scratch.predict(X_test_avg), axis=1)
f1_w2v = f1_score(df_test['intent_encoded'], y_test_pred_w2v, average='macro')
loss_w2v = log_loss(y_test, lstm_model_scratch.predict(X_test_avg))

# Embedding Pre-trained + LSTM
y_test_pred_pretrained = np.argmax(lstm_model_pretrained.predict(X_test_pad), axis=1)
f1_pretrained = f1_score(df_test['intent_encoded'], y_test_pred_pretrained, average='macro')
loss_pretrained = log_loss(y_test, lstm_model_pretrained.predict(X_test_pad))

# Embedding học từ đầu + LSTM
y_test_pred_scratch = np.argmax(lstm_model_scratch.predict(X_test_pad), axis=1)
f1_scratch = f1_score(df_test['intent_encoded'], y_test_pred_scratch, average='macro')
loss_scratch = log_loss(y_test, lstm_model_scratch.predict(X_test_pad))

# Tạo bảng tổng hợp
import pandas as pd

results = pd.DataFrame({
    'Pipeline': [
        'TF-IDF + Logistic Regression',
        'Word2Vec (Avg) + Dense',
        'Embedding (Pre-trained) + LSTM',
        'Embedding (Scratch) + LSTM'
    ],
    'F1-score (Macro)': [f1_tf, f1_w2v, f1_pretrained, f1_scratch],
    'Test Loss': [0, loss_w2v, loss_pretrained, loss_scratch]
})

print(results)
```

```
34/34 ━━━━━━━━━━━━━━━━━━━━ 4s 122ms/step
34/34 ━━━━━━━━━━━━━━━━━━━━ 3s 87ms/step
34/34 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step
34/34 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step
34/34 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step
34/34 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step
                         Pipeline  F1-score (Macro)  Test Loss
0    TF-IDF + Logistic Regression          0.822567   0.000000
1          Word2Vec (Avg) + Dense          0.000533   4.128409
2  Embedding (Pre-trained) + LSTM          0.054002   3.392387
3      Embedding (Scratch) + LSTM          0.000533   4.128409
```

```
# Các câu thử nghiệm
test_sentences = [
    "can you remind me to not call my mom",
    "is it going to be sunny or rainy tomorrow",
    "find a flight from new york to london but not through paris"
]

# Hàm dự đoán cho TF-IDF + Logistic Regression
def predict_tf(sentence):
    return label_encoder.inverse_transform(tfidf_lr_pipeline.predict([sentence]))[0]

# Hàm dự đoán cho Word2Vec trung bình + Dense
def predict_w2v(sentence):
    vec = sentence_to_avg_vector(sentence, w2v_model).reshape(1, -1)
    return label_encoder.inverse_transform([np.argmax(lstm_model_scratch.predict(vec))])[0]

# Hàm dự đoán cho LSTM Pre-trained
def predict_pretrained(sentence):
    seq = tokenizer.texts_to_sequences([sentence])
    pad_seq = pad_sequences(seq, maxlen=max_len, padding='post')
    return label_encoder.inverse_transform([np.argmax(lstm_model_pretrained.predict(pad_seq))])[0]

# Hàm dự đoán cho LSTM Scratch
def predict_scratch(sentence):
    seq = tokenizer.texts_to_sequences([sentence])
    pad_seq = pad_sequences(seq, maxlen=max_len, padding='post')
    return label_encoder.inverse_transform([np.argmax(lstm_model_scratch.predict(pad_seq))])[0]

# Kiểm tra dự đoán
for sent in test_sentences:
    print(f"Sentence: {sent}")
    print(f"TF-IDF + LR: {predict_tf(sent)}")
    print(f"Word2Vec (Avg) + Dense: {predict_w2v(sent)}")
    print(f"LSTM Pre-trained: {predict_pretrained(sent)}")
    print(f"LSTM Scratch: {predict_scratch(sent)}")
    print("-"*50)
```

```
Sentence: can you remind me to not call my mom
TF-IDF + LR: calendar_set
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
Word2Vec (Avg) + Dense: alarm_set
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
LSTM Pre-trained: datetime_query
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
LSTM Scratch: alarm_set
--------------------------------------------------
Sentence: is it going to be sunny or rainy tomorrow
TF-IDF + LR: weather_query
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
Word2Vec (Avg) + Dense: alarm_set
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
LSTM Pre-trained: qa_currency
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
LSTM Scratch: alarm_set
```

```
--------------------------------------------------
Sentence: find a flight from new york to london but not through paris
TF-IDF + LR: general_negate
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
Word2Vec (Avg) + Dense: alarm_set
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
LSTM Pre-trained: alarm_set
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
LSTM Scratch: alarm_set
--------------------------------------------------
```