GRU Weather Prediction - Binary Classification (RainToday)

Imports and CUDA

In [2]:
```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, random_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import os
```

```
/Users/raphaellong/anaconda3/lib/python3.11/site-packages/pandas/core/array
s/masked.py:60: UserWarning: Pandas requires version '1.3.6' or newer of 'b
ottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [3]:
```
# Use GPU if available, else use CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cpu
```

Load Data Paths

In [4]:
```
TRAIN_PATH = "/Users/raphaellong/Desktop/theory-and-practice-of-deep-learnir
TEST_PATH = "/Users/raphaellong/Desktop/theory-and-practice-of-deep-learning
SAVE_DIR = "saved_models"
os.makedirs(SAVE_DIR, exist_ok=True)
```

Configuration

In [5]:
```
SEQ_LENGTHS = [3, 5, 7]
BATCH_SIZE = 64
EPOCHS = 50
LEARNING_RATE = 1e-3
VALIDATION_SPLIT = 0.2
PATIENCE = 5
```

Custom Dataset

In [6]:
```
class RainDataset(Dataset):
    def __init__(self, sequences, labels):
        self.sequences = torch.tensor(sequences, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.float32)

    def __len__(self):
        return len(self.sequences)
```

```
    def __getitem__(self, idx):
        return self.sequences[idx], self.labels[idx]
```

Sequence Generator

In [7]:
```
def create_sequences(data, seq_len, feature_cols, target_col):
    sequences, labels = [], []
    for loc in data['Location'].unique():
        loc_data = data[data['Location'] == loc].sort_values('Date')
        X = loc_data[feature_cols].values
        y = loc_data[target_col].values
        for i in range(len(X) - seq_len):
            sequences.append(X[i:i+seq_len])
            labels.append(y[i+seq_len])
    return np.array(sequences), np.array(labels)
```

GRU Model

In [8]:
```
class RainGRU(nn.Module):
    def __init__(self, input_size, hidden_size=64):
        super().__init__()
        self.gru = nn.GRU(input_size, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(0.3)
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        out, _ = self.gru(x)
        out = self.dropout(out[:, -1, :])
        return self.fc(out)  # return raw logits
```

GRU + Transformer hybrid model

In [9]:
```
class RainGRUTransformer(nn.Module):
    def __init__(self, input_size, hidden_size=64, num_heads=4, num_layers=1
        super().__init__()
        self.gru = nn.GRU(input_size, hidden_size, batch_first=True)
        encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_size, nhea
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=n
        self.dropout = nn.Dropout(0.3)
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        gru_out, _ = self.gru(x)  # (B, T, H)
        trans_out = self.transformer(gru_out)  # (B, T, H)
        pooled = trans_out[:, -1, :]  # last time step
        out = self.dropout(pooled)
        return self.fc(out)
```

Training Function

In [10]:
```
def train(model, loader, optimizer, criterion):
    model.train()
    total_loss = 0
```

```
    for xb, yb in loader:
        optimizer.zero_grad()
        preds = model(xb).squeeze()
        loss = criterion(preds, yb)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(loader)
```

Evaluation Function

In [11]:
```
def evaluate(model, loader):
    model.eval()
    y_true, y_pred, y_prob = [], [], []
    with torch.no_grad():
        for xb, yb in loader:
            logits = model(xb).squeeze()
            probs = torch.sigmoid(logits)
            preds = probs > 0.5
            y_true.extend(yb.tolist())
            y_pred.extend(preds.int().tolist())
            y_prob.extend(probs.tolist())
    print(classification_report(y_true, y_pred))
    print(confusion_matrix(y_true, y_pred))

    # Plot predictions
    plt.figure(figsize=(12, 4))
    plt.plot(y_true[:100], label='Actual')
    plt.plot(y_prob[:100], label='Predicted Probability')
    plt.title("RainToday: Actual vs Predicted (First 100 Samples)")
    plt.legend()
    plt.show()
```

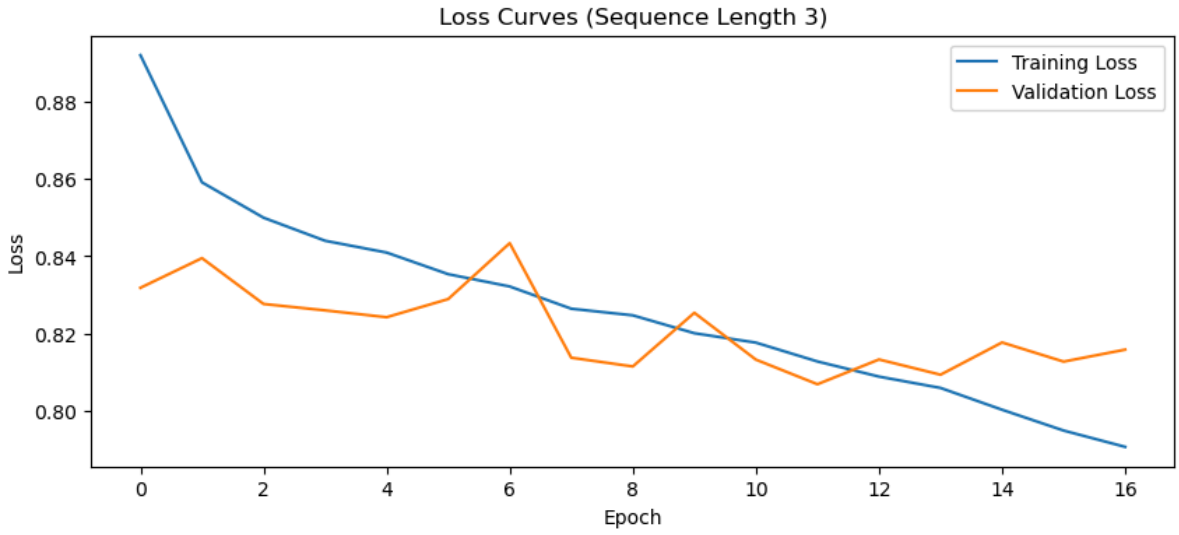
Run For Different Sequence Lengths

In [12]:
```
if __name__ == "__main__":
    train_df = pd.read_csv(TRAIN_PATH)
    test_df = pd.read_csv(TEST_PATH)

    TARGET_COL = "RainToday"
    FEATURE_COLS = train_df.columns.drop(TARGET_COL)

    scaler = MinMaxScaler()
    train_df[FEATURE_COLS] = scaler.fit_transform(train_df[FEATURE_COLS])
    test_df[FEATURE_COLS] = scaler.transform(test_df[FEATURE_COLS])

    # Compute class weights
    rain_count = train_df[TARGET_COL].sum()
    no_rain_count = len(train_df) - rain_count
    weight_ratio = no_rain_count / rain_count
    pos_weight = torch.tensor([weight_ratio], dtype=torch.float32)

    for seq_len in SEQ_LENGTHS:
        print(f"\n=== Training with sequence length {seq_len} ===")

        X_train, y_train = create_sequences(train_df, seq_len, FEATURE_COLS,
        X_test, y_test = create_sequences(test_df, seq_len, FEATURE_COLS, TA

        full_train_ds = RainDataset(X_train, y_train)
```

```
        val_size = int(len(full_train_ds) * VALIDATION_SPLIT)
        train_size = len(full_train_ds) - val_size
        train_ds, val_ds = random_split(full_train_ds, [train_size, val_size

        test_ds = RainDataset(X_test, y_test)

        train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=T
        val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE)
        test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE)

        # model = RainGRU(input_size=X_train.shape[2])
        model = RainGRUTransformer(input_size=X_train.shape[2])

        optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
        criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)


        best_val_loss = float('inf')
        patience_counter = 0
        best_model_path = os.path.join(SAVE_DIR, f"gru_seq{seq_len}.pt")

        train_losses = []
        val_losses = []

        for epoch in range(EPOCHS):
            loss = train(model, train_loader, optimizer, criterion)
            train_losses.append(loss)
            model.eval()
            val_loss = 0.0
            with torch.no_grad():
                for xb, yb in val_loader:
                    preds = model(xb).squeeze()
                    val_loss += criterion(preds, yb).item()
            val_loss /= len(val_loader)
            val_losses.append(val_loss)

            print(f"Epoch {epoch+1}/{EPOCHS} - Loss: {loss:.4f} - Val Loss:

            if val_loss < best_val_loss:
                best_val_loss = val_loss
                patience_counter = 0
                torch.save(model.state_dict(), best_model_path)
                print("Saved new best model.")
            else:
                patience_counter += 1
                if patience_counter >= PATIENCE:
                    print("Early stopping triggered.")
                    break

        # Plot training and validation loss
        plt.figure(figsize=(10, 4))
        plt.plot(train_losses, label='Training Loss')
        plt.plot(val_losses, label='Validation Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.title(f'Loss Curves (Sequence Length {seq_len})')
        plt.legend()
        plt.show()

        # Load best model
        model.load_state_dict(torch.load(best_model_path))

        print("\n--- Validation Set Evaluation ---")
        evaluate(model, val_loader)
```

```
        print("\n--- Test Set Evaluation ---")
        evaluate(model, test_loader)
```

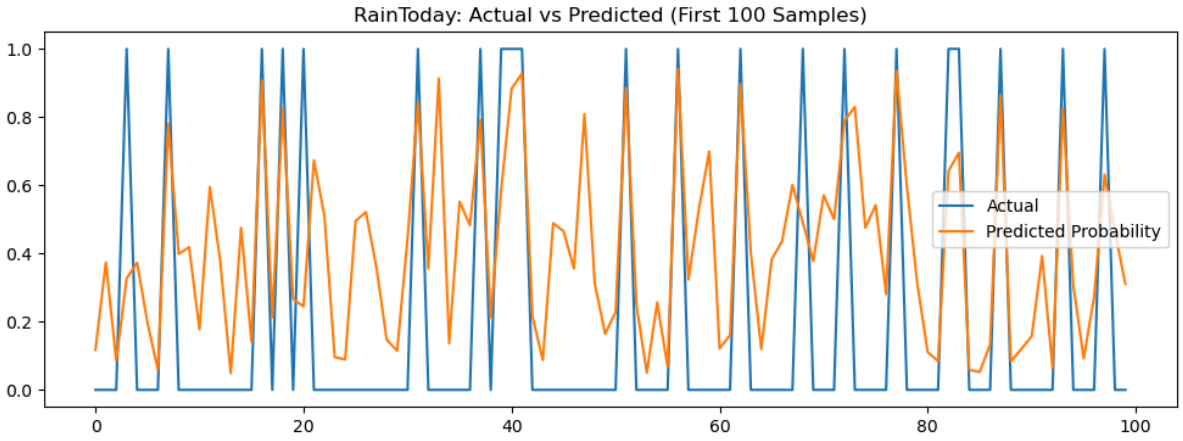
```
=== Training with sequence length 3 ===
Epoch 1/50 - Loss: 0.8921 - Val Loss: 0.8318
Saved new best model.
Epoch 2/50 - Loss: 0.8591 - Val Loss: 0.8395
Epoch 3/50 - Loss: 0.8500 - Val Loss: 0.8276
Saved new best model.
Epoch 4/50 - Loss: 0.8440 - Val Loss: 0.8260
Saved new best model.
Epoch 5/50 - Loss: 0.8410 - Val Loss: 0.8242
Saved new best model.
Epoch 6/50 - Loss: 0.8353 - Val Loss: 0.8289
Epoch 7/50 - Loss: 0.8322 - Val Loss: 0.8434
Epoch 8/50 - Loss: 0.8264 - Val Loss: 0.8137
Saved new best model.
Epoch 9/50 - Loss: 0.8247 - Val Loss: 0.8114
Saved new best model.
Epoch 10/50 - Loss: 0.8201 - Val Loss: 0.8253
Epoch 11/50 - Loss: 0.8176 - Val Loss: 0.8132
Epoch 12/50 - Loss: 0.8127 - Val Loss: 0.8068
Saved new best model.
Epoch 13/50 - Loss: 0.8088 - Val Loss: 0.8132
Epoch 14/50 - Loss: 0.8059 - Val Loss: 0.8093
Epoch 15/50 - Loss: 0.8002 - Val Loss: 0.8177
Epoch 16/50 - Loss: 0.7948 - Val Loss: 0.8127
Epoch 17/50 - Loss: 0.7906 - Val Loss: 0.8158
Early stopping triggered.
```



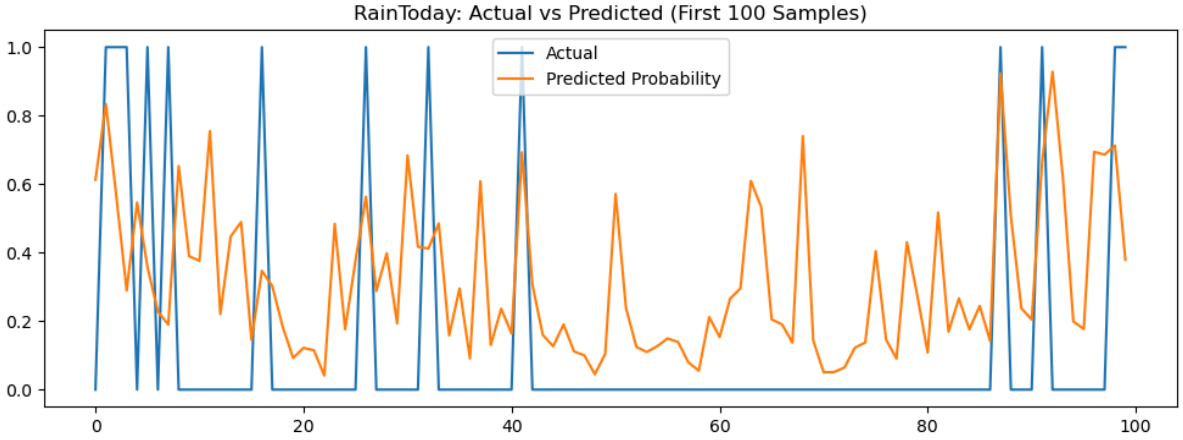

```
--- Validation Set Evaluation ---
```

```
/var/folders/n7/srp3978x3c1f8mjql0jyv_vh0000gn/T/ipykernel_92470/1861124988.py:85: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load(best_model_path))
```

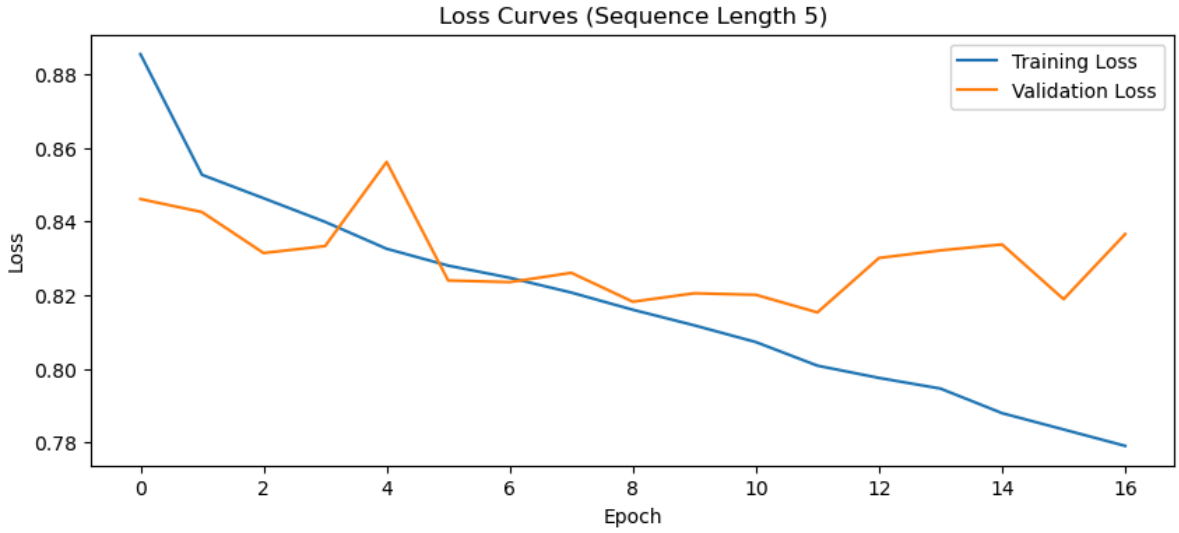
```
                precision    recall  f1-score   support

           0.0       0.91      0.76      0.83     15341
           1.0       0.46      0.73      0.56      4337

    accuracy                           0.75     19678
   macro avg       0.68      0.74      0.70     19678
weighted avg       0.81      0.75      0.77     19678

[[11660  3681]
 [ 1188  3149]]
```




```
--- Test Set Evaluation ---
                precision    recall  f1-score   support

           0.0       0.86      0.72      0.78     32735
           1.0       0.37      0.59      0.46      9368

    accuracy                           0.69     42103
   macro avg       0.62      0.65      0.62     42103
weighted avg       0.75      0.69      0.71     42103

[[23459  9276]
 [ 3848  5520]]
```



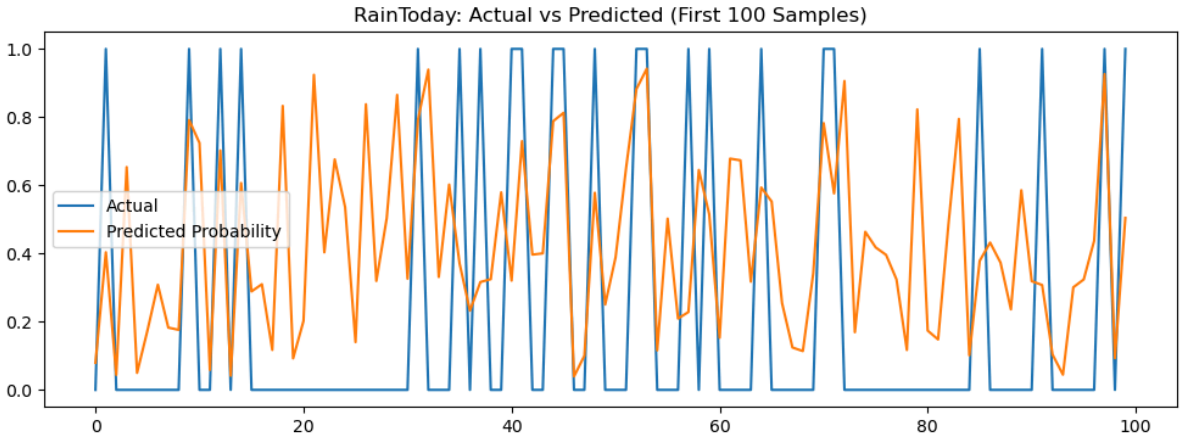
```
=== Training with sequence length 5 ===
Epoch 1/50 - Loss: 0.8855 - Val Loss: 0.8461
Saved new best model.
Epoch 2/50 - Loss: 0.8527 - Val Loss: 0.8426
Saved new best model.
Epoch 3/50 - Loss: 0.8463 - Val Loss: 0.8315
Saved new best model.
Epoch 4/50 - Loss: 0.8399 - Val Loss: 0.8333
Epoch 5/50 - Loss: 0.8326 - Val Loss: 0.8562
Epoch 6/50 - Loss: 0.8280 - Val Loss: 0.8240
Saved new best model.
Epoch 7/50 - Loss: 0.8247 - Val Loss: 0.8236
Saved new best model.
Epoch 8/50 - Loss: 0.8207 - Val Loss: 0.8261
Epoch 9/50 - Loss: 0.8161 - Val Loss: 0.8182
Saved new best model.
Epoch 10/50 - Loss: 0.8118 - Val Loss: 0.8205
Epoch 11/50 - Loss: 0.8073 - Val Loss: 0.8201
Epoch 12/50 - Loss: 0.8009 - Val Loss: 0.8153
Saved new best model.
Epoch 13/50 - Loss: 0.7975 - Val Loss: 0.8301
Epoch 14/50 - Loss: 0.7946 - Val Loss: 0.8322
Epoch 15/50 - Loss: 0.7880 - Val Loss: 0.8338
Epoch 16/50 - Loss: 0.7835 - Val Loss: 0.8189
Epoch 17/50 - Loss: 0.7791 - Val Loss: 0.8366
Early stopping triggered.
```



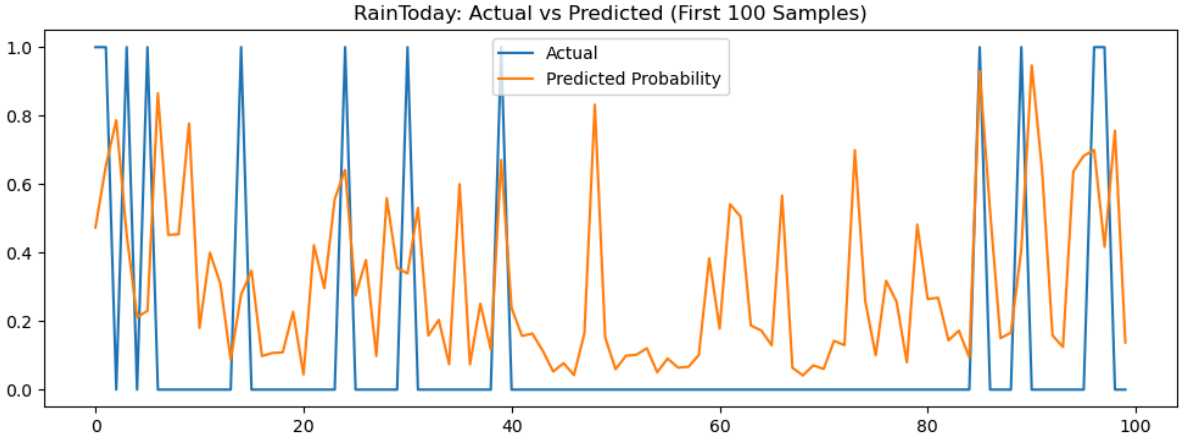

```
--- Validation Set Evaluation ---
```

```
/var/folders/n7/srp3978x3c1f8mjql0jyv_vh0000gn/T/ipykernel_92470/1861124988.py:85: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load(best_model_path))
```

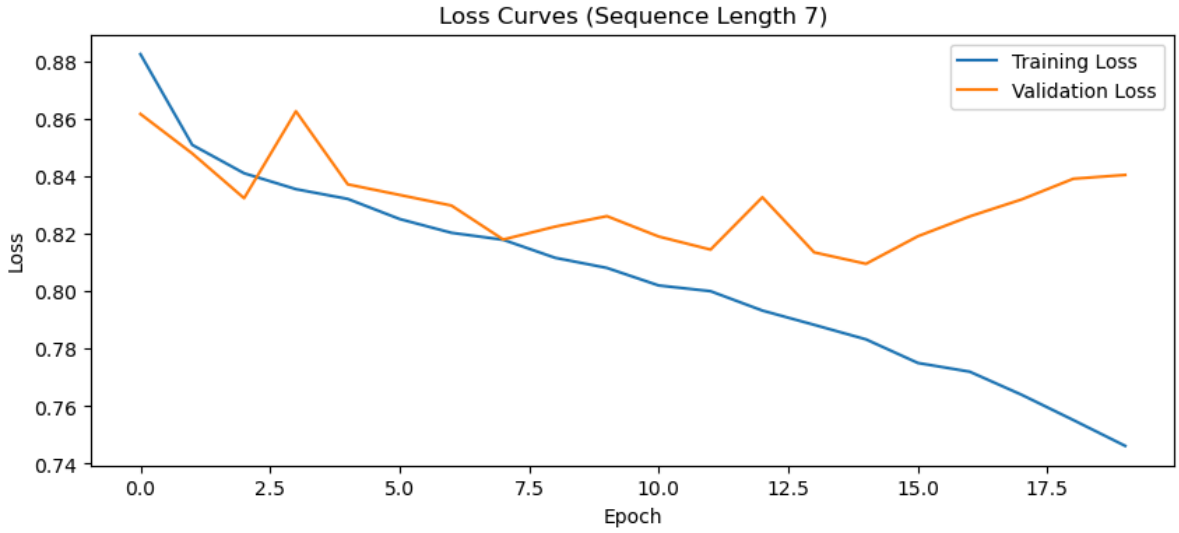
```
                precision    recall  f1-score   support

           0.0       0.90      0.79      0.84     15246
           1.0       0.49      0.70      0.57      4412

    accuracy                           0.77     19658
   macro avg       0.69      0.74      0.71     19658
weighted avg       0.81      0.77      0.78     19658

[[11994  3252]
 [ 1323  3089]]
```




```
--- Test Set Evaluation ---
                precision    recall  f1-score   support

           0.0       0.86      0.75      0.80     32656
           1.0       0.39      0.56      0.46      9349

    accuracy                           0.70     42005
   macro avg       0.62      0.65      0.63     42005
weighted avg       0.75      0.70      0.72     42005

[[24334  8322]
 [ 4123  5226]]
```

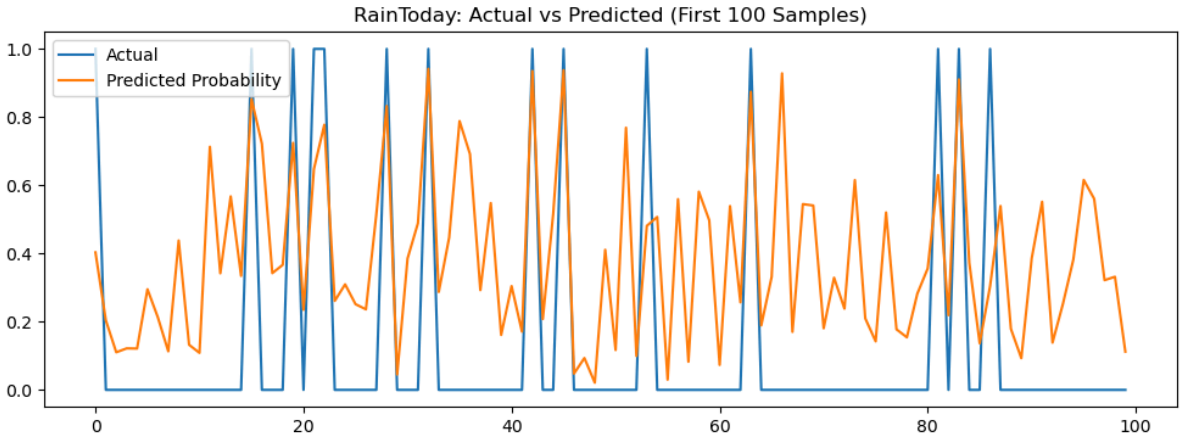


```
=== Training with sequence length 7 ===
Epoch 1/50 - Loss: 0.8827 - Val Loss: 0.8618
Saved new best model.
Epoch 2/50 - Loss: 0.8510 - Val Loss: 0.8481
Saved new best model.
Epoch 3/50 - Loss: 0.8412 - Val Loss: 0.8325
Saved new best model.
Epoch 4/50 - Loss: 0.8356 - Val Loss: 0.8628
Epoch 5/50 - Loss: 0.8322 - Val Loss: 0.8373
Epoch 6/50 - Loss: 0.8252 - Val Loss: 0.8336
Epoch 7/50 - Loss: 0.8204 - Val Loss: 0.8299
Saved new best model.
Epoch 8/50 - Loss: 0.8180 - Val Loss: 0.8181
Saved new best model.
Epoch 9/50 - Loss: 0.8117 - Val Loss: 0.8226
Epoch 10/50 - Loss: 0.8082 - Val Loss: 0.8262
Epoch 11/50 - Loss: 0.8021 - Val Loss: 0.8191
Epoch 12/50 - Loss: 0.8001 - Val Loss: 0.8146
Saved new best model.
Epoch 13/50 - Loss: 0.7933 - Val Loss: 0.8328
Epoch 14/50 - Loss: 0.7884 - Val Loss: 0.8136
Saved new best model.
Epoch 15/50 - Loss: 0.7833 - Val Loss: 0.8096
Saved new best model.
Epoch 16/50 - Loss: 0.7751 - Val Loss: 0.8192
Epoch 17/50 - Loss: 0.7720 - Val Loss: 0.8261
Epoch 18/50 - Loss: 0.7640 - Val Loss: 0.8320
Epoch 19/50 - Loss: 0.7552 - Val Loss: 0.8392
Epoch 20/50 - Loss: 0.7462 - Val Loss: 0.8406
Early stopping triggered.
```




```
--- Validation Set Evaluation ---
/var/folders/n7/srp3978x3c1f8mjql0jyv_vh0000gn/T/ipykernel_92470/1861124988.py:85: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load(best_model_path))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.78      0.84     15260
         1.0       0.48      0.71      0.57      4378

    accuracy                           0.76     19638
   macro avg       0.69      0.74      0.70     19638
weighted avg       0.81      0.76      0.78     19638

[[11891  3369]
 [ 1287  3091]]
```




```
--- Test Set Evaluation ---
              precision    recall  f1-score   support

         0.0       0.86      0.73      0.79     32572
         1.0       0.38      0.58      0.46      9335

    accuracy                           0.70     41907
   macro avg       0.62      0.66      0.63     41907
weighted avg       0.75      0.70      0.72     41907

[[23910  8662]
 [ 3949  5386]]
```