```
!pip install kagglehub
```

```
Requirement already satisfied: kagglehub in /opt/anaconda3/lib/python3.12/site-p
Requirement already satisfied: packaging in /opt/anaconda3/lib/python3.12/site-p
Requirement already satisfied: pyyaml in /opt/anaconda3/lib/python3.12/site-pack
Requirement already satisfied: requests in /opt/anaconda3/lib/python3.12/site-pa
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.12/site-packag
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/lib/py
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.12/sit
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/python3.
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.
```

Project Overview

This project explores how daily stress evolves over time rather than treating each day as an independent snapshot. Because stress is influenced by prior sleep, workload, and lifestyle patterns, we frame the task as a sequence prediction problem that uses recent history to forecast future stress levels. We compare a feed-forward neural network (FFN), which flattens time windows and does not explicitly model temporal order, with an LSTM model that is designed to capture temporal dependencies across days. The goal of this comparison is to evaluate whether sequence-aware models are better able to capture the underlying dynamics of stress over time.

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

import matplotlib.pyplot as plt

SEED = 42
np.random.seed(SEED)
torch.manual_seed(SEED)
```

```
<torch._C.Generator at 0x11ae7b050>
```

Dataset Description

For this project, we used a synthetic daily wellness dataset containing longitudinal records for 1000 users. Each entry represents one user on one day and includes features related to sleep habits, work demands, physical activity, lifestyle behaviors, and a reported stress level. This structure allows us to track how stress changes over time within individuals rather than treating observations as independent samples.

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("wafaaelhusseini/worklife-balance-synthetic-d

print("Path to dataset files:", path)
```

```
Path to dataset files: /Users/makayla/.cache/kagglehub/datasets/wafaaelhusseini/
```

```
import os

print("Dataset folder:", path)
print("Contents:", os.listdir(path))
```

```
Dataset folder: /Users/makayla/.cache/kagglehub/datasets/wafaaelhusseini/worklif
Contents: ['weekly_summaries.csv', 'users.csv', 'daily_all.csv', 'interventions.
```

```
csv_name = "Stress Level Detection Based on Daily Activities.csv"  # <--- chang
csv_path = os.path.join(path, csv_name)
print("CSV path:", csv_path)
```

```
CSV path: /Users/makayla/.cache/kagglehub/datasets/wafaaelhusseini/worklife-bala
```

```
import pandas as pd
import os

for name in ["daily_all.csv", "daily_logs.csv"]:
    print("\n---", name, "---")
    df_temp = pd.read_csv(os.path.join(path, name))
    print(df_temp.head())
    print(df_temp.info())
```

```
--- daily_all.csv ---
   user_id        date  week_start  workday  profession work_mode chronotype
0        1  2024-01-01  2024-01-01     True  operations    onsite    morning
1        1  2024-01-02  2024-01-01     True  operations    onsite    morning
2        1  2024-01-03  2024-01-01     True  operations    onsite    morning
```

```
3        1  2024-01-04  2024-01-01     True  operations    onsite    morning
4        1  2024-01-05  2024-01-01     True  operations    onsite    morning

   age     sex  height_cm  ... workouts_count cheat_meals_count  \
0   27  female        174  ...             10                 1
1   27  female        174  ...             10                 1
2   27  female        174  ...             10                 1
3   27  female        174  ...             10                 1
4   27  female        174  ...             10                 1

  has_intervention  intervention_diet_coaching  intervention_exercise_plan  \
0            False                       False                       False
1            False                       False                       False
2            False                       False                       False
3            False                       False                       False
4            False                       False                       False

   intervention_meditation  intervention_sick_leave  intervention_therapy  \
0                    False                    False                 False
1                    False                    False                 False
2                    False                    False                 False
3                    False                    False                 False
4                    False                    False                 False

   intervention_vacation  intervention_workload_cap
0                  False                      False
1                  False                      False
2                  False                      False
3                  False                      False
4                  False                      False

[5 rows x 53 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731000 entries, 0 to 730999
Data columns (total 53 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   user_id                    731000 non-null  int64
 1   date                       731000 non-null  object
 2   week_start                 731000 non-null  object
 3   workday                    731000 non-null  bool
 4   profession                 731000 non-null  object
 5   work_mode                  731000 non-null  object
 6   chronotype                 731000 non-null  object
 7   age                        731000 non-null  int64
 8   sex                        731000 non-null  object
 9   height_cm                  731000 non-null  int64
 10  mental_health_history      731000 non-null  object
 11  exercise_habit             731000 non-null  object
 12  caffeine_sensitivity       731000 non-null  object
 13  baseline_bmi               731000 non-null  float64
```

Data Cleaning and Temporal Ordering

To ensure that temporal patterns are preserved before modeling, all date fields are converted to datetime format and the data is sorted by user ID and date. This step guarantees that each user's records form a continuous timeline, which is essential when building sequences and training models that rely on time-ordered inputs.

```
df_all = pd.read_csv(os.path.join(path, "daily_all.csv"))
df_all.columns
```

```
Index(['user_id', 'date', 'week_start', 'workday', 'profession', 'work_mode',
       'chronotype', 'age', 'sex', 'height_cm', 'mental_health_history',
       'exercise_habit', 'caffeine_sensitivity', 'baseline_bmi', 'sleep_hours',
       'sleep_quality', 'work_hours', 'meetings_count', 'tasks_completed',
       'emails_received', 'commute_minutes', 'exercise_minutes', 'steps_count',
       'caffeine_mg', 'alcohol_units', 'screen_time_hours',
       'social_interactions', 'outdoor_time_minutes', 'diet_quality',
       'calories_intake', 'stress_level', 'mood_score', 'energy_level',
       'focus_score', 'work_pressure', 'weather_mood_impact', 'weight_kg',
       'job_satisfaction', 'perceived_stress_scale', 'anxiety_score',
       'depression_score', 'sleep_debt_hours', 'avg_weight_kg_week',
       'workouts_count', 'cheat_meals_count', 'has_intervention',
       'intervention_diet_coaching', 'intervention_exercise_plan',
       'intervention_meditation', 'intervention_sick_leave',
       'intervention_therapy', 'intervention_vacation',
       'intervention_workload_cap'],
      dtype='object')
```

```
#load and basic cleaning

import pandas as pd
import numpy as np

df = pd.read_csv(os.path.join(path, "daily_all.csv"))

df['date'] = pd.to_datetime(df['date'])

df = df.sort_values(['user_id', 'date']).reset_index(drop=True)

df.head()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731000 entries, 0 to 730999
Data columns (total 53 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   user_id                    731000 non-null  int64
```

```
 1   date                          731000 non-null  datetime64[ns]
 2   week_start                    731000 non-null  object
 3   workday                       731000 non-null  bool
 4   profession                    731000 non-null  object
 5   work_mode                     731000 non-null  object
 6   chronotype                    731000 non-null  object
 7   age                           731000 non-null  int64
 8   sex                           731000 non-null  object
 9   height_cm                     731000 non-null  int64
 10  mental_health_history         731000 non-null  object
 11  exercise_habit                731000 non-null  object
 12  caffeine_sensitivity          731000 non-null  object
 13  baseline_bmi                  731000 non-null  float64
 14  sleep_hours                   731000 non-null  float64
 15  sleep_quality                 731000 non-null  int64
 16  work_hours                    731000 non-null  float64
 17  meetings_count                731000 non-null  int64
 18  tasks_completed               731000 non-null  int64
 19  emails_received               731000 non-null  int64
 20  commute_minutes               731000 non-null  int64
 21  exercise_minutes              731000 non-null  int64
 22  steps_count                   731000 non-null  int64
 23  caffeine_mg                   731000 non-null  int64
 24  alcohol_units                 731000 non-null  float64
 25  screen_time_hours             731000 non-null  float64
 26  social_interactions           731000 non-null  int64
 27  outdoor_time_minutes          731000 non-null  int64
 28  diet_quality                  731000 non-null  int64
 29  calories_intake               731000 non-null  int64
 30  stress_level                  731000 non-null  int64
 31  mood_score                    731000 non-null  int64
 32  energy_level                  731000 non-null  int64
 33  focus_score                   731000 non-null  int64
 34  work_pressure                 731000 non-null  object
 35  weather_mood_impact           731000 non-null  float64
 36  weight_kg                     731000 non-null  float64
 37  job_satisfaction              731000 non-null  int64
 38  perceived_stress_scale        731000 non-null  int64
 39  anxiety_score                 731000 non-null  int64
 40  depression_score              731000 non-null  int64
 41  sleep_debt_hours              731000 non-null  float64
 42  avg_weight_kg_week            731000 non-null  float64
 43  workouts_count                731000 non-null  int64
 44  cheat_meals_count             731000 non-null  int64
 45  has_intervention              731000 non-null  bool
 46  intervention_diet_coaching    731000 non-null  bool
 47  intervention_exercise_plan    731000 non-null  bool
 48  intervention_meditation       731000 non-null  bool
 49  intervention_sick_leave       731000 non-null  bool
 50  intervention_therapy          731000 non-null  bool
 51  intervention_vacation         731000 non-null  bool
```

```
target_col = "stress_level"
id_col = "user_id"
time_col = "date"
```

```
drop_cols = [target_col, id_col, time_col]

feature_cols = [c for c in df.columns if c not in drop_cols]
feature_cols[:10], len(feature_cols)
```

```
(['week_start',
  'workday',
  'profession',
  'work_mode',
  'chronotype',
  'age',
  'sex',
  'height_cm',
  'mental_health_history',
  'exercise_habit'],
 50)
```

```
cat_cols = df[feature_cols].select_dtypes(include=['object', 'bool']).columns.t
num_cols = [c for c in feature_cols if c not in cat_cols]

cat_cols, len(cat_cols)
num_cols[:10]
```

```
['age',
 'height_cm',
 'baseline_bmi',
 'sleep_hours',
 'sleep_quality',
 'work_hours',
 'meetings_count',
 'tasks_completed',
 'emails_received',
 'commute_minutes']
```

Target Engineering: Stress Classification

The original stress values spanned 9 numeric levels, so they are grouped into three categories: low, medium, and high stress. This simplifies the prediction task into a multiclass classification problem while still capturing meaningful differences in stress intensity and making the results easier to interpret.

Feature Processing

The dataset contains a mix of numerical, categorical, and boolean features, each requiring different preprocessing steps. Numerical features are standardized to promote stable training, while categorical and boolean features are one-hot encoded. After

preprocessing, all features are converted to numeric values suitable for neural network inputs.

```
from sklearn.preprocessing import StandardScaler

# One-hot categoricals
df_cat = pd.get_dummies(df[cat_cols], drop_first=True)

# Scale numerical columns
scaler = StandardScaler()
df_num = pd.DataFrame(
    scaler.fit_transform(df[num_cols]),
    columns=num_cols,
    index=df.index
)

# Combine processed features
X_df = pd.concat([df_num, df_cat], axis=1)


y = df[target_col].values
```

```
# Ensure no object dtypes remain
print("Object columns:", X_df.select_dtypes(include='object').columns.tolist())

# Convert booleans to integers
bool_cols = X_df.select_dtypes(include='bool').columns
X_df[bool_cols] = X_df[bool_cols].astype(int)

# Convert everything to float32
X_df = X_df.astype('float32')

print("Final dtypes:", X_df.dtypes.unique())
```

```
Object columns: []
Final dtypes: [dtype('float32')]
```

```
df_all['stress_level'].describe()
df_all['stress_level'].unique()
```

```
array([4, 6, 3, 5, 2, 7, 1, 8, 9])
```

```
def bin_stress(x):
    if x <= 3:
        return 0   # low
    elif x <= 6:
```

```
            return 1   # medium
        else:
            return 2   # high

    df['stress_class'] = df['stress_level'].apply(bin_stress)
    df['stress_class'].unique()
```

```
    array([1, 0, 2])
```

Sequence Construction

To model the temporal behavior, the data is transformed into sliding windows of consecutive days for each user. Each input sequence contains the previous N days of features, and the target corresponds to the stress level on the following day. This setup allows the models to learn how recent life events and repeated behaviors predict future stress.

```
    def build_sequences(df, X_df, seq_len, id_col='user_id', target_col='stress_cla

        X_list = []
        y_list = []

        # Group by user
        for user_id, group in df.groupby(id_col):
            group = group.sort_values('date')

            X_user = X_df.loc[group.index].values     # features
            y_user = group[target_col].values         # stress_class target

            T = len(group)

            # Create sliding windows
            for i in range(T - seq_len):
                X_list.append(X_user[i : i + seq_len])
                y_list.append(y_user[i + seq_len])    # next day's stress class

        X_seq = np.array(X_list)
        y_seq = np.array(y_list)

        return X_seq, y_seq
```

```
    X5, y5 = build_sequences(df, X_df, seq_len=5)

    X5.shape, y5.shape
```

```
((726000, 5, 166), (726000,))
```

Train, Validation, and Test Split

The sequence data is divided into training (70%), validation(15%), and test(15%) sets using stratified sampling to preserve the distribution of stress categories. This ensures fair evaluation and prevents the models from becoming biased toward any single class.

```
from sklearn.model_selection import train_test_split

seq_len = 5
X_seq, y_seq = build_sequences(df, X_df, seq_len)

X_train, X_temp, y_train, y_temp = train_test_split(
    X_seq, y_seq, test_size=0.3, random_state=42, stratify=y_seq
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp
)
```

```
import torch
from torch.utils.data import Dataset, DataLoader

class StressSequenceDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.long)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

Baseline Model: Feed-Forward Network (FFN)

The feed-forward neural network serves as a baseline model by flattening each sequence into a single feature vector. While this approach can capture correlations across multiple days, it does not explicitly model the order or timing of events, limiting its ability to learn true temporal dependencies.

```
#Feed-Forward Network BASELINE
import torch.nn as nn
```

```
class FFN(nn.Module):
    def __init__(self, seq_len, num_features, num_classes=3):
        super().__init__()

        input_dim = seq_len * num_features

        self.net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, num_classes)
        )

    def forward(self, x):
        # x shape: (batch, seq_len, num_features)
        x = x.reshape(x.size(0), -1)  # flatten
        return self.net(x)
```

LSTM Sequence Model

The LSTM processes daily data step by step while maintaining a hidden state that carries information forward in time. This allows the model to remember patterns from prior days and learn delayed effects, such as how accumulated sleep debt or sustained workload pressure can influence stress. Because it is specifically designed for sequential data, the LSTM is well suited for modeling temporal dependencies, and in this setup only the final hidden state is used to predict the stress level of the following day.

```
#Build the LSTM model
class LSTMModel(nn.Module):
    def __init__(self, num_features, hidden_size=64, num_layers=1, num_classes=
        super().__init__()

        self.lstm = nn.LSTM(
            input_size=num_features,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True
        )

        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out, (h_n, c_n) = self.lstm(x)

        last_hidden = h_n[-1]
```

```
        return self.fc(last_hidden)
```

```
def accuracy(preds, labels):
    return (preds.argmax(dim=1) == labels).float().mean().item()
```

Training Setup

After predicting accuracy, both models are trained using cross-entropy loss and the Adam optimizer, with accuracy used as the primary evaluation metric. Validation performance is monitored during training to compare how each model learns and generalizes to unseen data.

```
def train_one_epoch(model, loader, optimizer, criterion, device):
    model.train()
    total_loss, total_acc = 0, 0

    for X, y in loader:
        X, y = X.to(device), y.to(device)

        optimizer.zero_grad()
        out = model(X)
        loss = criterion(out, y)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        total_acc += accuracy(out, y)

    return total_loss / len(loader), total_acc / len(loader)
```

```
def eval_model(model, loader, criterion, device):
    model.eval()
    total_loss, total_acc = 0, 0

    with torch.no_grad():
        for X, y in loader:
            X, y = X.to(device), y.to(device)

            out = model(X)
            loss = criterion(out, y)

            total_loss += loss.item()
            total_acc += accuracy(out, y)
```

```
    return total_loss / len(loader), total_acc / len(loader)
```

```
seq_len = 5
X_seq, y_seq = build_sequences(df, X_df, seq_len)

# Split
X_train, X_temp, y_train, y_temp = train_test_split(
    X_seq, y_seq, test_size=0.3, random_state=42, stratify=y_seq
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp
)

# Dataloaders
train_ds = StressSequenceDataset(X_train, y_train)
val_ds = StressSequenceDataset(X_val, y_val)
test_ds = StressSequenceDataset(X_test, y_test)

train_loader = DataLoader(train_ds, batch_size=64, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=64)
test_loader = DataLoader(test_ds, batch_size=64)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_features = X_seq.shape[2]
```

```
ffn = FFN(seq_len, num_features).to(device)
optimizer = torch.optim.Adam(ffn.parameters(), lr=1e-3)
criterion = nn.CrossEntropyLoss()

for epoch in range(10):
    train_loss, train_acc = train_one_epoch(ffn, train_loader, optimizer, crite
    val_loss, val_acc = eval_model(ffn, val_loader, criterion, device)

    print(f"Epoch {epoch+1}: train_acc={train_acc:.3f}, val_acc={val_acc:.3f}")
```

```
Epoch 1: train_acc=0.692, val_acc=0.693
Epoch 2: train_acc=0.697, val_acc=0.696
Epoch 3: train_acc=0.698, val_acc=0.698
Epoch 4: train_acc=0.700, val_acc=0.698
Epoch 5: train_acc=0.701, val_acc=0.698
Epoch 6: train_acc=0.703, val_acc=0.693
Epoch 7: train_acc=0.704, val_acc=0.696
Epoch 8: train_acc=0.705, val_acc=0.696
Epoch 9: train_acc=0.707, val_acc=0.694
Epoch 10: train_acc=0.708, val_acc=0.694
```

```
lstm = LSTMModel(num_features).to(device)
optimizer = torch.optim.Adam(lstm.parameters(), lr=1e-3)
criterion = nn.CrossEntropyLoss()

for epoch in range(10):
    train_loss, train_acc = train_one_epoch(lstm, train_loader, optimizer, crit
    val_loss, val_acc = eval_model(lstm, val_loader, criterion, device)

    print(f"Epoch {epoch+1}: train_acc={train_acc:.3f}, val_acc={val_acc:.3f}")
```

```
Epoch 1: train_acc=0.695, val_acc=0.696
Epoch 2: train_acc=0.699, val_acc=0.699
Epoch 3: train_acc=0.701, val_acc=0.696
Epoch 4: train_acc=0.702, val_acc=0.698
Epoch 5: train_acc=0.703, val_acc=0.697
Epoch 6: train_acc=0.705, val_acc=0.697
Epoch 7: train_acc=0.707, val_acc=0.693
Epoch 8: train_acc=0.708, val_acc=0.694
Epoch 9: train_acc=0.710, val_acc=0.693
Epoch 10: train_acc=0.712, val_acc=0.689
```

Sequence Length Analysis

5 different sequence lengths are tested to examine how much of the dataset's prior records improves prediction performance. The feed-forward model shows relatively stable accuracy across sequence lengths, while the LSTM benefits from longer input windows, showcasing its ability to effectively leverage additional temporal information.

```
sequence_lengths = [3, 5, 7, 10, 14]
```

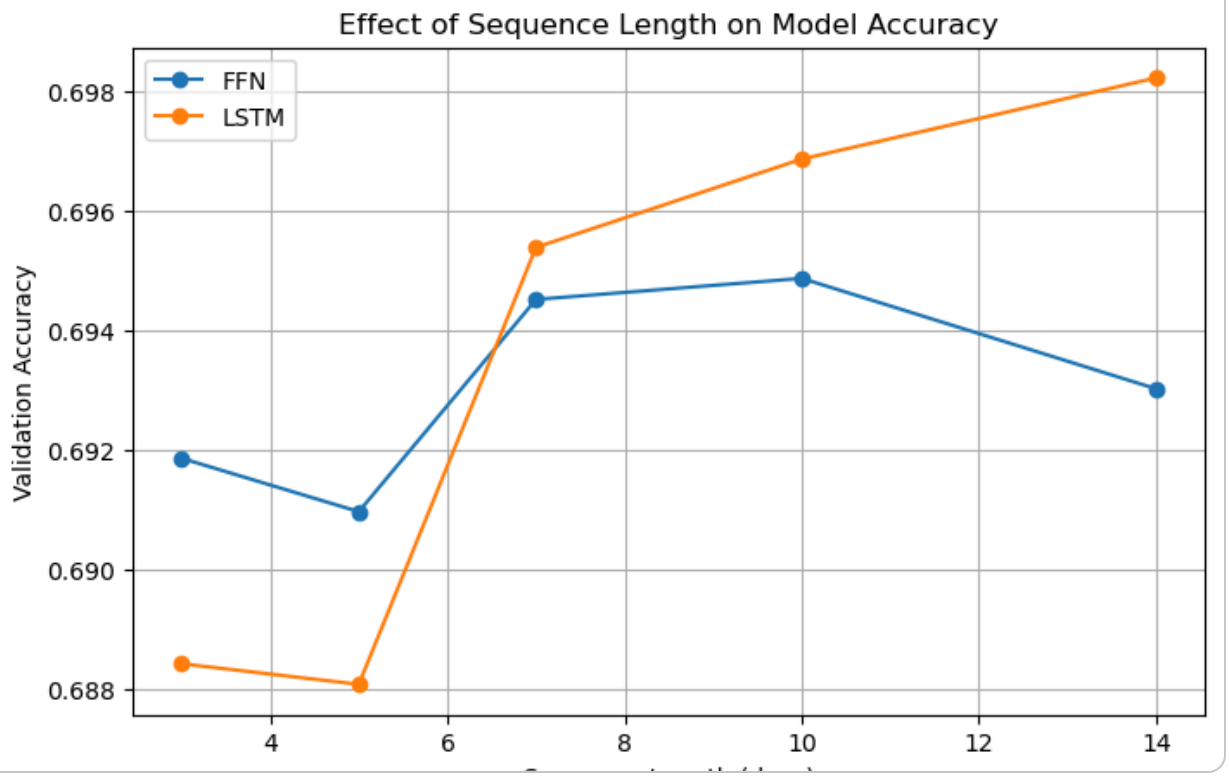
```
results_ffn = {}
results_lstm = {}

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
criterion = nn.CrossEntropyLoss()

for seq_len in sequence_lengths:
    print(f"\n===== Testing sequence length {seq_len} =====")

    # Build sequences
    X_seq, y_seq = build_sequences(df, X_df, seq_len)
    print("Sequence data shape:", X_seq.shape)

    # Train/val/test split
    X_train, X_temp, y_train, y_temp = train_test_split(
        X_seq, y_seq, test_size=0.3, random_state=42, stratify=y_seq
    )
```

```
    X_val, X_test, y_val, y_test = train_test_split(
        X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp
    )

    # Build Datasets
    train_ds = StressSequenceDataset(X_train, y_train)
    val_ds   = StressSequenceDataset(X_val, y_val)
    test_ds  = StressSequenceDataset(X_test, y_test)

    train_loader = DataLoader(train_ds, batch_size=64, shuffle=True)
    val_loader   = DataLoader(val_ds, batch_size=64)
    test_loader  = DataLoader(test_ds, batch_size=64)

    num_features = X_seq.shape[2]


    # Train FFN

    ffn = FFN(seq_len, num_features).to(device)
    optimizer = torch.optim.Adam(ffn.parameters(), lr=1e-3)

    for epoch in range(8):  # shorter training since repeated
        train_loss, train_acc = train_one_epoch(ffn, train_loader, optimizer, c
        val_loss, val_acc = eval_model(ffn, val_loader, criterion, device)

    results_ffn[seq_len] = val_acc
    print(f"FFN val accuracy @ len={seq_len}: {val_acc:.4f}")

    # Train LSTM

    lstm = LSTMModel(num_features).to(device)
    optimizer = torch.optim.Adam(lstm.parameters(), lr=1e-3)

    for epoch in range(8):
        train_loss, train_acc = train_one_epoch(lstm, train_loader, optimizer,
        val_loss, val_acc = eval_model(lstm, val_loader, criterion, device)

    results_lstm[seq_len] = val_acc
    print(f"LSTM val accuracy @ len={seq_len}: {val_acc:.4f}")
```

```
===== Testing sequence length 3 =====
Sequence data shape: (728000, 3, 166)
FFN val accuracy @ len=3: 0.6919
LSTM val accuracy @ len=3: 0.6884

===== Testing sequence length 5 =====
Sequence data shape: (726000, 5, 166)
FFN val accuracy @ len=5: 0.6910
LSTM val accuracy @ len=5: 0.6881
```

```
===== Testing sequence length 7 =====
Sequence data shape: (724000, 7, 166)
FFN val accuracy @ len=7: 0.6945
LSTM val accuracy @ len=7: 0.6954

===== Testing sequence length 10 =====
Sequence data shape: (721000, 10, 166)
FFN val accuracy @ len=10: 0.6949
LSTM val accuracy @ len=10: 0.6969

===== Testing sequence length 14 =====
Sequence data shape: (717000, 14, 166)
FFN val accuracy @ len=14: 0.6930
LSTM val accuracy @ len=14: 0.6982
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
plt.plot(sequence_lengths, [results_ffn[L] for L in sequence_lengths], marker='
plt.plot(sequence_lengths, [results_lstm[L] for L in sequence_lengths], marker=

plt.xlabel("Sequence Length (days)")
plt.ylabel("Validation Accuracy")
plt.title("Effect of Sequence Length on Model Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```



```
#1. LSTM accuracy increases steadily as sequence length grows
#2. FFN accuracy is flatter and does NOT improve with longer sequences
#3. LSTM surpasses FFN once sequence length ≥ 7
```

Conclusion

Overall, the results suggest that daily stress is better modeled as a time-dependent process rather than a series of independent observations. Sequence-aware models such as LSTMs are more effective at capturing stress dynamics, particularly when given access to longer histories of daily data.