In [1]:
```
import matplotlib.pyplot as plt
import csv
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
```

Load and preprocess the data

We will use data from past 7 days to predict next day temperature. Since every feature has values with varying ranges, we do normalization to confine feature values to a range of [0, 1] before training a neural network. We do this by subtracting the mean and dividing by the standard deviation of each feature.

80 % of the data will be used to train the model, i.e. 3650 * 0.8 rows. split_fraction can be changed to alter this percentage.

In [11]:
```
import pandas as pd
df = pd.read_csv('daily-min-temperatures.csv')
df.head()
```

Out[11]:

	Date	Temp
0	1981-01-01	20.7
1	1981-01-02	17.9
2	1981-01-03	18.8
3	1981-01-04	14.6
4	1981-01-05	15.8

In [12]:
```
df["Date"]
```

Out[12]:
```
0       1981-01-01
1       1981-01-02
2       1981-01-03
3       1981-01-04
4       1981-01-05
           ...
3645    1990-12-27
3646    1990-12-28
3647    1990-12-29
3648    1990-12-30
3649    1990-12-31
Name: Date, Length: 3650, dtype: object
```

B

```
In [16]: from sklearn.model_selection import train_test_split

         # Normalize the temperatures
         df['Temp'] = (df['Temp'] - df['Temp'].mean()) / df['Temp'].std()

         X_train, X_test, y_train, y_test = train_test_split(df['Date'], df['Temp'],

         print(y_train.std())
         print(y_train.mean())
```

```
0.9958783604892041
-0.0026863443783818773
```

C

```
In [27]: import torch
         from torch.utils.data import Dataset

         class TempSequenceDataset(Dataset):
             def __init__(self, temps, sequence_length=7):
                 self.sequence_length = sequence_length
                 self.temps = torch.tensor(temps.values, dtype=torch.float32)
                 self.samples = []
                 self.targets = []

                 for i in range(len(self.temps) - sequence_length):
                     self.samples.append(self.temps[i:i+sequence_length])    # sequer
                     self.targets.append(self.temps[i+sequence_length])      # target

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

             def __getitem__(self, idx):
                 return self.samples[idx], self.targets[idx]


         # Only pass in the normalized temperature column
         train_dataset = TempSequenceDataset(df['Temp'][:int(len(df)*0.8)])
         test_dataset = TempSequenceDataset(df['Temp'][int(len(df)*0.8):])
         train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True)
         test_loader = DataLoader(test_dataset, batch_size=256)

         train_dataset[0]
```

```
Out[27]: (tensor([2.3386, 1.6509, 1.8719, 0.8405, 1.1352, 1.1352, 1.1352]),
          tensor(1.5281))
```

D

```
In [35]: import torch.optim as optim
```

```
# Define the LSTM model
class LSTMModel(nn.Module):
    def __init__(self, input_size=1, hidden_dim=128, num_layers=1):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_dim, num_layers=num_layers, b
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        last_out = lstm_out[:, -1, :]  # use last time step output
        out = self.fc(last_out)
        return out.squeeze()

# Initialize model, loss, optimizer
model = LSTMModel().to(device)
```

E

In [33]:
```
# Training loop
epochs = 100
learning_rate = 0.001

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
for epoch in range(epochs):
    model.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        output = model(X_batch)
        loss = criterion(output, y_batch)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch {epoch+1}/{epochs} - Loss: {total_loss / len(train_loader)

# Optional: Evaluate on test set
model.eval()
with torch.no_grad():
    test_losses = []
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        output = model(X_batch)
        loss = criterion(output, y_batch)
        test_losses.append(loss.item())

    print(f"Test Loss: {sum(test_losses) / len(test_losses):.4f}")
```

```
Epoch 1/100 - Loss: 0.8821
Epoch 2/100 - Loss: 0.5781
Epoch 3/100 - Loss: 0.5046
Epoch 4/100 - Loss: 0.4843
Epoch 5/100 - Loss: 0.4700
Epoch 6/100 - Loss: 0.4754
Epoch 7/100 - Loss: 0.4579
Epoch 8/100 - Loss: 0.4497
Epoch 9/100 - Loss: 0.4466
Epoch 10/100 - Loss: 0.4422
Epoch 11/100 - Loss: 0.4303
Epoch 12/100 - Loss: 0.4202
Epoch 13/100 - Loss: 0.4037
Epoch 14/100 - Loss: 0.3884
Epoch 15/100 - Loss: 0.3925
Epoch 16/100 - Loss: 0.3868
Epoch 17/100 - Loss: 0.3801
Epoch 18/100 - Loss: 0.3781
Epoch 19/100 - Loss: 0.3758
Epoch 20/100 - Loss: 0.3726
Epoch 21/100 - Loss: 0.3744
Epoch 22/100 - Loss: 0.3685
Epoch 23/100 - Loss: 0.3708
Epoch 24/100 - Loss: 0.3742
Epoch 25/100 - Loss: 0.3717
Epoch 26/100 - Loss: 0.3714
Epoch 27/100 - Loss: 0.3724
Epoch 28/100 - Loss: 0.3726
Epoch 29/100 - Loss: 0.3685
Epoch 30/100 - Loss: 0.3748
Epoch 31/100 - Loss: 0.3661
Epoch 32/100 - Loss: 0.3691
Epoch 33/100 - Loss: 0.3702
Epoch 34/100 - Loss: 0.3743
Epoch 35/100 - Loss: 0.3764
Epoch 36/100 - Loss: 0.3670
Epoch 37/100 - Loss: 0.3720
Epoch 38/100 - Loss: 0.3702
Epoch 39/100 - Loss: 0.3764
Epoch 40/100 - Loss: 0.3682
Epoch 41/100 - Loss: 0.3673
Epoch 42/100 - Loss: 0.3699
Epoch 43/100 - Loss: 0.3635
Epoch 44/100 - Loss: 0.3695
Epoch 45/100 - Loss: 0.3684
Epoch 46/100 - Loss: 0.3662
Epoch 47/100 - Loss: 0.3684
Epoch 48/100 - Loss: 0.3741
Epoch 49/100 - Loss: 0.3711
Epoch 50/100 - Loss: 0.3658
Epoch 51/100 - Loss: 0.3730
Epoch 52/100 - Loss: 0.3666
Epoch 53/100 - Loss: 0.3731
Epoch 54/100 - Loss: 0.3697
Epoch 55/100 - Loss: 0.3734
Epoch 56/100 - Loss: 0.3634
Epoch 57/100 - Loss: 0.3627
Epoch 58/100 - Loss: 0.3669
Epoch 59/100 - Loss: 0.3663
Epoch 60/100 - Loss: 0.3611
Epoch 61/100 - Loss: 0.3627
Epoch 62/100 - Loss: 0.3649
Epoch 63/100 - Loss: 0.3645
Epoch 64/100 - Loss: 0.3628
Epoch 65/100 - Loss: 0.3677
Epoch 66/100 - Loss: 0.3650
Epoch 67/100 - Loss: 0.3646
Epoch 68/100 - Loss: 0.3639
Epoch 69/100 - Loss: 0.3620
Epoch 70/100 - Loss: 0.3649
Epoch 71/100 - Loss: 0.3642
Epoch 72/100 - Loss: 0.3671
Epoch 73/100 - Loss: 0.3693
Epoch 74/100 - Loss: 0.3662
Epoch 75/100 - Loss: 0.3618
Epoch 76/100 - Loss: 0.3687
Epoch 77/100 - Loss: 0.3707
Epoch 78/100 - Loss: 0.3618
Epoch 79/100 - Loss: 0.3653
Epoch 80/100 - Loss: 0.3677
Epoch 81/100 - Loss: 0.3658
Epoch 82/100 - Loss: 0.3670
Epoch 83/100 - Loss: 0.3636
Epoch 84/100 - Loss: 0.3643
Epoch 85/100 - Loss: 0.3635
Epoch 86/100 - Loss: 0.3705
Epoch 87/100 - Loss: 0.3620
Epoch 88/100 - Loss: 0.3678
Epoch 89/100 - Loss: 0.3618
Epoch 90/100 - Loss: 0.3675
Epoch 91/100 - Loss: 0.3649
Epoch 92/100 - Loss: 0.3709
Epoch 93/100 - Loss: 0.3621
Epoch 94/100 - Loss: 0.3671
Epoch 95/100 - Loss: 0.3587
Epoch 96/100 - Loss: 0.3633
Epoch 97/100 - Loss: 0.3679
Epoch 98/100 - Loss: 0.3674
Epoch 99/100 - Loss: 0.3597
Epoch 100/100 - Loss: 0.3617
Test Loss: 0.2971
```

F

In [36]:

```
model = LSTMModel(num_layers=2).to(device)
# Training loop
epochs = 100
```

```python
learning_rate = 0.001

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
for epoch in range(epochs):
    model.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        output = model(X_batch)
        loss = criterion(output, y_batch)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch {epoch+1}/{epochs} - Loss: {total_loss / len(train_loader)

# Optional: Evaluate on test set
model.eval()
with torch.no_grad():
    test_losses = []
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        output = model(X_batch)
        loss = criterion(output, y_batch)
        test_losses.append(loss.item())

    print(f"Test Loss: {sum(test_losses) / len(test_losses):.4f}")
```

```
Epoch 1/100 - Loss: 0.8684
Epoch 2/100 - Loss: 0.5510
Epoch 3/100 - Loss: 0.5105
Epoch 4/100 - Loss: 0.4992
Epoch 5/100 - Loss: 0.4896
Epoch 6/100 - Loss: 0.4798
Epoch 7/100 - Loss: 0.4745
Epoch 8/100 - Loss: 0.4567
Epoch 9/100 - Loss: 0.4399
Epoch 10/100 - Loss: 0.4162
Epoch 11/100 - Loss: 0.3856
Epoch 12/100 - Loss: 0.3728
Epoch 13/100 - Loss: 0.3725
Epoch 14/100 - Loss: 0.3664
Epoch 15/100 - Loss: 0.3759
Epoch 16/100 - Loss: 0.3862
Epoch 17/100 - Loss: 0.3744
Epoch 18/100 - Loss: 0.3689
Epoch 19/100 - Loss: 0.3703
Epoch 20/100 - Loss: 0.3755
Epoch 21/100 - Loss: 0.3719
Epoch 22/100 - Loss: 0.3734
Epoch 23/100 - Loss: 0.3724
Epoch 24/100 - Loss: 0.3703
Epoch 25/100 - Loss: 0.3706
Epoch 26/100 - Loss: 0.3711
Epoch 27/100 - Loss: 0.3680
Epoch 28/100 - Loss: 0.3770
Epoch 29/100 - Loss: 0.3662
Epoch 30/100 - Loss: 0.3719
Epoch 31/100 - Loss: 0.3712
Epoch 32/100 - Loss: 0.3721
Epoch 33/100 - Loss: 0.3715
Epoch 34/100 - Loss: 0.3623
Epoch 35/100 - Loss: 0.3710
Epoch 36/100 - Loss: 0.3697
Epoch 37/100 - Loss: 0.3720
Epoch 38/100 - Loss: 0.3695
Epoch 39/100 - Loss: 0.3656
Epoch 40/100 - Loss: 0.3644
Epoch 41/100 - Loss: 0.3644
Epoch 42/100 - Loss: 0.3698
Epoch 43/100 - Loss: 0.3772
Epoch 44/100 - Loss: 0.3649
Epoch 45/100 - Loss: 0.3617
Epoch 46/100 - Loss: 0.3633
Epoch 47/100 - Loss: 0.3676
Epoch 48/100 - Loss: 0.3640
Epoch 49/100 - Loss: 0.3592
Epoch 50/100 - Loss: 0.3646
Epoch 51/100 - Loss: 0.3692
Epoch 52/100 - Loss: 0.3661
Epoch 53/100 - Loss: 0.3601
Epoch 54/100 - Loss: 0.3647
Epoch 55/100 - Loss: 0.3589
Epoch 56/100 - Loss: 0.3644
Epoch 57/100 - Loss: 0.3630
Epoch 58/100 - Loss: 0.3627
Epoch 59/100 - Loss: 0.3649
Epoch 60/100 - Loss: 0.3606
Epoch 61/100 - Loss: 0.3596
Epoch 62/100 - Loss: 0.3611
Epoch 63/100 - Loss: 0.3604
Epoch 64/100 - Loss: 0.3581
Epoch 65/100 - Loss: 0.3581
Epoch 66/100 - Loss: 0.3586
Epoch 67/100 - Loss: 0.3675
Epoch 68/100 - Loss: 0.3601
Epoch 69/100 - Loss: 0.3566
Epoch 70/100 - Loss: 0.3569
Epoch 71/100 - Loss: 0.3575
Epoch 72/100 - Loss: 0.3610
Epoch 73/100 - Loss: 0.3569
Epoch 74/100 - Loss: 0.3561
Epoch 75/100 - Loss: 0.3576
Epoch 76/100 - Loss: 0.3559
Epoch 77/100 - Loss: 0.3686
Epoch 78/100 - Loss: 0.3569
Epoch 79/100 - Loss: 0.3520
Epoch 80/100 - Loss: 0.3560
Epoch 81/100 - Loss: 0.3586
Epoch 82/100 - Loss: 0.3609
Epoch 83/100 - Loss: 0.3570
Epoch 84/100 - Loss: 0.3541
Epoch 85/100 - Loss: 0.3572
Epoch 86/100 - Loss: 0.3533
Epoch 87/100 - Loss: 0.3568
Epoch 88/100 - Loss: 0.3554
Epoch 89/100 - Loss: 0.3555
Epoch 90/100 - Loss: 0.3514
Epoch 91/100 - Loss: 0.3569
Epoch 92/100 - Loss: 0.3553
Epoch 93/100 - Loss: 0.3532
Epoch 94/100 - Loss: 0.3506
Epoch 95/100 - Loss: 0.3490
Epoch 96/100 - Loss: 0.3508
Epoch 97/100 - Loss: 0.3465
Epoch 98/100 - Loss: 0.3499
Epoch 99/100 - Loss: 0.3472
Epoch 100/100 - Loss: 0.3462
Test Loss: 0.2996
```

In [37]:

```
model = LSTMModel(num_layers=3).to(device)
# Training loop
epochs = 100
learning_rate = 0.001

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
for epoch in range(epochs):
    model.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        output = model(X_batch)
        loss = criterion(output, y_batch)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch {epoch+1}/{epochs} - Loss: {total_loss / len(train_loader)

# Optional: Evaluate on test set
model.eval()
with torch.no_grad():
    test_losses = []
```

```python
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        output = model(X_batch)
        loss = criterion(output, y_batch)
        test_losses.append(loss.item())

    print(f"Test Loss: {sum(test_losses) / len(test_losses):.4f}")
```

```
Epoch 1/100 - Loss: 0.8964
Epoch 2/100 - Loss: 0.5672
Epoch 3/100 - Loss: 0.5283
Epoch 4/100 - Loss: 0.5213
Epoch 5/100 - Loss: 0.5166
Epoch 6/100 - Loss: 0.5071
Epoch 7/100 - Loss: 0.4969
Epoch 8/100 - Loss: 0.4846
Epoch 9/100 - Loss: 0.4754
Epoch 10/100 - Loss: 0.4346
Epoch 11/100 - Loss: 0.4036
Epoch 12/100 - Loss: 0.3841
Epoch 13/100 - Loss: 0.3764
Epoch 14/100 - Loss: 0.3753
Epoch 15/100 - Loss: 0.3732
Epoch 16/100 - Loss: 0.3741
Epoch 17/100 - Loss: 0.3767
Epoch 18/100 - Loss: 0.3734
Epoch 19/100 - Loss: 0.3748
Epoch 20/100 - Loss: 0.3835
Epoch 21/100 - Loss: 0.3728
Epoch 22/100 - Loss: 0.3669
Epoch 23/100 - Loss: 0.3679
Epoch 24/100 - Loss: 0.3697
Epoch 25/100 - Loss: 0.3674
Epoch 26/100 - Loss: 0.3665
Epoch 27/100 - Loss: 0.3677
Epoch 28/100 - Loss: 0.3621
Epoch 29/100 - Loss: 0.3668
Epoch 30/100 - Loss: 0.3627
Epoch 31/100 - Loss: 0.3636
Epoch 32/100 - Loss: 0.3653
Epoch 33/100 - Loss: 0.3681
Epoch 34/100 - Loss: 0.3672
Epoch 35/100 - Loss: 0.3643
Epoch 36/100 - Loss: 0.3670
Epoch 37/100 - Loss: 0.3685
Epoch 38/100 - Loss: 0.3631
Epoch 39/100 - Loss: 0.3689
Epoch 40/100 - Loss: 0.3648
Epoch 41/100 - Loss: 0.3694
Epoch 42/100 - Loss: 0.3682
Epoch 43/100 - Loss: 0.3633
Epoch 44/100 - Loss: 0.3641
Epoch 45/100 - Loss: 0.3669
Epoch 46/100 - Loss: 0.3662
Epoch 47/100 - Loss: 0.3680
Epoch 48/100 - Loss: 0.3675
Epoch 49/100 - Loss: 0.3596
Epoch 50/100 - Loss: 0.3671
Epoch 51/100 - Loss: 0.3603
Epoch 52/100 - Loss: 0.3659
Epoch 53/100 - Loss: 0.3587
Epoch 54/100 - Loss: 0.3643
Epoch 55/100 - Loss: 0.3609
Epoch 56/100 - Loss: 0.3620
Epoch 57/100 - Loss: 0.3588
Epoch 58/100 - Loss: 0.3610
Epoch 59/100 - Loss: 0.3653
Epoch 60/100 - Loss: 0.3586
Epoch 61/100 - Loss: 0.3612
Epoch 62/100 - Loss: 0.3558
Epoch 63/100 - Loss: 0.3609
Epoch 64/100 - Loss: 0.3609
Epoch 65/100 - Loss: 0.3612
Epoch 66/100 - Loss: 0.3596
Epoch 67/100 - Loss: 0.3570
Epoch 68/100 - Loss: 0.3586
Epoch 69/100 - Loss: 0.3588
Epoch 70/100 - Loss: 0.3585
Epoch 71/100 - Loss: 0.3525
Epoch 72/100 - Loss: 0.3518
Epoch 73/100 - Loss: 0.3647
Epoch 74/100 - Loss: 0.3535
Epoch 75/100 - Loss: 0.3650
Epoch 76/100 - Loss: 0.3542
Epoch 77/100 - Loss: 0.3517
Epoch 78/100 - Loss: 0.3557
Epoch 79/100 - Loss: 0.3504
Epoch 80/100 - Loss: 0.3499
Epoch 81/100 - Loss: 0.3522
Epoch 82/100 - Loss: 0.3495
Epoch 83/100 - Loss: 0.3496
Epoch 84/100 - Loss: 0.3491
Epoch 85/100 - Loss: 0.3484
Epoch 86/100 - Loss: 0.3514
Epoch 87/100 - Loss: 0.3536
Epoch 88/100 - Loss: 0.3465
Epoch 89/100 - Loss: 0.3484
Epoch 90/100 - Loss: 0.3393
Epoch 91/100 - Loss: 0.3434
Epoch 92/100 - Loss: 0.3452
Epoch 93/100 - Loss: 0.3471
Epoch 94/100 - Loss: 0.3450
Epoch 95/100 - Loss: 0.3440
Epoch 96/100 - Loss: 0.3402
Epoch 97/100 - Loss: 0.3382
Epoch 98/100 - Loss: 0.3400
Epoch 99/100 - Loss: 0.3403
Epoch 100/100 - Loss: 0.3378
Test Loss: 0.3096
```

The loss gets worse with more layers

G

In [38]:

```
import pandas as pd
import torch
from torch.utils.data import DataLoader, Dataset
import torch.nn as nn
import torch.optim as optim

# Load and normalize the dataset
df = pd.read_csv('daily-min-temperatures.csv')
df['Temp'] = (df['Temp'] - df['Temp'].mean()) / df['Temp'].std()

# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Custom Dataset
class TempSequenceDataset(Dataset):
    def __init__(self, temps, sequence_length=7, t=1):  # t-day ahead predic
        self.sequence_length = sequence_length
        self.t = t
        self.temps = torch.tensor(temps.values, dtype=torch.float32)
        self.samples = []
        self.targets = []

        for i in range(len(self.temps) - sequence_length - t + 1):
            self.samples.append(self.temps[i:i+sequence_length])
            self.targets.append(self.temps[i+sequence_length + t - 1])

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

    def __getitem__(self, idx):
        return self.samples[idx].unsqueeze(1), self.targets[idx]  # (seq_ler


# Split and create datasets
train_dataset = TempSequenceDataset(df['Temp'][:int(len(df)*0.8)])
test_dataset = TempSequenceDataset(df['Temp'][int(len(df)*0.8):])

train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=256)

# Define the LSTM model
class LSTMModel(nn.Module):
    def __init__(self, input_size=1, hidden_dim=128, num_layers=3):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_dim, num_layers=num_layers, b
        self.fc = nn.Linear(hidden_dim, 1)
```

```python
    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        last_out = lstm_out[:, -1, :]  # last time step
        out = self.fc(last_out)
        return out.squeeze()

def train_and_evaluate(t):
    print(f"\n=== Predicting {t}-day ahead ===")
    train_dataset = TempSequenceDataset(df['Temp'][:int(len(df)*0.8)], t=t)
    test_dataset = TempSequenceDataset(df['Temp'][int(len(df)*0.8):], t=t)

    train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=256)

    model = LSTMModel().to(device)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(30):  # Reduce for quicker testing
        model.train()
        total_loss = 0
        for X_batch, y_batch in train_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            optimizer.zero_grad()
            output = model(X_batch)
            loss = criterion(output, y_batch)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        print(f"Epoch {epoch+1}/30 - Loss: {total_loss / len(train_loader):.

    # Evaluate
    model.eval()
    with torch.no_grad():
        test_losses = []
        for X_batch, y_batch in test_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            output = model(X_batch)
            loss = criterion(output, y_batch)
            test_losses.append(loss.item())
        print(f"Test Loss for t={t}: {sum(test_losses) / len(test_losses):.4

for t in [1, 3, 5]:
    train_and_evaluate(t)
```

```
=== Predicting 1-day ahead ===
Epoch 1/30 - Loss: 0.8754
Epoch 2/30 - Loss: 0.5777
Epoch 3/30 - Loss: 0.5423
Epoch 4/30 - Loss: 0.5220
Epoch 5/30 - Loss: 0.5198
Epoch 6/30 - Loss: 0.5064
Epoch 7/30 - Loss: 0.5032
Epoch 8/30 - Loss: 0.4948
Epoch 9/30 - Loss: 0.4708
Epoch 10/30 - Loss: 0.4569
Epoch 11/30 - Loss: 0.4176
Epoch 12/30 - Loss: 0.3961
Epoch 13/30 - Loss: 0.3791
Epoch 14/30 - Loss: 0.3745
Epoch 15/30 - Loss: 0.3727
Epoch 16/30 - Loss: 0.3725
Epoch 17/30 - Loss: 0.3758
Epoch 18/30 - Loss: 0.3715
Epoch 19/30 - Loss: 0.3712
Epoch 20/30 - Loss: 0.3775
Epoch 21/30 - Loss: 0.3748
Epoch 22/30 - Loss: 0.3778
Epoch 23/30 - Loss: 0.3722
Epoch 24/30 - Loss: 0.3696
Epoch 25/30 - Loss: 0.3723
Epoch 26/30 - Loss: 0.3686
Epoch 27/30 - Loss: 0.3660
Epoch 28/30 - Loss: 0.3671
Epoch 29/30 - Loss: 0.3767
Epoch 30/30 - Loss: 0.3681
Test Loss for t=1: 0.3085

=== Predicting 3-day ahead ===
Epoch 1/30 - Loss: 0.9242
Epoch 2/30 - Loss: 0.5881
Epoch 3/30 - Loss: 0.5591
Epoch 4/30 - Loss: 0.5507
Epoch 5/30 - Loss: 0.5495
Epoch 6/30 - Loss: 0.5415
Epoch 7/30 - Loss: 0.5426
Epoch 8/30 - Loss: 0.5461
Epoch 9/30 - Loss: 0.5453
Epoch 10/30 - Loss: 0.5338
Epoch 11/30 - Loss: 0.5324
Epoch 12/30 - Loss: 0.5322
Epoch 13/30 - Loss: 0.5339
Epoch 14/30 - Loss: 0.5424
Epoch 15/30 - Loss: 0.5349
Epoch 16/30 - Loss: 0.5325
Epoch 17/30 - Loss: 0.5342
Epoch 18/30 - Loss: 0.5328
Epoch 19/30 - Loss: 0.5260
Epoch 20/30 - Loss: 0.5310
Epoch 21/30 - Loss: 0.5246
Epoch 22/30 - Loss: 0.5306
Epoch 23/30 - Loss: 0.5285
Epoch 24/30 - Loss: 0.5281
Epoch 25/30 - Loss: 0.5267
Epoch 26/30 - Loss: 0.5311
Epoch 27/30 - Loss: 0.5265
Epoch 28/30 - Loss: 0.5238
Epoch 29/30 - Loss: 0.5286
Epoch 30/30 - Loss: 0.5345
Test Loss for t=3: 0.4474

=== Predicting 5-day ahead ===
Epoch 1/30 - Loss: 0.8806
Epoch 2/30 - Loss: 0.6091
Epoch 3/30 - Loss: 0.5813
Epoch 4/30 - Loss: 0.5838
Epoch 5/30 - Loss: 0.5608
Epoch 6/30 - Loss: 0.5548
Epoch 7/30 - Loss: 0.5604
Epoch 8/30 - Loss: 0.5544
Epoch 9/30 - Loss: 0.5464
Epoch 10/30 - Loss: 0.5472
Epoch 11/30 - Loss: 0.5504
Epoch 12/30 - Loss: 0.5505
Epoch 13/30 - Loss: 0.5594
Epoch 14/30 - Loss: 0.5555
Epoch 15/30 - Loss: 0.5587
Epoch 16/30 - Loss: 0.5537
Epoch 17/30 - Loss: 0.5436
Epoch 18/30 - Loss: 0.5477
Epoch 19/30 - Loss: 0.5488
Epoch 20/30 - Loss: 0.5421
Epoch 21/30 - Loss: 0.5406
Epoch 22/30 - Loss: 0.5446
Epoch 23/30 - Loss: 0.5447
Epoch 24/30 - Loss: 0.5478
Epoch 25/30 - Loss: 0.5393
Epoch 26/30 - Loss: 0.5461
Epoch 27/30 - Loss: 0.5424
Epoch 28/30 - Loss: 0.5401
Epoch 29/30 - Loss: 0.5366
Epoch 30/30 - Loss: 0.5403
Test Loss for t=5: 0.4509
```

The loss gets worse as you try to predict more days in the future