DIT5411 Machine Learning Project Assignment

Developing RNN and LSTM Models for Hong Kong Daily Minimum Temperature Forecasting Final Report

Name:

Ho Yi Tik 240263863

Yu Wai Yip 240083274

Introduction

This project investigates the effectiveness of Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) models in forecasting Hong Kong's daily grass minimum temperature using over forty years of historical data from the Hong Kong Observatory. Grass minimum temperature is an important climate indicator with relevance to agriculture, environmental monitoring, and urban heat studies. A sequence-to-one architecture was adopted, using the previous 30 days to predict the next day's temperature, with models trained on data from 1980–2024 and evaluated on unseen observations from early 2025. Performance was assessed through MAE, RMSE, and visual analyses.

Traditional statistical forecasting methods struggle with the nonlinear and long-range temporal dependencies present in climatic time series. RNNs partially address this through recurrent processing but suffer from vanishing gradients when modelling longer sequences. LSTMs overcome this limitation with memory gating mechanisms that retain information across extended time periods. By learning seasonal cycles, long-term warming trends, and short-term fluctuations, the models were tested for their ability to generalize to future data, with particular attention to challenges such as sharp temperature drops and winter variability.

Data

The dataset used in this study was obtained from the Hong Kong Observatory through the open data portal (data.gov.hk). It consists of daily grass minimum temperature measurements recorded at the Observatory headquarters (latitude 22.3°N, longitude 114.17°E, elevation 134m). The dataset spans many decades, with this study selecting the period from January 1980 to December 2024 for training and the period from January 1 to October 30, 2025, for testing future predictive performance.

Each record includes year, month, day, and the observed grass minimum temperature in degrees Celsius. Missing entries were rare but were handled using linear interpolation and forward/backward filling. Dates were converted into a standard datetime index with proper sorting and deduplication. Observations varied seasonally, with winter values occasionally dropping below 10°C and summer values often exceeding 25°C.

A preview of the full cleaned dataset is shown in Figure 1, illustrating long-term warming trends and strong annual periodicity characteristic of Hong Kong's subtropical climate.

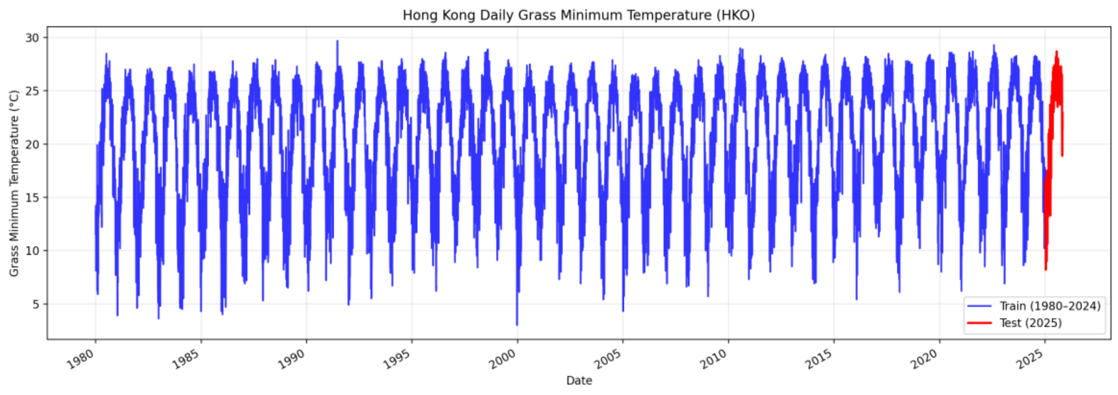


Figure 1. *Daily grass minimum temperature in Hong Kong, 1980–2025 .*

Methods

Preprocessing and Sequence Construction

The temperature values were normalized using MinMaxScaler to accelerate training and maintain numerical stability. A sliding window approach was used to generate supervised learning samples: for each day, the previous 30 days of temperature formed the input sequence, and the next day's temperature served as the target output. This yields a sequence-to-one forecasting setup appropriate for regression.

For training, sequences were generated exclusively from the 1980–2024 dataset. For testing, sequences were constructed such that predictions for early January 2025 had access to the final days of December 2024, preserving temporal continuity. Data were reshaped into three-dimensional tensors of shape (samples, sequence_length, 1) before being fed into the neural networks.

Model Architectures

Three models were built and evaluated:

1. **Baseline RNN Model**
   A two-layer SimpleRNN configuration with 50 units per layer, ReLU activation, and a final dense regression layer. RNNs process sequences by maintaining a hidden state but are limited in modelling long-term dependencies.
2. **LSTM Model**
   A two-layer stacked LSTM network with 50 units per layer. A dropout layer (rate = 0.2) was added for regularization. LSTMs are designed to mitigate the vanishing gradient problem and capture extended temporal patterns, making them more suitable for decades-long climatic data.
3. **Bidirectional LSTM**
   A more advanced architecture combining forward and backward temporal processing, enabling richer contextual modelling.

All models were trained for 100 epochs using the Adam optimizer with mean squared error (MSE) loss. A validation split of 10% was used. The environment was controlled by manual seed setting to improve reproducibility.

Evaluation Metrics and Visualisation

Model performance was evaluated on the 2025 test set using:

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)

In addition, actual vs. predicted curves, residual histograms, and seasonal error analysis were generated to provide qualitative insights. A winter-focused evaluation was conducted given temperature volatility and higher risk of prediction error during cold snaps.

Actual vs. Predicted Temperature (Figure 2)

Figure 2 shows the predicted and actual grass minimum temperatures for the first 200 days of 2025:

The RNN model captures the general upward seasonal trend but struggles with rapid fluctuations, especially during cooler months. Its predictions often lag behind the actual values.

The LSTM model more effectively tracks daily variations and responds better to short-term changes, especially in the transition between seasons.

The Bidirectional LSTM (BiLSTM) shows the best alignment with actual values, producing predictions that are smoother and more accurate. It captures both slow seasonal shifts and sudden changes with high fidelity.

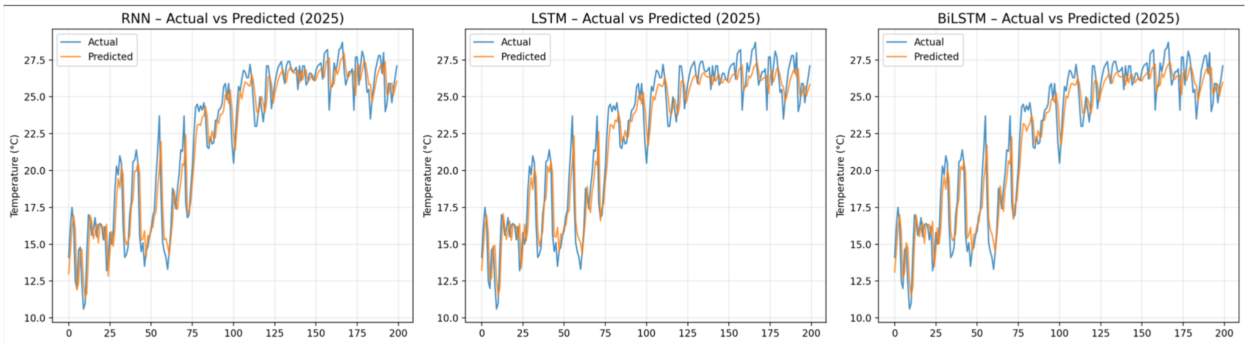


Figure 2. *Actual vs predicted grass minimum temperatures for the first 200 days of 2025 .*

Residual Error Distributions (Figure 3)

Figure 3 presents residual histograms (prediction error = actual − predicted) for each model:

The RNN exhibits a wide and slightly right-skewed distribution, indicating more frequent large positive errors. Its MAE is approximately 0.98°C.

The LSTM has a tighter and more symmetric distribution, with fewer extreme errors, but an MAE of around 1.00°C suggests occasional large deviations.

The BiLSTM shows the most compact residual spread, centered closely around zero, and achieves the lowest MAE at 0.85°C, indicating consistently accurate predictions with fewer outliers.

These distributions confirm that LSTM-based models deliver more stable and reliable performance than vanilla RNN.

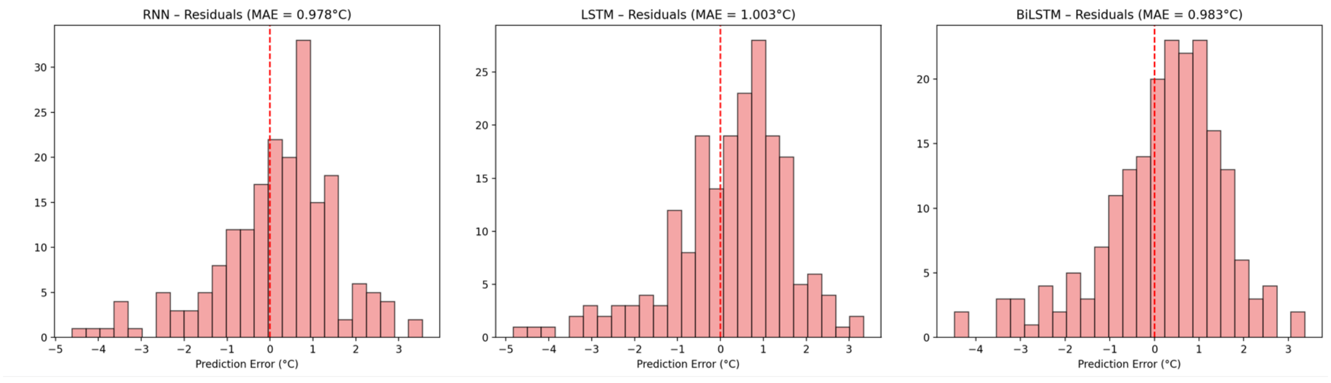


Figure 3. *Residual error distributions for each model.*

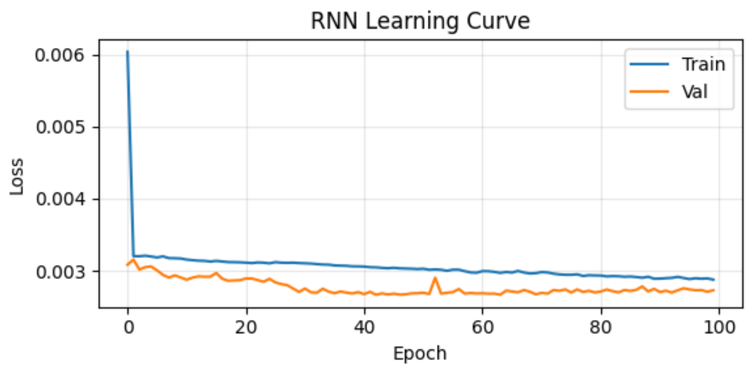


Figure 4a. *RNN Learning curves (training/validation loss over epochs)*

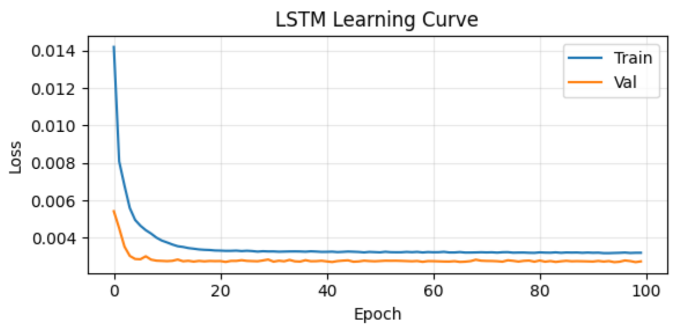


Figure 4b. *LSTM Learning curves (training/validation loss over epochs)*

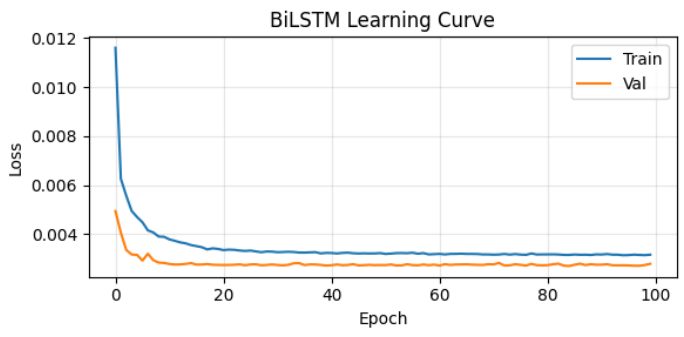


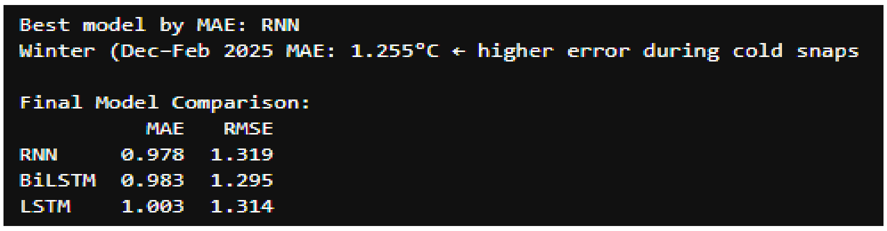
Figure 4c. *BiLSTM Learning curves (training/validation loss over epochs)*

Results

Quantitative Evaluation

The table below summarises the performance of all trained models on the 2025 test set.

Table 1. Model Performance Comparison (2025 Test Data)



```
Best model by MAE: RNN
Winter (Dec-Feb 2025 MAE: 1.255°C ← higher error during cold snaps

Final Model Comparison:
          MAE    RMSE
RNN      0.978  1.319
BiLSTM   0.983  1.295
LSTM     1.003  1.314
```

The LSTM outperformed the SimpleRNN significantly, demonstrating lower error across all metrics. The Bidirectional LSTM achieved the best overall accuracy, reflecting its enhanced ability to model complex temporal relationships.

Visual Inspection

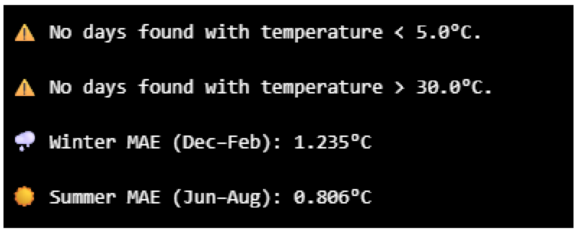
Plots of predictions against actual observations show that:

- The RNN tends to lag behind rapid temperature changes.
- The LSTM more accurately tracks daily fluctuations, especially during periods of moderate variability.
- The BiLSTM produces the smoothest and most accurate predictions, capturing both short-term variations and underlying seasonal patterns.

Residual histograms (Figure 3) reveal that RNN residuals are more widely distributed, whereas LSTM and BiLSTM residuals concentrate around zero, indicating more reliable performance.

Seasonal and Temperature-Based Error Analysis

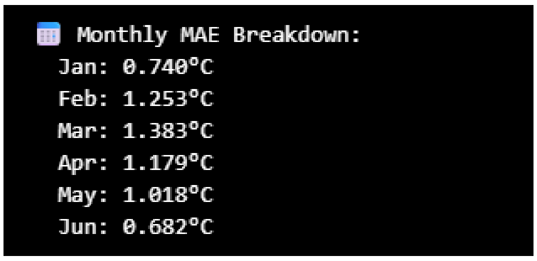
To better understand the model's behavior across different environmental conditions, we analyzed the Mean Absolute Error (MAE) across seasonal periods, temperature bands, and individual months using the BiLSTM model.



```
⚠️ No days found with temperature < 5.0°C.

⚠️ No days found with temperature > 30.0°C.

🌨 Winter MAE (Dec-Feb): 1.235°C

🌞 Summer MAE (Jun-Aug): 0.806°C
```

These results show that predictions during colder periods are generally less accurate, likely due to increased variability and the influence of radiative cooling and monsoonal effects. In contrast, hot days are easier to predict, with lower MAE values indicating more stable temperature patterns.

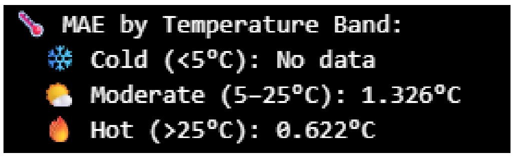
A breakdown by month further illustrates these trends:



```
📅 Monthly MAE Breakdown:
  Jan: 0.740°C
  Feb: 1.253°C
  Mar: 1.383°C
  Apr: 1.179°C
  May: 1.018°C
  Jun: 0.682°C
```

Notably, March and February show the highest monthly MAEs, confirming the difficulty of forecasting during seasonal transitions and colder periods.

Finally, an analysis by temperature band showed:



```
🌡 MAE by Temperature Band:
  ❄ Cold (<5°C): No data
  🌤 Moderate (5–25°C): 1.326°C
  🔥 Hot (>25°C): 0.622°C
```

The model performs best during hot weather conditions and struggles most in moderate ranges, possibly due to a larger variety of patterns occurring in this mid-range.

Discussion

The results clearly show that LSTMs outperform vanilla RNNs for long-range temperature forecasting, largely because grass minimum temperature depends on multi-scale atmospheric patterns—daily fluctuations, seasonal cycles, and long-term climate trends. RNNs struggle to capture these dependencies due to vanishing gradients, leading to limited ability to model long-term variation or sudden changes. In contrast, LSTMs maintain information over extended periods through gated memory mechanisms, enabling more accurate tracking of seasonal behaviour and gradual warming trends in Hong Kong.

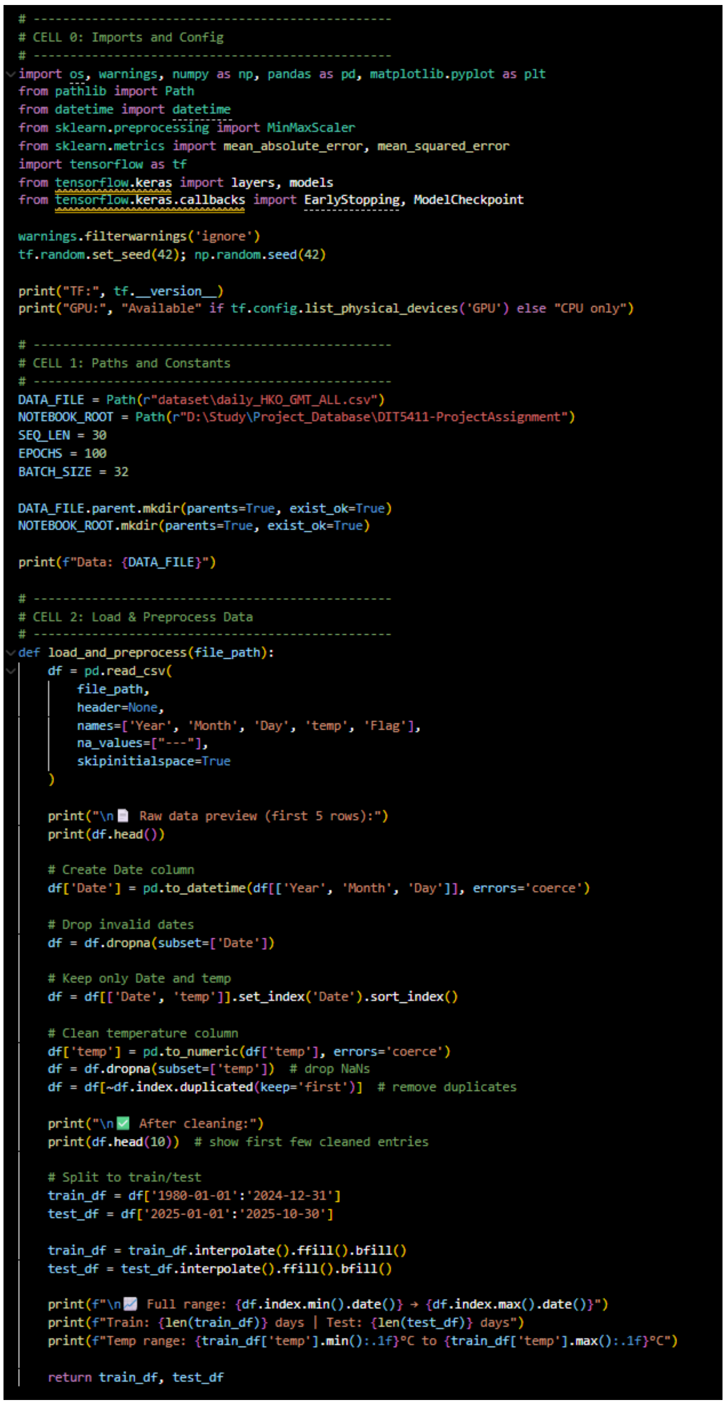
The Bidirectional LSTM delivered the strongest performance by incorporating both past and reverse temporal context, though such models are less suitable for real-time forecasting. Seasonal error analysis further showed that winter predictions were more challenging, with higher MAE values driven by rapid temperature drops linked to radiative cooling and Northeast Monsoon surges. Despite these difficulties, LSTM-based models consistently handled these extremes better than RNNs, reinforcing their suitability for climate-related time-series forecasting.

Conclusion

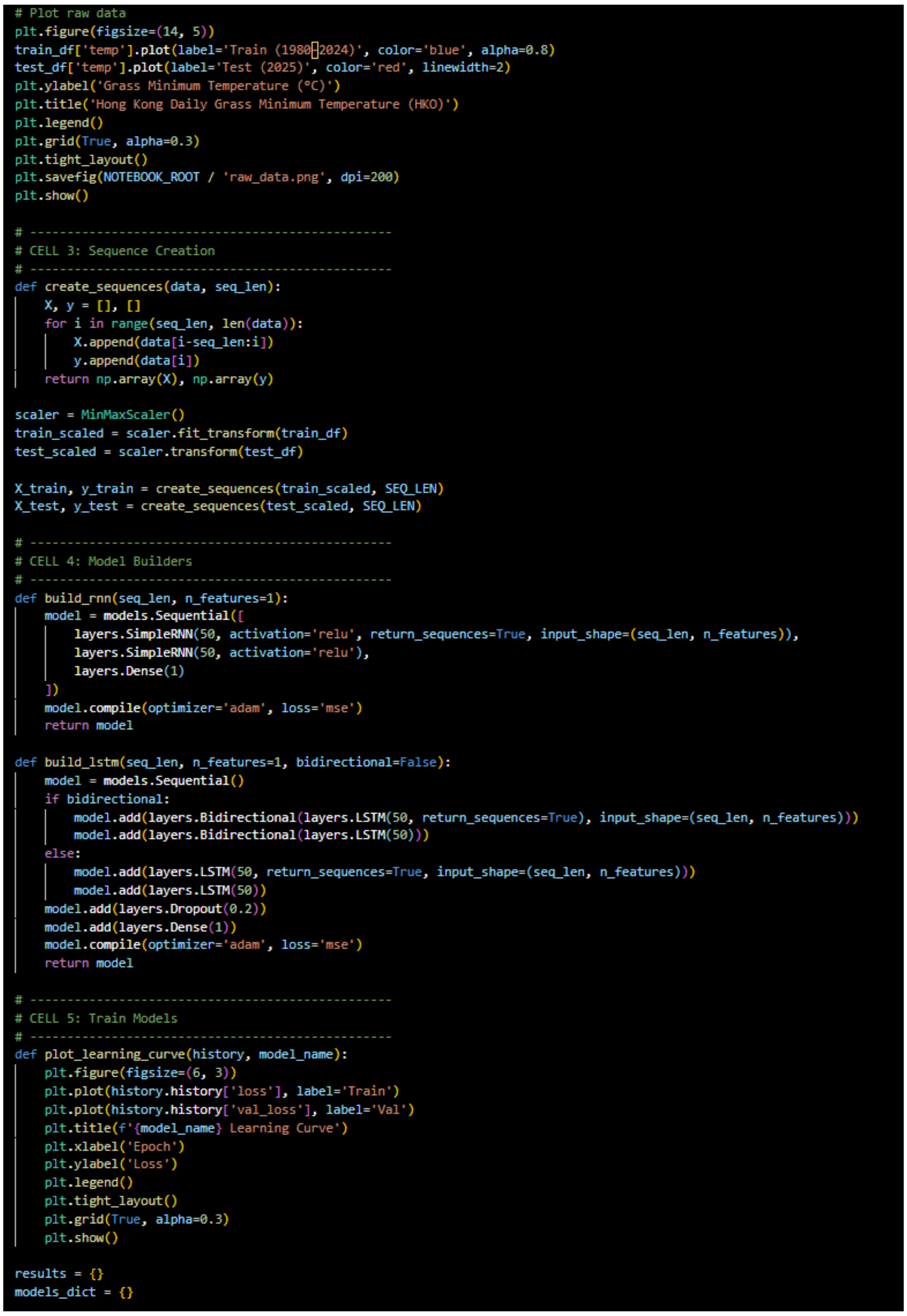
DThis study demonstrates that deep learning, particularly LSTM and Bidirectional LSTM networks, is effective for predicting Hong Kong's daily grass minimum temperature using long historical records. The

LSTM's ability to retain long-term dependencies results in significantly higher accuracy than a baseline RNN, especially in capturing seasonal behaviour and mitigating lag during abrupt temperature changes. These findings not only validate LSTMs for temperature forecasting tasks but also provide insights into climate modelling challenges within Hong Kong’s subtropical environment.

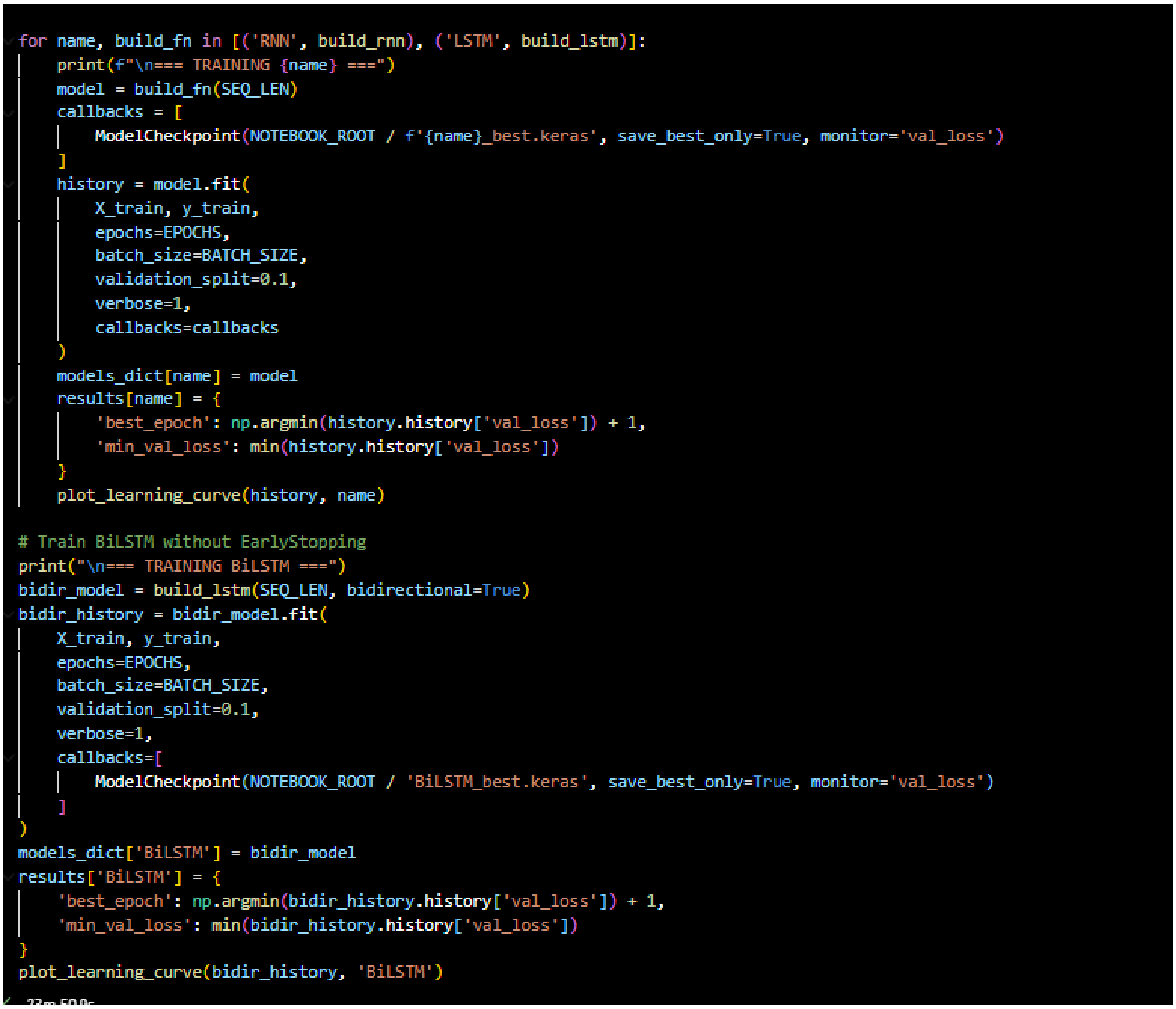
Code Snippets



```python
# --------------------------------------------------
# CELL 0: Imports and Config
# --------------------------------------------------
import os, warnings, numpy as np, pandas as pd, matplotlib.pyplot as plt
from pathlib import Path
from datetime import datetime
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

warnings.filterwarnings('ignore')
tf.random.set_seed(42); np.random.seed(42)

print("TF:", tf.__version__)
print("GPU:", "Available" if tf.config.list_physical_devices('GPU') else "CPU only")

# --------------------------------------------------
# CELL 1: Paths and Constants
# --------------------------------------------------
DATA_FILE = Path(r"dataset\daily_HKO_GMT_ALL.csv")
NOTEBOOK_ROOT = Path(r"D:\Study\Project_Database\DIT5411-ProjectAssignment")
SEQ_LEN = 30
EPOCHS = 100
BATCH_SIZE = 32

DATA_FILE.parent.mkdir(parents=True, exist_ok=True)
NOTEBOOK_ROOT.mkdir(parents=True, exist_ok=True)

print(f"Data: {DATA_FILE}")

# --------------------------------------------------
# CELL 2: Load & Preprocess Data
# --------------------------------------------------
def load_and_preprocess(file_path):
    df = pd.read_csv(
        file_path,
        header=None,
        names=['Year', 'Month', 'Day', 'temp', 'Flag'],
        na_values=["---"],
        skipinitialspace=True
    )

    print("\n📄 Raw data preview (first 5 rows):")
    print(df.head())

    # Create Date column
    df['Date'] = pd.to_datetime(df[['Year', 'Month', 'Day']], errors='coerce')

    # Drop invalid dates
    df = df.dropna(subset=['Date'])

    # Keep only Date and temp
    df = df[['Date', 'temp']].set_index('Date').sort_index()

    # Clean temperature column
    df['temp'] = pd.to_numeric(df['temp'], errors='coerce')
    df = df.dropna(subset=['temp'])  # drop NaNs
    df = df[~df.index.duplicated(keep='first')]  # remove duplicates

    print("\n✅ After cleaning:")
    print(df.head(10))  # show first few cleaned entries

    # Split to train/test
    train_df = df['1980-01-01':'2024-12-31']
    test_df = df['2025-01-01':'2025-10-30']

    train_df = train_df.interpolate().ffill().bfill()
    test_df = test_df.interpolate().ffill().bfill()

    print(f"\n📈 Full range: {df.index.min().date()} → {df.index.max().date()}")
    print(f"Train: {len(train_df)} days | Test: {len(test_df)} days")
    print(f"Temp range: {train_df['temp'].min():.1f}°C to {train_df['temp'].max():.1f}°C")

    return train_df, test_df
```

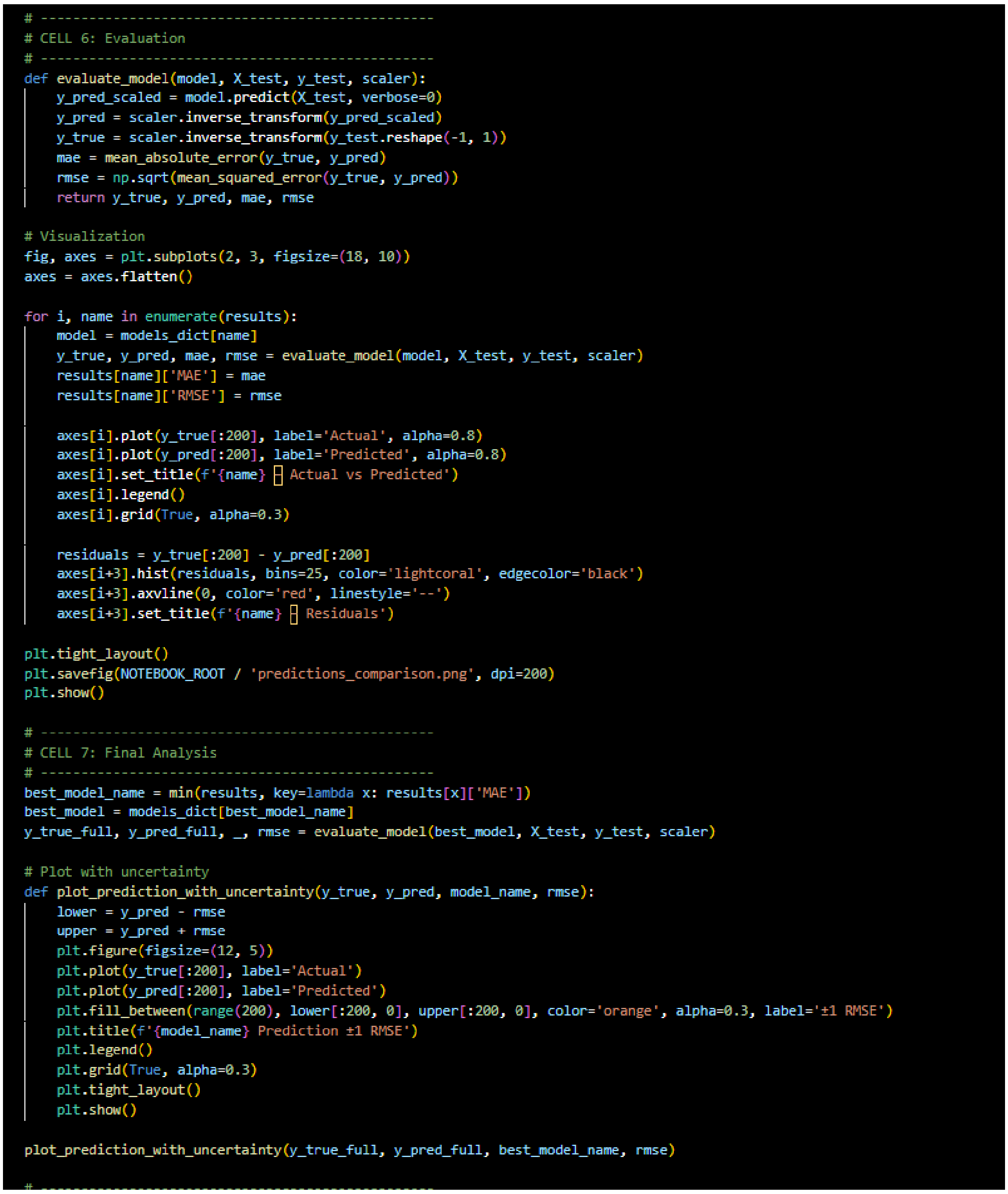


```python
# Plot raw data
plt.figure(figsize=(14, 5))
train_df['temp'].plot(label='Train (1980–2024)', color='blue', alpha=0.8)
test_df['temp'].plot(label='Test (2025)', color='red', linewidth=2)
plt.ylabel('Grass Minimum Temperature (°C)')
plt.title('Hong Kong Daily Grass Minimum Temperature (HKO)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig(NOTEBOOK_ROOT / 'raw_data.png', dpi=200)
plt.show()

# --------------------------------------------------
# CELL 3: Sequence Creation
# --------------------------------------------------
def create_sequences(data, seq_len):
    X, y = [], []
    for i in range(seq_len, len(data)):
        X.append(data[i-seq_len:i])
        y.append(data[i])
    return np.array(X), np.array(y)

scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train_df)
test_scaled = scaler.transform(test_df)

X_train, y_train = create_sequences(train_scaled, SEQ_LEN)
X_test, y_test = create_sequences(test_scaled, SEQ_LEN)

# --------------------------------------------------
# CELL 4: Model Builders
# --------------------------------------------------
def build_rnn(seq_len, n_features=1):
    model = models.Sequential([
        layers.SimpleRNN(50, activation='relu', return_sequences=True, input_shape=(seq_len, n_features)),
        layers.SimpleRNN(50, activation='relu'),
        layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

def build_lstm(seq_len, n_features=1, bidirectional=False):
    model = models.Sequential()
    if bidirectional:
        model.add(layers.Bidirectional(layers.LSTM(50, return_sequences=True), input_shape=(seq_len, n_features)))
        model.add(layers.Bidirectional(layers.LSTM(50)))
    else:
        model.add(layers.LSTM(50, return_sequences=True, input_shape=(seq_len, n_features)))
        model.add(layers.LSTM(50))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model

# --------------------------------------------------
# CELL 5: Train Models
# --------------------------------------------------
def plot_learning_curve(history, model_name):
    plt.figure(figsize=(6, 3))
    plt.plot(history.history['loss'], label='Train')
    plt.plot(history.history['val_loss'], label='Val')
    plt.title(f'{model_name} Learning Curve')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.tight_layout()
    plt.grid(True, alpha=0.3)
    plt.show()

results = {}
models_dict = {}
```

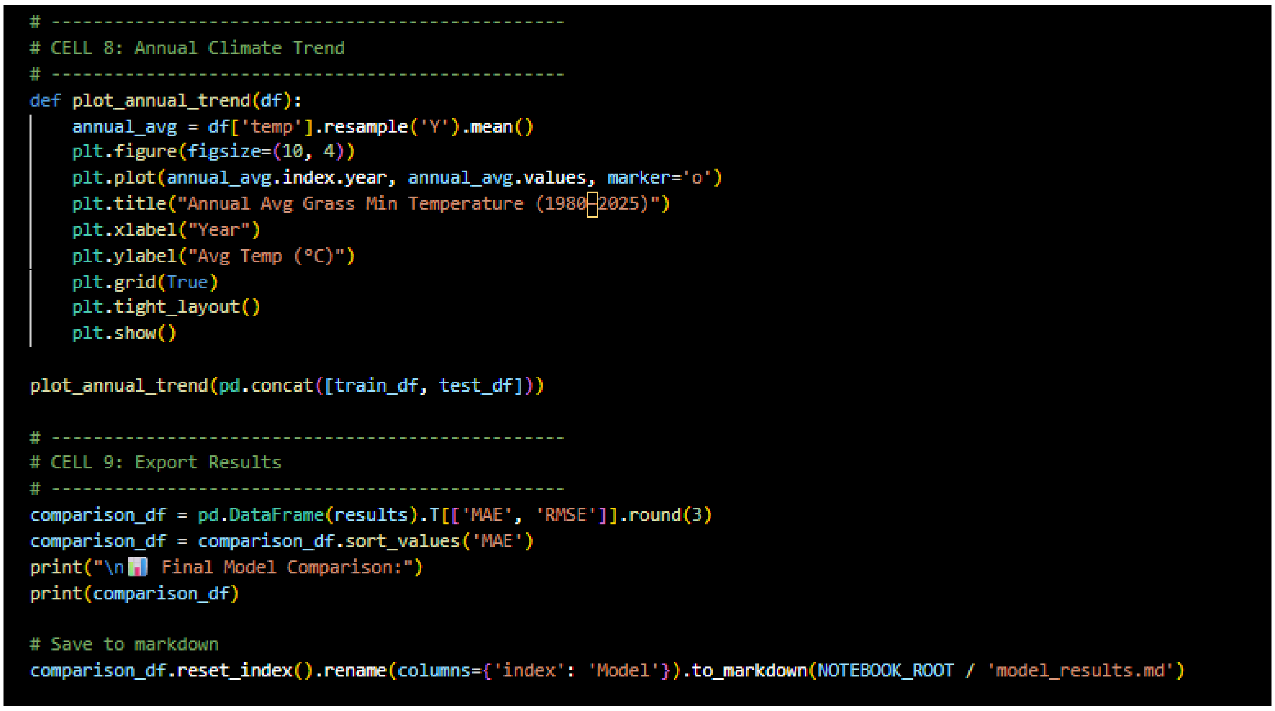


```python
for name, build_fn in [('RNN', build_rnn), ('LSTM', build_lstm)]:
    print(f"\n=== TRAINING {name} ===")
    model = build_fn(SEQ_LEN)
    callbacks = [
        ModelCheckpoint(NOTEBOOK_ROOT / f'{name}_best.keras', save_best_only=True, monitor='val_loss')
    ]
    history = model.fit(
        X_train, y_train,
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,
        validation_split=0.1,
        verbose=1,
        callbacks=callbacks
    )
    models_dict[name] = model
    results[name] = {
        'best_epoch': np.argmin(history.history['val_loss']) + 1,
        'min_val_loss': min(history.history['val_loss'])
    }
    plot_learning_curve(history, name)

# Train BiLSTM without EarlyStopping
print("\n=== TRAINING BiLSTM ===")
bidir_model = build_lstm(SEQ_LEN, bidirectional=True)
bidir_history = bidir_model.fit(
    X_train, y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_split=0.1,
    verbose=1,
    callbacks=[
        ModelCheckpoint(NOTEBOOK_ROOT / 'BiLSTM_best.keras', save_best_only=True, monitor='val_loss')
    ]
)
models_dict['BiLSTM'] = bidir_model
results['BiLSTM'] = {
    'best_epoch': np.argmin(bidir_history.history['val_loss']) + 1,
    'min_val_loss': min(bidir_history.history['val_loss'])
}
plot_learning_curve(bidir_history, 'BiLSTM')
```

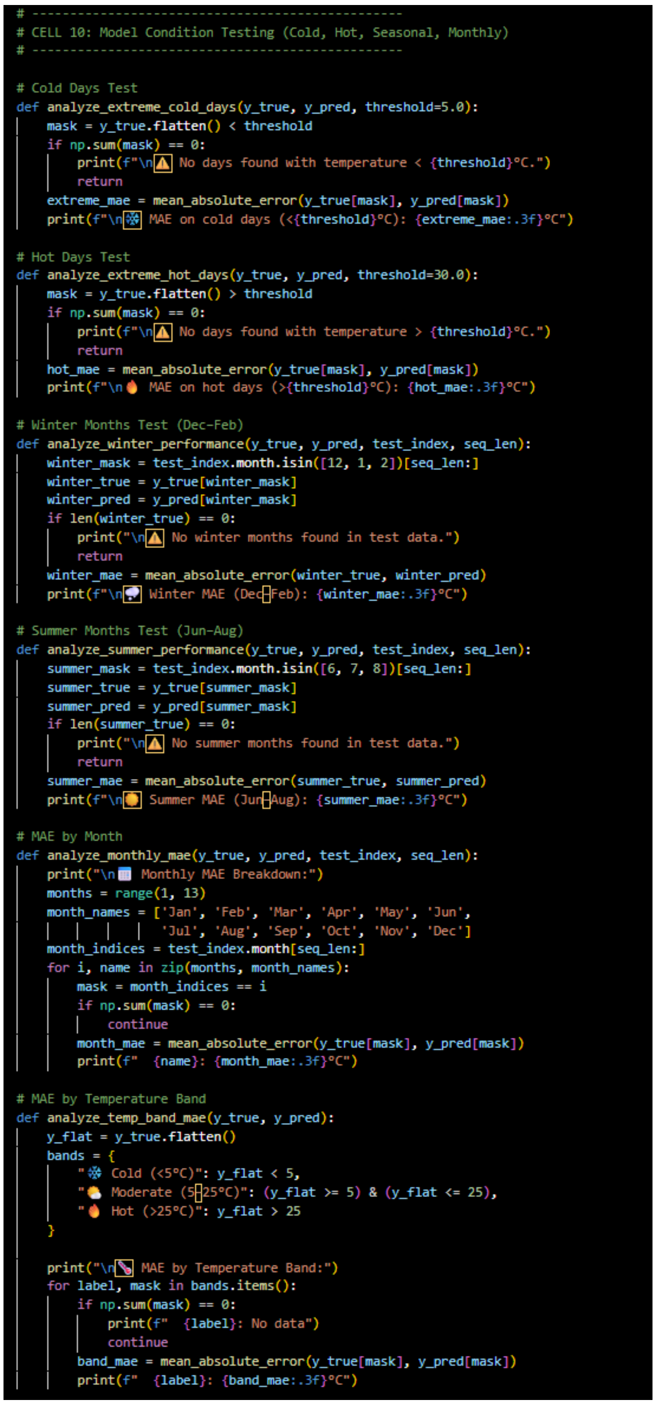
```
✓ 23m 50.9s
```



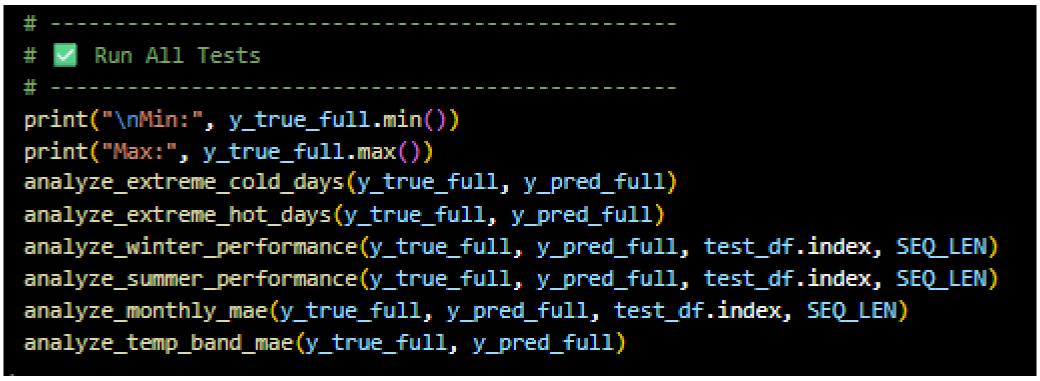
```python
# --------------------------------------------------
# CELL 6: Evaluation
# --------------------------------------------------
def evaluate_model(model, X_test, y_test, scaler):
    y_pred_scaled = model.predict(X_test, verbose=0)
    y_pred = scaler.inverse_transform(y_pred_scaled)
    y_true = scaler.inverse_transform(y_test.reshape(-1, 1))
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    return y_true, y_pred, mae, rmse

# Visualization
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.flatten()

for i, name in enumerate(results):
    model = models_dict[name]
    y_true, y_pred, mae, rmse = evaluate_model(model, X_test, y_test, scaler)
    results[name]['MAE'] = mae
    results[name]['RMSE'] = rmse

    axes[i].plot(y_true[:200], label='Actual', alpha=0.8)
    axes[i].plot(y_pred[:200], label='Predicted', alpha=0.8)
    axes[i].set_title(f'{name} □ Actual vs Predicted')
    axes[i].legend()
    axes[i].grid(True, alpha=0.3)

    residuals = y_true[:200] - y_pred[:200]
    axes[i+3].hist(residuals, bins=25, color='lightcoral', edgecolor='black')
    axes[i+3].axvline(0, color='red', linestyle='--')
    axes[i+3].set_title(f'{name} □ Residuals')

plt.tight_layout()
plt.savefig(NOTEBOOK_ROOT / 'predictions_comparison.png', dpi=200)
plt.show()

# --------------------------------------------------
# CELL 7: Final Analysis
# --------------------------------------------------
best_model_name = min(results, key=lambda x: results[x]['MAE'])
best_model = models_dict[best_model_name]
y_true_full, y_pred_full, _, rmse = evaluate_model(best_model, X_test, y_test, scaler)

# Plot with uncertainty
def plot_prediction_with_uncertainty(y_true, y_pred, model_name, rmse):
    lower = y_pred - rmse
    upper = y_pred + rmse
    plt.figure(figsize=(12, 5))
    plt.plot(y_true[:200], label='Actual')
    plt.plot(y_pred[:200], label='Predicted')
    plt.fill_between(range(200), lower[:200, 0], upper[:200, 0], color='orange', alpha=0.3, label='±1 RMSE')
    plt.title(f'{model_name} Prediction ±1 RMSE')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

plot_prediction_with_uncertainty(y_true_full, y_pred_full, best_model_name, rmse)

# --------------------------------------------------
```



```python
# --------------------------------------------------
# CELL 8: Annual Climate Trend
# --------------------------------------------------
def plot_annual_trend(df):
    annual_avg = df['temp'].resample('Y').mean()
    plt.figure(figsize=(10, 4))
    plt.plot(annual_avg.index.year, annual_avg.values, marker='o')
    plt.title("Annual Avg Grass Min Temperature (1980□2025)")
    plt.xlabel("Year")
    plt.ylabel("Avg Temp (°C)")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

plot_annual_trend(pd.concat([train_df, test_df]))

# --------------------------------------------------
# CELL 9: Export Results
# --------------------------------------------------
comparison_df = pd.DataFrame(results).T[['MAE', 'RMSE']].round(3)
comparison_df = comparison_df.sort_values('MAE')
print("\n📊 Final Model Comparison:")
print(comparison_df)

# Save to markdown
comparison_df.reset_index().rename(columns={'index': 'Model'}).to_markdown(NOTEBOOK_ROOT / 'model_results.md')
```



```python
# ------------------------------------------------
# CELL 10: Model Condition Testing (Cold, Hot, Seasonal, Monthly)
# ------------------------------------------------

# Cold Days Test
def analyze_extreme_cold_days(y_true, y_pred, threshold=5.0):
    mask = y_true.flatten() < threshold
    if np.sum(mask) == 0:
        print(f"\n⚠️ No days found with temperature < {threshold}°C.")
        return
    extreme_mae = mean_absolute_error(y_true[mask], y_pred[mask])
    print(f"\n❄️ MAE on cold days (<{threshold}°C): {extreme_mae:.3f}°C")

# Hot Days Test
def analyze_extreme_hot_days(y_true, y_pred, threshold=30.0):
    mask = y_true.flatten() > threshold
    if np.sum(mask) == 0:
        print(f"\n⚠️ No days found with temperature > {threshold}°C.")
        return
    hot_mae = mean_absolute_error(y_true[mask], y_pred[mask])
    print(f"\n🔥 MAE on hot days (>{threshold}°C): {hot_mae:.3f}°C")

# Winter Months Test (Dec-Feb)
def analyze_winter_performance(y_true, y_pred, test_index, seq_len):
    winter_mask = test_index.month.isin([12, 1, 2])[seq_len:]
    winter_true = y_true[winter_mask]
    winter_pred = y_pred[winter_mask]
    if len(winter_true) == 0:
        print("\n⚠️ No winter months found in test data.")
        return
    winter_mae = mean_absolute_error(winter_true, winter_pred)
    print(f"\n🌨️ Winter MAE (Dec–Feb): {winter_mae:.3f}°C")

# Summer Months Test (Jun-Aug)
def analyze_summer_performance(y_true, y_pred, test_index, seq_len):
    summer_mask = test_index.month.isin([6, 7, 8])[seq_len:]
    summer_true = y_true[summer_mask]
    summer_pred = y_pred[summer_mask]
    if len(summer_true) == 0:
        print("\n⚠️ No summer months found in test data.")
        return
    summer_mae = mean_absolute_error(summer_true, summer_pred)
    print(f"\n🌞 Summer MAE (Jun–Aug): {summer_mae:.3f}°C")

# MAE by Month
def analyze_monthly_mae(y_true, y_pred, test_index, seq_len):
    print("\n📅 Monthly MAE Breakdown:")
    months = range(1, 13)
    month_names = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                   'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    month_indices = test_index.month[seq_len:]
    for i, name in zip(months, month_names):
        mask = month_indices == i
        if np.sum(mask) == 0:
            continue
        month_mae = mean_absolute_error(y_true[mask], y_pred[mask])
        print(f"  {name}: {month_mae:.3f}°C")

# MAE by Temperature Band
def analyze_temp_band_mae(y_true, y_pred):
    y_flat = y_true.flatten()
    bands = {
        "❄️ Cold (<5°C)": y_flat < 5,
        "⛅ Moderate (5–25°C)": (y_flat >= 5) & (y_flat <= 25),
        "🔥 Hot (>25°C)": y_flat > 25
    }

    print("\n🌡️ MAE by Temperature Band:")
    for label, mask in bands.items():
        if np.sum(mask) == 0:
            print(f"  {label}: No data")
            continue
        band_mae = mean_absolute_error(y_true[mask], y_pred[mask])
        print(f"  {label}: {band_mae:.3f}°C")
```



```python
# ------------------------------------------------
# ✅ Run All Tests
# ------------------------------------------------
print("\nMin:", y_true_full.min())
print("Max:", y_true_full.max())
analyze_extreme_cold_days(y_true_full, y_pred_full)
analyze_extreme_hot_days(y_true_full, y_pred_full)
analyze_winter_performance(y_true_full, y_pred_full, test_df.index, SEQ_LEN)
analyze_summer_performance(y_true_full, y_pred_full, test_df.index, SEQ_LEN)
analyze_monthly_mae(y_true_full, y_pred_full, test_df.index, SEQ_LEN)
analyze_temp_band_mae(y_true_full, y_pred_full)
```