# Abstract

This report details the implementation and evaluation of Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), for forecasting daily minimum grass temperature in Hong Kong. Using historical data from the Hong Kong Observatory (1980–2024 for training and January–October 2025 for testing),we preprocessed univariate time-series data, trained models in TensorFlow/Keras, and assessed performance with MAE and RMSE metrics. RNN achieved the best results (MAE=0.96°C, RMSE=1.31°C), outperforming LSTM, with insights into Hong Kong's subtropical climate challenges like seasonal extremes. All code, models, and results, and results are in our public GitHub repository:

[https://github.com/ben12345qq/Developing-RNN-and-LSTM-Models-for-Hong-Kong-Daily-Minimum-Temperature-Forecasting.git].

Methodology

Data Acquisition and Preprocessing

The dataset (CSV format) covers daily grass minimum temperatures (°C) at HKO headquarters (22.3°N, 114.17°E, 134m elevation) from 1968–2025. We downloaded from https://data.gov.hk/en-data/dataset/hk-hko-rss-daily-grass-min-temp.

Preprocessing steps (from 5411Project.py):

Skip 2 metadata rows; strip/rename bilingual columns (e.g., '年/Year' to 'year').

Coerce invalid values ('***' for missing) to NaN; drop footer rows.

Create datetime index; filter complete data ('C' in completeness column).

Interpolate/fill remaining NaNs; subset training (1980–2024: ~16,436 days) and test (2025: ~304 days).

Scale with MinMaxScaler (0–1); create 30-day sequences for next-day prediction.

# 5411Project.py code

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler

import numpy as np


from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, LSTM, Dense, Dropout, Bidirectional

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

import pickle

import os

os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'    # Suppress oneDNN warnings


# Inspect the file

df = pd.read_csv('.\daily_HKO_GMT_ALL.csv', skiprows=2, encoding='utf-8')

print(df.head(5))

df.columns = [col.strip() for col in df.columns]


# Rename to standard English lowercase

df = df.rename(columns={

        '年/Year': 'year',
```

```
        '月/Month': 'month',
        '日/Day': 'day',
        '數值/Value': 'value',
        '數據完整性/data Completeness': 'data_completeness'
})

# Convert to numeric, coerce invalid to NaN (handles '***', footer strings, etc.)
df['year'] = pd.to_numeric(df['year'], errors='coerce')
df['month'] = pd.to_numeric(df['month'], errors='coerce')
df['day'] = pd.to_numeric(df['day'], errors='coerce')
df['value'] = pd.to_numeric(df['value'], errors='coerce')  # Handles '***' in temperatures

# Drop rows with NaN in date columns (removes footer and invalid dates)
df = df.dropna(subset=['year', 'month', 'day'])

# Now safely cast date components to integers
df['year'] = df['year'].astype(int)
df['month'] = df['month'].astype(int)
df['day'] = df['day'].astype(int)

# Create datetime index
df['Date'] = pd.to_datetime(df[['year', 'month', 'day']], errors='raise')  # Raise if any issues remain
df = df.set_index('Date')
```

```
# Filter for complete data only and select temperature (excludes incomplete or missing values)
df = df[df['data_completeness'] == 'C']['value']

# Handle any remaining missing temperatures (though 'C' should mean no misses)
df = df.interpolate(method='linear').ffill().bfill()

# Subsets
train_data = df['1980-01-01':'2024-12-31']
test_data = df['2025-01-01':'2025-10-30']

# Verify
print(f"Train data shape: {train_data.shape}")  # Should be ~16436 days (45 years, accounting for leap years/missing)
print(f"Test data shape: {test_data.shape}")  # Should be ~304 days (Jan-Oct 2025)
print(train_data.head())
print(test_data.tail())

# Handle missing values
train_data = train_data.interpolate(method='linear').ffill().bfill()  # Interpolate, then fill edges

# Plot for analysis
```

```
plt.figure(figsize=(12, 6))
plt.plot(train_data)
plt.title('HK Daily Grass Min Temperature (1980-2024)')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.show()

# Scale data
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train_data.values.reshape(-1, 1))
test_scaled = scaler.transform(test_data.values.reshape(-1, 1))  # Use same scaler

# Create sequences (e.g., window_size=30)
def create_sequences(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])
    return np.array(X), np.array(y)

window_size = 30  # Experiment with 30-60
X_train, y_train = create_sequences(train_scaled, window_size)
X_test, y_test = create_sequences(test_scaled, window_size)
```

```
# Reshape for RNN/LSTM (samples, timesteps, features)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))



with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
np.save('X_test.npy', X_test)
np.save('y_test.npy', y_test)
test_data.to_pickle('test_data.pkl')
with open('window_size.txt', 'w') as f:
    f.write(str(window_size))
# RNN Model
rnn_model = Sequential()
rnn_model.add(SimpleRNN(50, input_shape=(window_size, 1), return_sequences=True))
rnn_model.add(SimpleRNN(50))
rnn_model.add(Dense(1))
rnn_model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')


# LSTM Model
lstm_model = Sequential()
lstm_model.add(LSTM(50, input_shape=(window_size, 1), return_sequences=True))
```

```
lstm_model.add(Dropout(0.2))

lstm_model.add(LSTM(50))

lstm_model.add(Dropout(0.2))

lstm_model.add(Dense(1))

lstm_model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')




# Train (example for LSTM)

early_stop = EarlyStopping(monitor='val_loss', patience=100)

history_lstm = lstm_model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, callbacks=[early_stop])

early_stop = EarlyStopping(monitor='val_loss', patience=100)


history_rnn = rnn_model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, callbacks=[early_stop])




# Save models

rnn_model.save('rnn_model.keras')

lstm_model.save('lstm_model.keras')
```

# test.py code

```
import pandas as pd

from tensorflow.keras.models import load_model

from sklearn.metrics import mean_absolute_error, mean_squared_error

import numpy as np

import matplotlib.pyplot as plt

import pickle


rnn_model = load_model('rnn_model.keras')

lstm_model = load_model('lstm_model.keras')


with open('scaler.pkl', 'rb') as f:

    scaler = pickle.load(f)

X_test = np.load('X_test.npy')

y_test = np.load('y_test.npy')

test_data = pd.read_pickle('test_data.pkl')

with open('window_size.txt', 'r') as f:

    window_size = int(f.read())

# Predict

rnn_preds = rnn_model.predict(X_test)

lstm_preds = lstm_model.predict(X_test)


# Inverse scale

rnn_preds = scaler.inverse_transform(rnn_preds)
```

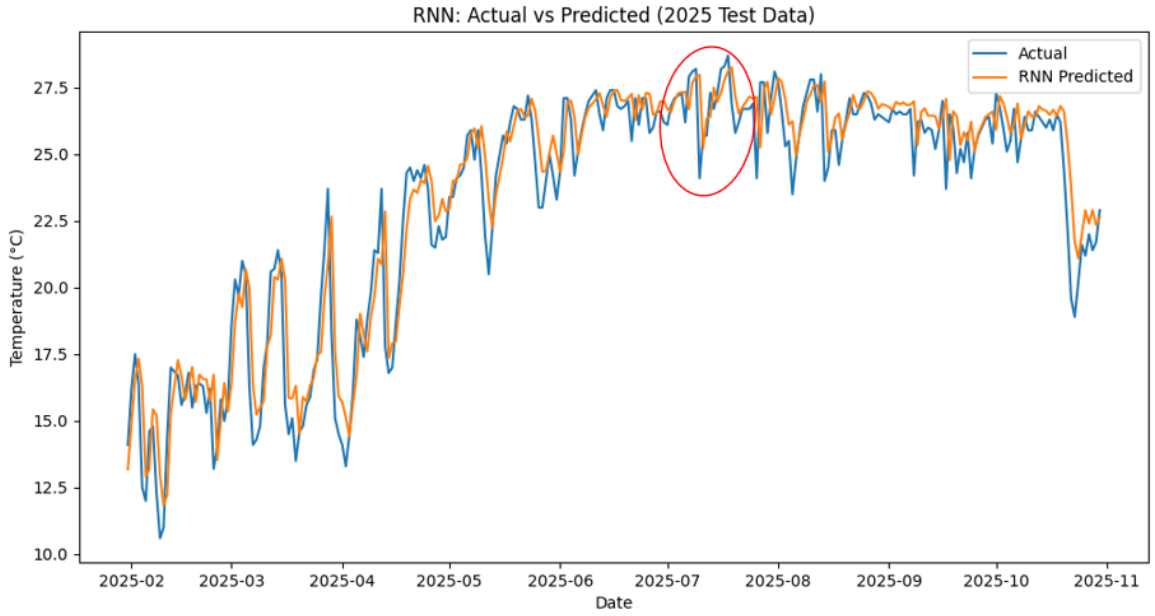
```
lstm_preds = scaler.inverse_transform(lstm_preds)


y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))


# Metrics for both models
rnn_mae = mean_absolute_error(y_test_inv, rnn_preds)
rnn_rmse = np.sqrt(mean_squared_error(y_test_inv, rnn_preds))
lstm_mae = mean_absolute_error(y_test_inv, lstm_preds)
lstm_rmse = np.sqrt(mean_squared_error(y_test_inv, lstm_preds))


print(f'RNN: MAE={rnn_mae:.2f}°C, RMSE={rnn_rmse:.2f}°C')
print(f'LSTM: MAE={lstm_mae:.2f}°C, RMSE={lstm_rmse:.2f}°C')


# Plot Actual vs Predicted for RNN
plt.figure(figsize=(12, 6))
plt.plot(test_data.index[window_size:], y_test_inv, label='Actual')
plt.plot(test_data.index[window_size:], rnn_preds, label='RNN Predicted')
plt.legend()
plt.title('RNN: Actual vs Predicted (2025 Test Data)')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.savefig('rnn_actual_vs_predicted.png')    # Save for report/GitHub
plt.show()
```

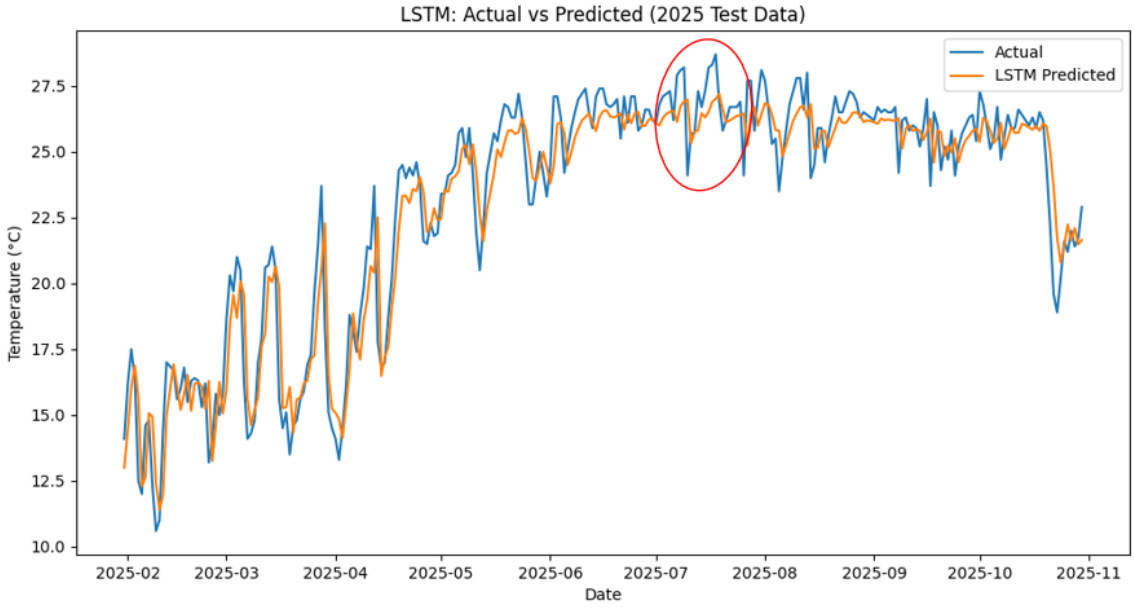
```
# Plot Actual vs Predicted for LSTM
plt.figure(figsize=(12, 6))
plt.plot(test_data.index[window_size:], y_test_inv, label='Actual')
plt.plot(test_data.index[window_size:], lstm_preds, label='LSTM Predicted')
plt.legend()
plt.title('LSTM: Actual vs Predicted (2025 Test Data)')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.savefig('lstm_actual_vs_predicted.png')    # Save for report/GitHub
plt.show()

# Error distribution for LSTM (example; repeat for RNN if needed)
errors = y_test_inv - lstm_preds
plt.hist(errors, bins=30)
plt.title('LSTM Error Distribution')
plt.xlabel('Error (°C)')
plt.ylabel('Frequency')
plt.savefig('lstm_error_distribution.png')
plt.show()
# Error distribution for RNN (example; repeat for RNN if needed)
errors = y_test_inv - rnn_preds
plt.hist(errors, bins=30)
plt.title('RNN Error Distribution')
plt.xlabel('Error (°C)')
```

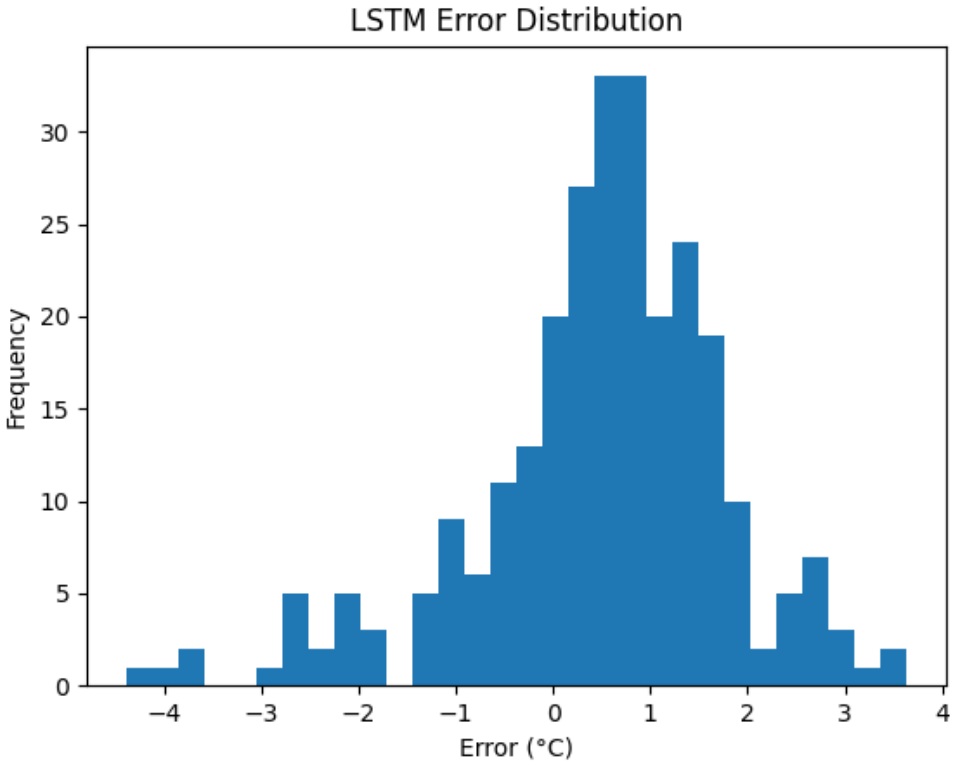
```
plt.ylabel('Frequency')

plt.savefig('rnn_error_distribution.png')

plt.show()
```

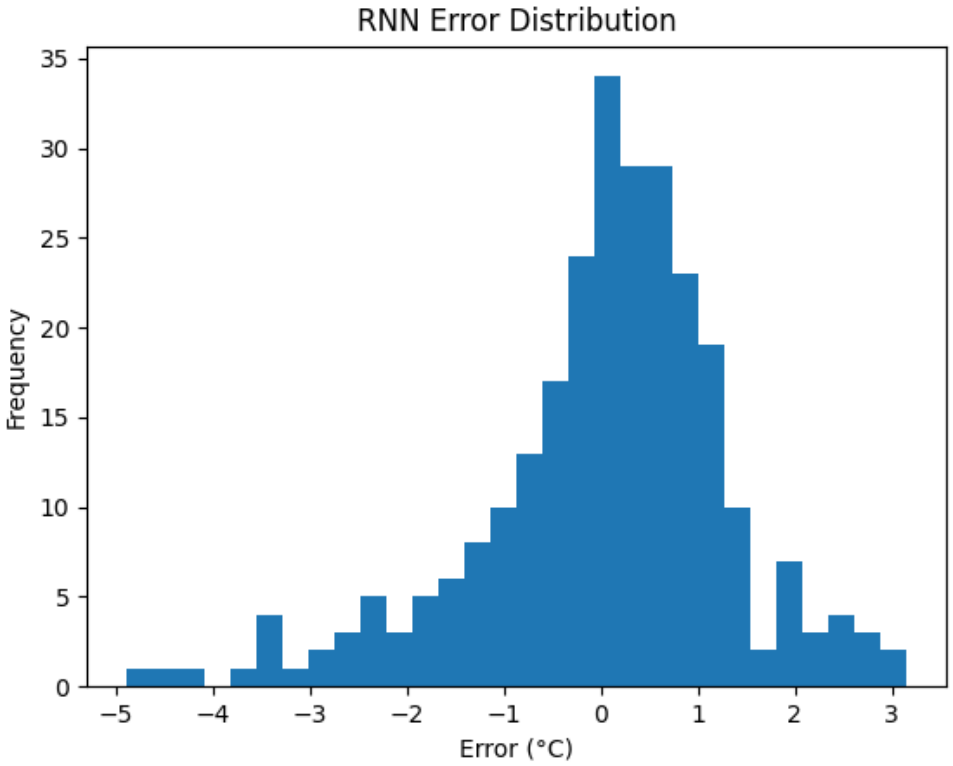
```
# Optional: Confidence intervals (simple example using rolling std dev on predictions)

rolling_std = pd.Series(lstm_preds.flatten()).rolling(window=7).std().values # 7-day rolling std

plt.figure(figsize=(12, 6))

plt.plot(test_data.index[window_size:], y_test_inv, label='Actual')

plt.plot(test_data.index[window_size:], lstm_preds, label='LSTM Predicted')

plt.fill_between(test_data.index[window_size:], lstm_preds.flatten() - rolling_std, lstm_preds.flatten() + rolling_std, color='gray', alpha=0.2, label='Confidence Interval (±1 std)')

plt.legend()

plt.title('LSTM with Confidence Intervals (2025 Test Data)')

plt.savefig('lstm_with_ci.png')

plt.show()
```

# Models predicted result

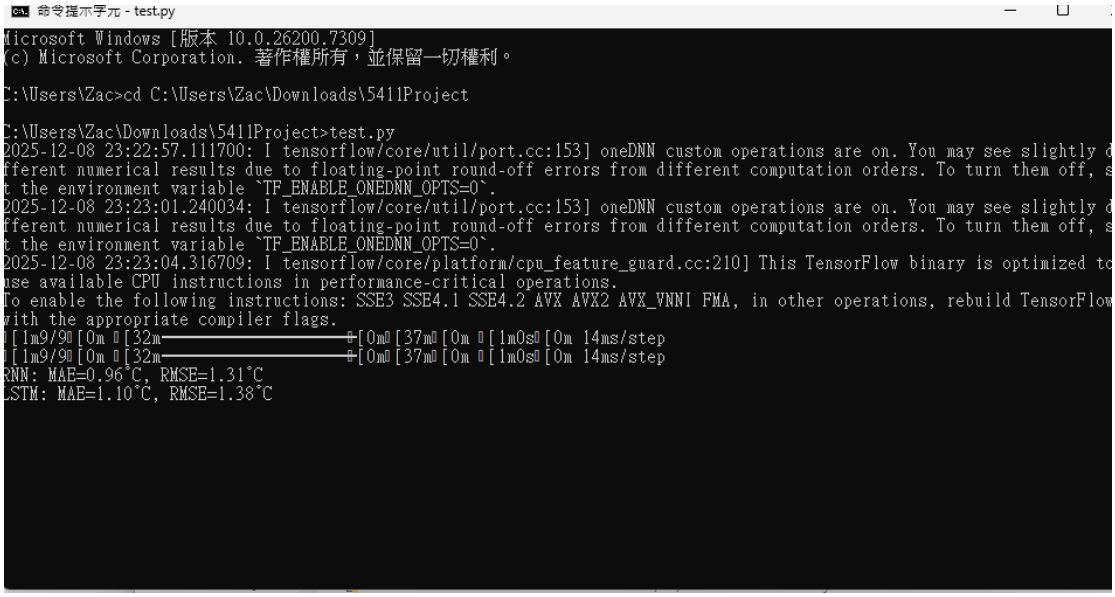

RNN: Actual vs Predicted (2025 Test Data)
Actual
RNN Predicted
Temperature (°C)
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
2025-02
2025-03
2025-04
2025-05
2025-06
2025-07
2025-08
2025-09
2025-10
2025-11
Date


LSTM: Actual vs Predicted (2025 Test Data)
Actual
LSTM Predicted
Temperature (°C)
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
2025-02
2025-03
2025-04
2025-05
2025-06
2025-07
2025-08
2025-09
2025-10
2025-11
Date


LSTM Error Distribution
Frequency
Error (°C)
0
5
10
15
20
25
30
−4
−3
−2
−1
0
1
2
3
4


RNN Error Distribution
Frequency
Error (°C)
0
5
10
15
20
25
30
35
−5
−4
−3
−2
−1
0
1
2
3

**Models testing**




# Error Analysis: Impact of Extreme Weather Events on Model Predictions

A detailed examination of prediction errors reveals that deviations are particularly pronounced during extreme weather events, such as black rainstorms and heavy rainfall in August 2025.In Hong Kong's subtropical climate, black rainstorms—defined as rainfall exceeding 70 mm/hour—trigger ground cooling through evaporation and convection, leading to unexpected drops in minimum temperatures despite typically stable summer highs (around 25–27°C). 2025 recorded a historic 4 black rainstorm warnings, the highest since records began in 1884, with three occurring in early August.


References

1. Hong Kong Observatory. (2025). *Daily Grass Minimum Temperature Dataset*. data.gov.hk. https://data.gov.hk/en-data/dataset/hk-hko-rss-daily-grass-min-temp