National College of Ireland

Configuration Manual

MSc Practicum Part 2
Master of Science in Data Analytics

Krishna Reeja Santhosh Kumar
Student ID: 23325071

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Krishna Reeja Santhosh Kumar
Student ID:	23325071
Programme:	Master of Science in Data Analytics **Year:** 2025
Module:	MSc Practicum Part 2
Supervisor:	Vikas Sahni
Submission Due Date:	28/01/2026
Project Title:	Modelling Crime Dynamics with a Spatio-Temporal CBAM Attention-Enhanced Deep Learning Model
Word Count:	1274 **Page Count: 14**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: 26/01/2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Krishna Reeja Santhosh Kumar
23325071

1 Introduction

The structure of this configuration guide explains how the project *Modelling Crime Dynamics with a Spatio-Temporal CBAM Attention-Enhanced Deep Learning Model* could be reproduced.

This project aims at crime count prediction in Chicago on a daily basis using:

- Historical crime data (2019–2025)
- Daily aggregated Weather variables and a calculated Discomfort Index.

i. Crime dataset: https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2/about_data
ii. Weather dataset: https://rp5.ru/Weather_archive_in_Chicago,_Midway_(airport)

The manual explains:

- System and environment setup
- Dataset acquisition and storage path
- Data preprocessing and feature engineering
- Model implementation (LSTM, BiLSTM, BiGRU, BiLSTM+Attention, BiLSTM+CBAM)
- Evaluation, forecasting and explainability (LIME)

The whole procedure is executed in a single Google Colab notebook.

2 Environment Setup / System Configuration

The proper configuration of the environment is essential to guarantee the reproducibility and proper correct execution of the notebook and model training pipeline.

2.1 Google Colab Runtime (Primary Implementation Environment)

Component	Specification
CPU	Intel(R) Xeon(R) CPU 2.20GHz (2 vCPUs)
GPU	None (CPU-only runtime)
RAM	12.7 GB available
Architecture	x86_64
Storage	Approx. 50 GB temporary disk (auto provided by Colab)

Component	Specification
Environment Type	Virtualized (KVM hypervisor)

2.2 Local Machine

Component	Specification
Device Name	LAPTOP-025KI6PL
CPU	Intel 11th Gen Core i5-11320H @ 3.20 GHz
RAM	16 GB
OS	Windows 11 Home Single Language, Version 24H2 (OS Build 10.0.26100 Build 26100)
System Type	64-bit OS, x64 CPU

2.3 Software Requirements

2.3.1 Operating System

- Windows 11 Home Single Language Version 24H2 (OS Build 10.0.26100).

2.4 Python Version

- Python 3.12.12

2.5 Required Libraries

The major libraries used in the project are:

2.5.1 Core Scientific Libraries

- Numpy: 2.0.2
- pandas: 2.2.2
- matplotlib: 3.10.0
- seaborn: 0.13.2
- scipy: 1.16.3
- statsmodels: 0.14.5

2.5.2 Machine Learning / Deep Learning

- scikit-learn: 1.6.1
- tensorflow: 2.19.0
- keras: 3.10.0
- joblib: 1.5.2

2.5.3 Explainable AI

- Lime: 0.2.0.1

2.5.4 Visualization

- matplotlib.pyplot: 3.10.0
- seaborn: 0.13.2

2.6 Installation Commands

The environment can be recreated with the following:

```
pip install numpy pandas matplotlib seaborn scikit-learn tensorflow keras
pip install lime
pip install statsmodels joblib
```

2.6.1 TensorFlow GPU

```
pip install tensorflow-gpu
```

3 Dataset Collection

Two main datasets are used:

1. **Chicago Crime Data (2019–2025)**
    - Source: Chicago Data Portal
    - file used: Crimes_-_2019_to_Present_20250930.csv
    - Contains individual crime incidents with fields such as Date, Primary Type, Community Area, Arrest, District, Latitude, Longitude, etc.
2. **Chicago Weather Data (2019–2025)**
    - Source: RP5 Weather Archive
    - file used: weather_data.xls
    - Columns used:
        - Local time in Chicago Midway Airport (timestamp)
        - T (temperature)
        - U (relative humidity)
        - Ff (wind speed)

3.1 Directory Structure

Both datasets are stored on Google Drive in:

/drive/MyDrive/Crime_hotspot/Dataset/

The notebook assumes this exact path and if changed, the file paths in the code must be updated.

4 Notebook & Drive Setup

1. **Mount Google Drive**

```
from google.colab import drive
drive.mount('/content/drive')
```

2. **Load Libraries**

Import required packages:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from tensorflow.keras import layers, Model, Input
from lime import lime_tabular
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn import metrics
import time
import psutil
import os
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, Input, Permute, Multiply, Flatten, GRU, Lambda, RepeatVector
from tensorflow.keras import Model
import plotly.express as px
from datetime import timedelta
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, Flatten, GlobalAveragePooling1D, GlobalMaxPooling1D, Reshape, Conv1D, Add, Activ
from tensorflow.keras.models import Model
```

3. **Install Necessary Packages**

```
!pip install lime
```

```
Requirement already satisfied: lime in /usr/local/lib/python3.12/dist-packages (0.2.0.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (from lime) (3.10.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from lime) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from lime) (1.16.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from lime) (4.67.1)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.12/dist-packages (from lime) (1.6.1)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.12/dist-packages (from lime) (0.25.2)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (3.6)
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (11.3.0)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (2.37.2)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (2025.10.16)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (25.0)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.18->lime) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.18->lime) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (1.4.9)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib->lime) (1.17.0)
```

5 Data Loading

```
# Load main crime dataset (CSV)
df = pd.read_csv('/content/drive/MyDrive/Crime_hotspot/Dataset/Crimes_-_2019_to_Present_20250930.csv')

# Load weather dataset (Excel file)
weather = pd.read_excel('/content/drive/MyDrive/Crime_hotspot/Dataset/weather_data.xls')
```

6 Exploratory Data Analysis (EDA)

Matplotlib, seaborn and plotly are used to perform EDA to gain an insight into spatial and temporal trends.

6.1 Crime EDA

- Plots:
  - Top 10 locations as bar chart
  - 2D KDE heatmap using Latitude vs Longitude
  - Yearly crime counts
  - Arrest vs non-arrest by year
  - Top Location Description categories

These are produced based on the notebook; no further setup is needed than executing the EDA cells.

6.2 Weather EDA

- Plots:
  - Histogram and KDE of temperature
  - Time series of temperature and humidity
  - Average temperature by hour and by month
  - Scatter plot of temperature vs humidity coloured by wind speed

7 Data Pre-processing & Feature Engineering

7.1 Daily Crime Aggregation

1. **Produce counts of crimes and auxiliary features on a daily basis:**

```
filtered_df['crime_count'] = 1

# 1. Daily aggregation
daily_df = filtered_df.resample('D').agg({
    'crime_count': 'count',
    'Arrest': 'sum',
    'District': 'nunique'
})
```

2. **Time-based features:**

```
# 2. Time-based features
daily_df['day_of_week'] = daily_df.index.dayofweek
daily_df['is_weekend'] = daily_df['day_of_week'].apply(lambda x: 1 if x >= 5 else 0)
```

3. **Holiday indicator (US):**

```
# 3. Holiday indicator (US holidays)
us_holidays = holidays.UnitedStates()
daily_df['is_holiday'] = daily_df.index.to_series().apply(lambda x: 1 if x in us_holidays else 0)
```

4. **Rolling group-based averages:**

```
# 4. Group-based averages
daily_df['weekday_avg'] = (
    daily_df.groupby(daily_df.index.dayofweek)['crime_count']
    .transform(lambda x: x.shift(1).expanding().mean())   # cumulative mean by weekday
)

daily_df['month_avg'] = (
    daily_df.groupby(daily_df.index.month)['crime_count']
    .transform(lambda x: x.shift(1).expanding().mean())   # cumulative mean by month
)

daily_df['weekend_avg'] = (
    daily_df.groupby(daily_df.index.dayofweek)['crime_count']
    .transform(lambda x: x.shift(1).expanding().mean())   # weekend mean
)
daily_df.loc[daily_df['is_weekend'] == 0, 'weekend_avg'] = 0

daily_df[['weekday_avg', 'month_avg', 'weekend_avg']] = (
    daily_df[['weekday_avg', 'month_avg', 'weekend_avg']].fillna(method="bfill")
)
```

5. **Lag features:**

```
# Add lag features
daily_df['lag_1'] = daily_df['crime_count'].shift(1)    # previous day crime count
daily_df['lag_7'] = daily_df['crime_count'].shift(7)    # last week's same-day count
daily_df = daily_df.dropna()
```

7.2 Weather Aggregation & Discomfort Index

1. **Prepare weather time series:**

```
weather_df['datetime'] = pd.to_datetime(weather_df['datetime'], dayfirst=True)
weather_df.set_index('datetime', inplace=True)
weather_df = weather_df.sort_index()
```

2. **Handle missing values:**

- temperature and humidity: time interpolation.
- wind_speed: fill missing with 0

3. **Daily aggregation + Discomfort Index (DI):**

```
daily_weather = weather_df.resample('D').agg({
    'temperature': 'mean',
    'humidity': 'mean',
    'wind_speed': 'mean'
})
```

```
# Discomfort Index (DI) calculation using temperature + humidity formula
daily_weather['DI'] = 0.5 * (
    daily_weather['temperature'] + 61 +
    ((daily_weather['temperature'] - 68) * 1.2) +
    (daily_weather['humidity'] * 0.094)
)
```

7.3 Merge Crime & Weather Features

```
# Merging daily aggregated crime data with daily averaged weather data
merged_df = daily_df.join(daily_weather, how='left')

merged_df.interpolate(method="time", inplace=True)
```

7.4 Scaling & Sliding Windows

1. **Train/test split (time-based, 80/20).**
2. **Standardization:**

```
# Train/test split (time-wise) BEFORE scaling
n_total = len(daily_df)
split_row = int(0.8 * n_total)
train_df = merged_df.iloc[:split_row].copy()
test_df  = merged_df.iloc[split_row:].copy()

# Fit scalers on train only
feature_scaler = StandardScaler().fit(train_df[feature_cols].values)
target_scaler  = StandardScaler().fit(train_df[[target_col]].values)

# Apply the scaler to the whole dataset
features_scaled = feature_scaler.transform(merged_df[feature_cols].values)
target_scaled   = target_scaler.transform(merged_df[[target_col]].values)
```

3. **Create rolling windows for supervised learning:**

```
WINDOW_SIZE = 7
def create_rolling_windows_from_scaled(features_arr, target_arr, window_size=WINDOW_SIZE):
    X, y = [], []
    for i in range(len(features_arr) - window_size):
        X.append(features_arr[i:i+window_size])
        y.append(target_arr[i+window_size])   # next day target
    X = np.array(X)   # (n_samples, window_size, n_features)
    y = np.array(y)   # (n_samples, 1)
    return X, y

X_all, y_all = create_rolling_windows_from_scaled(features_scaled, target_scaled, window_size=WINDOW_SIZE)
```

4. **Final split:**

```
n_samples = X_all.shape[0]
split_sample = int(0.8 * n_samples) # 80% split index

X_train, X_test = X_all[:split_sample], X_all[split_sample:] # split features
y_train, y_test = y_all[:split_sample], y_all[split_sample:] # split labels

print("X_train.shape:", X_train.shape, "X_test.shape:", X_test.shape)
print("y_train.shape:", y_train.shape, "y_test.shape:", y_test.shape)
```

5. **Callbacks configuration:**

```
early_stop = EarlyStopping(
monitor='val_loss',
patience=5,
restore_best_weights=True,
verbose=1
)

lr_reduce = ReduceLROnPlateau(
monitor='val_loss',
factor=0.5,
patience=2,
verbose=1
)
```

8 Model Implementation

The input shape (window size, n features) and target (number of crimes) of all the models are the same.

8.1 Baseline LSTM

- Single LSTM layer (128 units, return_sequences=False)
- Several dense layers with ReLU activation and heavy dropout (0.7)
- Output: 1 neuron (linear regression)

Compile:

```
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

Train:

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=50,
    batch_size=64,
    verbose=1,
    callbacks=[early_stop, lr_reduce]
)
```

8.2 BiLSTM

- 2 layers of Bidirectional LSTM (128, then 64 units) stacked together.
- Dropout value at every recurrent layer (0.7)
- Flatten + dense layers (64 units) + dropout
- Output layer: 1 neuron

Training and compilation identical to LSTM.

8.3 Bidirectional GRU

build_gru_model() defines:

- Two stacked Bidirectional GRU layers (64 units each, return_sequences=True then False)
- Dropout (0.3) after each GRU layer
- Final dense output layer

Training: 20 epochs max with early stopping and LR scheduling.

8.4 BiLSTM with Cross-Attention

Time-based custom attention mechanism:

- Bidirectional LSTM (128 units, return_sequences=True) × 2
- The elements of the cross-attention layer were implemented using Permute, Dense, Lambda, RepeatVector, Multiply.
- Flatten + dense layers
- Output layer: 1 neuron

This model identifies critical time lapses and aspects that will lead to the prediction.

8.5 Spatio-Temporal CBAM Attention Model

CBAM block combines channel attention and spatio-temporal attention:

1. **Backbone network:**
    - Four stacked Bidirectional LSTMs (128, 128, 64, 64 units) with dropout 0.7 after each.

2. **CBAM block:**

   - Channel attention using GlobalAveragePooling 1D, GlobalMaxPooling 1D, shared dense layers, as well as, sigmoid gating.
   - Spatial (temporal) attention using mean/max along feature dimension along with Conv1D having kernel size 7.
   - Element-wise multiplication with the original feature map.

3. **Classifier head:**

   - Flatten
   - Two dense layers (64 units, ReLU) with dropout
   - Output layer of 1 neuron in the end.

Compile and train as before. This is the primary model to be proposed in the project.

9 Evaluation, Forecasting and Explainability

9.1 Regression Metrics

Metrics:

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

Example:

```
start_exec = time.time()  # start measuring execution time

predicted = model.predict(X_test)  # generate predictions on test set

end_exec = time.time()
execution_time = end_exec - start_exec  # total execution time

print('Mean Squared Error(MSE):', metrics.mean_squared_error(y_test, predicted))
print('Root Mean Squared Error(RMSE):', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
print('Mean Absolute Error(MAE):', metrics.mean_absolute_error(y_test, predicted))
```

3. Plot Actual vs Predicted using Plotly line plots.

```
fig = px.line(actualpred_df.reset_index(), x=actualpred_df.index, y=actualpred_df.columns, title='Actual vs Predicted')
fig.show()
```

9.2 Efficiency Metrics

For each model:

- Execution time for batch prediction
- Latency per prediction
- Throughput – predictions per second for a batch.
- Memory usage

9.3 7-Day Forecasting Procedure

For each trained model:

1. Extract the last WINDOW_SIZE=7 days from features_scaled.
2. Iteratively predict one day ahead, append prediction into the window, and roll the window forward.
3. Inverse-transform predictions to get the numbers of crimes per day in original scale.
4. Create a forecast_df indexed by future dates (last_date + 1 … +7) and plot predicted counts.
5. Alternatively, merge the 20 final days of merged with the 7 days prediction to have a joint visual representation.

9.4 LIME Explainability (for BiLSTM + CBAM)

```
#  CONFIG
explain_index = -1
n_features_to_show = 20
n_samples_in_lime = 5000


# Sanity checks
assert 'X_train' in globals() and 'X_test' in globals(), "X_train / X_test must exist"
assert X_train.ndim == 3 and X_test.ndim == 3, "X_train and X_test should be (N, window, n_features)"
window_size = X_train.shape[1]
n_features = X_train.shape[2]
assert window_size * n_features == X_train.reshape(len(X_train), -1).shape[1]

# Build feature names for each timestep and feature
feat_names = []
for t in range(window_size):
    lag = window_size - t
    for f in feature_cols:
        feat_names.append(f"{f}_t-{lag}")

# Flatten training and test data for LIME
X_train_flat = X_train.reshape(len(X_train), -1)
X_test_flat = X_test.reshape(len(X_test), -1)


# Define predict function for LIME
def lime_predict_fn(flat_batch):
    """
    Predict function for LIME
    flat_batch: np.array shape (B, window*n_features)
    returns: np.array shape (B,) of model outputs
    """
    X_batch = flat_batch.reshape((flat_batch.shape[0], window_size, n_features)).astype(np.float32)
    preds = model.predict(X_batch, verbose=0)
    preds = np.array(preds).reshape(-1)
    return preds

#  Build LIME explainer
explainer = lime_tabular.LimeTabularExplainer(
    training_data=X_train_flat.astype(np.float32),
    feature_names=feat_names,
    mode='regression',
    discretize_continuous=False,
)
```

LIME determines what past days and features have the biggest impact on the forecasted amount of crime (e.g. recent crime_count and lag_7 features, weekend indicators, etc.).

10 Reproduction Steps (Summary)

To fully reproduce the study:

1. Open Google Colab.
2. Install Google Drive and make sure that dataset files are located in /content/drive/MyDrive/Crime_hotspot/Dataset/.
3. Installation of additional packages (lime) and all libraries importation.
4. Load crime data and weather data.
5. Run EDA cells (optional but recommended).
6. Execute all preprocessing steps:
    - Filter to Community Area 28 & THEFT
    - Handle missing spatial values
    - Aggregate daily crime and calculate calendar/holiday/lag features
    - Aggregate weather data and compute DI
    - Merge crime and weather features
    - Standardise features and create rolling windows
7. Configure callbacks (EarlyStopping, ReduceLROnPlateau).
8. Train models in the following order:
    - LSTM
    - BiLSTM
    - BiGRU
    - BiLSTM + Attention
    - BiLSTM + CBAM
9. For each model, run the evaluation cells: metrics, plots, efficiency calculation.
10. Run forecasting cells to generate the 7-day predictions and visualisations.
11. For the CBAM model, run the LIME section and can see the explanation.