National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Revanth Reddy Mallidi
Student ID: 23408308

School of Computing
National College of Ireland

Supervisor: Dr.Muhammad Asif Razzaq

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Revanth Reddy Mallidi

Student ID: 23408308

Programme: Master's in Data Analytics **Year:** 2025-2026

Module: Research Practicum

Lecturer: Dr.Muhammad Asif Razzaq

Submission Due Date: 28/01/2026

Project Title: Crime Forecasting Using a Hybrid BiGRU–Transformer Neural Architecture: A Deep Learning Approach on Austin Crime Data

Word Count: 1399 **Page Count: 11**

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: **Revanth Reddy Mallidi**

Date: **28-01-2026**

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

Revanth Reddy Mallidi
23408308

1. Introduction

The Hybrid BiGRU–Transformer Forecasting System, which is intended to forecast daily crime incidents in Austin, is fully documented in this configuration manual along with its architecture, workflow, setup, and execution details. To produce precise time-series forecasts, the system applies sophisticated deep learning structures, such as recurrent neural networks and attention processes, to past police-recorded incidents. Future researchers or practitioners can easily replicate, configure, and deploy this manual.

2. Overview of the Data

From unprocessed Austin crime data, the technology builds an organized forecasting pipeline. It performs:

- Preprocessing of data
- Lag generation and feature engineering
- Generation of time-series windows
- Hyperparameter control and model training
- Five neural network designs are evaluated.
- Comparison of efficiency and performance
- Outcomes of prediction and visualization

The system's modular architecture allows each component to function independently, which simplifies experimentation and troubleshooting.

3. Libraries Used

- numpy: Used for numerical operations, arrays.
- pandas: Handles data loading, preprocessing, cleaning, grouping, and time-series manipulation.
- matplotlib.pyplot: Used for visualizations such as line plots, bar charts, trend graphs, and heat maps
- seaborn: Creates visualizations of data
- os: Provides interaction with the operating system, such as file path handling, directory access, and environment variable management.

- MinMaxScaler: Scales features into 0–1 range for stable neural network training.
- Metrics: Calculates MSE, RMSE, MAE, R^2, and other evaluation metrics.
- TensorFlow & Keras: Primary deep learning architecture of creating and training model.
- Time: Measures execution time, latency, and throughput of predictions.

```
import pandas as pd   # Import pandas for data manipulation and analysis
import numpy as np   # Import NumPy for numerical operations
import matplotlib.pyplot as plt   # Import Matplotlib for data visualization
import seaborn as sns   # Import Seaborn for advanced statistical visualizations
from sklearn.preprocessing import MinMaxScaler # Import MinMaxScaler for feature scaling
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import Input, LSTM, Dropout, Flatten, Dense
import time
from sklearn import metrics
import psutil
import os
from tensorflow.keras.layers import MultiHeadAttention, LayerNormalization, Add
import tensorflow as tf
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import GRU, Bidirectional
```

Figure 1: Importing Libraries

4. Environment Setup

4.1 Hardware Requirements

- Recommended GPU
- RAM: 16 GB minimum
- Storage: 5–10 GB free space

4.2 Software Requirements

- Python 3.11
- TensorFlow, Keras
- Pandas & NumPy
- Scikit-Learn
- Matplotlib, Seaborn

5. Dataset Description

Dataset: Crimes in the City of Austin[1]

The main fields that are used from the dataset

Column	Description
Occurred Date	Date of reported crime
Occurred Time	Time of the incident
Council District	Austin district unique code
Category Description	Crime Type
Family Violence	Binary domestic violence indicator
Highest Offense Description	Most occured offense type
Location Type	location of the crime
Incident Number	Unique crime ID

6. Data Preprocessing

Dataset is loaded by using pd.read_csv

```
df = pd.read_csv('/content/drive/MyDrive/crime_prediction/dataset/Crime_Reports.csv')
```

Figure 2: Importing the Dataset

Time-series models require proper dates,
The figure 3 shows

dropna(subset=\['Occurred Date']): Eliminates all rows without the crime date. **pd.to_datetime(..., errors='coerce'**): converts the Occurred Date field to a suitable datetime type.
Actually, Family_violence column is in text (Y,N)-for model to understand I converted categorical value to numeric value 1,0(1-Yes,0-No)

```
df = df.dropna(subset=['Occurred Date'])  # Drop rows where the 'Occurred Date' column has missing (NaN) values

# Convert to datetime
df['Occurred Date'] = pd.to_datetime(df['Occurred Date'], errors='coerce')  # Convert 'Occurred Date' column to datetime format, coercing invalid values to NaT

# Convert flags to binary
df['Family Violence_Flag'] = df['Family Violence'].map({'Y': 1, 'N': 0, 'n': 0})  # Map 'Y' to 1 and 'N'/'n' to 0 to create a binary 'Family Violence_Flag' column
```

Figure 3: Cleaning the data

Figure 4 shows,
1. Stores the year in **df['year']** after extracting it from the **'Occurred Date'** column.
2. Stores the month in **df['month']** after extracting it from the **'Occurred Date'** column.
3. Stores the day of the week (0–6) in **df['day_of_week']** after extracting it from **'Occurred Date'.**
4. Extracts the hour and stores it in **df['hour']** after converting **'Occurred Date Time'** to datetime.

[1] https://catalog.data.gov/dataset/crime-reports-bf2b7

```
df['year'] = df['Occurred Date'].dt.year   # Extract the year from 'Occurred Date' and store it in a new column 'year'
df['month'] = df['Occurred Date'].dt.month   # Extract the month from 'Occurred Date' and store it in a new column 'month'
df['day_of_week'] = df['Occurred Date'].dt.dayofweek   # Extract the day of the week (0=Monday, 6=Sunday) and store it in 'day_of_week'
df['hour'] = pd.to_datetime(df['Occurred Date Time']).dt.hour   # Convert 'Occurred Date Time' to datetime and extract the hour component
```

Figure 4: Date and time feature extraction

7. Exploratory Data Analysis (EDA)

To better understand the underlying patterns in the crime dataset, many visualizations highlig hting the temporal, spatial, and categorical elements of crime incidence were developed.
These visual insights are an essential component of the exploratory data analysis (EDA) proc ess and help pinpoint areas that require attention in subsequent modeling and decision- making phases.

8. Feature Engineering

from figure 5..., The code filters the dataset to include only theft incidents that occurred within Council District 4. Following these filters, the data is grouped by the "Occurred Date" to determine the total number of family-violence-related cases for the same period as well as the number of theft incidents reported each day.
To maintain the natural time order, the resulting daily dataset is then sorted chronologically. Several time-based features, such as the year, month, day, day of the week, and week number of the year, are extracted from the date index to facilitate additional analysis and modelling.

```
district_filter = [4.0]   # Define the list of council districts to filter (e.g., district 4.0)
category_filter = ["Theft"]   # Define the list of crime categories to filter (e.g., 'Theft')

df = df[
    (df['Council District'].isin(district_filter)) &   # Filter rows where 'Council District' matches the specified list
    (df['Category Description'].isin(category_filter))   # Filter rows where 'Category Description' matches the specified list
]   # Apply both filters to retain only matching records


daily_df = df.groupby('Occurred Date').agg({   # Group the dataset by 'Occurred Date' and aggregate statistics
    'Incident Number': 'count',   # Count the number of incidents per day
    'Family Violence_Flag': 'sum'   # Sum up the family violence flags per day
}).rename(columns={'Incident Number': 'crime_count'})   # Rename 'Incident Number' to 'crime_count' for clarity

daily_df = daily_df.sort_index()   # Sort the DataFrame by date to maintain chronological order

#  Time-based feature engineering
daily_df['year'] = daily_df.index.year   # Extract the year from the date index
daily_df['month'] = daily_df.index.month   # Extract the month from the date index
daily_df['day'] = daily_df.index.day   # Extract the day from the date index
daily_df['day_of_week'] = daily_df.index.dayofweek   # Extract the day of the week (0=Monday, 6=Sunday)
daily_df['weekofyear'] = daily_df.index.isocalendar().week.astype(int)   # Extract the week number of the year
```

Figure 5: Data filtering and feature engineering

From figure 6…., To capture how past crime values affect current crime levels, I created lag characteristics. I created three new columns—lag_1, lag_7, and lag_30—that show the number of crimes that occurred one day, seven days, and thirty days prior, respectively.
Now the model can identify short-term, weekly, and monthly patterns in crime trends by these lag features.
The code defines features_col, which contains all predictors used for training, including temporal features (year, month, day, day of week, week of year) and the newly formed lag features, after generating the lag values and displaying the updated list of columns. Finally, I assigned target_col as the crime count.

```
#  Lag and rolling features
daily_df['lag_1'] = daily_df['crime_count'].shift(1)   # Create a feature for previous day's crime count (1-day lag)
daily_df['lag_7'] = daily_df['crime_count'].shift(7)   # Create a feature for crime count from the same day last week (7-day lag)
daily_df['lag_30'] = daily_df['crime_count'].shift(30)   # Create a feature for crime count from 30 days ago (monthly lag)
```

```
daily_df.columns
```

```
Index(['crime_count', 'Family Violence_Flag', 'year', 'month', 'day',
       'day_of_week', 'weekofyear', 'lag_1', 'lag_7', 'lag_30'],
      dtype='object')
```

```
features_col = ['crime_count', 'Family Violence_Flag', 'year', 'month', 'day',   # Define the list of feature columns for model training
       'day_of_week', 'weekofyear', 'lag_1', 'lag_7', 'lag_30']   # Include temporal and lag features
target_col = ['crime_count']   # Define the target variable column for prediction
```

Figure 6: Creating Lag Features

Figure 7 shows the relationship between each numerical feature and the target variable is visualized in this stage.
We can see how strongly each attribute is related to crime_count by using the corr() function, which calculates pairwise correlations between numerical variables.

```
numeric_merged_df = daily_df.select_dtypes(include=np.number)   # Select only numeric columns from 'daily_df' for correlation analysis

corr_matrix = numeric_merged_df.corr()   # Compute the correlation matrix between all numeric features
plt.figure(figsize=(15, 8))   # Set the figure size for the heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)   # Plot the correlation heatmap with annotations and color gradien
plt.title('Correlation Matrix')   # Set the title of the heatmap
plt.show()   # Display the heatmap
```

Figure 7: Generating Correlation matrix

9. Data Scaling

The figure 8, I used MinMax scaler to scale the target variable and the input features to 0–1 range. It then delivers clean, scaled DataFrames that are prepared for time-series windowing or machine learning.

```
from sklearn.preprocessing import MinMaxScaler
daily_df = daily_df.dropna()   # Remove rows with missing values to ensure data consistency before scaling

feature_scaler = MinMaxScaler().fit(daily_df[features_col].values)   # Fit the StandardScaler on feature columns to compute mean and standard deviation
target_scaler  = MinMaxScaler().fit(daily_df[target_col].values)   # Fit another StandardScaler on target column for consistent scaling

# Apply the scaler to the whole dataset (useful so windowing uses consistent scaling)
features_scaled = feature_scaler.transform(daily_df[features_col].values)   # Standardize the feature values (mean=0, std=1)
target_scaled   = target_scaler.transform(daily_df[target_col].values)   # Standardize the target values

features_scaled = pd.DataFrame(features_scaled, columns=features_col)   # Convert scaled feature array back to a DataFrame with original column names
target_scaled = pd.DataFrame(target_scaled, columns=target_col)   # Convert scaled target array back to a DataFrame with original column names
```

Figure 8: Scaling the dataset using MinMaxScaler

10. Window Construction

Crime counts are grouped by relevant dimensions and transformed through feature engineering and scaling.
Time-series windows (size = 15) are generated to convert the data into supervised learning sequences.
To prepare input-output samples for model training, each window is paired with the target value for the following day.

```python
# WINDOW ROLLING
WINDOW_SIZE = 15   # Define the size of the rolling window (e.g., 7 days for weekly pattern learning)

def create_rolling_windows_from_scaled(features_arr, target_arr, window_size=WINDOW_SIZE):
    X, y = [], []   # Initialize lists to hold feature windows and corresponding target values
    for i in range(len(features_arr) - window_size):   # Loop through the dataset, creating rolling windows
        X.append(features_arr[i:i+window_size])   # Append a sequence of features spanning 'window_size' days
        y.append(target_arr[i+window_size])   # Append the target value for the next day after the window
    X = np.array(X)   # Convert list of feature windows to a NumPy array with shape (samples, window_size, features)
    y = np.array(y)   # Convert list of targets to a NumPy array with shape (samples, 1)
    return X, y   # Return feature and target arrays for model training
```

Figure 9: Window Rolling for Time-Series Data

11. Train–Test Split

The dataset is split chronologically into 80% training and 20% testing to maintain temporal order. This ensures realistic model evaluation on unseen future data.

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

# Train-test split (chronological)
train_size = int(len(daily_df) * 0.8)   # Define training set size as 80% of the total data to maintain time order

X_train = X_all[:train_size]   # Use the first 80% of windows for training
y_train = y_all[:train_size]   # Corresponding targets for the training set

X_test = X_all[train_size:]   # Remaining 20% of windows for testing
y_test = y_all[train_size:]   # Corresponding targets for the testing set
```

Figure 10: splitting the data

12. Model Construction

Multiple architectures—LSTM, MLP, GRU, BiGRU, and a BiGRU-Transformer hybrid—are built and trained.

12.1 LSTM Model

Figure 11 shows implementation of a 128-unit LSTM model, and dense layers and dropout for regularisation.
The Adam optimiser for regression is used to compile the model, and the evaluation metrics are MAE and MSE loss.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, LSTM, Dropout, Flatten, Dense

model = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),  # Input layer specifying time steps and features (1 feature here)
    LSTM(128, activation="relu", return_sequences=True),  # LSTM layer with 128 units, outputs last timestep only
    Dropout(0.7),
     # Flatten layer
    Flatten(),
    # Dense layers
    Dense(64, activation='relu'),
    Dropout(0.7),
    Dense(64, activation='relu'),
    Dropout(0.7),
    Dense(64, activation="relu"),       # Fully connected hidden layer with 64 neurons and ReLU activation
    Dense(32, activation="relu"),       # Another Dense layer with 32 neurons and ReLU
    Dense(1, activation="linear")       # Output layer with 1 neuron for regression (linear activation)
])

# Compile the model with Adam optimizer, mean squared error loss, and mean absolute error metric
model.compile(optimizer='adam', loss='mse',metrics=['mae'])
```

Figure 11: LSTM Model Architecture

12.2 MLP Model

This figure 12 shows the MLP model, which flattens the input and then routes it through dense layers with dropout for regularization.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout, Input

model = Sequential([   # Initialize a sequential model
    Input(shape=(X_train.shape[1], X_train.shape[2])), # Explicit Input layer
    Flatten(),   # Flatten 3D input (window, features) into 2D
    Dense(128, activation='relu'),   # First hidden layer with 128 neurons and ReLU activation
    Dropout(0.2),   # Dropout to prevent overfitting
    Dense(64, activation='relu'),   # Second hidden layer with 64 neurons
    Dropout(0.2),   # Dropout again for regularization
    Dense(1)   # Output layer for regression (predicting crime count)
])

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

Figure 12: MLP Model Architecture

12.3 GRU Model

Figure 13 illustrates a stacked GRU model that learns patterns from the time-series data by using several GRU layers with dropout. The number of crimes for the following day is predicted by the last dense layer.

```
# Model 2: GRU
from tensorflow.keras.layers import GRU, Bidirectional

def build_gru_model(window_size, n_features):
    model = Sequential()
    model.add(GRU(128, return_sequences=True, input_shape=(window_size, n_features)))
    model.add(Dropout(0.7))
    model.add(GRU(128, return_sequences=True))
    model.add(Dropout(0.7))
    model.add(GRU(64, return_sequences=True))
    model.add(Dropout(0.7))
    model.add(GRU(64, return_sequences=True))
    model.add(Dropout(0.7))
    model.add(GRU(32, return_sequences=False))
    model.add(Dropout(0.7))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

model = build_gru_model(WINDOW_SIZE, X_train.shape[2])
model.summary()
```

Figure 13: GRU Model Architecture

12.4 Bi-GRU Model

This figure 14 presents a Bidirectional GRU model that processes the time-series data both forward and backward to reduce the time-series data to capture more temporal patterns. Learning and overfitting are enhanced by several layers of stacked BiGRU with dropout, and the final dense layer is used to make a prediction.

```
# Model 2: GRU (Bidirectional)
from tensorflow.keras.layers import GRU, Bidirectional

def build_gru_model(window_size, n_features):
    model = Sequential()
    model.add(Bidirectional(GRU(256, return_sequences=True, input_shape=(window_size, n_features))))
    model.add(Dropout(0.7))
    model.add(Bidirectional(GRU(128, return_sequences=True)))
    model.add(Dropout(0.7))
    model.add(Bidirectional(GRU(64, return_sequences=True)))
    model.add(Dropout(0.7))
    model.add(Bidirectional(GRU(64, return_sequences=True)))
    model.add(Dropout(0.7))
    model.add(Bidirectional(GRU(32, return_sequences=False))) # Changed to False
    model.add(Dropout(0.7))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

model = build_gru_model(WINDOW_SIZE, X_train.shape[2])
model.build(input_shape=(None, WINDOW_SIZE, X_train.shape[2]))
model.summary()
```

Figure 14: BiGRU Model Architecture

12.5 Bi-GRU Transformer Attention

This figure 15 shows the hybrid BiGRU-Transformer Attention that integrates temporal learning in BiGRU with multi-head self-attention to detect long-range patterns more effectively. It has layer normalization, residual connections and feed-forward layers to enhance stability and accuracy of prediction**.**

```
from tensorflow.keras.layers import MultiHeadAttention, LayerNormalization, Add
import tensorflow as tf
from tensorflow.keras.models import Model, Sequential

def build_gru_transformer_model(window_size, n_features, num_heads=4, key_dim=32):
    inputs = Input(shape=(window_size, n_features))

    # Step 1: GRU for temporal encoding
    x = Bidirectional(GRU(64, return_sequences=True))(inputs)
    x = Dropout(0.3)(x)

    # Step 2: Transformer-style Multi-Head Attention
    attn_output = MultiHeadAttention(num_heads=num_heads, key_dim=key_dim)(x, x)
    x = Add()([x, attn_output])  # Residual connection
    x = LayerNormalization(epsilon=1e-6)(x)

    # Step 3: Feed-forward + pooling
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.3)(x)
    x = tf.keras.layers.GlobalAveragePooling1D()(x)

    # Step 4: Output
    outputs = Dense(1)(x)

    model = Model(inputs, outputs)
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

# Build Model 2
model = build_gru_transformer_model(WINDOW_SIZE, X_train.shape[2])
model.summary()
```

Figure 15: BiGRU Transformer Attention Model Architecture

13. Evaluation

Figure 16 shows that testing forecasting models were evaluated on the test set that was held-out with a mix of accuracy-based and efficiency-based measures. Mean Squared error (MSE), Root mean squared error (RMSE), mean absolute error (MAE), Symmetric mean absolute percentage error (SMAPE) and coefficient of determination (R2) was calculated using the inverse scaled predictions against the actual number of crime per day. Along with such measures of errors, three runtime measures were also taken: average latency per single prediction, throughput of predictions per second when scoring the entire test set, and the overall amount of time used to compute all test forecasts.

```
#   MODEL PERFORMANCE METRICS
start_exec = time.time()

predicted = model.predict(X_test)
real_value = target_scaler.inverse_transform(y_test.reshape(-1, 1))
predicted = target_scaler.inverse_transform(predicted)


end_exec = time.time()
execution_time = end_exec - start_exec
bigru_mse = metrics.mean_squared_error(real_value, predicted)
bigru_rmse =  np.sqrt(metrics.mean_squared_error(real_value, predicted))
bigru_mae = metrics.mean_absolute_error(real_value, predicted)
def smape(y_true, y_pred):
    return 100 * np.mean(
        2 * np.abs(y_true - y_pred) /
        (np.abs(y_true) + np.abs(y_pred) + 1e-8)
    )

mape = smape(real_value, predicted)
residuals = y_test - predicted
print('Mean Squared Error(MSE):', metrics.mean_squared_error(real_value, predicted))
print('Root Mean Squared Error(RMSE):', np.sqrt(metrics.mean_squared_error(real_value, predicted)))
print('R-squared(R2-Score):', metrics.r2_score(real_value, predicted))
print('Mean Absolute Error(MAE):', metrics.mean_absolute_error(real_value, predicted))
print('Mean Absolute Percentage Error(MAPE):', str(round(mape))+'%')
```

Figure 16: Model Performance Metrics

14. Conclusion

The current configuration document outlines the overall architecture and process of the crime forecasting system, involving data preparation, feature engineering, model building and evaluation processes. There were five different neural structures developed and compared to each other, during which the capacity to learn both temporal dynamics and long-range relationships was the strongest in the hybrid BiGRU -Transformer model. The recorded methodology ensures complete reproducibility of the system and can be extended to further researchers, therefore, extending the applications to increased usage in crime prediction and timeseries forecasting tasks.

15. References

Dataset Source: https://catalog.data.gov/dataset/crime-reports-bf2b7