Configuration Manual for Enhancement of Anomaly Detection in Time Series Data with Hybrid Deep Learning Approach Methodologies

MSc Research Project
Msc in data analytics

Wonderpaul pallikona
Student ID: X23218142

School of Computing
National College of Ireland

Supervisor:Musfira Jilani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……wonderpaul pallikona. ………………
Student ID:	…………………X23218142………………
Programme:	…………Msc in Data analytics………………… **Year:** ……2024-2025
Module:	……………Msc Research project………..………
Lecturer:	…………………Musfira Jilani……………….………
Submission Due Date:	…………29-01-2025…………………….………
Project Title:	…………Enhancement of anomaly detection in time series data with hybrid deep learning approach methodologies….………
Word Count:	……936………… **Page Count:** ……10……………

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: ……………………Wonderpaul pallikona……………………

Date: ……………29-01-2025……………………………………………

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 for Enhancement of Anomaly Detection in Time Series Data with Hybrid Deep Learning Approach Methodologies

Forename Surname
Student ID:

1. Introduction

This configuration manual provides the necessary instructions to set up and execute the hybrid deep learning approach for time series anomaly detection, as described in the research. The methodology incorporates LSTM, RNN, and GRU Autoencoders, alongside the Isolation Forest algorithm for anomaly detection in a New York City taxi dataset. The steps outlined in this manual will guide you through the configuration, installation, and execution process for the entire workflow.

2. System Requirements

To run the anomaly detection models, ensure that your system meets the following hardware and software requirements:

Hardware Requirements:
- CPU: At least 4 cores for model training
- RAM: 8 GB or more for smooth execution
- Disk Space: 10 GB minimum for dataset and model storage

Software Requirements:
- Operating System: Linux, macOS, or Windows
- Python 3.7 or higher
- Required Python Libraries:
- numpy==1.23.1
- pandas==1.5.3
- scikit-learn==1.2.0
- tensorflow==2.10.0
- keras==2.10.0
- matplotlib==3.6.0
- seaborn==0.11.2
- scipy==1.9.3
- sklearn==0.0
- openpyxl==3.0.10

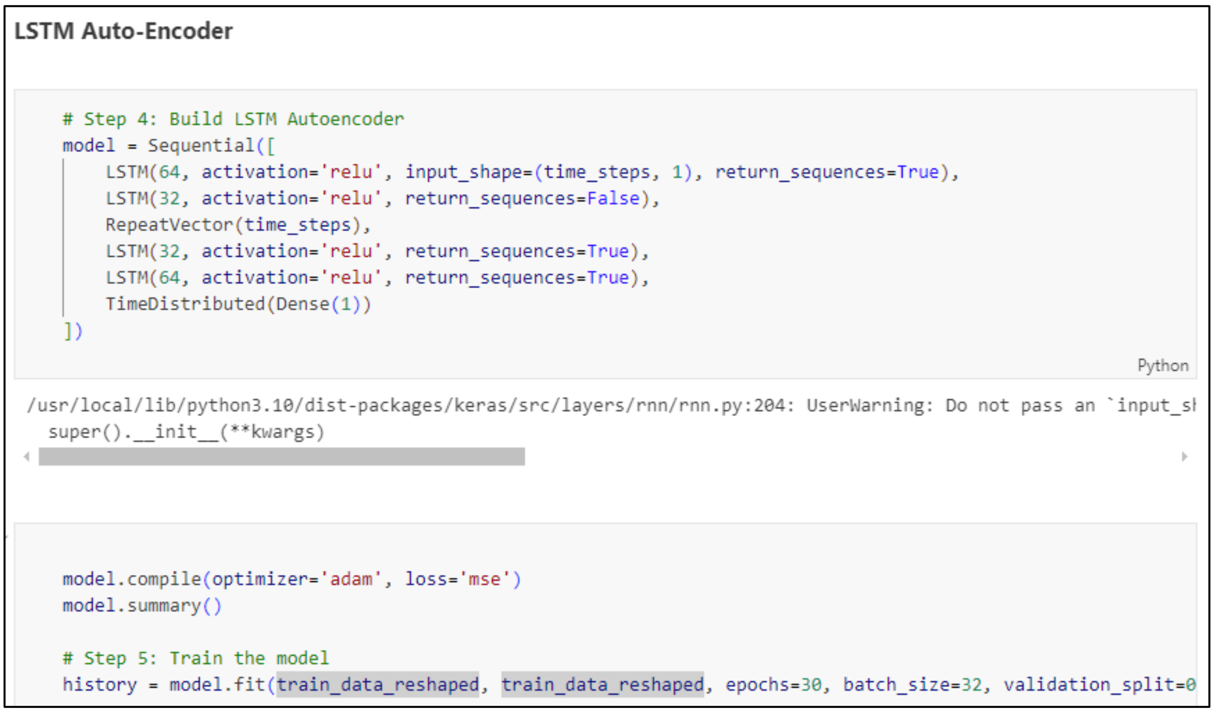
3. Dataset Preparation
The dataset used in this research is a New York City taxi dataset containing time series data.

Steps for preparing the dataset:
- Download the dataset: Where the dataset is sourced from the kaggle for Numenta Anamaly Benchmark (NAB)

Preprocessing:
- Ensure the timestamp column is in datetime format.
- Normalize the 'value' column using MinMaxScaler (scikit-learn).
- Create time sequences for model training using a rolling window approach. The size of the window can be adjusted based on the nature of the dataset.

4. Model Configuration
There are four main models in this research: LSTM Autoencoder, GRU Autoencoder, RNN Autoencoder, and Isolation Forest.

4.1 LSTM Autoencoder Configuration
- Model Architecture:
  - Input Layer: A sequence of time steps.
  - Encoder: LSTM layers with decreasing units.
  - Decoder: LSTM layers to reconstruct the input.
- Compilation:
  - Optimizer: Adam
  - Loss function: Mean Squared Error (MSE)
- Training:
  - Epochs: 30
  - Batch Size: 32


LSTM Auto-Encoder

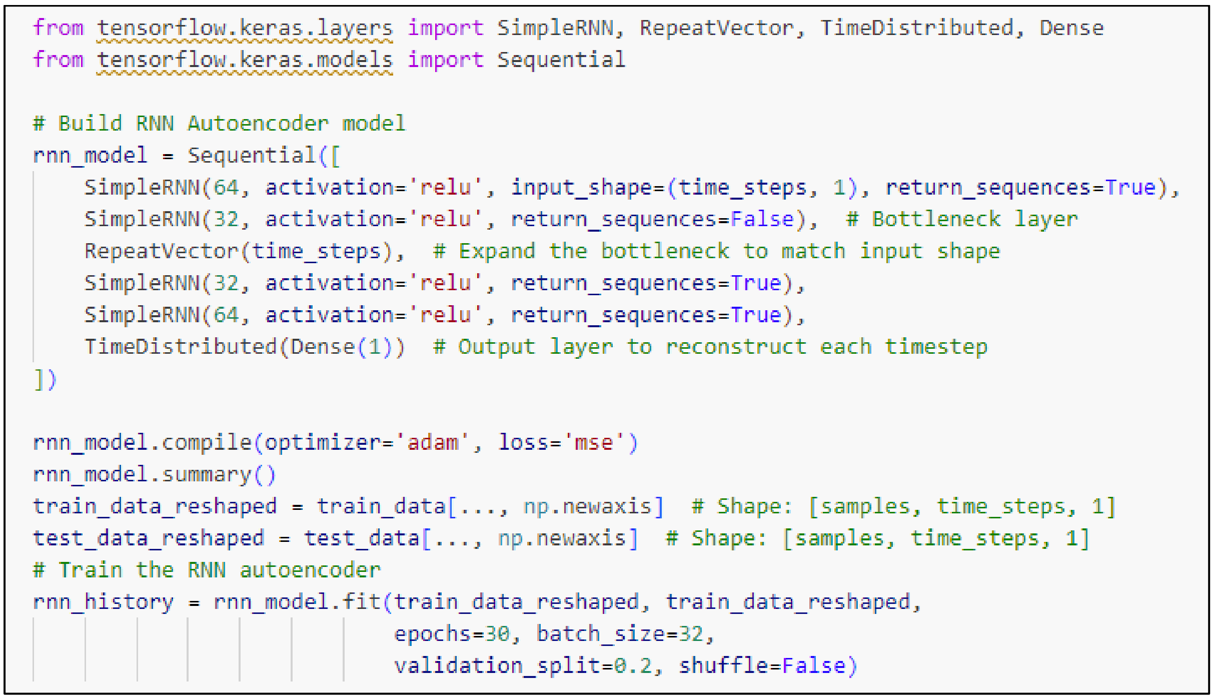
```
# Step 4: Build LSTM Autoencoder
model = Sequential([
    LSTM(64, activation='relu', input_shape=(time_steps, 1), return_sequences=True),
    LSTM(32, activation='relu', return_sequences=False),
    RepeatVector(time_steps),
    LSTM(32, activation='relu', return_sequences=True),
    LSTM(64, activation='relu', return_sequences=True),
    TimeDistributed(Dense(1))
])
```
Python

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_sh
  super().__init__(**kwargs)
```

```
model.compile(optimizer='adam', loss='mse')
model.summary()

# Step 5: Train the model
history = model.fit(train_data_reshaped, train_data_reshaped, epochs=30, batch_size=32, validation_split=0
```

4.2 RNN Autoencoder Configuration

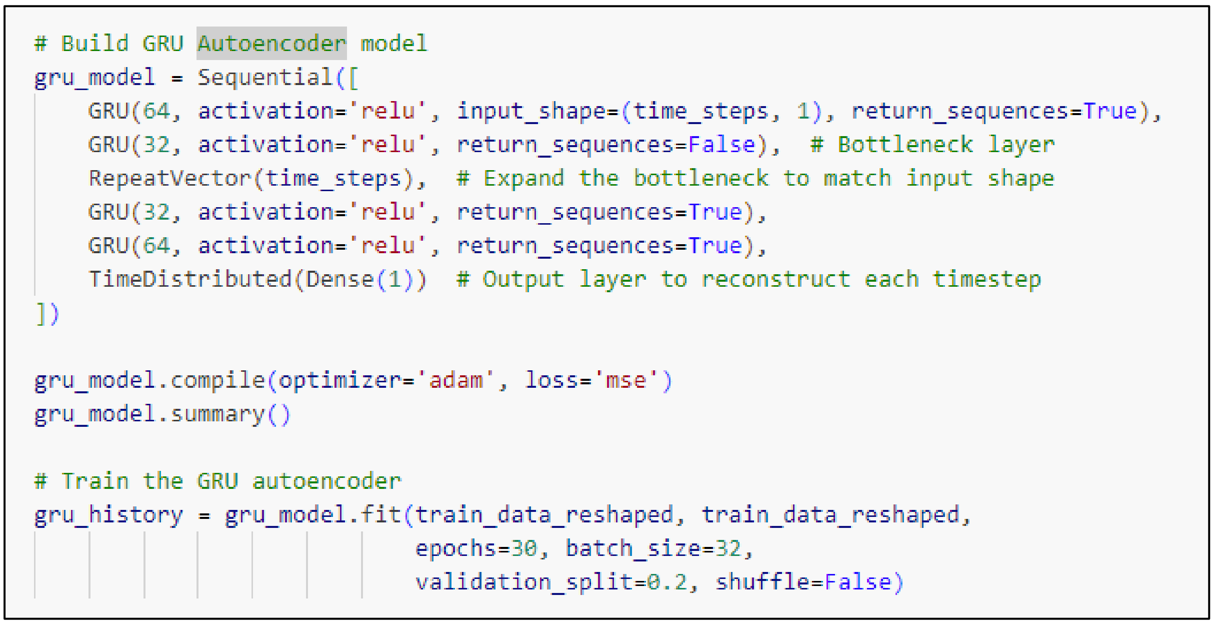
- Model Architecture:
  - Similar to the LSTM Autoencoder but using standard RNN layers.
- Compilation:
  - Optimizer: Adam
  - Loss function: Mean Squared Error (MSE)
- Training:
  - Epochs: 30
  - Batch Size: 32



```python
from tensorflow.keras.layers import SimpleRNN, RepeatVector, TimeDistributed, Dense
from tensorflow.keras.models import Sequential

# Build RNN Autoencoder model
rnn_model = Sequential([
    SimpleRNN(64, activation='relu', input_shape=(time_steps, 1), return_sequences=True),
    SimpleRNN(32, activation='relu', return_sequences=False),  # Bottleneck layer
    RepeatVector(time_steps),  # Expand the bottleneck to match input shape
    SimpleRNN(32, activation='relu', return_sequences=True),
    SimpleRNN(64, activation='relu', return_sequences=True),
    TimeDistributed(Dense(1))  # Output layer to reconstruct each timestep
])

rnn_model.compile(optimizer='adam', loss='mse')
rnn_model.summary()
train_data_reshaped = train_data[..., np.newaxis]  # Shape: [samples, time_steps, 1]
test_data_reshaped = test_data[..., np.newaxis]  # Shape: [samples, time_steps, 1]
# Train the RNN autoencoder
rnn_history = rnn_model.fit(train_data_reshaped, train_data_reshaped,
                            epochs=30, batch_size=32,
                            validation_split=0.2, shuffle=False)
```

4.3 GRU Autoencoder Configuration

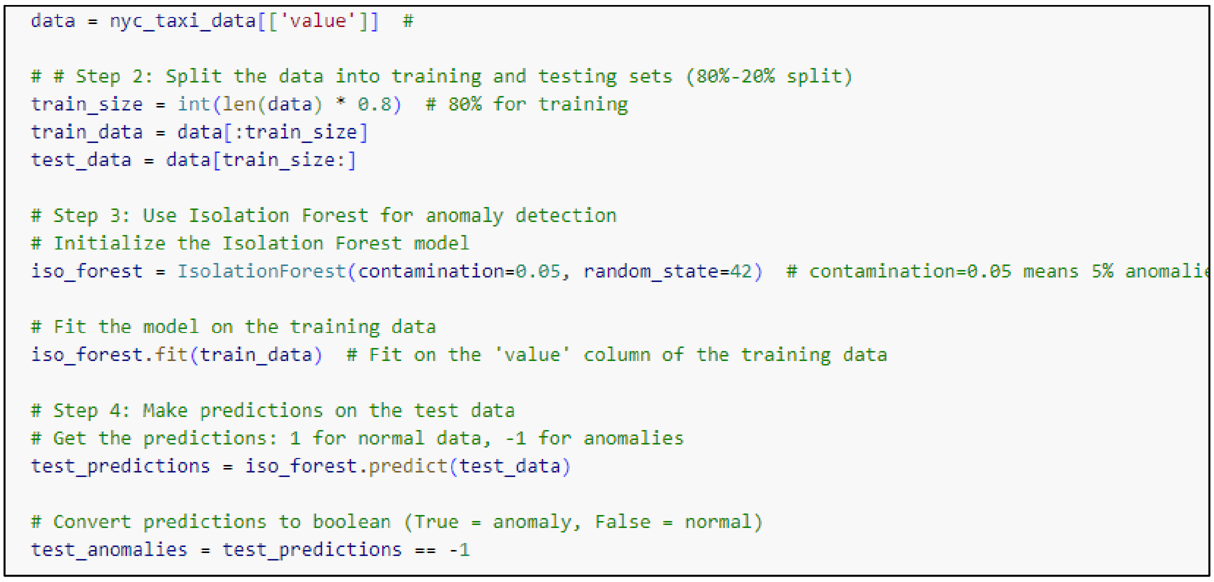
- Model Architecture:
  - Similar to the LSTM Autoencoder but using GRU layers.
- Compilation:
  - Optimizer: Adam
  - Loss function: Mean Squared Error (MSE)
- Training:
  - Epochs: 30
  - Batch Size: 32



```
# Build GRU Autoencoder model
gru_model = Sequential([
    GRU(64, activation='relu', input_shape=(time_steps, 1), return_sequences=True),
    GRU(32, activation='relu', return_sequences=False),  # Bottleneck layer
    RepeatVector(time_steps),  # Expand the bottleneck to match input shape
    GRU(32, activation='relu', return_sequences=True),
    GRU(64, activation='relu', return_sequences=True),
    TimeDistributed(Dense(1))  # Output layer to reconstruct each timestep
])

gru_model.compile(optimizer='adam', loss='mse')
gru_model.summary()

# Train the GRU autoencoder
gru_history = gru_model.fit(train_data_reshaped, train_data_reshaped,
                            epochs=30, batch_size=32,
                            validation_split=0.2, shuffle=False)
```

4.4 Isolation Forest Configuration

- Model Setup:
  - Contamination: 5% (set based on expected percentage of anomalies in the data).
- Training:
  - Fit the model on the preprocessed training data.
  - Predict anomalies on the test data.



```
data = nyc_taxi_data[['value']]  #

# # Step 2: Split the data into training and testing sets (80%-20% split)
train_size = int(len(data) * 0.8)  # 80% for training
train_data = data[:train_size]
test_data = data[train_size:]

# Step 3: Use Isolation Forest for anomaly detection
# Initialize the Isolation Forest model
iso_forest = IsolationForest(contamination=0.05, random_state=42)  # contamination=0.05 means 5% anomali

# Fit the model on the training data
iso_forest.fit(train_data)  # Fit on the 'value' column of the training data

# Step 4: Make predictions on the test data
# Get the predictions: 1 for normal data, -1 for anomalies
test_predictions = iso_forest.predict(test_data)

# Convert predictions to boolean (True = anomaly, False = normal)
test_anomalies = test_predictions == -1
```

5. Running the Models

The following steps outline the process for running the models:

5.1 Preprocessing the Data:

- Load your dataset (e.g., NYC taxi data).

```
# Step 1: Load the dataset
file_path = '/content/sample_data/nyc_taxi.csv'
nyc_taxi_data = pd.read_csv(file_path)
```

- Convert the timestamp to datetime format.

```
# Convert timestamp to datetime and set as index
nyc_taxi_data['timestamp'] = pd.to_datetime(nyc_taxi_data['timestamp'])
nyc_taxi_data.set_index('timestamp', inplace=True)

# Use only the 'value' column
data = nyc_taxi_data[['value']]
```

- Normalize the 'value' column using MinMaxScaler.

```
# Step 2: Preprocessing
# Scale the data between 0 and 1 for better performance of LSTM
scaler = MinMaxScaler()
data['value_scaled'] = scaler.fit_transform(data[['value']])
```

- Create rolling time windows for sequences and split the data into training and testing sets.

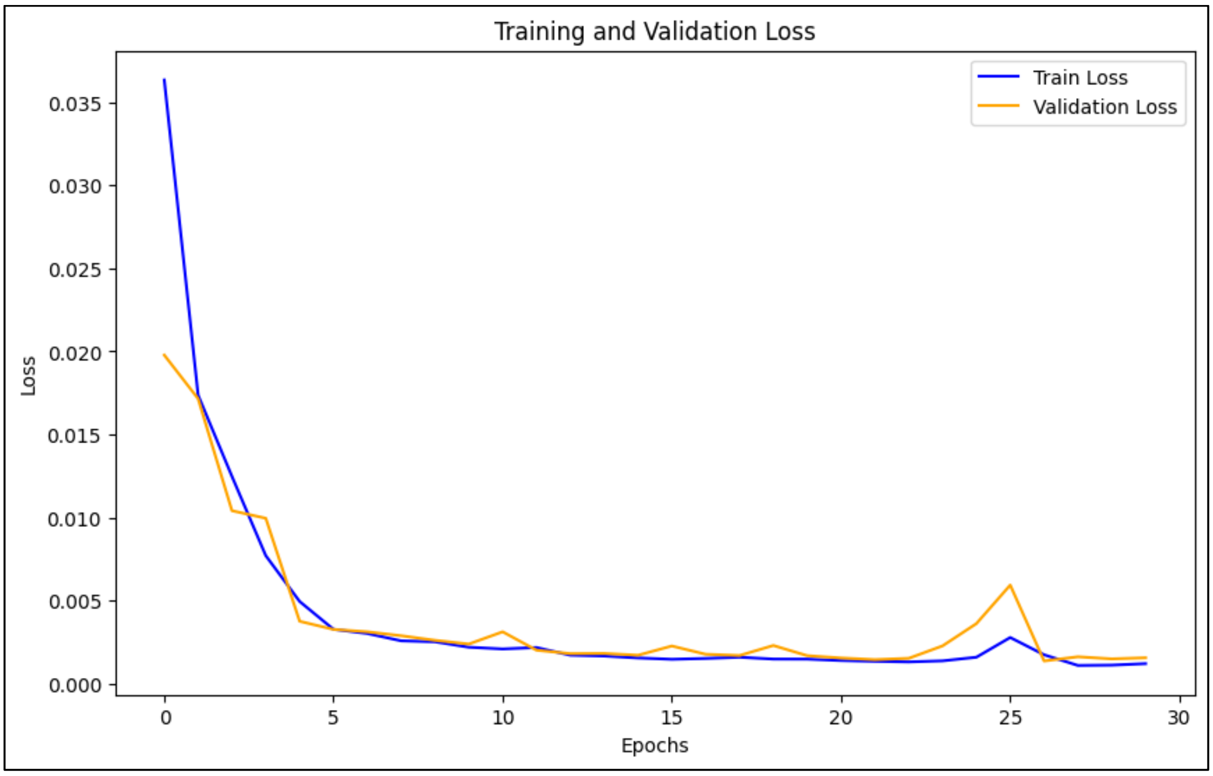
```
time_steps = 30  # Number of previous time steps to consider
sequence_data = create_sequences(data['value_scaled'].values, time_steps)


# Split into training and test sets
train_size = int(len(sequence_data) * 0.8)
train_data, test_data = sequence_data[:train_size], sequence_data[train_size:]
train_data_reshaped = train_data[..., np.newaxis]  # Shape: [samples, time_steps, 1]
test_data_reshaped = test_data[..., np.newaxis]  # Shape: [samples, time_steps, 1]
```

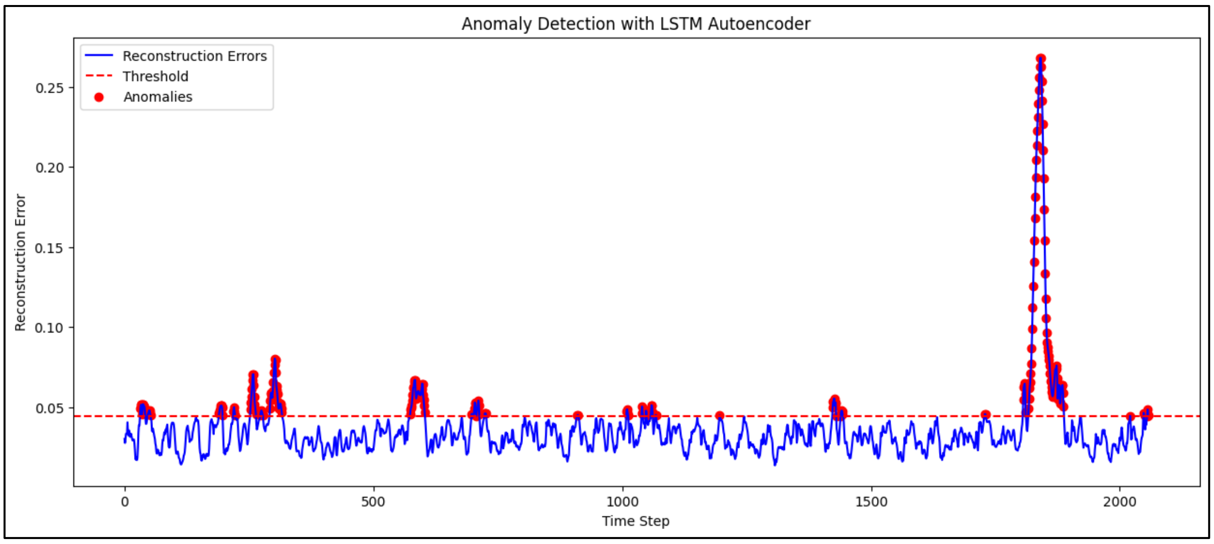
5.2 Train LSTM, RNN, and GRU Autoencoders:

- Define and compile each Autoencoder model.
- Train them on the training data and monitor the reconstruction error.

```python
# Step 4: Build LSTM Autoencoder
model = Sequential([
    LSTM(64, activation='relu', input_shape=(time_steps, 1), return_sequences=True),
    LSTM(32, activation='relu', return_sequences=False),
    RepeatVector(time_steps),
    LSTM(32, activation='relu', return_sequences=True),
    LSTM(64, activation='relu', return_sequences=True),
    TimeDistributed(Dense(1))
])
```

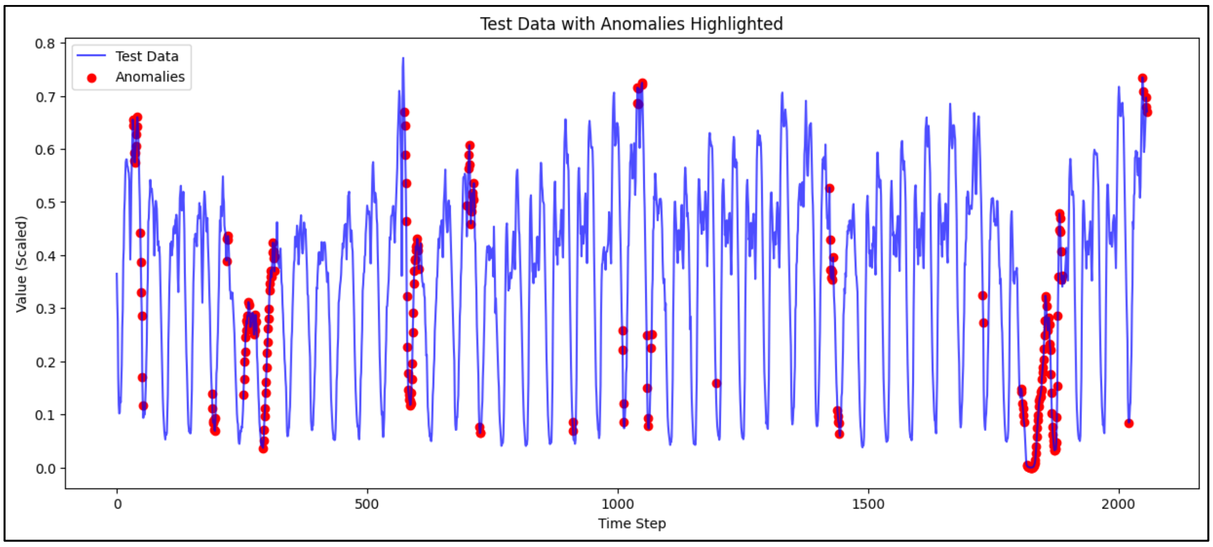
```python
model.compile(optimizer='adam', loss='mse')
model.summary()

# Step 5: Train the model
history = model.fit(train_data_reshaped, train_data_reshaped, epochs=30, batch_size=32, validation_split=0.2, shuffle=False)
```





5.3 Detect Anomalies:

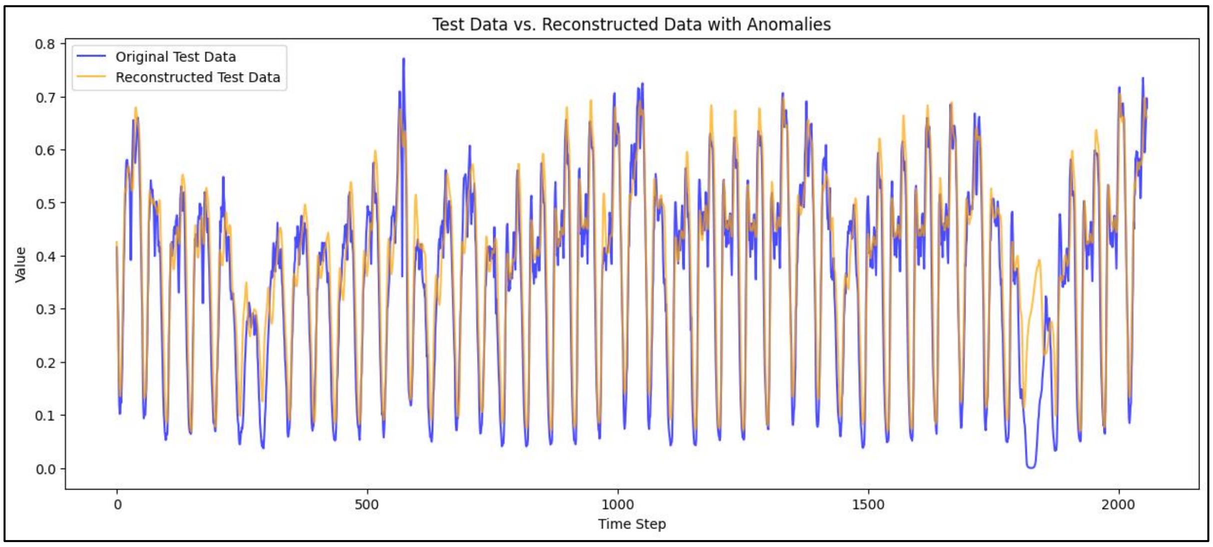
- Calculate reconstruction error for both training and test sets.
- Set a threshold based on the 95th percentile of the training error.
- Anomalies are detected if the reconstruction error exceeds this threshold.

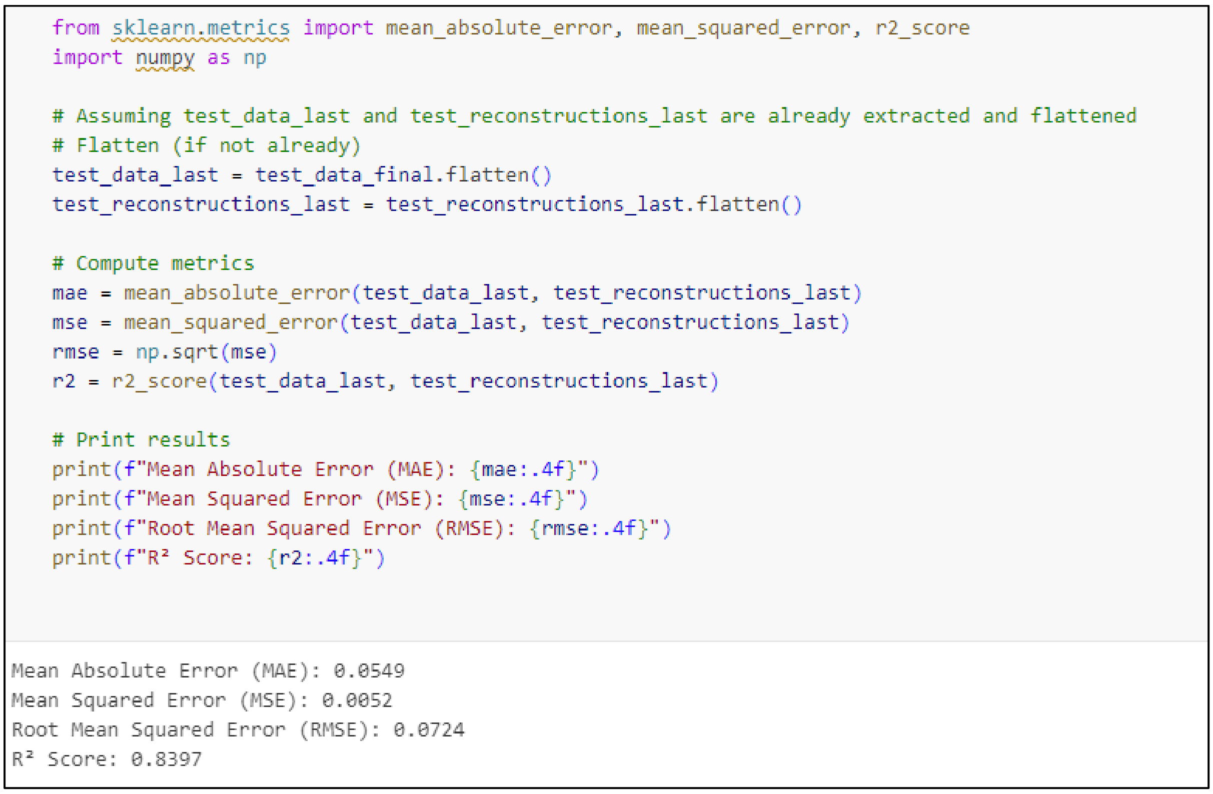


6. Evaluation and Visualization

Performance Metrics:

- Evaluate models using MAE, MSE, RMSE, and R² scores to measure reconstruction error.
- Compare the performance of the LSTM, RNN, and GRU models along with the Isolation Forest model.





```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Assuming test_data_last and test_reconstructions_last are already extracted and flattened
# Flatten (if not already)
test_data_last = test_data_final.flatten()
test_reconstructions_last = test_reconstructions_last.flatten()

# Compute metrics
mae = mean_absolute_error(test_data_last, test_reconstructions_last)
mse = mean_squared_error(test_data_last, test_reconstructions_last)
rmse = np.sqrt(mse)
r2 = r2_score(test_data_last, test_reconstructions_last)

# Print results
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"R² Score: {r2:.4f}")
```

```
Mean Absolute Error (MAE): 0.0549
Mean Squared Error (MSE): 0.0052
Root Mean Squared Error (RMSE): 0.0724
R² Score: 0.8397
```

Conclusion

This configuration manual provides you with the steps necessary to set up, execute, and evaluate hybrid deep learning models for time series anomaly detection. By following these instructions, you can replicate the research and adapt it to your specific use case.

References

Python: https://www.python.org
Dataset Source: https://www.kaggle.com/datasets/boltzmannbrain/nab