National College of Ireland

Attention-Augmented LSTM Autoencoders for Improved Anomaly Detection in Time-Series Data

MSc Research Project
Master of Science in Data Analytics

Karthik Shankar Naik
Student ID: X23242701

School of Computing
National College of Ireland

Supervisor: Hamilton V. Niculescu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Karthik Shankar Naik
Student ID:	X23242701
Programme:	Master of Science in Data Analytics **Year:** 2024-2025
Module:	MSc (Research)Practicum Part 2
Lecturer:	Hamilton V. Niculescu
Submission Due Date:	11th August 2025
Project Title:	Attention-Augmented LSTM Autoencoders for Improved Anomaly Detection in Time-Series Data
Word Count:	962 **Page Count:** 8

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: Karthik Shankar Naik

Date: 11th August 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):	

Attention-Augmented LSTM Autoencoders for Improved Anomaly Detection in Time-Series Data

Karthik Shankar Naik
Student ID: X23242701

Overview of Code

1. **Data Preparation**

- Load the NYC Taxi Passenger Count dataset (CSV format).
- Ensure time-indexed structure with hourly intervals.
- Filter out anomalies for training (retain only normal data).
- Split dataset:
  - 80% for training (normal only)
  - 20% for testing (normal + anomalies)

2. Model Architecture

- Construct an LSTM Autoencoder:
  - **Encoder**: LSTM layers to compress input sequences.
  - **Attention Layer**: Soft attention mechanism to weigh time steps.
  - **Decoder**: LSTM layers to reconstruct the input.
- Output: Reconstruction of input sequence.

3. Training

- Use **only normal** training data.
- Loss function: Mean Squared Error (MSE) between input and reconstruction.
- Optimizer: Adam (or similar).
- Epochs: Until convergence or early stopping on validation loss.

4. Evaluation

- Compute reconstruction errors on test data.
- Compare against threshold to classify anomaly (tuned via validation).
- Metrics: Precision, Recall, F1-score, AUC-ROC.

5. **Anomaly Detection**

- Input sequence → Autoencoder → Reconstructed sequence.
- Calculate reconstruction error.
- Flag as anomaly if error > threshold.

Hardware Requirements:

A GPU-enabled system is recommended for training the model efficiently, especially when working with long time sequences and deep architecture. At a minimum, the system should have:

- 8 GB of RAM or more
- A CUDA-compatible GPU (e.g., NVIDIA GTX 1060 or higher)
- Adequate storage space for datasets, models, and logs

Software Requirements:

The following software components and libraries should be installed and compatible:

- Python (version 3.8 or later)
- TensorFlow (for model development and training)
- PyTorch (optional, depending on implementation variant)
- NumPy and Pandas (for data handling and manipulation)
- Scikit-learn (for evaluation and metrics)
- Matplotlib and Seaborn (for visualization)
- Jupyter Notebook or JupyterLab (for experimentation and prototyping)

The use of virtual environments or containers (e.g., Conda or Docker) is encouraged to isolate dependencies and prevent compatibility issues. Logging and experiment tracking tools such as Weights & Biases or TensorBoard are also beneficial for monitoring training progress and tuning hyperparameters.

This environment setup provides a reliable foundation for replicating the model training pipeline, experimenting with various configurations, and deploying the solution in a controlled and efficient manner.

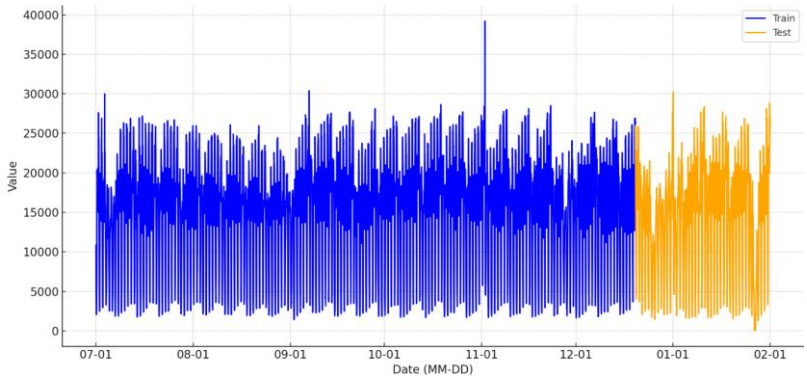
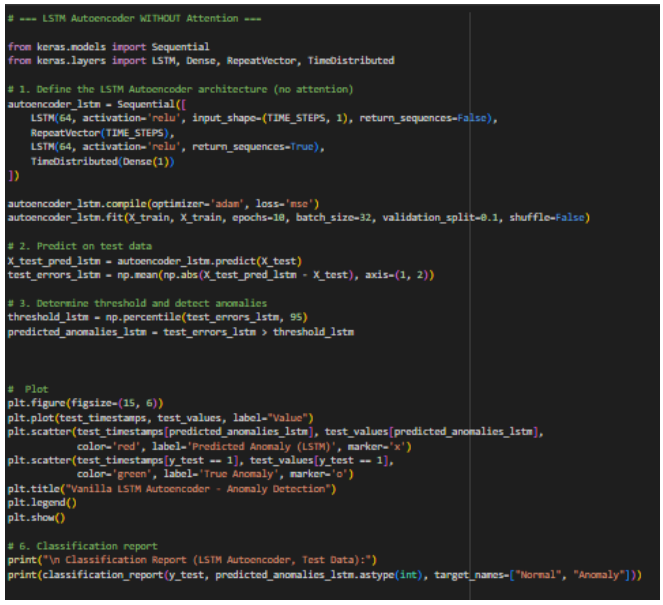


Figure 1. Train and Test Split of NYC Taxi Data

Figure 1 illustrates the temporal distribution of the train and test sets used in this study.

Model Architecture

- **Encoder:**
  - Two stacked **LSTM layers**
  - Extract hidden states representing temporal dependencies
- **Attention Layer:**
  - Computes attention weights across encoder hidden states
  - Generates a **context vector** via weighted sum of hidden states
  - Highlights **informative time steps**
- **Decoder:**
  - LSTM layers followed by a **dense output layer**
  - Reconstructs input sequence from context vector



```
# === LSTM Autoencoder WITHOUT Attention ===

from keras.models import Sequential
from keras.layers import LSTM, Dense, RepeatVector, TimeDistributed

# 1. Define the LSTM Autoencoder architecture (no attention)
autoencoder_lstm = Sequential([
    LSTM(64, activation='relu', input_shape=(TIME_STEPS, 1), return_sequences=False),
    RepeatVector(TIME_STEPS),
    LSTM(64, activation='relu', return_sequences=True),
    TimeDistributed(Dense(1))
])

autoencoder_lstm.compile(optimizer='adam', loss='mse')
autoencoder_lstm.fit(X_train, X_train, epochs=10, batch_size=32, validation_split=0.1, shuffle=False)

# 2. Predict on test data
X_test_pred_lstm = autoencoder_lstm.predict(X_test)
test_errors_lstm = np.mean(np.abs(X_test_pred_lstm - X_test), axis=(1, 2))

# 3. Determine threshold and detect anomalies
threshold_lstm = np.percentile(test_errors_lstm, 95)
predicted_anomalies_lstm = test_errors_lstm > threshold_lstm



#  Plot
plt.figure(figsize=(15, 6))
plt.plot(test_timestamps, test_values, label="Value")
plt.scatter(test_timestamps[predicted_anomalies_lstm], test_values[predicted_anomalies_lstm],
            color='red', label='Predicted Anomaly (LSTM)', marker='x')
plt.scatter(test_timestamps[y_test == 1], test_values[y_test == 1],
            color='green', label='True Anomaly', marker='o')
plt.title("Vanilla LSTM Autoencoder - Anomaly Detection")
plt.legend()
plt.show()

# 6. Classification report
print("\n Classification Report (LSTM Autoencoder, Test Data):")
print(classification_report(y_test, predicted_anomalies_lstm.astype(int), target_names=["Normal", "Anomaly"]))
```

Figure 2. Implementation of code

Evaluation Procedure

- **Macro-averaged metrics:**
  - **Precision**, **Recall**, **F1-score** (balanced evaluation across classes)
- **Anomaly-specific metrics:**
  - Targeted **Precision**, **Recall**, and **F1-score** for rare event detection
- **Visual analysis:**
  - Plot **reconstruction errors** over time

- Overlay **predicted vs. actual anomalies** for interpretability

- **Statistical significance testing:**

  - Use **paired t-tests** or **Wilcoxon signed-rank tests**
  - Compare with baselines: **RNN**, **GRU**, **Vanilla LSTM**

Baseline Models for Comparison

- RNN Autoencoder: serves as the simplest baseline

```
from keras.models import Sequential
from keras.layers import SimpleRNN

# 1. Define RNN Autoencoder architecture
rnn_autoencoder = Sequential([
    SimpleRNN(64, activation='relu', input_shape=(TIME_STEPS, 1), return_sequences=False),
    RepeatVector(TIME_STEPS),
    SimpleRNN(64, activation='relu', return_sequences=True),
    TimeDistributed(Dense(1))
])

rnn_autoencoder.compile(optimizer='adam', loss='mse')
rnn_autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, validation_split=0.1, shuffle=False)

# 2. Predict on test data
X_test_pred_rnn = rnn_autoencoder.predict(X_test)
test_errors_rnn = np.mean(np.abs(X_test_pred_rnn - X_test), axis=(1, 2))

# 3. Detect anomalies
threshold_rnn = np.percentile(test_errors_rnn, 95)
predicted_anomalies_rnn = test_errors_rnn > threshold_rnn



#  Plot predictions
plt.figure(figsize=(15, 6))
plt.plot(test_timestamps, test_values, label="Value")
plt.scatter(test_timestamps[predicted_anomalies_rnn], test_values[predicted_anomalies_rnn],
            color='red', label='Predicted Anomaly (RNN)', marker='x')
plt.scatter(test_timestamps[y_test == 1], test_values[y_test == 1],
            color='green', label='True Anomaly', marker='o')
plt.title("RNN Autoencoder - Anomaly Detection")
plt.legend()
plt.show()
```

Figure 3. Implementation of RNN Autoencoder code

- GRU Autoencoder: offers faster training with moderate performance

```
# 1. Define GRU Autoencoder model
gru_autoencoder = Sequential([
    GRU(64, activation='relu', input_shape=(TIME_STEPS, 1), return_sequences=False),
    RepeatVector(TIME_STEPS),
    GRU(64, activation='relu', return_sequences=True),
    TimeDistributed(Dense(1))
])

gru_autoencoder.compile(optimizer='adam', loss='mse')
gru_autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, validation_split=0.1, shuffle=False)

# 2. Predict on test data
X_test_pred_gru = gru_autoencoder.predict(X_test)
test_errors_gru = np.mean(np.abs(X_test_pred_gru - X_test), axis=(1, 2))

# 3. Determine threshold and detect anomalies
threshold_gru = np.percentile(test_errors_gru, 95)
predicted_anomalies_gru = test_errors_gru > threshold_gru



# 5. Plot GRU predictions
plt.figure(figsize=(15, 6))
plt.plot(test_timestamps, test_values, label="Value")
plt.scatter(test_timestamps[predicted_anomalies_gru], test_values[predicted_anomalies_gru],
            color='red', label='Predicted Anomaly (GRU)', marker='x')
plt.scatter(test_timestamps[y_test == 1], test_values[y_test == 1],
            color='green', label='True Anomaly', marker='o')
plt.title("GRU Autoencoder - Anomaly Detection")
plt.legend()
plt.show()
```

Figure 4. Implementation of GRU Autoencoder code

- Vanilla LSTM Autoencoder: captures long-term dependencies but lacks attention mechanism

```python
# === LSTM Autoencoder WITHOUT Attention ===

from keras.models import Sequential
from keras.layers import LSTM, Dense, RepeatVector, TimeDistributed

# 1. Define the LSTM Autoencoder architecture (no attention)
autoencoder_lstm = Sequential([
    LSTM(64, activation='relu', input_shape=(TIME_STEPS, 1), return_sequences=False),
    RepeatVector(TIME_STEPS),
    LSTM(64, activation='relu', return_sequences=True),
    TimeDistributed(Dense(1))
])

autoencoder_lstm.compile(optimizer='adam', loss='mse')
autoencoder_lstm.fit(X_train, X_train, epochs=10, batch_size=32, validation_split=0.1, shuffle=False)

# 2. Predict on test data
X_test_pred_lstm = autoencoder_lstm.predict(X_test)
test_errors_lstm = np.mean(np.abs(X_test_pred_lstm - X_test), axis=(1, 2))

# 3. Determine threshold and detect anomalies
threshold_lstm = np.percentile(test_errors_lstm, 95)
predicted_anomalies_lstm = test_errors_lstm > threshold_lstm



#  Plot
plt.figure(figsize=(15, 6))
plt.plot(test_timestamps, test_values, label="Value")
plt.scatter(test_timestamps[predicted_anomalies_lstm], test_values[predicted_anomalies_lstm],
            color='red', label='Predicted Anomaly (LSTM)', marker='x')
plt.scatter(test_timestamps[y_test == 1], test_values[y_test == 1],
            color='green', label='True Anomaly', marker='o')
plt.title("Vanilla LSTM Autoencoder - Anomaly Detection")
plt.legend()
plt.show()
```

Figure 5. Implementation of Vanilla LSTM Autoencoder code

- Each baseline is evaluated under the same training and evaluation pipeline

Performance Summary

- Attention-LSTM Autoencoder outperforms all baselines
- Achieves highest macro and anomaly-specific scores
- Demonstrates improved sensitivity to subtle and localized anomalies

Classification Report (Test Data):

	precision	recall	f1-score	support
Normal	0.98	0.98	0.98	1920
Anomaly	0.66	0.66	0.66	102
accuracy			0.97	2022
macro avg	0.82	0.82	0.82	2022
weighted avg	0.97	0.97	0.97	2022

Figure 6. Final Results of LSTM Auto Encoder with Attention

This configuration manual has outlined a comprehensive framework for implementing an attention-based LSTM Autoencoder for time-series anomaly detection.

References

Python Official Docs: https://www.python.org/doc/

PyTorch: https://pytorch.org/

TensorFlow: https://www.tensorflow.org/

Matplotlib: https://matplotlib.org/

PyTorch Geometric: https://pytorch-geometric.readthedocs.io/en/latest/

Jupyter Notebooks: https://jupyter.org/documentation

Scikit-learn: https://scikit-learn.org/

Seaborn: https://seaborn.pydata.org/