National College of Ireland

Improving Energy Forecasting and Anomaly Detection in Multivariate Environments

MSc Research Project
MSc in Data Analytics

Parashuram Vavilala
Student ID: x23337583

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …….PARASHURAM VAVILALA……

Student ID: ……x23337583……..……

Programme: ……MSc in Data Analytics…… **Year:** ……2025……

Module: ……Research Practicum……..……

Lecturer: ……Furqan Rustam……..……

Submission Due Date: ……15-09-2025……..……

Project Title: ……Improving Energy Forecasting and Anomaly Detection in Multivariate Environments……

Word Count: ……517…… **Page Count:** ……07……..……..……

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: ……Parashuram Vavilala…….

Date: ……15-09-2025……

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):	

Improving Energy Forecasting and Anomaly Detection in Multivariate Environments

Parashuram Vavilala
Student ID: x23337583

System Hardware Requirements

The implementation and evaluation of all models—including classical machine learning classifiers, deep learning autoencoders, and the attention-enhanced GRU autoencoder
All experiments were executed on a local workstation with the following specifications:

- **Processor (CPU):** Intel Core i7-12700K (12-Core, 3.6 GHz)
- **Memory (RAM):** 32 GB DDR4
- **Storage:** 1 TB SSD (NVMe)
- **Graphics Processing Unit (GPU):** NVIDIA RTX 3060 Ti (8 GB VRAM)
- **Operating System:** Windows 11 (64-bit)
- **Python Version:** 3.10 (Anaconda Distribution)

Minimum Recommended Specifications
To ensure smooth execution of the models, the following minimum hardware is recommended:

- **CPU:** Quad-Core Processor (≥ 2.5 GHz)
- **RAM:** 16 GB
- **Storage:** 500 GB SSD
- **GPU:** Optional, but recommended for training deep learning models (e.g., NVIDIA GTX 1660 or higher)
- **OS:** Windows 10/11, Ubuntu 20.04+, or macOS (M1/M2 recommended)

Ideal Configuration for Deep Learning
For larger datasets, frequent hyperparameter tuning, or multiple model training iterations, the following configuration is considered optimal:

- **CPU:** High-performance multi-core processor (e.g., AMD Ryzen 9 or Intel i9)
- **RAM:** ≥ 32 GB
- **GPU:** NVIDIA RTX 3080/3090 or equivalent with ≥ 10 GB VRAM
- **Storage:** ≥ 1 TB SSD
- **Environment:** Conda or virtualenv with CUDA/cuDNN support if using TensorFlow-GPU

Software Requirements:

- pandas, numpy, matplotlib, seaborn, scikit-learn, plotly
- torch, torchvision, torch_geometric, networks
- Data Loader, TensorDataset, StandardScaler, CrossEntropyLoss, Adam, AdamW, conda,

Dataset Details

The dataset used in this project was sourced from a publicly accessible and reputable platform offering real-world multivariate energy time series data.

datetime	nat_demand	T2M_toc	QV2M_toc	TQL_toc	W2M_toc	T2M_san	QV2M_san	TQL_san	W2M_san	T2M_dav	QV2M_dav	TQL_dav	W2M_dav	Holiday_ID	holiday	school	anomaly
2019-01-03 01:00:00	970.3450	25.865259	0.018576	0.016174	21.850546	23.482446	0.017272	0.001855	10.328949	22.662134	0.016562	0.096100	5.364148	0	0	0	0
2019-01-03 02:00:00	912.1755	25.899255	0.018653	0.016418	22.166944	23.399255	0.017265	0.001327	10.681517	22.578943	0.016509	0.087646	5.572471	0	0	0	0
2019-01-03 03:00:00	900.2688	25.937280	0.018768	0.015480	22.454911	23.343530	0.017211	0.001428	10.874924	22.531030	0.016479	0.078735	5.871184	0	0	0	0
2019-01-03 04:00:00	889.9538	25.957544	0.018890	0.016273	22.110481	23.238794	0.017128	0.002599	10.518620	22.512231	0.016487	0.068390	5.883621	0	0	0	0
2019-01-03 05:00:00	893.6865	25.973840	0.018981	0.017281	21.186089	23.075403	0.017059	0.001729	9.733589	22.481653	0.016456	0.064362	5.611724	0	0	0	0

Figure 1. Representative Sample from the Dataset

Model Configuration

Baseline Models

To benchmark the performance of the attention-enhanced GRU autoencoder, several baseline models were implemented using both classical machine learning and standard deep learning techniques. Classical models included Logistic Regression, Decision Tree, Random Forest, and XGBoost, which were trained on engineered features such as lag values and reconstruction residuals. These models served as lightweight classifiers for anomaly detection. In addition, deep learning-based autoencoders—ANN, CNN, LSTM, and GRU—were developed to reconstruct time series data and identify anomalies through reconstruction errors. These baselines helped evaluate the added value of sequential modeling and feature weighting in the final architecture.

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Train Decision Tree model
dt_model = DecisionTreeClassifier(max_depth=6, random_state=42)
dt_model.fit(X_train, y_train)

# Predict class probabilities
y_proba_normal = dt_model.predict_proba(X_test)[:, 0]

# Compute pseudo-reconstruction error
reconstruction_like_error = 1 - y_proba_normal

# Define anomaly threshold
threshold = np.mean(reconstruction_like_error) + 2 * np.std(reconstruction_like_error)

# Predict anomalies
y_pred_anomaly = (reconstruction_like_error > threshold).astype(int)

# Evaluation Metrics
print("Evaluation Metrics for Decision Tree Anomaly Detection:")
print("Accuracy:", accuracy_score(y_test, y_pred_anomaly))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_anomaly))
print("\nClassification Report:\n", classification_report(y_test, y_pred_anomaly))
```

Figure 2. Implementation of the Baseline Model

Deep Learning Models

Multiple deep learning models were developed to capture temporal dependencies and reconstruct normal energy consumption patterns for anomaly detection. The **ANN Autoencoder** served as a basic feedforward model, lacking sequence awareness but providing a nonlinear reconstruction baseline. The **CNN Autoencoder** used 1D convolutional layers to extract local feature relationships across time steps. The **LSTM Autoencoder** incorporated memory units to model long-term temporal dependencies, while the **GRU Autoencoder** provided similar sequence learning capabilities with fewer parameters and faster training. These models enabled comparison across different architectures, offering insights into the effectiveness of temporal modeling in multivariate time series analysis.

```python
encoding_dim = 64
# Define custom Attention layer
class AttentionLayer(Layer):
    def __init__(self):
        super(AttentionLayer, self).__init__()

    def build(self, input_shape):
        self.W = self.add_weight(name="att_weight", shape=(input_shape[-1], input_shape[-1]),
                                 initializer="random_normal", trainable=True)
        self.b = self.add_weight(name="att_bias", shape=(input_shape[-1],),
                                 initializer="zeros", trainable=True)
        self.V = self.add_weight(name="att_var", shape=(input_shape[-1], 1),
                                 initializer="random_normal", trainable=True)
        super().build(input_shape)

    def call(self, inputs):
        score = tf.nn.tanh(tf.tensordot(inputs, self.W, axes=1) + self.b)
        attention_weights = tf.nn.softmax(tf.tensordot(score, self.V, axes=1), axis=1)
        context_vector = attention_weights * inputs
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attention_weights

# Define model
def build_gru_attention_autoencoder(sequence_length, n_features, encoding_dim=64):
    input_layer = Input(shape=(sequence_length,n_features ))
    x = GRU(128, activation='relu', return_sequences=True)(input_layer)
    x = Dropout(0.2)(x)
    x = GRU(encoding_dim, activation='relu', return_sequences=True)(x)
    x = Dropout(0.2)(x)
    context_vector, _ = AttentionLayer()(x)
    x = RepeatVector(sequence_length)(context_vector)
    x = GRU(encoding_dim, activation='relu', return_sequences=True)(x)
    x = Dropout(0.2)(x)
    x = GRU(128, activation='relu', return_sequences=True)(x)
    x = Dropout(0.2)(x)
    output_layer = TimeDistributed(Dense(n_features))(x)
    return Model(inputs=input_layer, outputs=output_layer)
```

Figure 3. Implementation of deep learning Models

Main Model Configuration – Attention-Based GRU Autoencoder

```python
from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras.layers import GRU

# Define GRU Autoencoder
def build_gru_autoencoder(sequence_length, n_features, encoding_dim=64):
    input_layer = Input(shape=(sequence_length, n_features), name='input')

    # ENCODER
    x = GRU(128, activation='relu', return_sequences=True, name='encoder_gru1')(input_layer)
    x = Dropout(0.2)(x)
    x = GRU(encoding_dim, activation='relu', return_sequences=False, name='encoder_gru2')(x)
    x = Dropout(0.2)(x)

    # DECODER
    x = RepeatVector(sequence_length, name='repeat_vector')(x)
    x = GRU(encoding_dim, activation='relu', return_sequences=True, name='decoder_gru1')(x)
    x = Dropout(0.2)(x)
    x = GRU(128, activation='relu', return_sequences=True, name='decoder_gru2')(x)
    x = Dropout(0.2)(x)

    output_layer = TimeDistributed(Dense(n_features, activation='linear'), name='output')(x)
    autoencoder = Model(input_layer, output_layer, name='GRU_Autoencoder')
    return autoencoder

# Build and compile model
gru_autoencoder = build_gru_autoencoder(SEQUENCE_LENGTH, n_features, encoding_dim=64)
gru_autoencoder.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

print("\nGRU Autoencoder Architecture:")
gru_autoencoder.summary()
```

Evaluation and Results

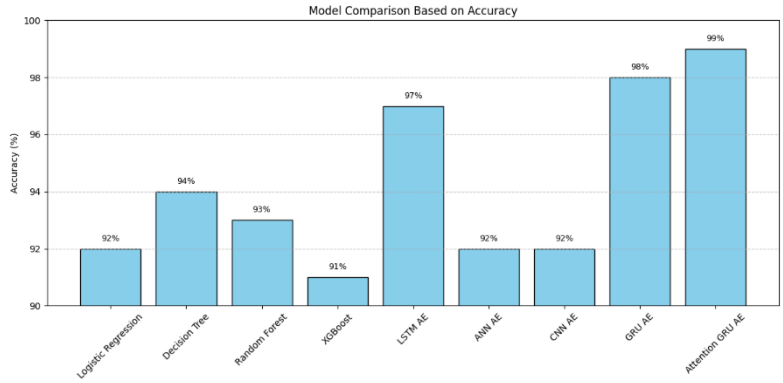
```
Threshold (95th percentile): 0.030708
Anomalies detected: 352 out of 8731 (4.0%)

 GRU Autoencoder - Classification Report:
              precision    recall  f1-score   support

      Normal       0.99      1.00      0.99      8294
     Anomaly       0.89      0.72      0.80       437

    accuracy                           0.98      8731
```

Comparison of Results



References

Python Official Documentation: https://www.python.org/doc/
PyTorch: https://pytorch.org/
Scikit-learn: https://scikit-learn.org/
Seaborn: https://seaborn.pydata.org/

Plotly: https://plotly.com/python/

XGBoost: https://xgboost.readthedocs.io/en/stable/

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