Enhancing Anomaly Detection in Cyber-Physical Systems with Event-Aware Graph Attention Networks (EA-GAT)

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE300 - MINI PROJECT

Submitted by

GAYAM HARSHA REDDY

(Reg No: 226003048, B.Tech-CSE)

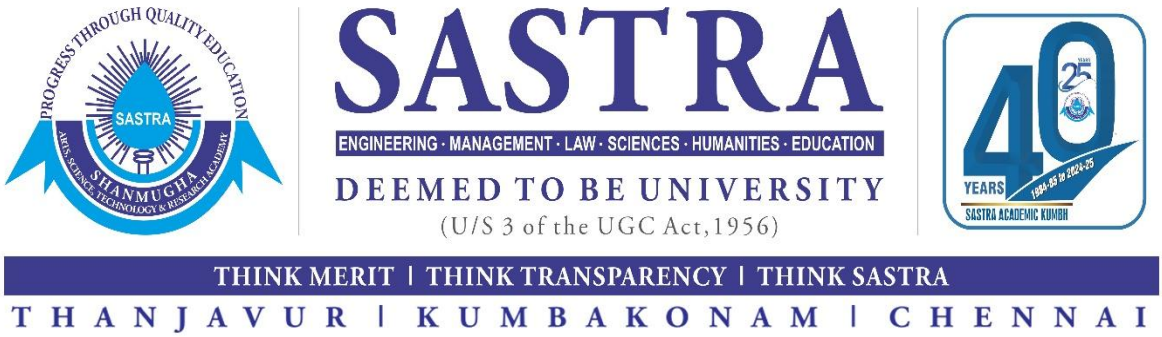
GUTTI LIKITHA

(Reg No:226003177, B.Tech-CSE)

CHIMMIRI SAHITHI VENNELA

(Reg No:226003032, B.Tech-CSE)

May 2025



Department of Computer Science and Engineering

SRINIVASA RAMANUJAN CENTRE

KUMBAKONAM, TAMIL NADU, INDIA-612001

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

Department of Computer Science and Engineering

SRINIVASA RAMANUJAN CENTRE

KUMBAKONAM, TAMIL NADU, INDIA-612001

Bonafide Certificate

This is to certify that the report titled **"Enhancing Anomaly Detection in Cyber-Physical Systems with Event-Aware Graph Attention Networks (EA-GAT)"** submitted as a requirement for the course, CSE300 : **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Ms.GAYAM HARSHA REDDY(222003048, B.Tech-CSE), Ms.GUTTI LIKITHA (222003177, B.Tech-CSE), Ms.CHIMMIRI SAHITHI VENNELA (222003032, B.Tech-CSE)** during the academic year 2024-25, in the Srinivasa Ramanujan Centre, under my supervision.

Signature of Project Supervisor : 24/4/25

Name with Affiliation : Dr. S.Priyanga/AP-II/CSE/SRC/SASTRA

Date : 24/4/25

Mini Project *Viva voce* held on _____

Examiner 1

Examiner 2

Acknowledgements

We would like to express our sincere gratitude to our Honorable Chancellor, **Prof. R. Sethuraman,** for providing us with the opportunity and the necessary infrastructure to carry out this project as part of our curriculum.

We are deeply thankful to our Honourable Vice-Chancellor, **Dr. S. Vaidhyasubramaniam, and Dr. S. Swaminathan**, Dean, Planning & Development, for their constant encouragement and strategic support throughout our college journey. We also extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University, for providing us with the opportunity, to pursue this project.

Our heartfelt thanks go to **Dr. V. Ramaswamy,** Dean, and **Dr. A. Alli Rani,** Associate Dean, Srinivasa Ramanujan Centre, SASTRA Deemed to be University. We are also pleased to express our gratitude to **Dr. V. Kalaichelvi**, Associate Professor, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre, for her encouragement and support during our project work.

Our guide, **Dr. S.Priyanga**/AP-II/CSE/SRC/SASTRA, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre, was the driving force behind this project from the very beginning. Her deep insights and invaluable suggestions guided us in making steady progress throughout the project.

We also thank the project review panel members, **Dr. Vaishnavi D** and **Sow**. **Rubidha Devi D**, for their valuable feedback and insights that significantly enhanced the quality of our work.

We would like to extend our special thanks to, **Dr. Vaishnavi D** Assistant Professor-II & Project Coordinator, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre, for organizing and supporting us in the successful completion of the project.

Finally, we gratefully acknowledge the unwavering encouragement and contributions from our families and friends, which played a vital role in the successful completion of this project.

We thank you all for giving us the opportunity to showcase our skills through this project

List of Figures

List of Tables

Abbreviations

EA-GAT	Event Aware Graph Attention Network
SWaT	Secure Water Treatment
WADI	Water Distribution
MSL	Mars Science Laboratory
GAT	Graph Attention Network
GAN	Generative Adversarial Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
PID	Proportional-Integral-Derivative
PLC	Programmable Logic Controller

Notations

English Symbols

A	Attention parameter
Y	Class probability
P	Dropout probability
FP	False Positive
FN	False Negative
H	Feature vector
TP	True Positive
TN	True Negative
R	Random value
W	Weight matrix

Greek Symbols

α(Alpha)	Attention Coefficient
∑(Sigma)	Summation
→	Vector notation

Miscellaneous Symbols

ϵ	epsilon in summation context
argmax	Function to get maximum argument
logSoftMax	Log of SoftMax function

Abstract

Reliable anomaly detection in Cyber-Physical Systems (CPS) is essential, particularly for multimodal time-series data. Traditional deep learning methods often fail to achieve the required accuracy and recall. Graph-based techniques suffer from data loss, and fixed window-size constraints limit spatio-temporal correlation analysis. Event Aware Graph Attention Networks (EA-GAT) introduces an event-based approach that dynamically models relationships in CPS by constructing graphs during specific event periods. Performance is enhanced through parallel processing, dividing data into subgroups for efficient analysis. Experiments on the SWaT dataset demonstrate that EA-GAT outperforms traditional methods, improving accuracy and recall. This approach offers a robust solution for real-time anomaly detection and system monitoring in CPS.

Keywords: *Event-Aware Graph Attention Networks (EA-GAT), Cyber-Physical Systems (CPS), Pruned Exact Linear Time (PELT), Anomaly Detection.*

Table Of Contents

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title: EA-GAT: Event aware graph attention network on cyber-physical systems.

Journal Name: Computers in Industry

Authors: Mehmet Yavuz Yağci, Muhammed Ali Aydin

Year: 2024

Indexing: SCIE, Scopus

1.1 Introduction

Cyber-Physical Systems (CPS) are a category of systems that closely couple physical processes with computational elements like controllers, sensors, and communication networks. CPS are being used more and more in critical infrastructures such as water treatment plants, energy systems, transportation systems, and manufacturing facilities. With their critical functions, it is essential to ensure the security and reliability of CPS. Yet CPS are extremely complicated, consisting of several sensors, actuators, and controllers that act across different modalities and time scales. With such complexity, CPS are especially susceptible to anomalies unforeseen deviations from the norm that result from hardware faults, software bugs, or malicious cyber attacks. An early and precise anomaly detection is critical to avoid physical infrastructure damage and allow uninterrupted operation.

1.2 Problem Statement

Cyber-Physical Systems generate multimodal time series data, making anomaly detection difficult, due to complex dependencies and different event durations. Deep learning models often fail to capture anomalies effectively due to their inability to model dynamic spatio-temporal relationships, leading to false alarms. Traditional graph-based approaches and many existing anomaly detection techniques suffer from data loss during time-series-to-graph conversion. The use of fixed window sizes in these methods further weakens the detection of spatio-temporal correlations, so there is a need for an adaptive, event-aware anomaly detection model that dynamically captures spatio-temporal relationships, efficiently processes multimodal data, and enhances system reliability by accurately identifying affected components.

1.3 Literature Survey

Lyu et al. (2022) introduced the Global-Local Integration Network (GLIN) for anomaly detection, which integrates local node features and the global network structure to enhance detection performance. GLIN beats ordinary models like GCN, GAT, and GraphSAGE. One major drawback of GLIN is its over-sensitivity to statistical noise, which can undermine its robustness in noisy environments [1].

Tang et al. (2023) proposed a graph-based anomaly detection algorithm for multivariate time series via attention-based sensor weighting. The algorithm is effective in detecting attacks with high accuracy through the focus on appropriate sensor readings. Though efficient, the model has the limitation of failing to parse long temporal dependencies, hence cannot capture fully the evolving patterns of time in the data [2].

Miele et al. (2022) used graph-based anomaly detection for wind turbines and introduced the application of a Graph Convolutional Autoencoder (GCAE) to time-series analysis. The approach efficiently discovers correlations among sensor data and, therefore, has utility in application to physical systems. It, however, uses correlation predominantly and does not express event-based dependency, which might be essential when identifying more involved anomalies [3].

Balla et al. (2022) investigated the use of deep learning methods on SCADA systems using CNN, RNN, Autoencoders, and DBN for identifying anomalies. Its method had satisfying detection rates and was effective when detecting intricate patterns. Nevertheless, it has immense computational expenses and is not highly interpretable and thus less desired for real-time monitoring in Cyber-Physical Systems (CPS) [4].

Nedeljkovic and Jakovljevic (2022) proposed a CNN-based anomaly detection in CPS, where separate CNN networks are trained for each sensor. It is effective for detecting anomalies in one sensor. Therefore, it does not account for correlations between different sensors, resulting in an incomplete system-level analysis [5].

Wang et al. (2022) identified stealthy attacks in CPS with a multi-layer LSTM model coupled with Random Forest feature selection. The approach enhances detection efficiency since relevant features are chosen. However, it comes with high resource usage and requires a lot of feature engineering, which might render it less practical in resource-limited settings [6].

Al-Asiri and El-Alfy (2020) proposed a Decision Tree-based anomaly detection method aimed at network packet analysis of critical infrastructures. It is easy to deploy and can perform well with structured network data. However, it is particular to specific infrastructures and does not generalize well across all Cyber-Physical Systems (CPS) [7].

Tabassum et al. (2022) explained anomaly detection in distributed systems with Federated Learning and Generative Adversarial Networks (GANs) for handling data imbalance. The suggested model efficiently handles data imbalance issues and can be used for large-scale CPS deployments. It is computation-intensive and does not support all forms of infrastructure [8].

1.4 Architecture Diagram

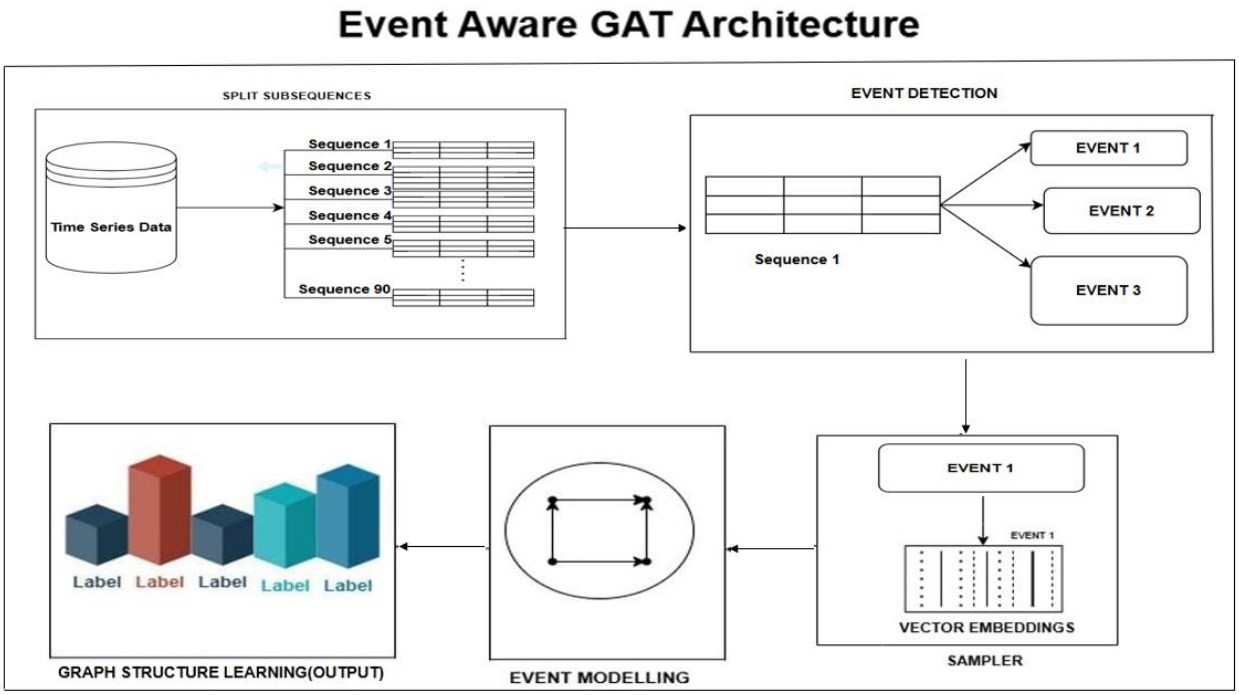


Fig. 1.4 Event Aware Graph Attention Network (EA-GAT) Architecture

1.5 Hardware and Software Requirements

Software Requirements

- Programming Language: Python
- Python IDLE and JupyterNotebook
- Libraries & Tools
- SciPy, NumPy, scikit, TensorFlow, pandas, seaborn, matplotlib

Hardware Requirements

- CPU: Intel Core i5
- RAM :16GB
- Storage: 100MB free space for dataset
- OS: Windows 11

1.6 Modules and Descriptions

1.6.1. Input Data (Multimodal Time Series): The model processes multimodal time series data generated by sensors and controllers in a cyber-physical system. Time series of multiple modalities are considered in concert to capture interdependencies between variables.

1.6.2. Split Subsequence: The big and complex CPS data hinders immediate processing, so the data set is divided into minor subsequences to improve processing efficiency. Parallel processing on autocorrelation is utilized to identify logical segmentation points to accelerate computation with the structural relationship of associated data.

1.6.3. Event Detection: In place of a pre-defined sliding window, the system utilizes the PELT algorithm to actively detect significant change points in the data. The change points identified in the data set, demarcates the start and end of the events, with accurate segmentation without defining event durations.

1.6.4. Sampler Module: Events from the previous step vary in length, and equal processing is required, for which standardization is needed. The sampling rate is determined as the ratio of the original vector length to the target vector length for proper event duration adjustment.
Down-sampling is done if the event slice is greater than the target length, preserving vital features by averaging and maintaining max-min differences intact.
Up-sampling is invoked when the slice of the event is shorter than the desired length, generating additional data points using interpolation from current values.

1.6.5. Event Modelling Module: Each detected event is translated into graph form, such that: Each sensor feature is a node. Sensor relationships are represented as edges via correlation-driven edge generation. The generated graphs preserve spatial and temporal relationships across events.

1.6.6. Graph Structure Learning Module: The GAT is utilized to learn the relationships between nodes and identify anomalies.
i. GATConv Layer: Utilizes an attention mechanism to give varying weights to the node connections based on their importance.
ii. LeakyReLU Activation: Adds non-linearity and prevents training from diverging.
iii. Mean Pooling: Concatenates node representations into a single graph-level embedding to encapsulate the event information.
iv. Dropout Mechanism: Prevents overfitting by randomly dropping connections during training.

1.6.7. Anomaly Detection: Log-softmax maps node embeddings into classification probabilities, i.e., whether an event is normal or anomalous. The system identifies the attacked or faulty sensors in real time and provides precise anomaly detection. Testing on datasets like SWaT indicates high accuracy in anomaly detection.

1.6.8. Evaluation Metrics

Precision – How many detected anomalies are correctly detected as anomalies.

$$precision = \frac{TP}{FP+TP} * 100 \tag{1}$$

Accuracy- How well the model classifies the Normal and Attack Data.

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} * 100 \tag{2}$$

Recall – How often a model correctly identifies anomalies from actual positive samples in dataset.

$$Recall = \frac{TP}{TP+FN} * 100 \tag{3}$$

F1-score – Harmonic mean of precision and recall.

$$F1_{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} * 100 \tag{4}$$

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

2.1 Merits

1.Event-Based Modelling: EA-GAT models the events of different durations dynamically to avoid the constraints of fixed window size and maintains spatio-temporal dependencies.

2.High Accuracy and F1 Score: It reports excellent performance on benchmark datasets such as SWaT.

3.Combines Data-Centric and Design-Centric Approaches: The model combines design and data and hence is adaptive and generalizable without requiring prior system-specific setup.

4.Split Subsequence Module Parallel Processing: EA-GAT framework allows faster data processing by decomposing multimodal time series into individual autocorrelated groups.

5. Application of GAT Layers: Using GAT layers enhances detection by discovering weighted node relationships, which makes the model sensitive to subtle dependencies.

6. Identifying Anomaly Component: EA-GAT can both detect attacked components and those affected indirectly, thus allowing further forensics.

7.Noisy Data Resilient: With attention mechanisms and dynamic sampling, the model is better able to deal with noisy and irregular sensor readings.

8.Better than Baseline Models: GAT method outperforms several state-of-the-art models like DAGMM, USAD, OmniAnomaly, and MST-GAT on several datasets (SWaT, WADI, MSL).

2.2 Demerits

1.Sensitivity to pen Parameter: The performance of the event detection largely relies on the adjustment of the pen parameter, which is different for dataset or system.

2. Increased Complexity: The architecture is composed of multiple steps (event detection, sampling, graph modelling, learning), which might augment implementation and maintenance overhead.

3. Needs Sufficient Training Data: Although generalizable, the model still relies on significant amounts of data in order to learn effective attention and correlation patterns.

4. Limited Testing on Real-Time Systems: Though it is for cyber-physical systems, the paper does not mention performance with real-time.

5.Computational Overhead: Graph processing, particularly with GAT layers, tends to be heavy in computational resources and hence less viable for edge or low-power devices.

6.Not Fully Explainable: Even with improved performance, the internal choice-making (through attention weights) can't necessarily be fully explained by system engineers.

7.Event Segmentation Ambiguity: Incorrect segmentation because of poor pen tuning may result in broken or merged events, impacting detection efficiency.

8.Lack of Varied Attack Scenarios: Though tested on some datasets, the research would be enhanced with a wider variety of real-world attack modes and severity levels.

CHAPTER 3

SOURCE CODE

3.1 Preprocessing

[3]:
```
if 'Timestamp' in sensor_data.columns:
    sensor_data.drop(columns=['Timestamp'], inplace=True)
    print("🗓️ Dropped 'Timestamp' column.")
```

[4]:
```
if 'Normal/Attack' in sensor_data.columns:
    sensor_data['Normal/Attack'] = sensor_data['Normal/Attack'].replace({'Normal': 1, 'Attack': 0})
    print("✅ Converted 'Normal' and 'Attack' labels to numeric values.")
```

```
✅ Converted 'Normal' and 'Attack' labels to numeric values.
```

[5]:
```
# Replace common error indicators with NaN
sensor_data.replace({'ERR': None, '---': None}, inplace=True)

# Convert all columns to numeric (forcing non-numeric values to NaN)
sensor_data = sensor_data.apply(pd.to_numeric, errors='coerce')

# Replace NaN values with the column median
sensor_data.fillna(sensor_data.median(), inplace=True)

print("✅ Replaced non-numeric values and handled missing data.")
```

```
✅ Replaced non-numeric values and handled missing data.
```

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Replace 'swat.csv' with the path to your SWaT dataset
dataset_path = 'C:/Users/gutti/OneDrive/Desktop/SWaT_Dataset_Attack_v0.csv'
data = pd.read_csv(dataset_path)
# Display the first few rows to inspect the data
data.head()

# Check for missing values
print(data.isnull().sum())
# Summary of the dataset
print(data.describe())
# Display column names
print(data.columns)

import pandas as pd
from sklearn.preprocessing import MinMaxScaler
# Load the SWaT dataset
data = pd.read_csv("C:/Users/gutti/OneDrive/Desktop/SWaT_Dataset_Attack_v0.csv") #
Replace with your actual file path
# Separate numerical and non-numerical columns
numerical_columns = data.select_dtypes(include=['number']).columns
non_numerical_columns = data.select_dtypes(exclude=['number']).columns
# Initialize MinMaxScaler
scaler = MinMaxScaler()
# Apply Min-Max Normalization only on numerical columns
normalized_numerical_data = scaler.fit_transform(data[numerical_columns])
# Convert the normalized numerical data back to a DataFrame
normalized_numerical_df = pd.DataFrame(normalized_numerical_data,
```

3.2 Split Subsequences

```python
import numpy as np
import pandas as pd
import multiprocessing
from numba import jit
from joblib import Parallel, delayed
import os  # Import os to manage directories

# Load dataset
file_path = "C:/Users/gutti/OneDrive/Documents/Project/SWaT_Dataset_NoHeaders.csv"

try:
    sensor_data = pd.read_csv(file_path, header=None)  # Since no headers
    print(f" Dataset loaded successfully! Total samples: {len(sensor_data)}")
except FileNotFoundError:
    print(" File not found! Please check the file path.")
    exit()

# Keep only numeric columns
sensor_data = sensor_data.select_dtypes(include=[np.number])

# Define thresholds
ACF_THRESHOLD = 0.3
VARIANCE_WINDOW = 5000  # Moving window size for variance detection
VARIANCE_THRESHOLD = 0.05  # Change detection threshold
MIN_SUBSEQUENCE_SIZE = 5000  # Min samples per sequence


@jit(nopython=True, parallel=True)
def fast_autocorrelation(series, nlags=50):
    """Compute autocorrelation using Numba for speed."""
    n = len(series)
    mean = np.mean(series)
```

3.3 Event Detection

3.3.1 Penalty 0.01

```python
import matplotlib.pyplot as plt
def plot_event_line_graph(event_dict_0_01):
    # Filter out sensors with no detected events
    event_counts = {sensor: len(changepoints) for sensor, changepoints in event_dict_0_01.items() if len(changepoints) > 0}
    if not event_counts:
        print("No sensors with detected events.")
        return
    # Sort sensors by event count for better visualization
    sorted_sensors = sorted(event_counts, key=event_counts.get)
    sorted_counts = [event_counts[sensor] for sensor in sorted_sensors]
    # Plot the line graph
    plt.figure(figsize=(14, 6))
    plt.plot(sorted_sensors, sorted_counts, marker='o', linestyle='-', color='b', markersize=6, label="Event Count")
    # Highlight each point
    for i, count in enumerate(sorted_counts):
        plt.text(i, count, str(count), ha='center', va='bottom', fontsize=10, color='black')
    # Formatting
    plt.xlabel("Sensor Name")
    plt.ylabel("Number of Detected Events")
    plt.title("Detected Event Counts Per Sensor")
    plt.xticks(rotation=45, ha="right")  # Rotate labels for better readability
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()
    plt.show()
# Call the function to generate the line graph
plot_event_line_graph(event_dict_0_01)
```

3.3.2 Penalty 0.001

```
import matplotlib.pyplot as plt
def plot_event_line_graph(event_dict_0_001):
    # Filter out sensors with no detected events
    event_counts = {sensor: len(changepoints) for sensor, changepoints in event_dict_0_001.items() if len(changepoints) > 0}
    if not event_counts:
        print("No sensors with detected events.")
        return
    # Sort sensors by event count for better visualization
    sorted_sensors = sorted(event_counts, key=event_counts.get)
    sorted_counts = [event_counts[sensor] for sensor in sorted_sensors]
    # Plot the line graph
    plt.figure(figsize=(14, 6))
    plt.plot(sorted_sensors, sorted_counts, marker='o', linestyle='-', color='b', markersize=6, label="Event Count")
    # Highlight each point
    for i, count in enumerate(sorted_counts):
        plt.text(i, count, str(count), ha='center', va='bottom', fontsize=10, color='black')
    # Formatting
    plt.xlabel("Sensor Name")
    plt.ylabel("Number of Detected Events")
    plt.title("Detected Event Counts Per Sensor")
    plt.xticks(rotation=45, ha="right")  # Rotate labels for better readability
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()
    plt.show()
# Call the function to generate the line graph
plot_event_line_graph(event_dict_0_001)
```

3.3.3 Penalty 0.1

```
[46]: import matplotlib.pyplot as plt

      def plot_event_line_graph(event_dict_0_1):
          """
          Plots a line graph showing the number of detected events for each sensor.
          """
          # Filter out sensors with no detected events
          event_counts = {sensor: len(changepoints) for sensor, changepoints in event_dict_0_1.items() if len(changepoints) > 0}

          if not event_counts:
              print("No sensors with detected events.")
              return

          # Sort sensors by event count for better visualization
          sorted_sensors = sorted(event_counts, key=event_counts.get)
          sorted_counts = [event_counts[sensor] for sensor in sorted_sensors]

          # Plot the line graph
          plt.figure(figsize=(14,7))
          plt.plot(sorted_sensors, sorted_counts, marker='o', linestyle='-', color='b', markersize=6, label="Event Count")

          # Highlight each point
          for i, count in enumerate(sorted_counts):
              plt.text(i, count, str(count), ha='center', va='bottom', fontsize=10, color='black')

          # Formatting
          plt.xlabel("Sensor Name")
          plt.ylabel("Number of Detected Events")
          plt.title("Detected Event Counts Per Sensor")
          plt.xticks(rotation=45, ha="right")  # Rotate labels for better readability
          plt.grid(True, linestyle='--', alpha=0.7)
          plt.legend()
```

3.3.4 Penalty 1

```
import matplotlib.pyplot as plt
def plot_event_line_graph(event_dict_1):
        # Filter out sensors with no detected events
    event_counts = {sensor: len(changepoints) for sensor, changepoints in event_dict_1.items() if len(changepoints) > 0}
    if not event_counts:
        print("No sensors with detected events.")
        return
    # Sort sensors by event count for better visualization
    sorted_sensors = sorted(event_counts, key=event_counts.get)
    sorted_counts = [event_counts[sensor] for sensor in sorted_sensors]
    # Plot the line graph
    plt.figure(figsize=(5, 3))
    plt.plot(sorted_sensors, sorted_counts, marker='o', linestyle='-', color='b', markersize=6, label="Event Count")

    # Highlight each point
    for i, count in enumerate(sorted_counts):
        plt.text(i, count, str(count), ha='center', va='bottom', fontsize=10, color='black')
    # Formatting
    plt.xlabel("Sensor Name")
    plt.ylabel("Number of Detected Events")
    plt.title("Detected Event Counts Per Sensor")
    plt.xticks(rotation=45, ha="right")  # Rotate labels for better readability
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()
    plt.show()
# Call the function to generate the line graph
plot_event_line_graph(event_dict_1)
```

3.3.5 Plotting All Penalty Parameters in one Graph for Comparison

```
import matplotlib.pyplot as plt
def plot_combined_event_line_graph(event_dicts, thresholds):
    plt.figure(figsize=(14, 7))
    colors = ['b', 'g', 'r', 'm']  # Colors for different penalties
    markers = ['o', 's', 'D', '^']  # Different markers for distinction
    # Get all sensors that have at least one detected event in any penalty
    all_sensors = set()
    for event_dict in event_dicts:
        all_sensors.update(event_dict.keys())
    all_sensors = sorted(all_sensors)  # Sort for better visualization
    x_positions = range(len(all_sensors))  # X-axis positions for sensor names
    for i, (event_dict, threshold) in enumerate(zip(event_dicts, thresholds)):
        event_counts = [len(event_dict.get(sensor, [])) for sensor in all_sensors]

        plt.plot(x_positions, event_counts, marker=markers[i], linestyle='-',
                 color=colors[i], markersize=6, label=f"Penalty {threshold}")
        for j, count in enumerate(event_counts):
            plt.text(x_positions[j], count, str(count), ha='center', va='bottom', fontsize=10, color='black')
    plt.xlabel("Sensor Name")
    plt.ylabel("Number of Detected Events")
    plt.title("Detected Event Counts Per Sensor for Different Penalties")
    plt.xticks(x_positions, all_sensors, rotation=45, ha="right")  # Rotate labels for clarity
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()
    plt.show()
# Example usage:
thresholds = [0.1, 0.01, 0.001, 1]  # Penalty values
event_dicts = [event_dict_0_1, event_dict_0_01, event_dict_0_001, event_dict_1]  # Replace with actual event dicts
plot_combined_event_line_graph(event_dicts, thresholds)
```

3.4 Sampler

```
import matplotlib.pyplot as plt
import random

def plot_resampled_events(resampled_data, penalties, num_samples=5):
    plt.figure(figsize=(14, 6))
    colors = ['b', 'g', 'r', 'm']  # Different colors for penalties

    for i, penalty in enumerate(penalties):
        if penalty not in resampled_data or len(resampled_data[penalty]) == 0:
            print(f"No resampled events available for penalty {penalty}.")
            continue

        # Select random samples to plot
        samples = random.sample(resampled_data[penalty], min(num_samples, len(resampled_data[penalty])))

        for sample in samples:
            plt.plot(sample, linestyle='-', alpha=0.7, color=colors[i], label=f"Penalty {penalty}" if sample is samples[0] else "")

    # Formatting
    plt.xlabel("Time Index (Fixed Length = 10)")
    plt.ylabel("Sensor Value (Normalized)")
    plt.title("Resampled Event Sequences for Different Penalties")
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

# Example usage:
plot_resampled_events(resampled_data, penalties, num_samples=5)
```

3.5 Event Modelling

[1]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

def read_events_from_file(filename):
    """
    Reads detected events from a file and converts them into a list of lists.
    :param filename: Path to the file containing detected events.
    :return: List of event samples.
    """
    detected_events = []

    try:
        with open(filename, "r") as file:
            lines = file.readlines()
            print(f"Reading {len(lines)} lines from the file.")

            for line in lines:
                if line.startswith("Sample"):
                    event_values = line.split(":")[1].strip().split()
                    detected_events.append([float(val) for val in event_values])
            print(f"Detected {len(detected_events)} event samples.")
    except FileNotFoundError:
        print(f"Error: The file {filename} was not found.")

    return detected_events

def create_event_graphs(detected_events):
    """
    Creates undirected event graphs using detected events in a linear (non-cyclic) fashion.
```

3.6 Graph Structure Learning

[15]:
```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GATConv, global_mean_pool
from torch_geometric.data import Data
import random
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

[17]:
```
class GATClassifier(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GATClassifier, self).__init__()
        self.conv1 = GATConv(in_channels, hidden_channels, heads=4, dropout=0.2)
        self.conv2 = GATConv(hidden_channels * 4, hidden_channels, heads=2, dropout=0.2)
        self.classifier = torch.nn.Linear(hidden_channels * 2, out_channels)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch
        x = F.elu(self.conv1(x, edge_index))
        x = F.elu(self.conv2(x, edge_index))
        x = global_mean_pool(x, batch)
        return self.classifier(x)
```

[19]:
```
def load_event_graphs(events_file, labels_file):
    graphs = []
    with open(events_file, 'r') as ef, open(labels_file, 'r') as lf:
        event_lines = ef.readlines()
        label_lines = lf.readlines()

        for event_line, label_line in zip(event_lines, label_lines):
            if not event_line.startswith("Sample"):
                continue

            parts = event_line.strip().split(':')
            values = list(map(float, parts[1].strip().split()))
            label_values = label_line.strip().split(',')

            try:
                label = int(label_values[-1])  # Adjust if necessary
            except ValueError:
                print(f"Error converting label: {label_values[-1]}")
                continue

            num_nodes = len(values)
            edge_index = [[i, i+1] for i in range(num_nodes - 1)] + [[i+1, i] for i in range(num_nodes - 1)]
            edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()

            x = torch.tensor([[v] for v in values], dtype=torch.float)
            y = torch.tensor([label], dtype=torch.long)

            data = Data(x=x, edge_index=edge_index, y=y)
            graphs.append(data)

    return graphs, [g.y.item() for g in graphs]
```

CHAPTER 4

SNAP SHOTS

4.1 Preprocessing

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	FIT101	LIT101	MV101	P101	P102	AIT201	AIT202	AIT203	FIT201	MV201	P201	P202	P203	P204	P205	P206	DPIT301	FIT301	LIT301	MV301	MV302	MV303	MV304
2	0.879322	0.452962	1	1	0	0.942783	0.876785	0.436773	0.865044	1	0	0	1	0	1	0	0.438853	0.928726	0.70735	0.5	1	0.5	0.5
3	0.886285	0.453015	1	1	0	0.942783	0.876785	0.436773	0.865044	1	0	0	1	0	1	0	0.438853	0.929319	0.70735	0.5	1	0.5	0.5
4	0.901833	0.452962	1	1	0	0.942783	0.876081	0.436773	0.863956	1	0	0	1	0	1	0	0.437572	0.92948	0.707733	0.5	1	0.5	0.5
5	0.918195	0.453122	1	1	0	0.942783	0.876081	0.436773	0.863956	1	0	0	1	0	1	0	0.437572	0.92948	0.708116	0.5	1	0.5	0.5
6	0.930842	0.453816	1	1	0	0.942783	0.876081	0.436773	0.864228	1	0	0	1	0	1	0	0.437572	0.92948	0.708451	0.5	1	0.5	0.5
7	0.945347	0.45435	1	1	0	0.942783	0.876081	0.436773	0.86459	1	0	0	1	0	1	0	0.437572	0.928941	0.708547	0.5	1	0.5	0.5
8	0.955442	0.454671	1	1	0	0.942783	0.876081	0.436773	0.86459	1	0	0	1	0	1	0	0.437572	0.930073	0.708355	0.5	1	0.5	0.5
9	0.960896	0.454831	1	1	0	0.942783	0.876081	0.436773	0.863774	1	0	0	1	0	1	0	0.437572	0.930073	0.708691	0.5	1	0.5	0.5
10	0.962172	0.455205	1	1	0	0.942783	0.876081	0.436773	0.863774	1	0	0	1	0	1	0	0.436932	0.930073	0.709074	0.5	1	0.5	0.5
11	0.960084	0.454617	1	1	0	0.942783	0.876081	0.436773	0.863774	1	0	0	1	0	1	0	0.436932	0.930073	0.708691	0.5	1	0.5	0.5
12	0.953005	0.454671	1	1	0	0.942783	0.876081	0.436773	0.863774	1	0	0	1	0	1	0	0.437501	0.930073	0.709074	0.5	1	0.5	0.5
13	0.949872	0.45419	1	1	0	0.942783	0.875144	0.436773	0.863774	1	0	0	1	0	1	0	0.438355	0.930073	0.708595	0.5	1	0.5	0.5
14	0.947784	0.453763	1	1	0	0.942783	0.874674	0.436773	0.863774	1	0	0	1	0	1	0	0.438995	0.930073	0.708978	0.5	1	0.5	0.5
15	0.94175	0.453229	1	1	0	0.942783	0.874674	0.436773	0.863774	1	0	0	1	0	1	0	0.439778	0.930073	0.709409	0.5	1	0.5	0.5
16	0.937804	0.452802	1	1	0	0.942783	0.874674	0.436773	0.863774	1	0	0	1	0	1	0	0.439778	0.929858	0.709888	0.5	1	0.5	0.5
17	0.930378	0.452695	1	1	0	0.942783	0.874674	0.436773	0.863774	1	0	0	1	0	1	0	0.43864	0.929858	0.709888	0.5	1	0.5	0.5
18	0.926317	0.452748	1	1	0	0.942783	0.874674	0.436773	0.863774	1	0	0	1	0	1	0	0.437999	0.928833	0.710223	0.5	1	0.5	0.5
19	0.918195	0.452748	1	1	0	0.942783	0.874674	0.436515	0.863095	1	0	0	1	0	1	0	0.437501	0.928294	0.710414	0.5	1	0.5	0.5
20	0.913669	0.452695	1	1	0	0.942783	0.874674	0.434186	0.863095	1	0	0	1	0	1	0	0.437501	0.928133	0.710127	0.5	1	0.5	0.5
21	0.906707	0.452748	1	1	0	0.942783	0.874674	0.434186	0.863095	1	0	0	1	0	1	0	0.437501	0.928133	0.710223	0.5	1	0.5	0.5
22	0.901833	0.452802	1	1	0	0.942783	0.874674	0.434186	0.863095	1	0	0	1	0	1	0	0.437501	0.928941	0.71051	0.5	1	0.5	0.5
23	0.896147	0.452535	1	1	0	0.942783	0.874674	0.436773	0.863095	1	0	0	1	0	1	0	0.437501	0.92948	0.710223	0.5	1	0.5	0.5
24	0.89081	0.453015	1	1	0	0.942783	0.874674	0.436773	0.863095	1	0	0	1	0	1	0	0.435083	0.92948	0.710271	0.5	1	0.5	0.5

4.2 Split Subsequences

```
✅ Dataset loaded successfully! Total samples: 449919

✅ Total subsequences generated: 90
Original Dataset Rows: 449919
Total Rows After Splitting: 449919
✅ Data fully split with NO extra rows! 🎯
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_1.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_2.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_3.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_4.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_5.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_6.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_7.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_8.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_9.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_10.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_11.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_12.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_13.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_14.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_15.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_16.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_17.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_18.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_19.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_20.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_21.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_22.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_23.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_24.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_25.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_26.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_27.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new split.csv\subsequence 28.csv (Rows: 5000)

📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_29.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_30.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_31.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_32.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_33.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_34.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_35.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_36.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_37.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_38.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_39.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_40.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_41.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_42.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_43.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_44.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_45.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_46.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_47.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_48.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_49.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_50.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_51.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_52.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_53.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_54.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_55.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_56.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_57.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_58.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_59.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_60.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_61.csv (Rows: 5000)
📁 Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_62.csv (Rows: 5000)
```

```
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_63.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_64.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_65.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_66.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_67.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_68.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_69.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_70.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_71.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_72.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_73.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_74.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_75.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_76.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_77.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_78.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_79.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_80.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_81.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_82.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_83.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_84.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_85.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_86.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_87.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_88.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_89.csv (Rows: 5000)
Saved: C:/Users/gutti/OneDrive/Documents/Project/Subsequences/new_split.csv\subsequence_90.csv (Rows: 4919)
```

🔹 Subsequence 1 (Rows: 5000)

	0	1	2	3	4	5	6	7	8	9	\
0	2.427057	522.8467	2	2	1	262.0161	8.396437	328.6337	2.445391	2	
1	2.446274	522.8860	2	2	1	262.0161	8.396437	328.6337	2.445391	2	
2	2.489191	522.8467	2	2	1	262.0161	8.394514	328.6337	2.442316	2	
3	2.534350	522.9645	2	2	1	262.0161	8.394514	328.6337	2.442316	2	
4	2.569260	523.4748	2	2	1	262.0161	8.394514	328.6337	2.443085	2	

	...	42	43	44	45	46	47	48	49	50	51
0	...	2	1	250.8652	1.649953	189.5988	0.000128	1	1	1	1
1	...	2	1	250.8652	1.649953	189.6789	0.000128	1	1	1	1
2	...	2	1	250.8812	1.649953	189.6789	0.000128	1	1	1	1
3	...	2	1	250.8812	1.649953	189.6148	0.000128	1	1	1	1
4	...	2	1	250.8812	1.649953	189.5027	0.000128	1	1	1	1

[5 rows x 52 columns]

🔹 Subsequence 2 (Rows: 5000)

	0	1	2	3	4	5	6	7	8	9	\
5000	2.535311	667.0323	2	1	1	263.3299	8.449308	317.0982	0.0	1	
5001	2.519617	668.0323	2	1	1	263.3299	8.449308	317.0982	0.0	1	
5002	2.503283	669.0323	2	1	1	263.3299	8.449308	317.0982	0.0	1	
5003	2.482785	670.0323	2	1	1	263.3299	8.449308	316.9956	0.0	1	
5004	2.465490	671.0323	2	1	1	263.3299	8.448988	316.7906	0.0	1	

	...	42	43	44	45	46	47	48	49	50	51
5000	...	2	1	250.2243	1.762086	189.0541	0.000256	1	1	1	0
5001	...	2	1	250.2243	1.762086	189.0541	0.000256	1	1	1	0
5002	...	2	1	250.2243	1.762086	189.0541	0.000256	1	1	1	0
5003	...	2	1	250.2243	1.762086	189.0541	0.000256	1	1	1	0
5004	...	2	1	250.2243	1.762086	189.0541	0.000256	1	1	1	0

[5 rows x 52 columns]

🔹 Subsequence 3 (Rows: 5000)

	0	1	2	3	4	5	6	7	8	9	...	\
10000	0.0	810.6872	1	1	1	263.5222	8.439375	316.7906	0.0	1	...	
10001	0.0	810.6087	1	1	1	263.5222	8.439375	316.9187	0.0	1	...	
10002	0.0	810.8442	1	1	1	263.5222	8.439375	316.9187	0.0	1	...	
10003	0.0	811.1975	1	1	1	263.5222	8.439375	316.9187	0.0	1	...	
10004	0.0	811.2760	1	1	1	263.5222	8.439375	316.9187	0.0	1	...	

	42	43	44	45	46	47	48	49	50	51
10000	2	1	248.0934	1.730048	187.2116	0.000128	1	1	1	1
10001	2	1	248.3017	1.730048	187.4039	0.000128	1	1	1	1
10002	2	1	248.4139	1.730048	187.4840	0.000128	1	1	1	1
10003	2	1	248.4139	1.730048	187.4840	0.000128	1	1	1	1
10004	2	1	248.4139	1.730048	187.6602	0.000128	1	1	1	1

[5 rows x 52 columns]

4.3 Event Detection

4.3.1 Penalty 0.01

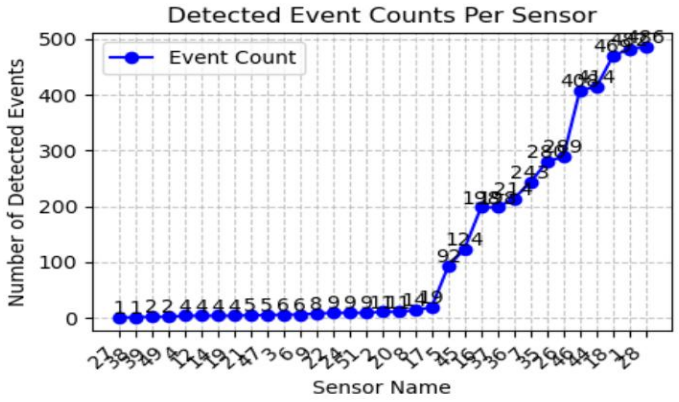


Fig. 4.3.1 Graph for penalty 0.01

4.3.2 Penalty 0.001

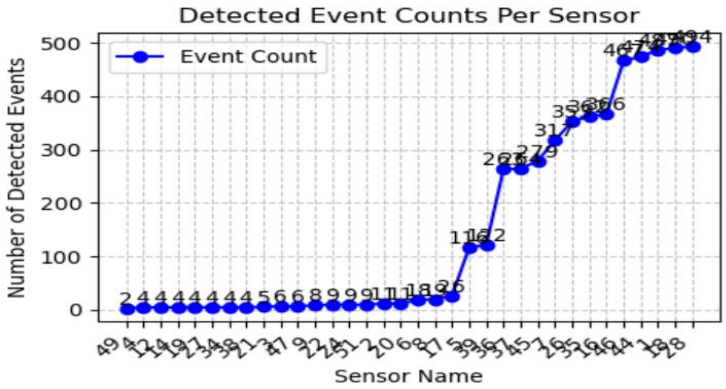


Fig. 4.3.2 Graph for penalty 0.001

4.3.3 Penalty 0.1

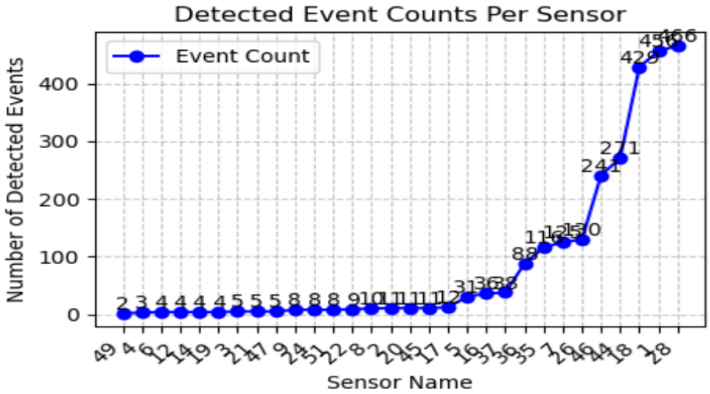


Fig. 4.3.3 Graph for penalty 0.1

4.3.4 Penalty 1

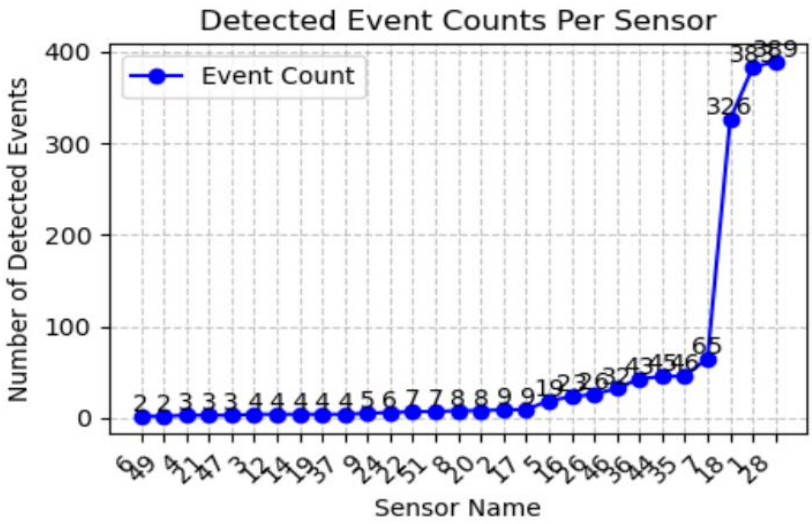


Fig. 4.3.4 Graph for penalty 1

4.3.5 Plotting All Penalty Parameters in one Graph for Comparison

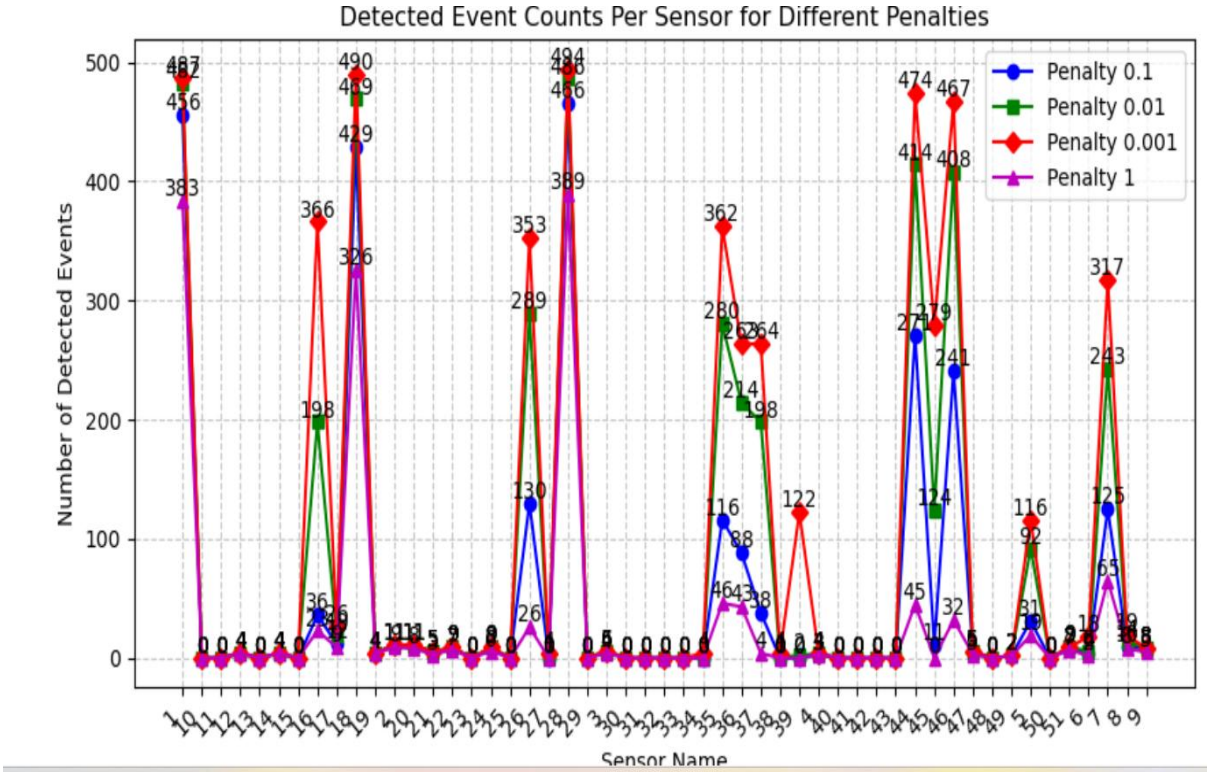


Fig. 4.3.5 Combined Graph for penalty 0.01,0.1.0.001,1

4.4 Sampler

```
Processing Sampler for Penalty 0.1...
System-dependent target length determined: 10
Resampled 2602 events to target length 10 for penalty 0.1.

Processing Sampler for Penalty 0.01...
System-dependent target length determined: 10
Resampled 4082 events to target length 10 for penalty 0.01.

Processing Sampler for Penalty 0.001...
System-dependent target length determined: 10
Resampled 5072 events to target length 10 for penalty 0.001.

Processing Sampler for Penalty 1...
System-dependent target length determined: 10
Resampled 1540 events to target length 10 for penalty 1.
```

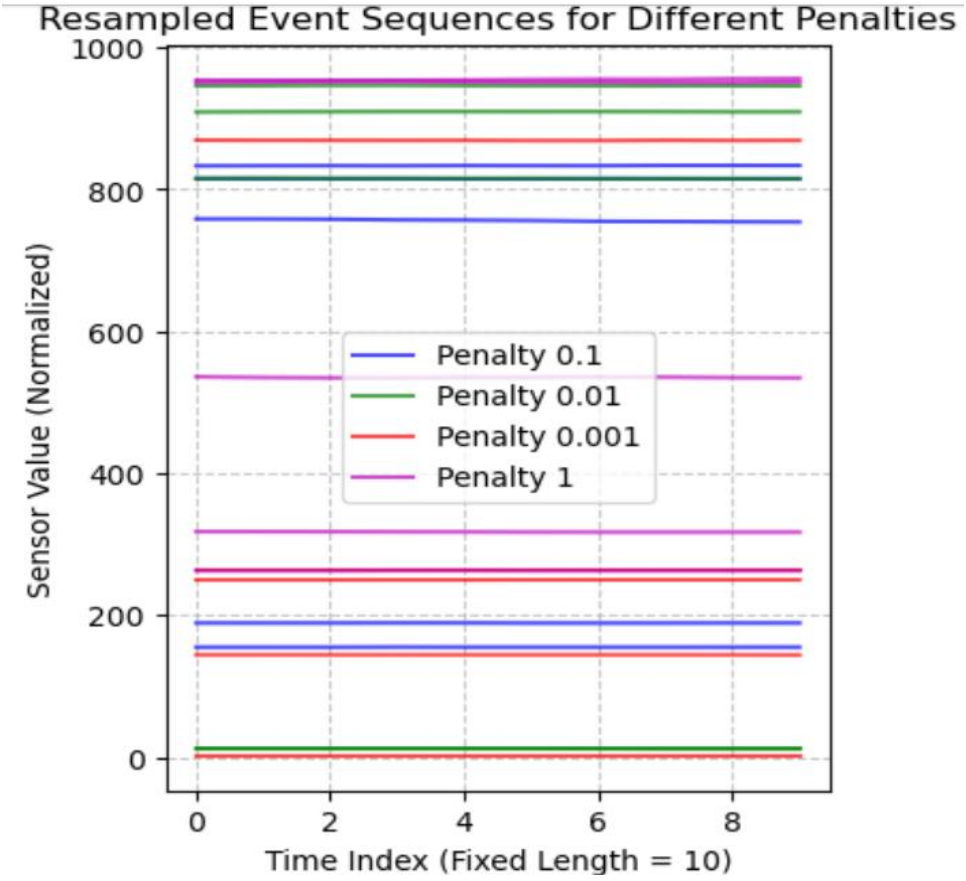


Fig. 4.4 Resampled Events

4.5 Event Modelling

Table 4.5 Features of event graph

Graph	1
Nodes	10
Edges	9
Avg Degree	1.8
Density	0.2
Diameter	9

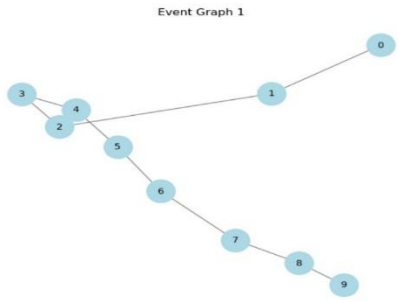


Fig. 4.5 Event Graph

```
Adjacency Matrix for Event Graph 1:
 [[0.     0.0393 0.     0.     0.     0.     0.     0.     0.     ]
 [0.0393 0.     0.0393 0.     0.     0.     0.     0.     0.     ]
 [0.     0.0393 0.     0.1178 0.     0.     0.     0.     0.     ]
 [0.     0.     0.1178 0.     0.5103 0.     0.     0.     0.     ]
 [0.     0.     0.     0.5103 0.     0.3925 0.     0.     0.     ]
 [0.     0.     0.     0.     0.3925 0.     0.2355 0.     0.     ]
 [0.     0.     0.     0.     0.     0.2355 0.     0.1178 0.     ]
 [0.     0.     0.     0.     0.     0.     0.1178 0.     0.2748 0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.2748 0.     0.4318]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.4318 0.    ]]
```

4.6 Graph Structure Learning

```
Total samples: 2602
Training samples: 2081, Testing samples: 521
Epoch 1/20, Loss: 0.0816, Training Accuracy: 0.9750
Epoch 2/20, Loss: 0.0394, Training Accuracy: 0.9789
Epoch 3/20, Loss: 0.0345, Training Accuracy: 0.9841
Epoch 4/20, Loss: 0.0339, Training Accuracy: 0.9856
Epoch 5/20, Loss: 0.0359, Training Accuracy: 0.9808
Epoch 6/20, Loss: 0.0367, Training Accuracy: 0.9813
Epoch 7/20, Loss: 0.0342, Training Accuracy: 0.9861
Epoch 8/20, Loss: 0.0312, Training Accuracy: 0.9875
Epoch 9/20, Loss: 0.0377, Training Accuracy: 0.9870
Epoch 10/20, Loss: 0.0315, Training Accuracy: 0.9837
Epoch 11/20, Loss: 0.0355, Training Accuracy: 0.9832
Epoch 12/20, Loss: 0.0370, Training Accuracy: 0.9846
Epoch 13/20, Loss: 0.0327, Training Accuracy: 0.9875
Epoch 14/20, Loss: 0.0290, Training Accuracy: 0.9856
Epoch 15/20, Loss: 0.0294, Training Accuracy: 0.9880
Epoch 16/20, Loss: 0.0263, Training Accuracy: 0.9894
Epoch 17/20, Loss: 0.0342, Training Accuracy: 0.9870
Epoch 18/20, Loss: 0.0340, Training Accuracy: 0.9832
Epoch 19/20, Loss: 0.0285, Training Accuracy: 0.9851
Epoch 20/20, Loss: 0.0317, Training Accuracy: 0.9827
```

```
--- Training Set Evaluation ---
Accuracy:  0.9962
Precision: 0.9744
Recall:    0.9268
F1-Score:  0.9500
Confusion Matrix:
[[1997    2]
 [   6   76]]
```

```
Sample 1: Predicted: Normal, True: Normal
Sample 2: Predicted: Normal, True: Normal
Sample 3: Predicted: Normal, True: Normal
Sample 4: Predicted: Normal, True: Normal
Sample 5: Predicted: Normal, True: Normal
Sample 6: Predicted: Normal, True: Normal
Sample 7: Predicted: Normal, True: Normal
Sample 8: Predicted: Normal, True: Normal
Sample 9: Predicted: Normal, True: Normal
Sample 10: Predicted: Normal, True: Normal
Sample 11: Predicted: Normal, True: Normal
Sample 12: Predicted: Normal, True: Normal
Sample 13: Predicted: Normal, True: Normal
Sample 14: Predicted: Normal, True: Normal
Sample 15: Predicted: Normal, True: Normal
Sample 16: Predicted: Normal, True: Normal
Sample 17: Predicted: Normal, True: Normal
Sample 18: Predicted: Abnormal, True: Abnormal
Sample 19: Predicted: Normal, True: Normal
Sample 20: Predicted: Normal, True: Normal
Sample 21: Predicted: Normal, True: Normal
Sample 22: Predicted: Normal, True: Normal
Sample 23: Predicted: Normal, True: Normal
Sample 24: Predicted: Normal, True: Normal
Sample 25: Predicted: Normal, True: Normal
Sample 26: Predicted: Normal, True: Normal
Sample 27: Predicted: Abnormal, True: Normal
Sample 28: Predicted: Normal, True: Normal
Sample 29: Predicted: Normal, True: Normal
Sample 30: Predicted: Normal, True: Normal
Sample 31: Predicted: Normal, True: Normal
```

```
--- Test Set Evaluation ---
Accuracy:  0.9923
Precision: 0.9444
Recall:    0.8500
F1-Score:  0.8947
Confusion Matrix:
[[500   1]
 [  3  17]]
```

Table 4.6.1 Evaluation Metrics Obtained for Training Set

Evaluation Metrics	Percentage Obtained (%)
Accuracy	99.62
Precision	97.44
Recall	92.68
F1-Score	95.00

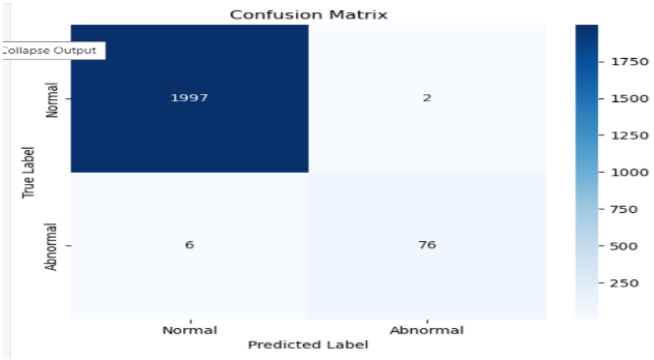


Fig.4.6.1 Confusion Matrix for Training Set

Table 4.6.2 Evaluation Metrics Obtained for Testing Set

Evaluation Metrics	Percentage Obtained (%)
Accuracy	99.23
Precision	94.44
Recall	85.00
F1-Score	89.47

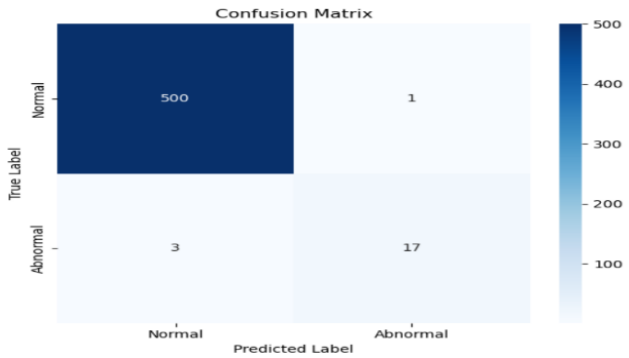


Fig 4.6.2 Confusion Matrix for Testing Set

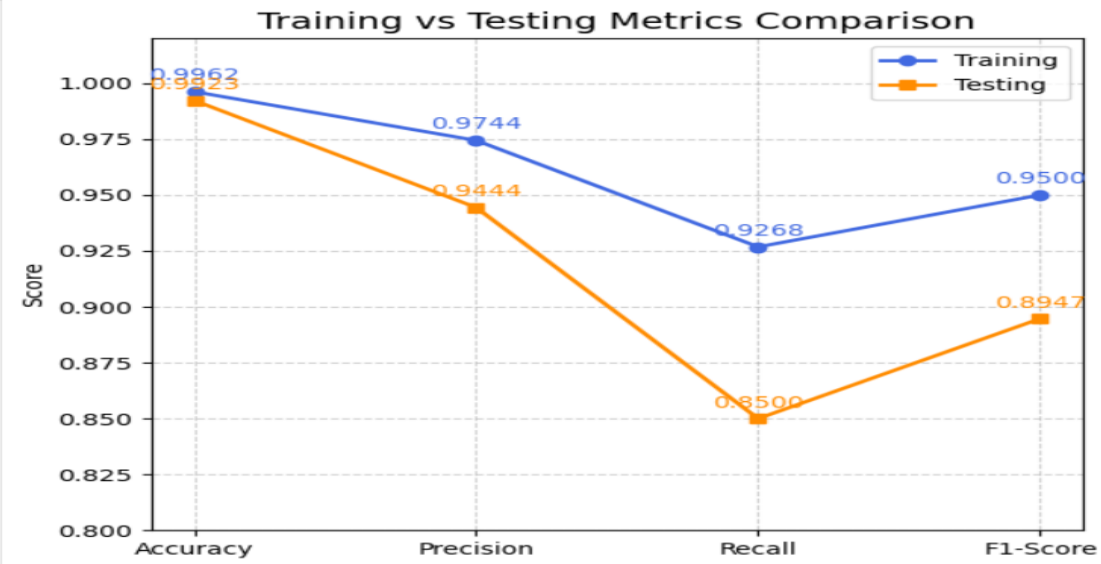


Fig. 4.6.3 Training and Testing Metrics Comparison

Table 4.6.3 Comparing EA-GAT with Competing Methods on SWaT

Method	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
GAT [12]	70.70	60.31	65.09
TABOR [16]	88.60	78.80	80.30
GRU [17]	99.86	59.09	74.96
1D CNN [18]	96.80	79.10	87.10
EA-GAT	**94.44**	**85.00**	**89.47**

CHAPTER 5

CONCLUSION AND FUTURE PLANS

5.1 Conclusion

The Event Aware-Graph Attention Networks offers a breakthrough for anomaly detection in cyber-physical systems overcoming the constraints of fixed window-based time series analysis. EA-GAT dynamically detects event boundaries with the Pruned Exact Linear Time (PELT) algorithm, transforms them into graph structures, and employs graph attention mechanisms to capture dependencies effectively. Having both data-focused and design-focused features, supported by event-aware modelling and concurrent processing via the split subsequence module, combines to make EA-GAT scalable and accurate enough for practical uses.

5.2 Future Plans

1. Multimodal Fusion with Non-Time-Series Data: The model is currently centred on multimodal time series. Future versions may include log files, configuration information, or external context (e.g., weather, schedules) to enhance event modelling.

2. Adaptive Pen Parameter Tuning: The paper mentions that the pen parameter employed in event detection plays a major role in performance. Future research can set a adaptive or self-tuning system for choosing the best penalty value depending on the dataset or system dynamics instead of manually setting it.

3. Resource Optimization: While EA-GAT is parallelized, additional improvements may aim to make it applicable to resource-limited environments (such as embedded systems) by reducing the architecture of the proposed concept.

CHAPTER 6

REFERENCES

1. **Lyu, Haoteng, Hongwei Li, Xiaotao Li, et al.** "GLIN: Global-Local Integration Network for Graph-Based Anomaly Detection." *IEEE Transactions on Knowledge and Data Engineering*, 2022. https://doi.org/10.1109/TKDE.2022.3200935.

2. **Tang, Jing, et al.** *"Anomaly Detection in Multivariate Time Series Using Graph Neural Networks." Applied Intelligence, 2023*. https://doi.org/10.1007/s10489-023-04633-x.

3. **Miele, Adriano, Fabio Tosti, Daniele Della Giustina, and Alfredo Vaccaro.** *"Using Graph Convolutional Networks for Early Anomaly Detection in Wind Turbines." Renewable Energy 187 (2022): 390–401*. https://doi.org/10.1016/j.renene.2022.01.108.

4. **Balla, Anett, Athanasios Zolotas, Fenghua Li, et al.** "Deep Learning for Anomaly Detection in SCADA Systems: A Comprehensive Survey." *Computers & Security* 117 (2022): 102693. https://doi.org/10.1016/j.cose.2022.102693.

5. **Nedeljkovic, Aleksandar, and Milorad Jakovljevic.** "Anomaly Detection in SCADA Systems Using Deep Convolutional Networks." *Journal of Intelligent & Fuzzy Systems* 43, no. 3 (2022): 3465–3477. https://doi.org/10.3233/JIFS-212888.

6. **Wang, Jun, et al.** "Detection of Stealthy Attacks in Industrial Control Systems Using Multi-Layer LSTM." *IEEE Transactions on Industrial Informatics* 18, no. 6 (2022): 3961–3970. https://doi.org/10.1109/TII.2021.3119420.

7. **Al-Asiri, Mohammed A., and El-Sayed M. El-Alfy.** "Anomaly Detection in Critical Infrastructure Based on Machine Learning Algorithms." *Journal of Ambient Intelligence and Humanized Computing*, 2020. https://doi.org/10.1007/s12652-020-02516-9.

8. **Tabassum, A., M. Asad, M. A. Jan, et al**. "Distributed Anomaly Detection Using GAN and Federated Learning." *Sensors* 22, no. 21 (2022): 8540. https://doi.org/10.3390/s22218540.

9. **Monzer, Tarek, et al**. "Health-Centric Intrusion Detection Systems in Critical Infrastructures." *ACM Transactions on Cyber-Physical Systems*, 2022. https://doi.org/10.1145/3551305.

10. **Das, Abhinav, et al.** “Real-Time Anomaly Detection Using Sensor Data in Cyber-Physical Systems.” *IEEE Internet of Things Journal*, 2020. https://doi.org/10.1109/JIOT.2020.3038702.

11. **Deng, A., and B. Hooi.** "Graph Neural Network-Based Anomaly Detection in Multivariate Time Series." *Proceedings of the AAAI Conference on Artificial Intelligence* 35, no. 1 (2021): 1495–1503. https://doi.org/10.1609/aaai.v35i1.16129.

12. **Ding, Shiqi, et al.** “MST-GAT: Multimodal Spatio-Temporal Graph Attention Networks for Anomaly Detection.” *Knowledge-Based Systems*, 2023. https://doi.org/10.1016/j.knosys.2023.109496.

13. **Drias, Z., A. Serhrouchni, and O. Vogel.** “Cyber Security: A Review of Anomaly Detection Systems: Techniques, Datasets, and Challenges.” *Procedia Computer Science* 58 (2015): 511–520. https://doi.org/10.1016/j.procs.2015.08.122.

14. **Farag, A. A., et al.** “Parallel Outlier Detection in Large Sequential Data Sets.” *Future Generation Computer Systems*, 2022. https://doi.org/10.1016/j.future.2022.03.050.

15. **Raman, Gauthama M. R., and Ankit Mathur.** “Data-Centric Versus Design-Centric Anomaly Detection in CPS.” *ACM Transactions on Cyber-Physical Systems*, 2022. https://doi.org/10.1145/3480071.

16. **Lin, Qian, Sandeep Adepu, Shonali Verwer, and Anish Mathur.** "TABOR: A Graphical Model-Based Approach for Anomaly Detection in Industrial Control Systems." In *Proceedings of the 2018 Asia Conference on Computer and Communications Security*, ASIACCS ’18, 525–536. New York, NY: Association for Computing Machinery, 2018. https://doi.org/10.1145/3196494.3196546.

17. **Tang, Chao, Lin Xu, Bo Yang, Yukun Tang, and Donghao Zhao.** “GRU-Based Interpretable Multivariate Time Series Anomaly Detection in Industrial Control Systems.” *Computers & Security* 127 (2023): 103094. https://doi.org/10.1016/j.cose.2023.103094.

18. **Kravchik, Matan, and Asaf Shabtai.** “1D Convolutional Neural Networks for Anomaly Detection in Time-Series Data.” In *Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 302–317. https://doi.org/10.1109/EuroSP.2018.00042.

CHAPTER 7

APPENDIX-BASE PAPER

Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.sciencedirect.com/journal/computers-in-industry

EA-GAT: Event aware graph attention network on cyber-physical systems

Mehmet Yavuz Yağci *, Muhammed Ali Aydin

Department of Computer Engineering, Istanbul University-Cerrahpaşa, Istanbul 34320, Turkey

ARTICLE INFO

Keywords:
Cyber-physical system
Anomaly detection
Graph attention network

ABSTRACT

Anomaly detection with high accuracy, recall, and low error rate is critical for the safe and uninterrupted operation of cyber-physical systems. However, detecting anomalies in multimodal time series with different modalities obtained from cyber-physical systems is challenging. Although deep learning methods show very good results in anomaly detection, they fail to detect anomalies according to the requirements of cyber-physical systems. In the use of graph-based methods, data loss occurs during the conversion of time series into graphs. The fixed window size used to transform time series into graphs causes a loss of spatio-temporal correlations. In this study, we propose an Event Aware Graph Attention Network (EA-GAT), which can detect anomalies by event-based cyber-physical system analysis. EA-GAT detects and tracks the sensors in cyber-physical systems and the correlations between them. The system analyzes and models the relationship between the components during the marked periods as a graph. Anomalies in the system are found through the created graph models. Experiments show that the EA-GAT technique is more effective than other deep learning methods on SWaT, WADI, MSL datasets used in various studies. The event-based dynamic approach is significantly superior to the fixed-size sliding window technique, which uses the same learning structure. In addition, anomaly analysis is used to identify the attack target and the affected components. At the same time, with the slip subsequence module, the data is divided into subgroups and processed simultaneously.

1. Introduction

Cybersecurity incidents in cyber-physical systems have risen in recent years, with a growing focus on integrating these systems into the new-generation communication systems. There has been a demand for better anomaly detection systems to protect against attacks and minimize the resulting physical damage. These systems are utilized to detect situations that display behavior that is outside the expected norms (Macas et al., 2022).

Anomaly detection systems designed for cyber-physical systems can be categorized into data-centric and design-centric, based on the input data they utilize (Gauthama Raman and Mathur, 2022). Data-centric approaches concentrate on data, including sensor measurements and network traffic. They depend on input data modeling to build a system's regular behavioral pattern. Due to inadequate data logging, the connection between sensors and events can be lost. Modelling physical events based on data collected from cyber-physical systems can be inaccurate.

Alternatively, design-centric approaches concentrate on system design details, including PID, sensor-based limits, and design patterns, which are fed into the anomaly detection system. The inputs comprise vital information that is hard to access, including sensor communications within the system and threshold values for each sensor. The inputs include critical information that is hard to access, including sensor communications within the system and threshold values for each sensor. Moreover, the system's complex design poses a challenge due to its vast array of inputs. It is imperative to reconfigure the anomaly detection system should any changes, additions, or expansions arise.

Traditionally, experienced engineers have manually determined static threshold values. However, the exponential rise in data volume and data points over the years has hindered the manual determination of threshold values. The increasing number of events in physical systems prevents the sensors from having fixed threshold values. Traditional studies that detect anomalies in multimodal time series created by multiple physical events do not yield accurate precision and recall metrics results.

Graph-based approaches have been shown to be successful in anomaly detection for cyber-physical systems where traditional methods have failed (Ding et al., 2023), (Tang et al., 2023). These approaches use edge, node, and weight mechanisms to model sensors and

* Corresponding author.
E-mail address: myavuz.yagci@iuc.edu.tr (M.Y. Yağci).

https://doi.org/10.1016/j.compind.2024.104097
Received 3 November 2023; Received in revised form 26 December 2023; Accepted 7 April 2024
Available online 26 April 2024

dependencies within physical systems. Time series data is transformed into a structured graph using fixed window sizes, and finite state diagram methods are applied to the graph model for anomaly detection studies.

Finite state diagram methods applied to the graph model outperform traditional methods. However, it has been observed that finite state diagrams are excessively sensitive, weakening the models because they only rely on statistical data extracted from visible features (Lyu et al., 2022). As the system expands and the amount of data and collection points grow, finite state diagrams yield less satisfactory outcomes than traditional methods. They are ineffective on mixed systems because they can only model a single flow and consider visible features. They cannot comprehensively analyze the system by evaluating all components together.

Time series are converted into graphs using the sliding window without finite state diagrams. The approach models a fixed-length data region as a graph. As such, modeling of cyber-physical systems requires a variable window length approach. However, cyber-physical systems may involve different events, which cannot be uniformly modeled with this technique. Furthermore, creating a universal model for cyber-physical systems using the sliding window approach is not feasible due to their inherent differences. Erroneous sampling leads to the loss of dependencies within the system, resulting in incomplete learning of the models.

In this paper, we propose an Event Aware Graph Attention Network (EA-GAT) to detect anomalies in cyber-physical systems. EA-GAT can model the sensors in cyber-physical systems and their interdependencies in an event-based manner. By monitoring the system, significant changes are detected and marked. The marked periods are modeled as a graph by analyzing the dependencies between all components in the system. This ensures that events of different lengths can be modeled holistically without losing dependencies. The modeled events are learned by a graph attention network built as four layers.

The main contributions of our work can be summarized as follows:

1. The proposed EA-GAT is an innovative anomaly detection method that combines the advantages of both data-centric and design-centric methods.
2. It minimizes service interruption due to system failure and undetected attacks due to erroneous decisions in anomaly detection in cyber-physical systems.
3. Detects the attacked component and other affected components in the system.
4. In contrast to studies that detect anomalies in multimodal time series, the Split Subsequence module allows the data to be processed in parallel by dividing the data into independent sections. Simultaneous processing of independent parts of the data reduces the time required to transform the multimodal time series into a graph.
5. A dynamic approach was adopted in contrast to the fixed sampling used in analyzing multimodal time series with graph-based methods. The dynamic approach allowed graph modeling of event-based dependencies by detecting events in the data. Within each graph, coefficients are used that are weighted according to the strength of the dependencies.

The rest of the paper is structured as follows. Section 2 presents the related works. Section 3 introduces the proposed EA-GAT. In Section 4, acquaint the experimental parameters. Section 5 experimental results demonstrate the success of the proposed method. Finally, the conclusion and future research directions are given in the last section.

2. Related work

Anomalies in cyber-physical systems can cause changes in the functioning of the physical process. Changes in the functioning of the process are usually detected when they reach a visible level. This situation often causes irreversible damage and loss of life and property. Therefore, modeling of cyber-physical systems and anomaly detection is significant. The research in the literature is analyzed under two main categories: the use and effectiveness of graph-based structures in cyber-physical systems and the key aspects of anomaly detection in cyber-physical systems.

This section briefly introduces anomaly detection in cyber-physical systems and graph-based anomaly detection studies in the literature.

2.1. Anomaly detection on cyber-physical systems

Z. Drias et al. (2015) examined cybersecurity attacks against cyber-physical systems. The study mentioned the importance of using anomaly detection systems to detect attacks. When designing anomaly detection systems, the requirements of cyber-physical systems should be analyzed. Anomalies occurring in cyber-physical systems are classified as infrequent anomalies. For this reason, it is expected that there will be few attack records in the test data, and insufficient attack records will lead to uncertainty about which system variable will be affected and at what rate. Feature selection methods are often used in anomaly detection systems to improve performance and reduce training time. However, feature selection methods reduce the number of parameters monitored in the system. The system parameters that do not affect the system for a long time can be removed from the monitored list using feature selection. However, the anomaly may affect the system parameter and cause disturbances in the system. The unexamined system parameter causes the system not to be fully analyzed. For this reason, although feature selection methods improve the performance of anomaly detection systems in the short term, they have the effect of missing anomalies in the long term.

Al-Asiri et al (Al-Asiri and El-Alfy, 2020). discuss the importance of anomaly detection in critical infrastructures, so-called cyber-physical systems. The concepts and terms of anomaly detection systems are analyzed. It is pointed out that the concepts of stateless and stateful are important parameters for classifying anomaly detection systems. Stateless anomaly detection systems do not evaluate the current input, or the input given as test data using historical patterns. They only assess the current data according to compliance with the parameters passed as conditions. Stateful anomaly detection systems evaluate the data using the correlation result extracted from the historical data. The gas distribution system test bed at Mississippi State University was used as an example of a critical infrastructure process. The gas distribution system is modeled using the Modbus TCP protocol in the test bed. A physical process was created through gas mains and storage tanks. The network data received in the test environment was analyzed to detect anomalies. The Modbus TCP packet address, CRC, length, type, Modbus function code, and payload data were analyzed from the network packets. Using the Weka application, the decision tree algorithm was run on the relevant data set. 10-cross validation was used as the validation method. A test environment created with a real-world scenario was used in the study. The anomaly detection system built in the study uses system variables such as the PID of the established infrastructure in the anomaly detection system. No comparison was made with other algorithms or studies. The static use of system variables in the model makes the proposed model work only on the proposed system. For this reason, the model is not considered a general model for cyber-physical systems.

A. Balla et al (Balla et al., 2022). evaluate deep learning techniques for use in SCADA systems. Deep learning models are introduced, and their compatibility with SCADA systems is mentioned. Among the algorithms analyzed are CNN, RNN, autoencoder, and DBN algorithms.

A. Tabassum et al (Tabassum et al., 2022). discuss the implementation issues of anomaly detection systems in distributed architectures. In distributed structures, data moving to the center is a problem that affects both processing and network performance. In addition, data imbalance problems, where one cluster is significantly less than the other cluster in the collected data, and inadequate sampling problems are examined.

The Internet of Things and some critical infrastructure processes operating in distributed architecture may face the mentioned problems. As a solution to the related issues, partial processing of the data in the region where the data is located, and transfer of the decision formed due to the processing to the center is suggested. It is based on the partial processing and transfer of data to the central authority by devices with limited resources connected in a centralized structure and geographically distributed architecture. The Generative Adversarial Network (GAN) algorithm is used with the federated learning approach. Problems such as imbalance and incomplete sampling in the data are attempted to be overcome with the synthetic data generation of the GAN algorithm. With Federated Learning, the structure is designed so that each component processes as much of its data as possible and then transmits it. In this way, network bandwidth problems are avoided. However, the proposed structure is not suitable for every critical infrastructure process.

The study (Monzer et al., 2022), mentions concepts such as stability, antifragile, health, and dispersion, which are called health metrics in Intrusion Detection System(IDS). It was noted that general-purpose IDS such as Suricata and Snort do not provide information about the system's overall health because they examine packets one at a time. In general-purpose anomaly detection systems, general rules based on the protocol are used to detect anomalies in the system. Commands sent to take action in the field may affect different components differently. For this reason, it is challenging to detect anomalies in critical infrastructures and to make decisions about the general state of the system using general-purpose structures. In some studies, anomaly detection is performed using system variables such as min-max values. There are also studies on automatic extraction of min-max values from system data. The system's limits are extracted in these studies using the autoregression method. By evaluating the instantaneous data of the system, information about its state is inferred. In the way proposed in the related study, the physical process, PLC, and infrastructure are entirely modeled using six virtual machines. The system information from the model construction has also been used to construct the anomaly detection system. The corresponding anomaly detection system requires a lot of details, like design information about the system and system variables. In the study by Fovino et al (Nai Fovino et al., 2012)., a similar approach was used to verify the commands sent to the physical layer before executing them in the field. The packets were analyzed in detail using a structure between the physical system and the component that controls the physical objects in the field. By knowing the range of data that should be sent to the appropriate physical component, the individual impact of the data on the physical system was evaluated. In addition to Fovino's work, the model-based IDS study found temporal and sequential patterns and examined the relationship between control variables and other variables. The architecture of the system is modeled using a graph structure. The functional behavior of the system is modeled using hybrid automata. While modeling the operational behavior, the code block running in the PLC was also used. In all modeling, process estimation structure, and synchronization structure were used to predict the system's current state.

Miele et al (Miele et al., 2022). analyzed data from a wind turbine operating for 20 months. It was studied on the detection of anomaly records in the data. The Graph Convolutional Autoencoder for Multivariate Time Series algorithm was used to analyze how the signals correlated with others. This study explains that the correlation of the data with other data is more important than the individual analysis. The advantages of correlation for anomaly detection are highlighted.

Authors in (Nedeljkovic and Jakovljevic, 2022), focused on detecting anomalies in physical data using the SWaT dataset (Goh et al., 2016). A separate CNN network is trained for data from each sensor and controller. The proposed work is constructed by training only with normal data without attack data in the system. The training phase is called the offline scheme. After training, the system is brought online, and test data consisting of normal data and attack data is applied. When the results obtained from the model are evaluated, it is seen that it is similar to the existing studies in the literature. In the study, the control of each signal by a separate CNN network causes the correlation between sensors to be ignored. The loss of correlation between sensors disturbs the analysis of the system as a whole. SWaT was able to detect 30 out of 36 attacks in the dataset.

Wang et al (Wang et al., 2022). developed a solution for stealthy attacks caused by inaccurate sensor measurements and sensor-directed effects. Stealth attacks are challenging to detect with traditional anomaly detection systems. Detecting stealth attacks usually requires a mathematical model of the system or physical security countermeasures. Physical methods are typically expensive and difficult to implement. Since mathematical models are not generalizable, a general model that can detect stealthy attacks cannot be created. STEP7 and SWaT datasets were used for the experimental studies. The corresponding study used heatmap and random forest algorithms to select the features with the highest impact on the system. The selected features were processed using the multi-layer LSTM method. Choosing the MFLSTM method consumes 52% less resources than the classical LSTM method. It was reported that the use of the MFLSTM method was as successful as the classical methods at lower values of the epoch parameter used during the model's training. The classical LSTM method has to work several times more to achieve the same performance.

Authors in (Umer et al., 2022) are categorized into two main areas: network-level analysis and physical data analysis. Network-level anomaly detection systems attempt to detect anomalies by analyzing the properties of network packets. Physical-level anomaly detection systems detect anomalies using physical process data that reflects the physical behavior of the system. Machine learning algorithms used in anomaly detection are analyzed under four main categories: supervised, unsupervised, semi-supervised, and reinforcement learning. The created taxonomy makes it easier to find the appropriate categories of algorithms used in anomaly detection.

Adephu et al (Adepu and Mathur, 2021). mentioned that faulty commands in the system, faulty components, incorrect control commands, communication errors, and cybersecurity attacks disrupt the physical process. It is emphasized that such anomalies can be detected after the physical process is affected and causes a visible disruption, leaving irreversible damage. To prevent this situation, it is argued that anomaly detection applications through the model of the system will be successful. Experimental studies were conducted using the SWaT dataset. The system model was extracted, and it was decided whether the sent command would be executed in the system or not. To use the associated structure, all parameters related to the design and the min-max values of each component must be known. Since it is not a general model, its applicability to all systems is very limited.

A study was conducted in (Das et al., 2020) to detect physical anomalies very close to real-time using low computational power. The healthy operation of infrastructures is made possible by the continuous monitoring of sensor readings of that structure. Anomalies in the physical process are detected by examining the sensor data. Using the SWaT dataset, a study was conducted on normal data and attack data that may occur in the physical process. In this study, a model of the system was extracted by examining past sensor records using the Logical Analysis of Data method. The extracted rules provide information about the current state of the system. The SWaT dataset has 36 attack scenarios, and 32 were detected with the proposed method. Since five detected attacks have a success rate below 0.5, they should be considered unsuccessful. With this conclusion, it can be seen in the related article that 9 of the 36 attacks could not be detected or were difficult to detect.

2.2. Graph based anomaly detection on cyber-physical systems

GNN-based methods try to predict the state at a node by encoding the state in its immediate neighbors. This model compromises the model's accuracy by giving more importance to the immediate neighborhood. This study (Lyu et al., 2022) uses the Global-Local Integration Network

Table 1
Compare Related Work.

Reference	Algorithms	Generalizable Model	Design Knowledge Requirement	Design Knowledge Synthesis	State Structure
(Gauthama Raman and Mathur, 2022)	PNN, SVM, RF, BayesNet, LR	no	yes	no	stateless
(Al-Asiri and El-Alfy, 2020)	Decition Tree	no	yes	no	stateless
(Faramondi et al., 2021)	KNN, RF, SVM, NB	none	no	no	stateful
(Monzer et al., 2022)	Hybrid Automata	no	yes	no	stateful
(Nedeljkovic and Jakovljevic, 2022)	CNN	yes	no	no	stateful
(Wang et al., 2022)	RF, MFLSTM	yes	no	no	stateful
(Adepu and Mathur, 2021)	Condition Structure	no	yes	no	stateless
(Das et al., 2020)	Logical Analysis of Data (LAD)	yes	no	no	stateful
(Lyu et al., 2022)	GCN GAT, SAGE, GLIN	yes	no	no	stateful
(Tang et al., 2023)	GRU	yes	no	no	stateful
(Ding et al., 2023)	GAT	yes	no	no	stateful
Our Model	**Event Aware Graph Attention Network**	**yes**	**no**	**yes**	**stateful**

(GLIN) structure for node-level anomaly detection by exploiting the local structure of nodes' and networks' global structure. The GLIN algorithm considers both the properties of the node and the data from the global network together when computing for the node. The GLIN model consists of five components: a preprocessor, an encoder for generating node representations, a pooling module that generates the global expression, an integration module that integrates the global expression vector into local node representations, and a decoder that performs label prediction. The preprocessor transforms the data stream into initial vectors, while the encoder performs message transfer and generates local representations for all nodes. The pooling module combines the local representations of all nodes in the output of the previous encoder to form the global expression. The integration module integrates the global expression vector into the local representations of the nodes, and the decoder performs label prediction based on the integrated vectors. GNN models encode temporal features separately. Therefore, data loss occurs when temporal data transients are encoded. To solve this problem, a sliding window approach is used, which allows a specific time interval to be analyzed. It is explained that the sliding window approach encodes typical transients. Two different sets of experiments were created using the SWAT dataset to validate the method's success. The first set contains positive samples obtained around the point where an anomaly begins. The second set contains negative and positive samples obtained sometime after the occurrence of an anomaly. Different combinations of GCN, GAT, SAGE, and GLIN were tested, and the results were reported.

Tang et al (Tang et al., 2023). mention that nowadays, anomaly detection in multivariate time series focuses on detecting anomaly locations and affected components. For this reason, anomaly detection systems that identify, learn, and interpret the dependencies between system variables have begun to be developed. Some significant challenges anomaly detection systems face are as follows: Deep learning-based approaches can model complex patterns by expanding to higher dimensions and are more robust to noise. However, they cannot deeply extract potential correlations in multivariate time series. By focusing only on attack detection, interpretation processes such as identifying the components affected by the attack are ignored. This paper proposes an algorithm for anomaly detection in multivariate time series to solve these problems. A graph-based architecture is used to learn sensors and other related components. The importance of the learned sensors is adjusted by giving them an attention coefficient. Each sensor is considered a node, and the similarity between it and its nearest neighbors is calculated and used as a node feature. Edges also represent neighborhoods between sensors. A portion of the time series data is retrieved using a fixed window size. The similarity between sensors and the node features is computed on the received data and transferred to the graph. The proposed model is tested on SWaT and WADI datasets. One attack from the SWAT dataset and two attacks from the WADI dataset are taken, and the success of the model is tested over three different attacks. In attack number three from the SWaT dataset, the accuracy is 0.88, precision is 0.99, and recall is 0.59.

Authors in (Farag et al., 2022) proposed a parallel outlier detection algorithm to detect outliers in sequential data. In particular, they focused on detecting structural anomalies in sequential data. The data set is first shown as a graph. A node is added to the graph for each data in the dataset. The data is sorted according to its structural properties. The data set is partitioned according to the number of cores in the processor. The fragmented data set is assigned to the corresponding CPUs. On the data assigned to the CPUs, the data is scanned in parallel using a fixed-length sliding window technique. An edge with a weight value is added between nodes in the same sliding window. The weight value is found by computing the Euclidean distance between behavioral features. Non-loop tree structures are explored using minimum spanning tree algorithms on the graph-transformed dataset. For each node, the edge with the least weight is found and added to the minimum spanning tree. Connecting the nodes creates a tree structure with a single vertex. The corresponding tree structure is converted into an edge list. The generated edge list is divided into groups and distributed to the processors. Classification is performed on the graph by deleting the edges. As a result of the operations, outlier factors are calculated for the data segments.

Authors in (Ding et al., 2023) mention that multimodal time series combines time series from multiple sources with various characteristics or measurements. Combining univariate time series with different sensor characteristics in industrial systems is an example of multimodal time series. This combination allows a more comprehensive analysis by combining data from various sensors. Deep learning-based models successfully detect anomalies in multimodal time series. However, these models generally cannot model the proportional and temporal relationships between univariate time series with different characteristics. This study generates and analyzes spatial and temporal correlations in multimodal time series with a graph attention network. In the MST-GAT method, sensors are represented as nodes, and their relationships are represented as edges. Each node represents a univariate time series. The node similarity is calculated, and k nodes with the highest similarity are generated as edges. The time series are converted to fixed-length inputs using a sliding window approach. Fixed-length inputs are analyzed with multi-head, intra-modal, and inter-modal attention modules. The multi-head attention module models spatial relationships independent of the style of the data. The intra- and inter-modal attention modules capture the correlation between time series. The intra-modal attention module models edge based on similarities between data. In the inter-modal attention module, edges are generated according to the differences between modalities. MLP is used as a prediction module. The Mars Science Laboratory (MSL) and the Soil Moisture Active Passive (SMAP)

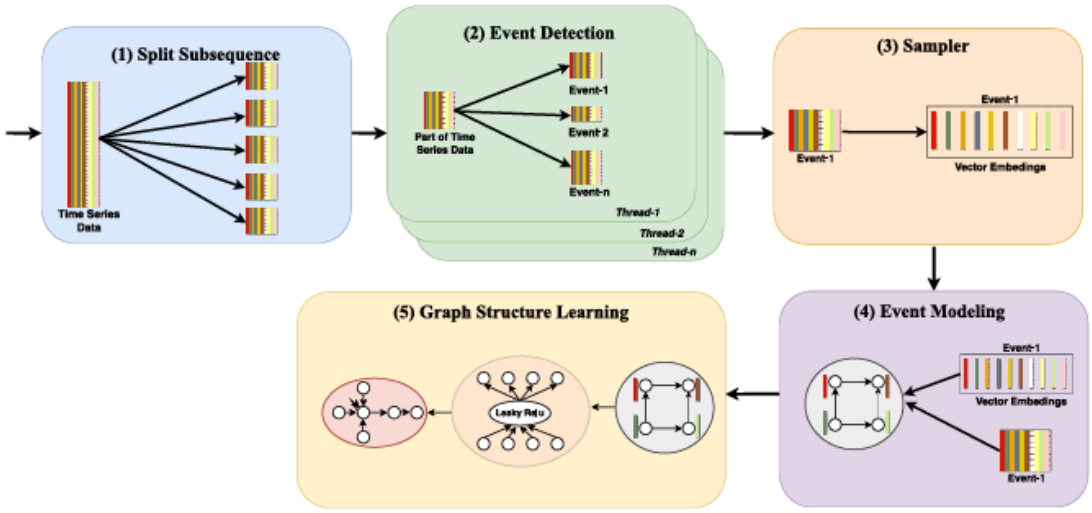


Fig. 1. Event Aware GAT Architecture.

(Hundman et al., 2018), Secure Water Treatment (SWaT) (Goh et al., 2016), and Water Distribution (WADI) (Ahmed et al., 2017) are used to test the model. The performance metrics for the SWaT dataset are 98.73% precision and 72.41% recall.

The prominent methods in the literature on anomaly detection in cyber-physical systems are examined in Table 1 with the properties of the generalizable model, design knowledge requirement, design knowledge synthesis, and state structure. The generalizable model property represents the model's suitability for other cyber-physical systems or datasets. The design information requirement means that there is a need for system-specific design information when constructing the systems. Design Information Synthesis represents the ability of the model to synthesize design information about the system without using system-specific parameters. State structure indicates the ability of the model to store information about the previous states of the system (Al-Asiri and El-Alfy, 2020).

It can be seen that the studies in the literature are generally based on two different model structures: those developed specifically for a single system and those developed in general and trying to adapt to systems. It is seen that the models developed for a single system cannot be applied to different systems due to the use of system-specific parameters such as PID, min-max, etc. On the other hand, models developed for general purposes and adapted according to the system to be used cannot reach a certain level of success due to the inability to use the system's design parameters to be adjusted. The EA-GAT model proposed in this study, on the other hand, can be applied to different systems like generalizable models since it can create the design of the system itself and can produce results at least as successful as models that use design information.

3. Methodology

This section describes the proposed EA-GAT method. The EA-GAT method aims to detect events by slicing the data of the correct length from the multimodal time series, transforming them into a graph, and detecting anomalies over the transformed data.

3.1. Problem definition

In operational technologies, various sensors and controllers work together harmoniously. Some of the sensors and controllers form specific patterns within different periods. Similarity and correlation within the data set mean a physical event occurs in the real world. To detect anomalies in physical events, it is necessary to be able to examine multimodal time series, extract information about system behavior, and express all this information in a data type that can model a physical system.

Multimodal Time Series (MMTS) combines time series with different characteristics or measurements. This time series is the type of data encountered in operational technologies that involve multiple modalities in the same period. Each modality has a self-repeating correlation within itself, as well as dynamics that depend on other modalities at specific time intervals. The main obstacle to making the right decisions for these systems is using anomaly detection algorithms unsuited to these dynamics. A review of innovative anomaly detection approaches shows that graph-based approaches are up-to-date and adaptable to the characteristics of MMTS. However, data loss occurs during the conversion of MMTS to graph data type. An analysis of the data loss shows that important parameters and relationships related to the system are lost. Sensor data from field events are referred to as physical data and are classified as MMTS.

Anomaly detection systems detect anomalies in the physical environment and generate relevant alarms. Anomaly detection systems using physical data include thresholding, finite state diagrams, and min-max determination methods. The common feature of the methods is that they require fixed inputs such as design information about the system, threshold, and parameters. The methods utilized can't cover the whole system due to the dynamic change of the system, the fact that the min-max values of the system vary based on events, and the different requirements in different modalities. Therefore, anomaly detection systems classify the system as an anomaly even when in a normal state, and it causes system failures. They cannot learn the natural changes in the system and are condemned to work in a fixed structure. They are also vulnerable to human intervention, flexibility and generalization problems, scalability, and false thresholds.

Representing physical data as a graph makes it easier to model and understand the system. However, there are some problems with graphing time series data. These problems are the size of the graph is too large, the information represented by the nodes and edges is determined, the time property is lost, and the relationship between the components is distorted. The most significant transformation that causes data loss is the time series sampling. The sliding window technique is the most widely used sampling method in the literature. It aims to sample

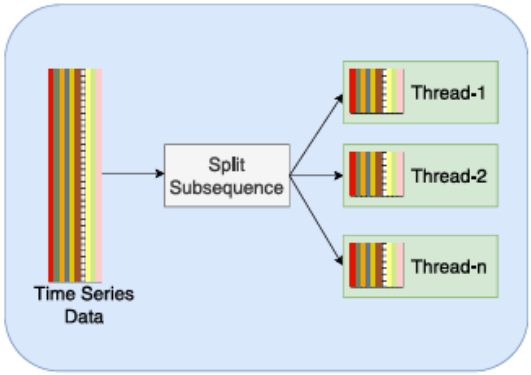


Fig. 2. Split Subsequence.

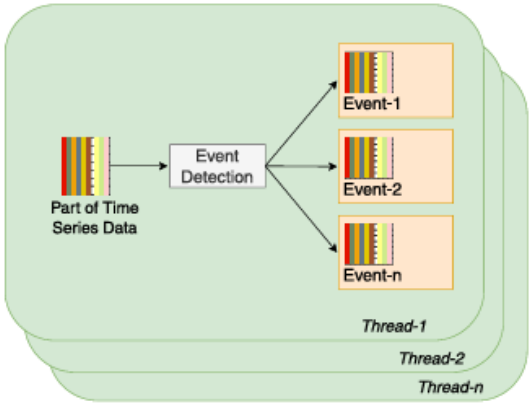


Fig. 3. Event Detection.

the data by sliding window with a fixed length. However, while the data is sampled at a constant length, it causes considerable data loss due to the loss of some temporal relationships and fragmentation of the data from the wrong points. This situation causes the system to fail to learn some of the relationships needed for anomaly detection.

3.2. Overview of EA-GAT

The EA-GAT method combines data-centric and design-centric approaches to extract design information from data and model it as a graph. It uses a stateful architecture that also stores the system's dependencies on past events. The EA-GAT architecture is illustrated in Fig. 1.

- Split Subsequence: Provides parallel processing by splitting the multimodal time series into subgroups.
- Event Detection: works in parallel on the data subsets, detecting the changes that can be considered as events within the data subsets and dividing the data into slices.
- Sampler: Reduces data slices of different lengths to a fixed length that can represent them all together and minimize data loss.
- Event Modeling: Models the correlation and relationship between features in data slices detected as events as a graph.
- Graph structure learning: Using the data modeled as a graph, it discovers system dependencies specific to events. It decides whether there is an anomaly in the system.

3.2.1. Split Subsequence

Learning large amounts of data by artificial intelligence algorithms and making decisions negatively affects data processing and response time. Parallel data processing and real-time results are achieved by dividing the data into subgroups that retain their meaning and are suitable for learning logic. This performs high-performance processing and faster results.

Dividing the data into subgroups involves considering all features and performing an autocorrelation calculation. Autocorrelation measures the relationship between consecutive observations in a time series. This analysis is used to understand patterns within data and relationships between variables. Autocorrelation analysis divides the data into slices by setting a specific cut-off point. Each subset is assigned to a thread, allowing the following modules to be run in parallel (Fig. 2). This method utilizes the principles of data fragmentation and parallel processing, allowing larger time series to be processed more efficiently.

Dividing data into relevant subgroups and providing it to artificial intelligence algorithms improves performance, scalability, resource management, fault isolation, and parallelization. Subdividing data into subsets and feeding them to artificial intelligence models ensures system performance is maintained and intermediate findings in very high-dimensional data can be quickly produced. The balanced distribution of subgroups across system resources facilitates resource management. Tracking errors that occur in subgroups helps maintain the overall success of the model. With the suitable grouping methodology, parallelization can be facilitated by reducing dependencies between subgroups. In addition, the coexistence of dependencies ensures that related inputs to the model are represented together. This has a positive impact on model performance.

3.2.2. Event detection

The sliding window technique is used in the studies in the literature that analyze multimodal time series using graph-based methods. The sliding window technique performs fixed-length sampling independent of the time series and the physical event. Fixed-length sampling should not be used because each data and event has different characteristics, and the events are completed at other times. Fixed-length sampling causes the data of the physical event to be analyzed with different parts, thus distorting the existence and integrity of the event in the analysis phase. The disruption of the event integrity causes the correlation between the sensors and controls in the system at the time of the event to degrade or even disappear.

In the split subsequence phase, the event detection phase is applied parallel to the data set, which is divided into subgroups and assigned to threads. Using the data in each thread, the data is divided into pieces of different sizes, in which compatible rows are collected together (Fig. 3).

It uses all features in data to detect change points and interprets them as beginning and ending points of events. The Power of the Pruned Exact Linear Time (PELT) algorithm (Wambui et al., 2015) detects data change points.

In the process of finding data change points, the PELT algorithm produces results by the predefined error value. The error value used in the PELT algorithm is called the pen parameter in the event detection phase. A very low pen parameter captures small data changes such as noise. In cases where the pen parameter is chosen high, it is seen to have difficulty capturing physical events. It is necessary to work with the most accurate pen parameter to accurately predict physical events from data with different modalities in multimodal time series. Different pen parameters, such as 1, 0.1, 0.01, and 0.001, were used to model the physical events in the SWaT dataset.

Regions detected as events with different lengths are formed for each pen parameter tested. A length histogram is used to visualize the length distribution of the regions detected as events and to analyze their

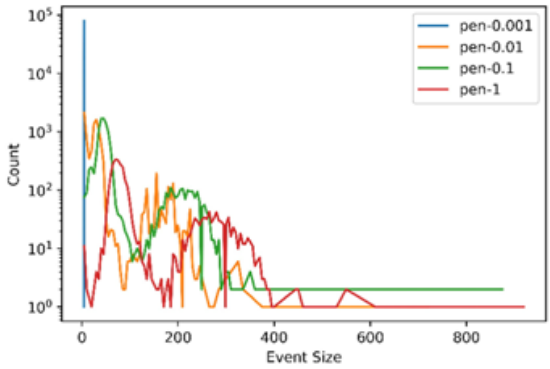


Fig. 4. Length Histogram.

density. The length histogram displays the lengths on the x-axis to represent the number of vectors of different lengths and the number of vectors of that length on the logarithmically scaled y-axis. The length histogram resulting from different pen values is shown in Fig. 4.

Analyzing the graph formed according to the Pen parameter, it can be seen that the data is spread over two peaks and their surroundings. The presence of two peaks basically indicates that the events are concentrated in two different length regions. Thus, events with regions of two different lengths can be examined. When the parameter pen is set to 0.001, the graph shows that the data is clustered in a single point and shows no distribution. The lack of distribution in the graph shows that all the data is divided into regions of fixed length. This situation is similar to the sliding window approach and is the same as the case where the window size parameter is selected as 5. When the pen parameter is 0.01 and 0.1, the data is divided into regions of the most minor lengths and then shows a uniform distribution. In the case where the Pen parameter is 0.01, the number of small regions of the data increases compared to the case where the Pen parameter is 0.1. When the Pen is set to 1, the two-peak distribution shifts to the right of the x-axis. The fact that the first peak has an enormous length value indicates that it divides the data into regions of greater length.

The ability to accurately predict the start and end points of the most minor events is essential when dividing the data into regions as events. Therefore, the pen parameter with the most minor first peak is the parameter value that can most accurately predict the start and end points. Due to the repetitive nature of the events in the SWaT data set, the number of regions detected as events after a certain length is close to zero.

3.2.3. Sampler

The time between the beginning and end of physical events is dissimilar. For this reason, the data slices obtained in the event detection phase are of different lengths. Processing data of various lengths within the same graph model is impossible. A data set consisting of the same length representing different events is required.

The Sampler module converts vectors of different lengths into vectors of a fixed length by sampling them at the appropriate rate. The sampling rate is calculated as the ratio of the original vector length to the target vector length (1). The function's main purpose is to convert vectors of different lengths into vectors of a given target length and, thus, to organize the data conveniently with the same length.

$$Ratio = original\quad vector\quad length / destination\quad vector\quad length \tag{1}$$

Depending on the sampling rate, the function works in two ways: down-sampling and up-sampling. In the case of down-sampling (ratio>1.0), the original vector length is greater than the output vector length and must be reduced. In this scenario, we decrease the sampling rate and combine the difference between the highest and lowest elements with the average of each group that corresponds to the given target sampling rate (2). This ensures that the impact of the maximum and minimum elements within each group is considered and reflected in the final output.

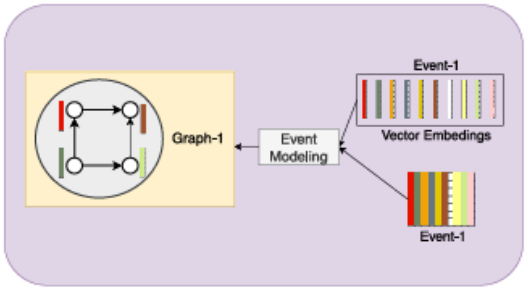


Fig. 5. Event Modeling.

$$Sampled\quad signal[i] = \frac{1}{ratio}\sum_{j=i*ratio}^{ratio*(i+1)} signal[j] + [\max(signal[j, ratio * (i+1)]) - \min(signal[j, ratio * (i+1)])] \Big/ 2 \tag{2}$$

The original vector length is smaller than the output vector length and must be increased when up-sampling (ratio<1.0). In this case, the sample rate is increased, and a group of data points corresponding to each target sample rate is taken from the original data set to create new sample data (3).

$$Sampled\quad signal[i] = signal[i * ratio] \tag{3}$$

3.2.4. Event modeling

The data slices obtained in the event detection phase have specific correlations. Detecting these correlations and modeling the relationship between them using proportional values is one of the most appropriate methods of modeling the structure of events. The data slices detected as events are processed in the event modeling phase and converted into a graph (Fig. 5).

In the event modeling phase, three different operations are performed: creating graphs, creating nodes, and creating edges that model the relationships between nodes. A graph is created for each data slice called an event. The features in the data slice are represented as nodes in the graph. Relationships between features are added to the graph as edges. The resulting graphs are models of the events that occur in the system. The pseudocode of the event modeling phase is shown below (Algorithm 1).

Algorithm 1. Event Modeling

(continued on next page)

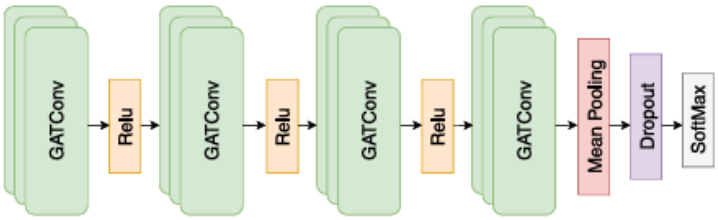


Fig. 6. Graph Learning Structure.

(*continued*)

Algorithm 1 Event Modeling

```
set G to Graph
for x in each columns:
    for y in each columns:
        call Correlation(x,y) return similarity
        if similarity>treshold then
            call Sampler(x) return x_embed
            call Sampler(y) return y_embed
            call CreateNode(G,x,x_embed)
            call CreateNode(G,y,y_embed)
            call CreateEdge(G,x,y,similarity)
        end if
    end for
end for
```

3.2.5. Creating graph

As a result of event detection, data slices are obtained that represent a physical event in the time domain. Each data slice represents a physical event in the time domain. A graph is created to represent each data slice. The graph takes the dominant label value in the data slices as the graph label.

The graph representation of multimodal time series can be used for the analysis of different modalities jointly. More meaningful graphs are obtained using event detection instead of the sliding window technique to transform data into graphs. Thus, the graph structure is independent of the size of the time series, and the data size is reduced.

3.2.6. Creating node

The nodes in the graph need to be labeled and have unique properties. These properties and labels are used to represent data as a graph. A node is created for each characteristic in the data. The node label is the name of the feature in the data. The node's features are derived from the univariate time series represented by the feature in the corresponding data.

3.2.7. Creating edge

In the graph, edges represent the existence of a relationship between nodes and the degree of importance of the relationship. The relationship between the features represented as nodes in the data slice is expressed by edges in the graph structure.

The data slice estimated as an event is analyzed independently from other data slices. The similarity ratio represents the relationship of each feature in the data slice with other features. The similarity ratio is calculated as the correlation between the two features being analyzed. If the absolute value of the similarity ratio is greater than 0.5, an edge is placed between the nodes, assuming a relationship between the features. A weight value represents the intensity of the relationship between the two nodes in the created edge. If the similarity ratio is less than 0.5, it is assumed that there is no relationship between the features. Lack of correlation is represented by no edges being added between nodes in the graph structure.

After the relationship analysis of all the features in the data slice, the transformation into the graph structure is complete. In the graph structure, nodes with degree zero are not accessible from other nodes. This indicates that the corresponding feature has no meaning in the data slice identified as an event. Therefore, nodes with degree zero are deleted from the graph.

3.2.8. Graph structure learning

Multimodal time series have multiple related events with different characteristics. In multimodal time series expressed as graphs, multiple relationships, and potential correlations need to be learned with artificial intelligence. In this study, a graph learning structure is developed using the Graph Attention Convolution (GATConv) layer Veličković et al., 2017, called the GAT algorithm's convolution process. The GATConv is a convolution layer that models relationships in graph structures using the attention mechanism. The attention mechanism determines the importance of each node to other nodes.

In this study, a graph learning structure is designed using the GATConv layer created by adapting the attention structure and the ReLU activation function. A visualization of the designed learning structure is given in Fig. 6.

The attention mechanism computes attention weights by combining the feature vectors and the parameter vector. These weights represent the importance of nodes to other nodes. Attention weights consider node properties and graph structure to determine the relationship of each node to other nodes. In the general formulation, the model allows each node to pay attention to any node while ignoring structural information. The masked attention mechanism is computed only with the first-degree neighbors of the node.

To compute the attention mechanism, this study used a single-layer feedforward neural network parameterizing the weight vector and a nonlinear activation function. When calculating the attention weight between two nodes (node n_i, node n_j), $\vec{h}_i$ is the feature vector of node n_i, W is the weight matrix used to transform the features of the nodes. In operation $\|$, the feature vectors $W\vec{h}_i$ and $W\vec{h}_k$ are concatenated. LeakyReLU assigns a low slope corresponding to negative inputs, while traditional ReLU leaks for values less than zero. The attention weight between nodes n_i and n_j is calculated by performing this calculation for all neighbors of node n_i. (4)

$$\alpha_{ij} = \frac{\exp\left(LeakyReLU\left(\vec{a}^T\left[W\vec{h}_i \| W\vec{h}_j\right]\right)\right)}{\sum_{k\in n_i}\exp\left(LeakyReLU\left(\vec{a}^T\left[W\vec{h}_i \| W\vec{h}_k\right]\right)\right)} \tag{4}$$

The features of node n are updated by crossing the features of its neighbors with the attention coefficient. The attention coefficient of node n with each neighbor node is denoted by α_{ik}. h_k denotes the feature of each neighbor of node i. W is the weight matrix used to transform the feature in the nodes. In this calculation, the attention coefficient

between the node and its neighbors is used to obtain the features of the neighboring nodes. The attention coefficient of the node (α_{ii}) itself is multiplied by its own feature and summed with the node features from its neighbors. Finally, we obtain the feature vector of node i updated with the attention coefficient.

$$h_i = \alpha_{ii} \quad Wh_i + \sum_{k \in n_i} \alpha_{ik} \quad Wh_k \tag{5}$$

The ReLU activation function stabilizes the output value to learn the nonlinear features. In graph data, each node has its own features, and these features are updated considering the neighborhood structure. In mean pooling, the features of each node are used to obtain a single feature of the graph. (6)

$$h_{graph} = \frac{1}{n} \sum_{i=1}^{n} h_i \tag{6}$$

The vectors from the mean pooling layer are transferred to the dropout layer to mitigate the risk of overfitting and enhance generalization performance. Dropout randomly eliminates network connections temporarily while the network learns, which prevents overfitting in the learning process. (7)

If r is a randomly generated value between 0 and 1, and p represents the dropout probability:

$$h_i = \begin{cases} \frac{h_i}{p}, r_i > p \\ 0, otherwise \end{cases} \tag{7}$$

The vectors from the dropout layer are transformed into classification probabilities through the logsoftmax layer, producing a probability distribution for each class generated by the model. The argmax function is utilized to select the class index with the highest probability, resulting in determination of the most suitable class from the graph structure outputs. (8)

$$argmax(y) = \underset{i}{argmax} \quad y_i \tag{8}$$

The process of graph learning structure involves taking the features of nodes in a graph and computing new features by considering the features of neighboring nodes. This method ensures that node neighborhoods' importance and properties are preserved. Once the features of the nodes are weighted, a feature vector representing the graph is calculated. Probability calculation is then utilized to determine whether the graph is abnormal or normal.

4. Experiments

In this section, we perform extensive experiments to demonstrate the effectiveness of EA-GAT. First, we introduce the SWaT dataset, which is at the forefront of anomaly detection in cyber-physical systems and uses devices and attacks that fit a real-world scenario. Next, we demonstrate the environment set up to evaluate EA-GAT. Finally, we present the performance metrics used in the related works and their meaning.

4.1. Dataset

Cyber-physical systems combine equipment and communication technologies used in information systems with physical control and monitoring equipment used in operational technologies. In cyber-physical systems, SWaT (Goh et al., 2016), WADI (Ahmed et al., 2017), and MSL (Hundman et al., 2018) datasets from real-world applications are widely used.

The SWaT data set is an experimental study that includes events and scenarios that can occur in a real-world industrial water treatment plant. The SWaT dataset contains 11 days of data, seven days under normal operating conditions, and four days under attack scenarios. It includes

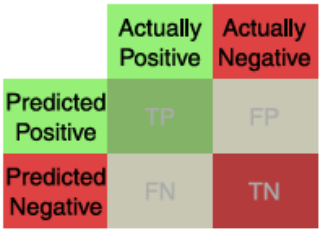


Fig. 7. Confusion Matrix.

normal operating conditions and potential attacks that may occur in water treatment systems. It contains 51 different attributes, including many measurement and control data such as water quality, flow rate, pressure levels, concentrations of chemical constituents, etc. The features contain some relationships within and among themselves. These correlations represent many water treatment processes, such as filling the tanks, mixing particular chemicals, filtering, and dechlorinating the water.

The Water Distribution (WADI) dataset was obtained from a water distribution test bed that operated continuously for 16 days as a real-world application. The WADI dataset is built with data from 123 sensors and actuators operating under 15 different attack scenarios applied for 2 days, 14 days normal.

The datasets from the Mars Science Laboratory rover (MSL) spacecraft are actual data collected in the real world. These datasets are described by NASA experts and consist of pre-segmented training and test sets. The training set is constructed from normal data, while the test set includes labeled anomalies. The dataset comprises 55 unlabeled features.

4.2. Experimental setup

The proposed model was developed on an Ubuntu server with Intel (R) Core (TM) i7–1280 P and Nvidia RTX 3050 Ti hardware. CUDA 11.6, Pytorch 1.13, and Pytorch Geometric 2.2 (Paszke et al., 2017) libraries were used in the development stage. The Adam optimizer was used in the training phase, and the learning rate parameter was set to 1×10^{-3}. The whole network was trained in about 30 epochs with batch size 128. The embedding dimension was set to 128. The pen value used in the event detection module was obtained experimentally.

4.3. Evaluation metrics

The metrics we discuss in this paper are the primary measures used to evaluate the performance of classification models. Using a confusion matrix is valuable for a more in-depth understanding and analysis of a classification model's performance. The confusion matrix (Fig. 7) allows us to examine the classification results of the model from different perspectives. This matrix is based on four basic terms: true positive (TP), false positive (FP), true negative (TN), and false negative (FN). TP indicates that the model correctly predicts actual positive examples, while FP demonstrates that the model makes wrong positive predictions. TN indicates that the model correctly identifies actual negative samples, while FN indicates that the model makes false negative predictions. The different metrics calculated using these terms allow us to better measure the model's performance.

The confusion matrix uses basic terms to calculate accuracy, precision, recall, and F1 score. The confusion matrix helps us better understand the weaknesses and strengths of the model by allowing us to determine how accurately or inaccurately the model predicts which class. Therefore, using the confusion matrix when evaluating performance will enable us to perform a more in-depth analysis of the results.

Table 2
Graph Size Resulting from Different Parameters and Methods on SWaT.

Methods	Graph Count
Line Graphs	1(450.000 node)
Sliding Window, Window Size =5	90.000
Sliding Window, Window Size =10	45.000
Sliding Window, Window Size =100	4.500
Event Aware, Pen=1	3.055
Event Aware, Pen=0.1	10.825
Event Aware, Pen=0.01	9.940
Event Aware, Pen=0.001	89.984

$$Acc = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{9}$$

$$Prec = \frac{TP}{(TP + FP)} \tag{10}$$

$$Rec = \frac{TP}{(TP + FN)} \tag{11}$$

$$F1 = \frac{2 * (Prec * Rec)}{(Prec + Rec)} \tag{12}$$

Accuracy (Acc), Precision (Prec), Recall (Rec), and F1 score(F1) were used as evaluation metrics. Accuracy is a metric often used to evaluate the performance of artificial intelligence models. It is calculated as the ratio of correct predictions to the total data. It shows the rate of correct detection of normal and abnormal situations. It is important when the data set is balanced. Precision is calculated as the ratio of actual positively predicted samples to all positively predicted samples in classification models. If this ratio is low, the system will classify normal behavior as anomalous and halt system operations. Therefore, it is an important parameter in anomaly detection systems. Recall is calculated as the ratio of samples predicted to be positive by classification models to all samples that are actually positive. It indicates the rate at which anomaly cases are correctly detected. It is the most critical metric to examine in low anomaly rate datasets such as SWaT.

5. Result

The results obtained in EA-GAT are categorized under two main headings: anomaly detection and anomaly analysis. In the anomaly detection section, the results obtained from event-driven graph formation, the results of the EA-GAT study using different parameters, and the results shared in other studies are presented together. In the anomaly analysis section, the two regions are detected as events in the time interval, and the correlations between the properties of each region and the definition of the correlations in the physical process are evaluated.

5.1. Anomaly detection

There are various methods for converting multimodal time series into graphs, such as line graphs and sliding window methods. Each second corresponds to a node in a line graph, and features are embedded in that node. The time series length is equal to the number of nodes in the graph structure. The fact that an edge connects the graphs in only one direction does not make much sense in the graph structure. The sliding window technique transforms the data set of a given window width into a graph. The number of graphs created is the ratio of the time series length to the window size. Since no fixed window size can cover events of different lengths, relational losses may occur during graph construction.

With the proposed event detection mechanism, events of different lengths are segmented according to their start and end times. This allows the meaningful part of the data to be analyzed and expressed as a graph without fragmentation. The number of graphs produced according to the methods and parameters used to convert time series into graphs based on SWaT dataset is listed in Table 2.

Table 3
Sliding window & Event Aware Performance Metrics on SWaT.

Methods	Parameters	Accuracy	Precision	Recall	F1-Score
Sliding Window	Window size= 5	93.98	98.98	83.99	90.87
	Window size= 10	94.90	99.65	85.16	91.83
	Window size= 100	91.00	96.88	79.49	87.32
Event Aware	Pen=1	86.67	63.64	63.64	63.64
	Pen=0.1	86.21	88.89	81.16	84.84
	Pen=0.01	**96.21**	**97.82**	**96.97**	**97.40**
	Pen=0.001	93.64	97.78	83.12	89.85

Various studies in the literature have shown that the performance of the anomaly detection system is significantly affected by the use of a fixed-length window size during the graph transformation stage of multimodal time series. The EA-GAT method has been observed to perform better than the sliding window approach when graphs generated using different window sizes are submitted to the graph structure learning stage. Moreover, when the sliding window approach is evaluated independently, changing the window size does not lead to significant changes in the performance metrics, indicating that the data's internal dependencies are lost when segmented.

On the other hand, the EA-GAT method uses a dynamic approach to data segmentation and relies on a precision parameter called the pen value to create data slices. This parameter is system-specific and is the only parameter that needs to be adjusted in the EA-GAT model. During the setup or training phase, the parameter is chosen only once, and its accuracy is maintained throughout the system's life. Changing the pen parameter alters the start and end points of the event region and the number of events. The following table, namely Table 3, presents a detailed analysis of the influence of altering the pen parameter on the SWaT dataset's anomaly detection performance.

The graphs generated by the sliding window and event aware approaches were tested in the same graph structure learning. It is observed that the sliding window approach, which takes samples of fixed length, significantly lags behind the event aware approach, which takes samples by detecting events. The event aware approach tested event detection sensitivity with different pen parameters. When the pen parameter is set to 0.001, fixed-length events are generated. In this case, it was observed that the same graph structure was formed as in the sliding window approach when the window size was selected as 5. Comparing the two cases where the pen parameter is selected as 0.1 and 0.01, it can be seen that the number of graphs is quite close to each other. However, it was observed that the pen parameter selected as 0.1 was not suitable for detecting events in the SWaT dataset. It is seen that the case where the pen parameter is selected as 0.01 can correctly detect the events in the SWaT dataset, and the detected events are converted into graphs in a way that is very compatible with the graph structure learning structure.

In previous research, it has been observed that fixed-length sliding window approach is commonly utilized by graph-based techniques to analyze time series data from Cyber-Physical Systems. The studies focused on analyzing the data transformed into graphs with different graph-based methods. When the success rates of the prominent methods in the literature are analyzed, it is seen that the EA-GAT method outperforms all other methods in the F1 score and gives similar results in other metrics. A high F1 score indicates that a classification model strikes a balance between recall and precision, thus trying to minimize both false positives and false negatives. The F1 score is important for evaluating classification performance on unbalanced data sets, such as detecting rare anomalies. The high F1 score of the EA-GAT model shows that it tries to minimize both false positives and false negatives. It achieves a good balance. This shows its success in anomaly detection in

Table 4
Compared EA-GAT with Competing Methods.

Method	SWaT			WADI			MSL		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
AE (Provotar et al., 2019)	72.63	52.63	61.03	34.35	34.35	34.35	71.66	50.08	58.96
PCA (Shyu et al., 2003)	24.92	21.63	23.16	39.53	5.63	9.86	29.37	24.14	26.50
USAD (Audibert et al., 2020)	98.51	66.18	79.17	64.51	32.20	42.96	93.08	89.17	91.08
OmniAnomaly (Su et al., 2019)	98.25	64.97	78.22	99.47	12.98	22.96	88.67	91.17	89.90
DAGMM (Zong et al., 2018)	27.46	69.52	39.37	54.44	26.99	36.09	49.11	55.62	52.16
GDN (Deng and Hooi, 2021)	99.35	68.12	80.82	97.50	40.19	56.92	91.35	86.12	88.66
MAD-GAN (Li et al., 2019)	98.97	63.74	77.54	41.44	33.92	37.30	85.17	89.91	87.48
LSTM-VAE (Park et al., 2018)	96.24	59.91	73.85	87.79	14.45	24.82	52.57	95.46	67.80
MST-GAT (Ding et al., 2023)	98.73	72.41	83.55	98.24	43.51	60.31	95.06	89.10	91.98
EA-GAT	**97.82**	**96.97**	**97.40**	**75.00**	**59.26**	**66.20**	**92.85**	**92.85**	**92.85**

Table 5
Compared EA-GAT with Competing Methods on SWaT.

Method	Prec	Rec	F1
RPCA (Al-Dhaheri et al., 2022)	100	86.10	92.50
STAE-AD (Macas and Wu, 2019)	96.0	81.50	88.0
1D CNN (Kravchik and Shabtai, 2018)	96.80	79.10	87.10
TABOR (Lin et al., 2018)	88.60	78.80	80.30
GAT (Lyu et al., 2022)	70.70	60.31	65.09
PCA-GLIN-GCA (Lyu et al., 2022)	96.55	95.97	96.26
GRU (Tang et al., 2023)	99.86	59.09	74.96
FATRAF (Truong et al., 2022)	93.89	97.75	95.78
EA-GAT	**97.82**	**96.97**	**97.40**

cyber-physical systems, which can minimize system failures due to wrong decisions and service interruptions due to undetected attacks. The comparison of performance metrics between the proposed EA-GAT model and other works in the literature is shown in Table 4.

Table 4 summarizes the comparison of EA-GAT and prominent studies in the literature on three datasets in terms of precision, recall, and F1-score. The results show that EA-GAT outperforms the existing works on the three datasets compared to the F1-score. Multivariate anomaly detection was carried out by MAD-GAN (Li et al., 2019) using LSTM-RNN and GAN structure to account for hidden variables. However, the recall value fell short due to the inability to learn some of the hidden connections. OmniAnomaly (Su et al., 2019) overcomes this challenge by employing stochastic variable linkage and planar normalization flow techniques to establish strong connections to normal cases. Ignoring these connections may result in low recall values, particularly in complex datasets like WADI. DAGMM (Zong et al., 2018) simultaneously optimizes the parameters of the deep autoencoder and the mixture model in an end-to-end fashion and leverages a separate prediction network to facilitate parameter learning of the mixture model. USAD (Audibert et al., 2020) and GDN (Deng and Hooi, 2021) outperform the other studies. However, GDN needs to improve at extracting temporal features from time series. USAD ignores spatial correlation in multimodal time series, and problems can arise when the underlying spatial dependence is complex. MST-GAT (Ding et al., 2023) is a notable exception to graphing-based anomaly detection systems which use all data unaltered. The MST-GAT method employs a sliding window technique for time series sampling, which does not consider the occurrence of events. This approach leads to incomplete representation of the system and events in the generated graphs and has a negative impact on performance metrics as well. EA-GAT stands out among other notable baseline studies with its exceptional performance. It achieved F-1 scores that were at least 5.89% and 13.85% higher than the WADI and SWaT datasets, respectively. Additionally, the MSL dataset also displayed better F-1 scores than other baseline studies.

The SWaT dataset contains various scenarios that reflect the complexity of industrial control systems and real-world conditions. For this reason, it has been observed that the SWaT dataset has been used in most studies in the field of CPS. Table 5 shows other prominent studies using only the SWaT dataset.

Among the studies with the best performance on the SWaT dataset, the GLIN, GRU, and FATRAF methods stand out with particularly impressive results. The GLIN (Lyu et al., 2022) method splits dataset into two subsets. The first subset contains negative and positive samples obtained when the anomaly occurred. The second subset consists of positive and negative samples collected after the occurrence of an anomaly. The GAT (Lyu et al., 2022) method divides the dataset into two subsets. However, successful results could not be obtained as in GLIN (Lyu et al., 2022). The GAT method could not learn spatio-temporal relationships because the edge weight was incorrectly modeled. However, grouping the data according to the anomaly removes the specificity of the time series and disrupts the flow of events over time. Therefore, if the GLIN method uses the dataset without splitting it, its success becomes questionable. Furthermore, the possibility of an anomaly occurring unexpectedly in tests or real-world scenarios invalidates the grouping rules used. The GRU (Tang et al., 2023) method observes that only attack number 36 is used in the SWaT dataset. The FATRAF (Truong et al., 2022) method applies feature selection by considering the correlations between each data point. This feature selection significantly reduces the computational burden and shortens the learning time. However, anomalies occurring in cyber-physical systems belong to the rare anomaly class. In high-dimensional cyber-physical systems, feature selection can improve immediate results, but its ability to detect long-term anomalies is poor (Drias et al., 2015) because of the eliminated feature. For example, 51 features in the SWaT dataset were reduced to 16 features by feature selection using the FATRAF method. In the moment, it can be assumed that 16 features represent 51 features, but this assumption may not be true in the long term. For this reason, it is argued in the literature that the high success of the FATRAF method cannot be maintained in long-term use and test scenarios. EA-GAT analyzes system changes and generates simple patterns from spatio-temporal correlations, significantly improving performance compared to other methods.

The proposed EA-GAT method can successfully detect anomalies in the system using graphs modeled with a dynamic approach. Thanks to its ability to detect events in the system obtain from the change of data, it models the current state as a graph using data sets related to each other. Dynamic approach yields more success than fixed sample lengths in graph models, as seen in literature. Fixed sampling lengths in studies interfere with distinguishing rare anomalous states from normal states of the system. The nested states are particularly characterized by low recall in anomaly detection studies. The recall value indicates the ability of the system to accurately detect the rare cases that are usually the system's focus. Therefore, a high recall indicates success in detecting anomalies in anomaly detection studies. Precision means how many of the samples the system correctly labels as positive are anomalies. That is, a high precision value indicates that the cases the system labels as anomalies have a high probability of actually being anomalies. The recall and precision values should be evaluated together for a correct performance evaluation. There is usually an inverse relationship between recall and

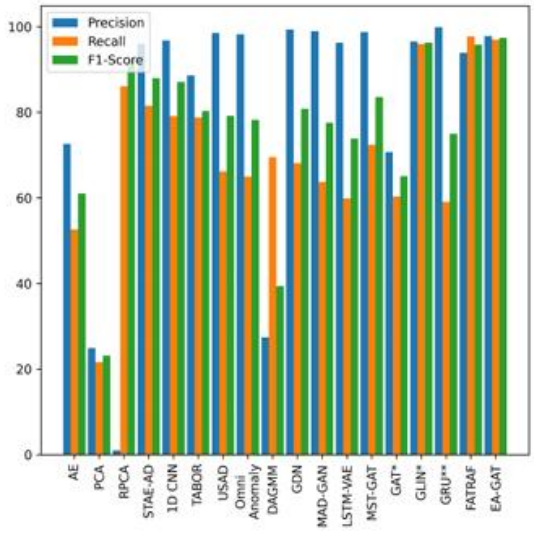


Fig. 8. Performance Comparison on SWaT.

precision. This results in low precision values in models with high recall. In the EA-GAT model, the carefully tuned pen parameter correctly segmented the data regions called events in the system. As a result of the correct modeling of the relationships within the data regions called events and their evaluation with the designed graph learning structure, the proposed model achieved high recall and precision values. The results from the SWaT dataset, which has been used in numerous studies, are displayed graphically in Fig. 8.

5.2. Anomaly analysis

The SWaT dataset originates from a water treatment plant that can produce 5 gallons of water per minute in 6 stages. (Fig. 9) The P1 process obtains, stores, and transfers raw water to the other stages. The P2 process, also known as the pre-treatment process, is responsible for maintaining the water quality. If the water quality is not within the desired limits, chemicals are added. The P3 process eliminates any unwanted substances from the water. The P4 process removes chlorine from the water. The water entering the P5 process is filtered, and reverse osmosis is used. At P6, the water is stored and made available for use. Tanks are used in the P1, P3, P4, and P6 processes, which ensures that the water flow is not interrupted in case of short-term anomalies.

Between 12/28/2015 at 12:15:50 PM and 12/28/2015 at 2:09:59 PM, the event detection module detected two different event regions. The relationship between the features was analyzed for each region detected as an event. The relationship between the features analyzed for two various events is shown in Fig. 10. When the regions separated as events were analyzed, it was seen that event-2 was the data region corresponding to attack-8. In attack-8, the design limits of the DPIT-301 component are changed. Fig. 10 clearly shows that it disrupts the current system relationship.

Comparing the matrices of event 1, which is estimated to be normal system behavior, and event 2, which is known to be an anomaly, it can be seen that the existing relationships of the DPIT-301 component have changed significantly. It is seen that the relationships of DPIT-301 with FIT-301, MV-302, and P-302 have significantly decreased or even disappeared altogether, while its insignificant relationship with the AIT-401 component has become more critical. The component affected by the attack is identified by taking advantage of the significant changes that occur.

When the relationship matrix is analyzed in detail, the following changes are observed:

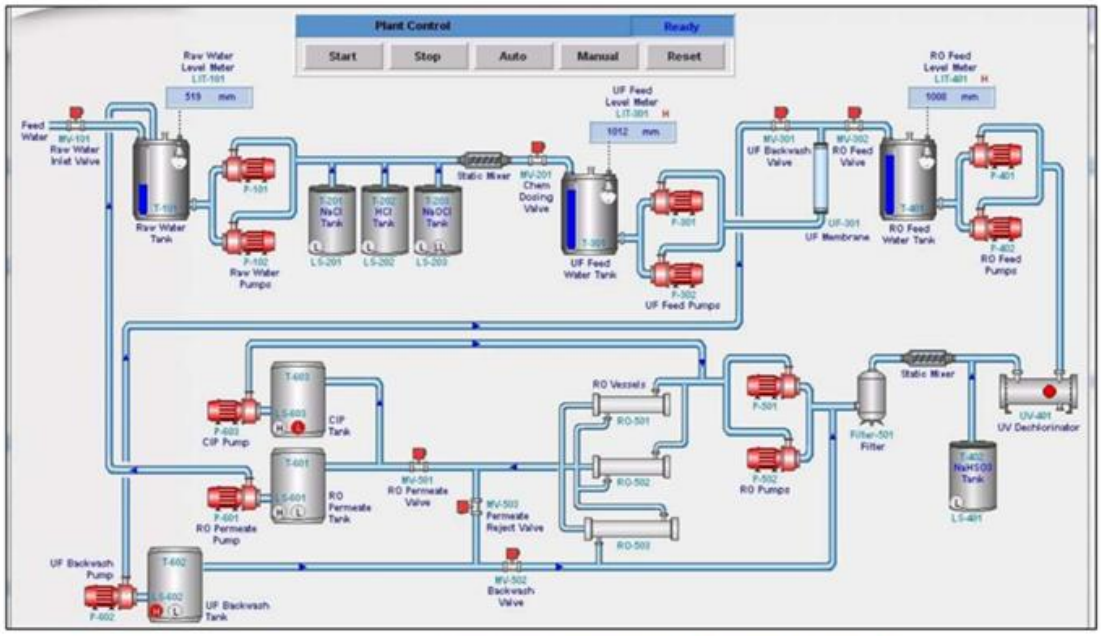


Fig. 9. SWaT Operation Structure (Secure Water Treatment (SWaT) Testbed Technical Details, 2023).

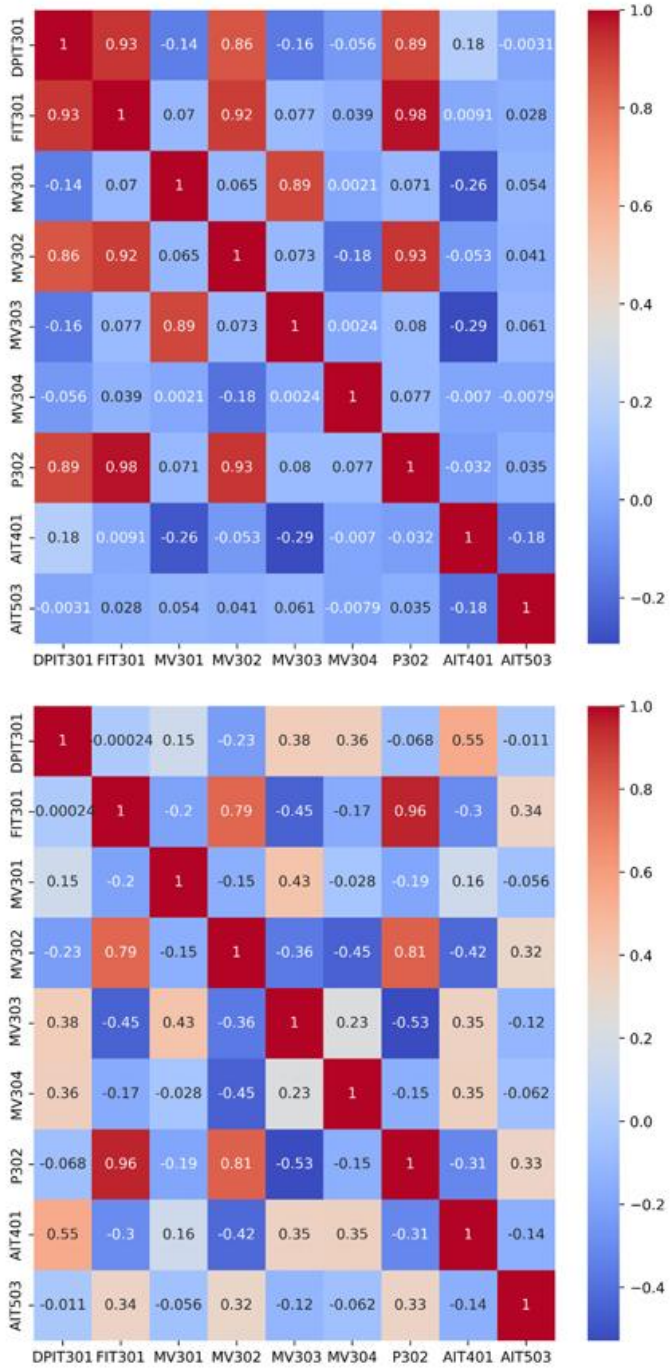


Fig. 10. Relationship Matrix (Above is Event-1, below is Event-2).

Fig. 11. Graph Model of Event (Left is Event-1, right is Event-2).

- DPIT-301 component's relationship with FIT-301, which was 0.93, decreased to −0.00024, its relationship with MV-302, which was 0.86, decreased to −0.23, and its relationship with P-302, which was 0.89, decreased to 0.55. The correlation with AIT-401 increased from 0.18 to 0.55.
- The correlation of FIT-301 with MV-302 decreased from 0.92 to 0.79.
- The correlation of MV-301 with MV-303 decreased significantly from 0.89 to 0.43.
- The correlation of MV-303 with P-302, which was 0.08, decreased significantly to −0.53.

Changes in the relationship coefficients change the graph shapes and edge weights resulting from event modeling. In the event modeling phase, the absolute value of the values in the relationship matrix is taken and the cases greater than 0.5, which is set as the threshold value, are expressed as edges. Therefore, the edges (DPIT-301, FIT-301), (DPIT-301, MV-302), (DPIT-301, P-302), (MV-301, MV-303) in the normal state are not present in the graph formed in the attack state. In addition, (DPIT-301, AIT-401) and (MV-303, P-302) edges that are not present in the normal behavior state appear in the graph modeling the attack. The graphs for the two event regions are shown in Fig. 11.

The graph named Event −1 shows that DPIT-301, MV-302, FIT-301, and P-302 are connected. Also, only MV-303 is connected to MV-301. All the nodes have a relationship with P3. The sensors and actuators used in the P3 stage were analyzed in the physical process. The sensor DPIT-301 measures the pressure at the membrane inlet. FIT-302 measures the flow rate of water pumped by P-302. MV-302 valve transfers water from the P3 stage to the P4 stage. There is also a connection between MV-301, which controls the water from P6, and MV-303, used for draining. An attack on the DPIT-301 sensor affects P-302, which pumps water to the membrane, FIT-302, which measures the amount of water pumped, and the MV-302 valve at the membrane outlet. It can be seen that the graph structure created as Event-1 corresponds to the operation structure in Fig. 9.

In the graph called Event-2, P-602 pumps the water sent from stage P6 to P3, and FIT-601 measures the flow rate. The water from P6 is transferred to P3 via MV-301. So, there is a connection between these three components. P-302 pumps the water produced in P3, and the flow is measured by FIT-301 and transferred by MV-302. MV-303 combines the water from P6 via MV-301 and the water from P3 via MV-302 and sends it to P4. At the same time, there is a connection between DPIT-301, which controls the pressure in P3, and the AIT-401 sensor, which measures the hardness of the water. Since the pressure of the water leaving the P3 stage is regulated, the pressure difference should not affect P4. In addition, there is no physical relationship between the hardness of the water. For these reasons, the connection made by the DPIT-301 sensor in Event 2 has no equivalent in the physical process.

The changes in the system because of the attack on the DPIT-301 component are shown in Fig. 12. The time interval during which the attack occurred is indicated by the red-shaded area. In the area outside the red shade, the relationships detected in event 1 are present. From the first moments of the attack, it was observed that the existing relationships were broken. The attack resulted in the formation of new relationships and the emergence of unprecedented behaviors within the system.

6. Conclusion

In this study, we introduce EA-GAT, a novel approach that models events in cyber-physical systems via a dynamic graph generation approach and detects anomalies in the system using the generated models. EA-GAT models systems as graphs employing a stateful architecture that integrates data-driven and design-centric approaches. Instead of the typically employed sliding window approach, the event detection module is utilized for graph system modeling to produce relevant data slices of varying lengths. The event detection module identifies physical events in the system, which groups significant correlations based on the data relationships. The event modeling module's relational analysis transforms correlation groups into graphs. The event detection and modeling modules enable the synthesis of objective design information regarding the events in the system, without requiring system-specific data. Only pertinent and coherent data slices are converted into graphs, resulting in the same system being represented with 5–10 times fewer graphs than alternative methods in the literature. This study showcases the remarkable success of the EA-GAT method using SWaT, WADI, and MSL datasets. Compared to existing studies, the EA-GAT model outperformed its predecessors by achieving impressive F-1 scores of 66.20% and 92.85% on WADI and MSL datasets, respectively. Notably, the EA-GAT model's F-1 score of 97.40% is unparalleled compared to other studies that focus on the SWaT dataset. The F-1 score is the most reliable criterion for measuring the effectiveness of anomaly detection systems, as it provides a thorough assessment of their ability to identify real anomalies. A high F-1 score proves that the EA-GAT model minimizes type-1 and type-2 errors, successfully detects system anomalies, and minimizes system interruptions caused by false detections.

Moreover, the graph model analysis proved that the EA-GAT model can detect the components affected by an attack. By analyzing the graph model of a state detected as an attack, it becomes evident that it differs from the normal state graphs of the system. Upon analyzing the differing nodes in the graph, we can discern the components that are attack targets and components affected by the attack. Moving forward, our plan is to optimize the model to use fewer graphs and consume less memory.

CRediT authorship contribution statement

Muhammed Ali AYDIN: Methodology, Supervision, Writing – review & editing. **MEHMET Yavuz YAGCI:** Methodology, Software, Visualization, Writing – original draft, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial

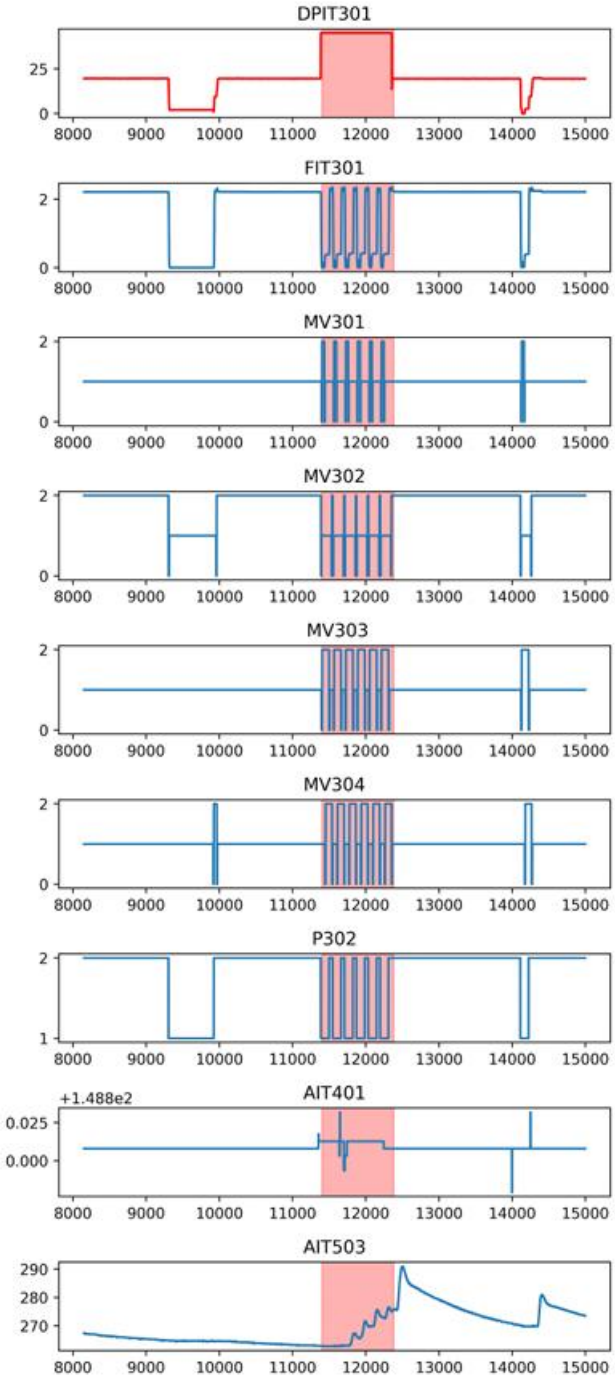


Fig. 12. System Data (The red shaded area indicates the system under attack).

interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

This work is a part of the Ph.D. thesis titled "Design Secure Architecture for Critical Infrastructure" at the Institute of Graduate Studies, Istanbul University-Cerrahpasa, Istanbul, Turkey. The study is supported by Istanbul University-Cerrahpasa Scientific Research Projects Commission as project number "36754". We would like to thank the Istanbul University-Cerrahpasa Scientific Research Projects Unit for their support.

References

C.M. Ahmed , V.R. Palleti , and A.P. Mathur , a water distribution testbed for research in the design of secure cyber physical systems, Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks, 2017, [Online]. Available: ⟨https://api.semanticscholar.org/CorpusID:16245468⟩.

Adepu, S., Mathur, A., 2021. SafeCI: Avoiding process anomalies in critical infrastructure. Int. J. Crit. Infrastruct. Prot. 34 https://doi.org/10.1016/j.ijcip.2021.100435.

Al-Asiri, M., El-Alfy, E.S.M., 2020. On using physical based intrusion detection in SCADA Systems (Elsevier B.V.). Procedia Comput. Sci. 34–42. https://doi.org/10.1016/j.procs.2020.03.007.

Al-Dhaheri, M., Zhang, P., Mikhaylenko, D., 2022. Detection of cyber attacks on a water treatment process (Elsevier B.V.). IFAC-Pap. 667–672. https://doi.org/10.1016/j.ifacol.2022.07.204.

J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M.A. Zuluaga, USAD: UnSupervised Anomaly Detection on Multivariate Time Series, in Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, in KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 3395–3404. doi: 10.1145/3394486.3403392.

Balla, A., Habaebi, M.H., Islam, M.R., Mubarak, S., 2022. Applications of deep learning algorithms for Supervisory Control and Data Acquisition intrusion detection system. Elsevier Ltd, Aug. 01 Clean. Eng. Technol. vol. 9. https://doi.org/10.1016/j.clet.2022.100532.

Das, T.K., Adepu, S., Zhou, J., 2020. Anomaly detection in industrial control systems using logical analysis of data. Comput. Secur vol. 96. https://doi.org/10.1016/j.cose.2020.101935.

A. Deng and B. Hooi, Graph Neural Network-Based Anomaly Detection in Multivariate Time Series, Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 5, pp. 4027–4035, May 2021, doi: 10.1609/aaai.v35i5.16523.

Ding, C., Sun, S., Zhao, J., 2023. MST-GAT: a multimodal spatial–temporal graph attention network for time series anomaly detection. Inf. Fusion vol. 89, 527–536. https://doi.org/10.1016/j.inffus.2022.08.011.

Z. Drias, A. Serhrouchni, and O. Vogel, Taxonomy of attacks on industrial control protocols, in 2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS), IEEE, Jul. 2015, pp. 1–6. doi: 10.1109/NOTERE.2015.7293513.

Farag, A., Abdelkader, H., Salem, R., 2022. Parallel graph-based anomaly detection technique for sequential data. J. King Saud. Univ. - Comput. Inf. Sci. vol. 34 (1), 1446–1454. https://doi.org/10.1016/j.jksuci.2019.09.009.

Faramondi, L., Flammini, F., Guarino, S., Setola, R., 2021. A hardware-in-the-loop water distribution testbed dataset for cyber-physical security testing. IEEE Access vol. 9, 122385–122396. https://doi.org/10.1109/ACCESS.2021.3109465.

Gauthama Raman, M.R., Mathur, A.P., 2022. AICrit: A unified framework for real-time anomaly detection in water treatment plants. J. Inf. Secur. Appl. vol. 64 https://doi.org/10.1016/j.jisa.2021.103046.

Goh, J., Adepu, S., Junejo, K.N., Mathur, A.P., 2016. A dataset to support research in the design of secure water treatment systems. Crit. Inf. Infrastruct. Secur. ⟨https://api.semanticscholar.org/CorpusID:3075307⟩.

K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding, in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, in KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 387–395. doi: 10.1145/3219819.3219845.

M. Kravchik and A. Shabtai, Detecting Cyber Attacks in Industrial Control Systems Using Convolutional Neural Networks, in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, in CPS-SPC '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 72–83. doi: 10.1145/3264888.3264896.

L(y)u, S., Wang, K., Zhang, L., Wang, B., 2022. Global-local integration for GNN-based anomalous device state detection in industrial control systems. Expert Syst. Appl. vol. 209 https://doi.org/10.1016/j.eswa.2022.118345.

D. Li, D. Chen, L. Shi, B. Jin, J. Goh, and S.-K. Ng, MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks, in International Conference on Artificial Neural Networks, 2019. [Online]. Available: ⟨https://api.semanticscholar.org/CorpusID:58007096⟩.

Q. Lin, S. Adepu, S. Verwer, and A. Mathur, TABOR: A Graphical Model-Based Approach for Anomaly Detection in Industrial Control Systems, in Proceedings of the 2018 on Asia Conference on Computer and Communications Security, in ASIACCS '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 525–536. doi: 10.1145/3196494.3196546.

M. Macas and C. Wu, An Unsupervised Framework for Anomaly Detection in a Water Treatment System, in 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), 2019, pp. 1298–1305. doi: 10.1109/ICMLA.2019.00212.

Macas, M., Wu, C., Fuertes, W., 2022. A survey on deep learning for cybersecurity: Progress, challenges, and opportunities. Elsevier B.V., Jul. 20 Comput. Netw. vol. 212. https://doi.org/10.1016/j.comnet.2022.109032.

Miele, E.S., Bonacina, F., Corsini, A., 2022. Deep anomaly detection in horizontal axis wind turbines using Graph Convolutional Autoencoders for Multivariate Time series. Energy AI vol. 8. https://doi.org/10.1016/j.egyai.2022.100145.

Monzer, M.H., Beydoun, K., Ghaith, A., Flaus, J.M., 2022. Model-based IDS design for ICSs. Reliab Eng. Syst. Saf. vol. 225 https://doi.org/10.1016/j.ress.2022.108571.

Nai Fovino, I., Coletta, A., Carcano, A., Masera, M., 2012. Critical state-based filtering system for securing SCADA network protocols. IEEE Trans. Ind. Electron. vol. 59 (10), 3943–3950. https://doi.org/10.1109/TIE.2011.2181132.

Nedeljković, D., Jakovljević, Z., 2022. CNN based method for the development of cyber-attacks detection algorithms in industrial control systems. Comput. Secur vol. 114. https://doi.org/10.1016/j.cose.2021.102585.

Park, D., Hoshi, Y., Kemp, C.C., 2018. A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder. IEEE Robot Autom. Lett. vol. 3 (3), 1544–1551. https://doi.org/10.1109/LRA.2018.2801475.

A. Paszke *et al.*, Automatic differentiation in PyTorch, 2017.

O.I. Provotar, Y.M. Linder, and M.M. Veres, Unsupervised Anomaly Detection in Time Series Using LSTM-Based Autoencoders, in 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT), 2019, pp. 513–517. doi: 10.1109/ATIT49449.2019.9030505.

Secure Water Treatment (SWaT) Testbed Technical Details. Accessed: Oct. 19, 2023. [Online]. Available: ⟨https://itrust.sutd.edu.sg/itrust-labs-home/itrust-labs_swat/⟩.

M.-L. Shyu, S. Chen, K. Sarinnapakorn, and L. Chang, A Novel Anomaly Detection Scheme Based on Principal Component Classifier, 2003. [Online]. Available: ⟨https://api.semanticscholar.org/CorpusID:6319694⟩.

Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, [Online]. Available: ⟨https://api.semanticscholar.org/CorpusID:196175745⟩.

Tabassum, A., Erbad, A., Lebda, W., Mohamed, A., Guizani, M., 2022. FEDGAN-IDS: privacy-preserving IDS using GAN and Federated Learning. Comput. Commun. vol. 192, 299–310. https://doi.org/10.1016/j.comcom.2022.06.015.

Tang, C., Xu, L., Yang, B., Tang, Y., Zhao, D., 2023. GRU-based interpretable multivariate time series anomaly detection in industrial control system. Comput. Secur vol. 127. https://doi.org/10.1016/j.cose.2023.103094.

Truong, H.T., et al., 2022. Light-weight federated learning-based anomaly detection for time-series data in industrial control systems (Sep). Comput. Ind. vol. 140. https://doi.org/10.1016/j.compind.2022.103692.

Umer, M.A., Junejo, K.N., Jilani, M.T., Mathur, A.P., 2022. Machine learning for intrusion detection in industrial control systems: applications, challenges, and recommendations. Elsevier B.V., Sep. 01 Int. J. Crit. Infrastruct. Prot. vol. 38. https://doi.org/10.1016/j.ijcip.2022.100516.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, Graph Attention Networks, Oct. 2017, [Online]. Available: ⟨http://arxiv.org/abs/1710.10903⟩.

Wambui, G.D., Gichuhi, A.Waititu, Wanjoya, A.K., 2015. The power of the pruned exact linear time(PELT) test in multiple changepoint detection. Am. J. Theor. Appl. Stat. vol. 4, 581. ⟨https://api.semanticscholar.org/CorpusID:4647879⟩.

Wang, J., Lai, Y., Liu, J., 2022. Stealthy attack detection method based on Multi-feature long short-term memory prediction model. Future Gener. Comput. Syst. vol. 137, 248–259. https://doi.org/10.1016/j.future.2022.07.014.

B. Zong *et al.*, Deep autoencoding gaussian mixture model for unsupervised anomaly detection, in International conference on learning representations, 2018.