Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 270 (2025) 3913–3923

Procedia Computer Science

www.elsevier.com/locate/procedia

29th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2025)

Towards Graph-based Self-learning of Industrial Process Behaviour for Anomaly Detection

Ankush Meshram[a,*], Markus Karch[b], Christian Haas[b], Jürgen Beyerer[a,b]

[a] *KASTEL Security Research Labs, Vision and Fusion Laboratory (IES), Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany*
[b] *Information Management and Production Control, Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB), 76131 Karlsruhe, Germany*

Abstract

The increasing sophistication of cyber threats targeting industrial control systems (ICS) necessitates advanced anomaly detection techniques capable of identifying attacks by analyzing industrial process data exchange. This paper addresses the challenge of representing and learning the spatio-temporal characteristics of industrial network communication as Graph for self-learning anomaly detection. We propose a novel framework that models the spatio-temporal characteristics as *graph snapshots* and applies Graph Neural Networks (GNNs) for anomaly detection. Each *graph snapshot* captures the structural and temporal dynamics of PROFINET-based industrial traffic, with edge features encoding the *payload transitions*, *timing intervals*, and *cycle counter differences*. We evaluate the performance of an isotropic *Graph Convolutional Network (GCN)* and anisotropic GNN variants – *Message Passing Neural Network (MPNN), Gated Graph ConvNet (GatedGCN) and Graph Transformer (GT)* – for the task of graph classification on real-world datasets from a miniaturized deterministic production plant. Our evaluation results demonstrate that anisotropic GNN models (MPNN, GatedGCN, GT) achieve complete anomaly detection (*specificity*) while maintaining perfect recall on normal behavior (*sensitivity*). In contrast, the isotropic GCN fails to distinguish between normal and anomalous states of the miniaturized plant. These findings highlight the efficacy of encoding spatio-temporal characteristics on graph edges and the capability of anisotropic GNNs to learn complex process behaviors for anomaly detection in the industrial networks of the evaluated production plant.

Keywords: industrial cybersecurity; network security; graph learning; anomaly detection; graph classification;

1. Introduction

A *Graph* is an ubiquitous representation of systems where the interactions between the constituent objects reflect functionality of the system. The objects are represented as *nodes/vertices* and the directional or mutual interac-

* Corresponding author.
E-mail address: ankush.meshram@kit.edu

tions/relations are represented by the *directed* or *undirected edges* between components. Anomaly detection on graph data has been an active research area [1]. Especially, graph-based anomaly detection finds its application in Cyber-security for spam and malware detection [6, 2] in Web network, socware (malware in social networks) detection in Social networks [23], and cyber-attacks and intrusion detection in computer networks [29, 15].

An industrial communication network can be represented as a *graph*, where industrial components and networking assets constitute nodes and the communication relationships between them are represented as edges of the graph. The process data communication relationships between industrial components could be modeled in a graph and used for process data analysis to detect anomalies in the industrial process. The network topology remains *static* however the values on the edges are *dynamic*. The structural information and temporally changing edge information offers insights into *spatio-temporal* characteristics of industrial networks.

Threat actors target the *spatial* and *temporal* characteristics (the periodic and deterministic requirements) of an industrial process through violating *integrity* and *availability* requirements of industrial communication [24]. The threat behaviours in the traffic could be detected efficiently through Graph-based analysis *only* when the industrial network's *Graph* representation aptly *represents* the *spatio-temporal* characteristics of its communication. After an appropriate *spatio-temporal* characteristics representation on a graph is available, a Machine Learning (ML) model can *learn* the *normal* process behaviours and detect the deviations as *anomalies* [20].

In a *self-learning* process behaviour analysis of industrial networks, the *spatio-temporal* characteristics are extracted from the industrial traffic and represented as a graph. A ML model learns the normal process behaviour from the normal operations of industrial process, and the model is utilized for anomaly detection in process data exchange. Graph Neural Networks (GNNs) [19] are the ML models exclusively developed for graph data analysis. In recent years, GNN have gained attention for analysis of graph-structured data and have shown to perform better in graph classification, node classification, edge classification and link prediction tasks [30]. Its applications in Cybersecurity for botnet detection [31], malware detection [5] and network intrusion detection [22] have been reported.

In the presented work, for a given *PROFINET*-based industrial system (the *Festo Demonstrator* [20]), its process data exchange is represented as a graph, as shown in Fig. 1. The edges represent *Connections* between industrial components and changing edge values represent process data exchange. A *spatio-temporal* graph representation is required to capture the *spatio-temporal* characteristics extracted from *PROFINET* traffic. Feasibility of GNN to model process behaviour based on the *spatio-temporal* graph representation for anomaly detection needs an evaluation. The research problem of *self-learning* the process behaviour in Graph represented data is formulated as follows:

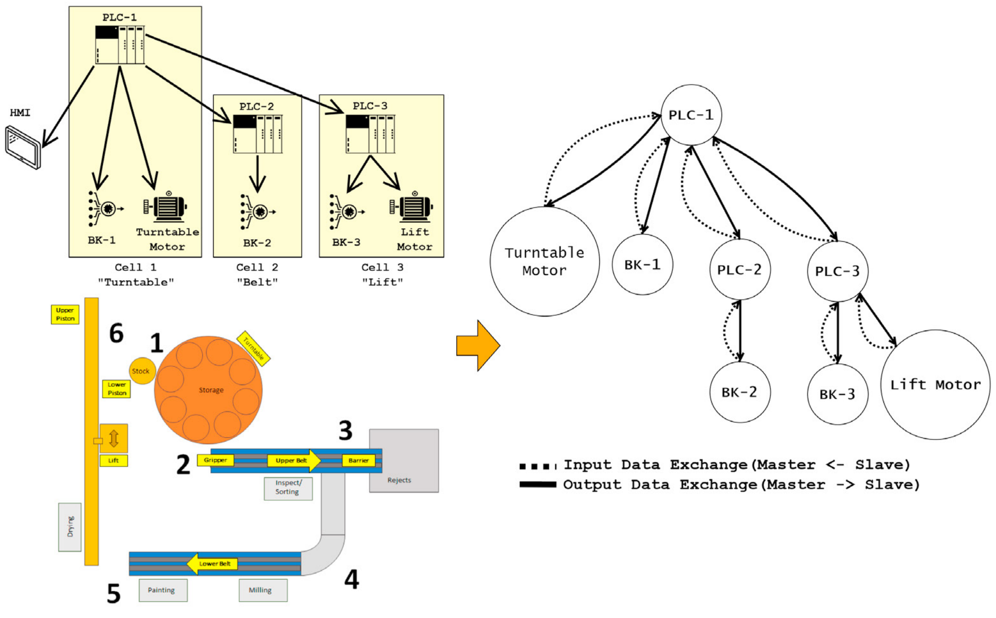


Fig. 1: Graph representation of the *Festo Demonstrator*'s industrial network communication.

- **RQ-1.** How to model *spatio-temporal* characteristic of industrial communication in Graph representation?
- **RQ-2.** How to learn Graph-represented process behaviour for anomaly detection?

In the next sections, the foundation on GNNs is provided in Section 2. The proposed solutions for the representation and self-learning of *spatio-temporal* characteristics of an industrial process as a *Graph* are described in Section 3 and Section 4, respectively. The paper is reviewed in comparison to related work in Section 5 followed with discussion and conclusion in Section 6.

2. Foundations

Graphs offer a flexible way to represent structural data, with nodes denoting entities and edges representing interactions. They may be homogeneous or heterogeneous, depending on the types of nodes and edges involved. Traditional graph-based ML methods rely on hand-crafted features or kernel similarities, but often struggle with scalability and generalization [13]. Graph representation learning addresses these challenges by using encoder-decoder frameworks to learn low-dimensional embeddings that capture both local and global structure [16], enabling more effective downstream tasks through Graph Neural Networks (GNNs), which have shown strong performance in domains such as cybersecurity and industrial systems [5].

GNNs are a neural network framework for graph data with encoders defined over the graph structure and its attributes. The motivation of utilizing neural networks for graph data is found in the works of Bruna et al. [4] to generalize convolutions to non-Euclidean data, Dai et al. [7] defining a differentiable variant of belief propagation and Hamilton et al. [14] outlining graph isomorphism tests. At its core, a GNN employs a form of *neural message passing*, where for a given graph G=($\mathcal{V}$, $\mathcal{E}$) and its node features X $\in \mathbb{R}^{|\mathcal{V}|\times d}$, the information as vector messages are exchanged between nodes and updated through neural networks to generate node embeddings $z_u, \forall u \in \mathcal{V}$ [12]. In message-passing iteration of GNN, a *hidden embedding* h_u for each node $u \in \mathcal{V}$ is updated with information aggregated from u's graph neighbourhood $\mathcal{N}(u)$. The message-passing update process [13] can be summarized as:

$$h_u^{(k+1)} = UPDATE^{(k)}(h_u^{(k)}, AGGREGATE^{(k)}(\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\})) = UPDATE^{(k)}(h_u^{(k)}, m_{\mathcal{N}(u)}^{(k)}) \tag{1}$$

where, $UPDATE$ and $AGGREGATE$ are differentiable functions and $m_{\mathcal{N}(u)}$ is the "*message*" containing aggregated information from node u's graph neighbourhood $\mathcal{N}(u)$. The superscripts represent embeddings and functions at different iterations of message passing, or different layers of GNN.

2.1. Basic Intuition

At each iteration/layer k of GNN, every node aggregates embeddings of its local neighbouring nodes in the graph and updates its own embeddings. As the iterations progresses, the information of the further reaches in the graph are aggregated into node's embeddings. The initial embedding for a node u is its input features i.e. at $k = 0, h_u = x_u, \forall u \in \mathcal{V}$. At $k = 1$, node u's 1-hop neighbourhood is aggregated in $h_u^{(1)}$ i.e. all the nodes that are immediate neighbours with graph path length 1. At $k = 2$, $h_u^{(2)}$ contains 2-hop neighbourhood information, and so on. After K iterations of message-passing and updates, the final layer outputs the embeddings of each node, $z_u = h_u^{(K)}, \forall u \in \mathcal{V}$. It summarizes *structural information* such as node degrees and *feature-based information* of nodes in its K-hop neighbourhood. The local neighbourhood information aggregation is analogous to convolutional operation in Convolutional Neural Network, where feature information is aggregated over local graph neighbourhood instead of spatially-defined patches in an image [8].

2.2. Basic GNN

The abstract definition of message-passing in a GNN described so far can be translated to a general instantiation in two classes as follows:
(a) *isotropic GNN*, where every neighbouring edge is treated equally in the update equation:

$$h_u^{(k)} = \sigma(W_{self}\, h_u^{(k-1)} + W_{neigh} \sum_{v\in\mathcal{N}(u)} h_v^{(k-1)})$$

(b) *anisotropic GNN*, where every neighbouring edge is weighted different:

$$\begin{aligned} h_u^{(k)} &= \sigma(W_{self}\, h_u^{(k-1)} + W_{neigh}\ \textstyle\sum_{v\in\mathcal{N}(u)} \eta_{uv}\, h_v^{(k-1)}) \\ \eta_{uv} &= f^{(k-1)}(h_u^{(k-1)}, h_v^{(k-1)}) \end{aligned} \tag{2}$$

where $W_{self}, W_{neigh} \in \mathbb{R}^{d^{(k)}\times d^{(k-1)}}$ are trainable parameter matrices, σ is a non-linear element-wise activation such as *ReLU* or *tanh*, and f is a parametrized function on edge between nodes u and v.

Different GNN instantiations of *isotropic* and *anisotropic* GNN, which are evaluated for process data analysis, are outlined in Section 4.

2.3. Graph Pooling

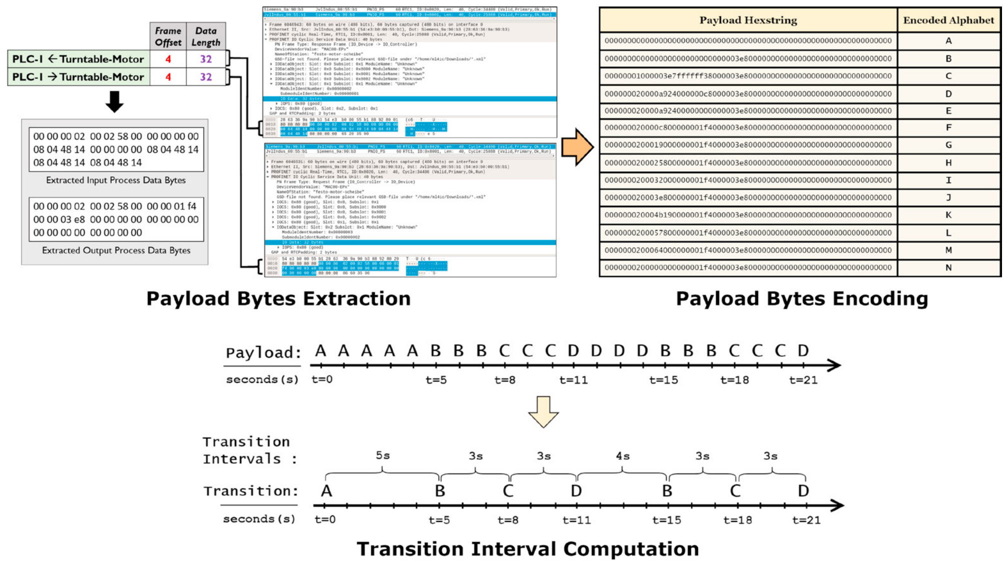
After K-iterations, the node embeddings $z_u, \forall u \in \mathcal{V}$, for a graph are computed. However, in a graph-level task, an embedding over the graph G, z_G is required. The graph embedding can be learnt through pooling the node embeddings over final GNN layer together to calculate their sum (or mean) [13]:

$$z_G = \frac{\sum_{u\in\mathcal{V}} z_u}{f_n(|\mathcal{V}|)},$$

where f_n is the normalizing function.

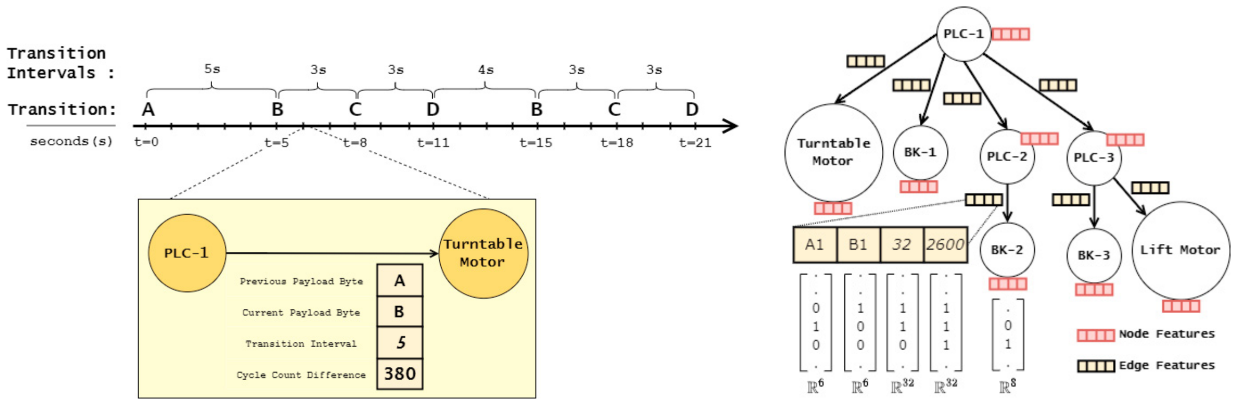
3. Spatio-temporal Graph Representation

In order to capture the *spatio-temporal* characteristics of an industrial system's network communication, sequential steps of Process Payload Profiling Framework (P3F) [20] from *Payload Byte Extraction* to *Transition Interval Computation* are employed, as shown in Fig. 2. The transition order of the alphabet-encoded process payload bytes corresponds to the *spatial* characteristics and the transition interval represents the *temporal* characteristics. In the current case of graph representation of industrial network communication, we encode the *spatio-temporal* characteristics of process data exchange over the edges of *Connection* between industrial components (IOController, IODevice), as shown in Fig. 3a. Additionally, the network transmission characteristics are added as the *Cycle Counter* value differences between transitioning process payload bytes. The cycle counter value in a *PNIO* frame serves the purpose


Frame Offset
Data Length
PLC-I ←Turntable-Motor 4 32
PLC-I →Turntable-Motor 4 32
00 00 00 02 00 02 58 00 00 00 00 00 08 04 48 14 00 00 00 00 08 04 48 14 08 04 48 14 08 04 48 14
Extracted Input Process Data Bytes
00 00 00 02 00 02 58 00 00 00 01 f4 00 00 03 e8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Extracted Output Process Data Bytes
Payload Bytes Extraction
Payload Bytes Encoding
Payload: A A A A A B B B C C C D D D D B B B C C C D
seconds(s) t=0 t=5 t=8 t=11 t=15 t=18 t=21
Transition Intervals : 5s 3s 3s 4s 3s 3s
Transition: A B C D B C D
seconds(s) t=0 t=5 t=8 t=11 t=15 t=18 t=21
Transition Interval Computation

Payload Hexstring	Encoded Alphabet
0000000000000000000000000000000000000000000000000000000000000000	A
00000000000000000000000000003e800000000000000000000000000000000	B
00000001000003e7ffffff38000003e800000000000000000000000000000000	C
000000020000a924000000c8000003e800000000000000000000000000000000	D
000000020000a92400000000000003e800000000000000000000000000000000	E
000000020000c800000001f4000003e800000000000000000000000000000000	F
0000000200019000000001f4000003e800000000000000000000000000000000	G
0000000200025800000001f4000003e800000000000000000000000000000000	H
0000000200032000000001f4000003e800000000000000000000000000000000	I
000000020003e800000001f4000003e800000000000000000000000000000000	J
000000020004b190000001f4000003e800000000000000000000000000000000	K
0000000200057800000001f4000003e800000000000000000000000000000000	L
0000000200064000000001f4000003e800000000000000000000000000000000	M
0000000200000000000001f4000003e800000000000000000000000000000000	N

Fig. 2: The Payload Byte Encoding steps from P3F. [20]



(a) Modeling of spatio-temporal industrial network characteristics (b) Binary Encoding of node and edge features.

Fig. 3: Spatio-temporal industrial network characteristics modeling on Graph.

of up-to-dateness determination of frame and detect duplicate frames. Based on the deterministic time interval in process data transmission, the cycle counter values are incremented in fixed iterations and reset after the cycle is complete. The difference in cycle counter values of transition process payload bytes encapsulates either transition from higher counter valued frames to lower counter frames (-ve) or vice-versa (+ve). In a nutshell, as shown in Fig. 3a, the edge features of the *Connection* between components is defined as tuple (`previous payload byte, current payload byte, transition interval, cycle counter difference`).

In order to represent the *spatio-temporal* characteristics of the process, a *graph snapshot* is defined on a fixed time window over the traffic, shown in Fig. 4. The *graph snapshot* contains the *homogeneous* graph representation of the industrial system's network communication, where features of each *Connection* edge are updated as per transition information of the window. The window size is defined as number of *Output Connections* of the industrial network.

Significance of encoding. Within the *graph snapshot*, the correlation of process payload bytes between *Connections* are encoded. It encodes following implicit information: *(1)* the relation between the current payloads of all *Connections*, and *(2)* the relation between *intra-Connection* transitions, along with their *temporal* transition intervals.

Especially, when the transition of payload bytes in one of the *Connection* remains persistent for long time, its effect on the behaviour of other *Connections* can be easily captured. In theory, stealthy attacks on industrial processes

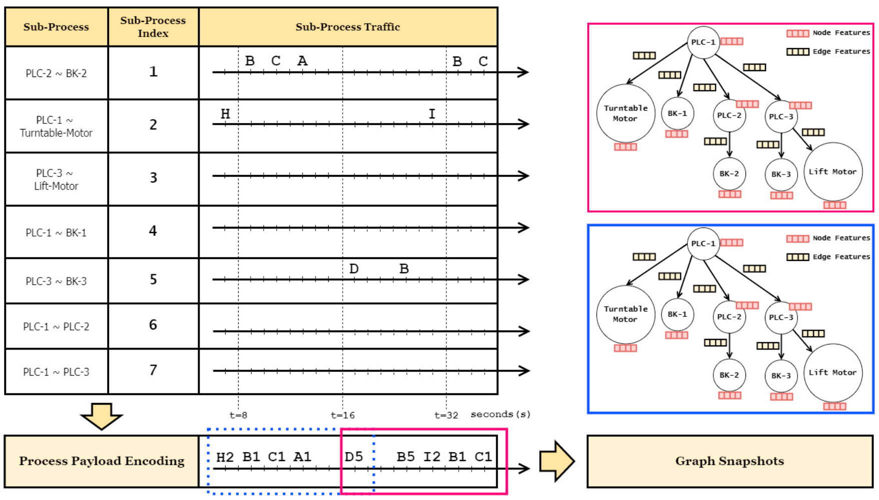


Fig. 4: Graph Snapshots over the encoded traffic of industrial process.

through manipulation of *Cycle Counters* in *PROFINET* traffic outlined by Ferrari et al. [10] can be detected. The cycle counter differences encoded over the edge will have inconsistent values compared to normal behaviour.

Binary Encoding. For representing numerically the encoded process payload byte and devices, one-hot encoding is incorporated. For encoding the values of intervals and counter differences, each *float* value is *32-bit* encoded. For example, A1 and B1 alphabet-encoded transitioning payload bytes for *Connection PLC-2˜BK-2* are one-hot encoded over the edges, as shown in Fig. 3b. The interval values 32.0 *seconds* and counter differences 2600 are binary encoded. The edge feature for every *Connection* in the *graph snapshot* has length 76 (6 + 6 + 32 + 32), and node feature size 8.

4. Graph-based Process Data Analysis

The industrial network communication characteristics are represented as *Graphs*, where edges represent logical network connection between industrial components for process data exchange. GNN are the ML models being employed to learn the characteristics. They learn the structural information of logical network topologies and process exchange traffic characteristics encoded over the edges in edge features, as outlined before. The ML task for GNN is to classify *graph snapshots* for *normal* or *abnormal* industrial process behaviour (*graph classification*). Different GNN architectures are evaluated for their performances on learning the *spatio-temporal* representation of normal process behaviour, and capability to distinguish anomalies triggered by process data targeted attack.

4.1. GNN Variants

There have been multiple GNN variants developed in recent years varying in their $UPDATE$ and $AGGREGATE$ functions. In this work, *Graph Convolutional Network* (GCN) for an isotropic GNN, and *Message Passing Neural Network* (MPNN), *Gated Graph ConvNet* (GatedGCN) and *Graph Transformer* (GT) anisotropic models have been employed for proposed solution. The selection of anisotropic models is driven by their usage of edge attributes for GNN computation and implementation availability amongst the published GNN architectures.

(a) Graph Convolutional Network (GCN) is the baseline GNN model, outlined by Kipf and Welling [16], that employs symmetric normalization for isotropic aggregation of neighbourhood information of a node along with its own information.

(b) Message Passing Neural Network (MPNN) is a GNN framework, outlined by Gilmer et al. [12], that operates on node features $x_u, u \in \mathcal{V}$ and edge features $e_{uv}, e_{uv} \in \mathcal{E}$ of a given graph to generate node embeddings.

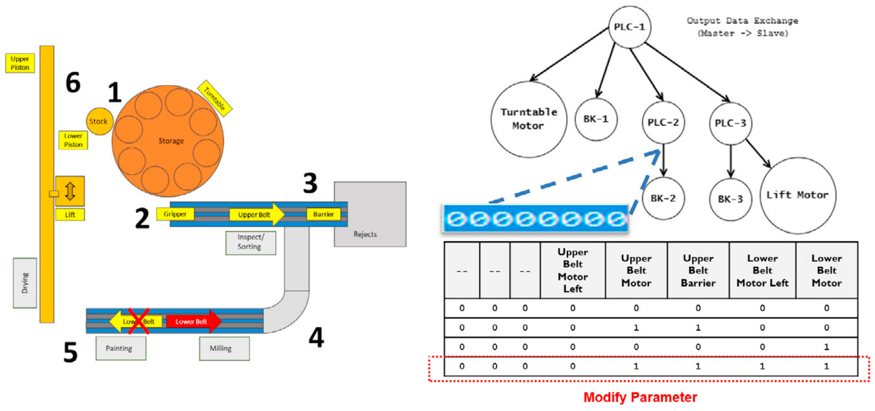
(c) Graph Transformer (GT) outlined in Shi et al. [25] adopts multi-head attention mechanism from *Transformer* architecture of Vaswani et. al. [27], with inclusion of edge features for graph learning. An "*attention*" defines the weight of an edge in a node's neighbourhood as per its influence in aggregation computation.

(d) Gated Graph ConvNet (GatedGCN) is the GNN architecture proposed by Bresson and Laurent [3] leveraging the vanilla graph ConvNet architecture of Sukhbaatar et al. [26] and the edge gating mechanism of Marcheggiani and Titov [18]. Unlike other anisotropic GNN models, GatedGCN explicitly update edge features along with node features at each iteration.

4.2. Dataset

Negative Sampling. The *spatio-temporal* modeling of network traffic contains only the normal behaviour of industrial network communication. For a ML model to self-learn the normality in absence of any known abnormality, the data with abnormal characteristics are generated artificially. They can be generated in multiple ways for a given domain, which is the case for current work. The following amongst the many other ways are chosen to generate abnormal characteristics data in the presented study: (1) abnormal transition order of payload bytes for *inter-Connection* transitions, (2) abnormal transition intervals outside the observed interval ranges of normal transitions, and (3) abnormal cycle counter differences outside the observed ranges of normal transitions.

Training Dataset. The training dataset consists of *graph snapshots* extracted over data captured from normal behaviour [20]. Every *graph snapshot* is a graph consisting of 8 nodes and 7 edges with feature size 8 for a node and 76 for an edge. The negative sampled data was added to weigh in abnormalities. The dataset was split into 80:20 ratio for train and validation for training of GNN models:



--	--	--	Upper Belt Motor Left	Upper Belt Motor	Upper Belt Barrier	Lower Belt Motor Left	Lower Belt Motor
0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1

Fig. 5: The anomaly use case of *modifying parameter*.

- Training Dataset: 1139 normal snapshots, 261 negative-sampled abnormal snapshots
- Validation Dataset: 489 normal snapshots, 113 negative-sampled abnormal snapshots

Test Dataset. To evaluate the anomaly detection performance, the traffic captured while executing network attacks is used for extracting snapshots. The anomaly use case is outlined in Fig. 5. The process parameters on the output data exchange for *Connection PLC-2˜BK-2* are modified, which causes *Lower Belt* to move in reverse direction. The faulty process payload byte transition is encoded over the edge between *PLC-2* and *BK-2* of the *graph snapshot*. All the *graph snapshots* containing the faulty payload byte transition are considered *abnormal*, otherwise *normal*. The test dataset contains 41 normal snapshots and 47 abnormal/anomalous snapshots. Every *graph snapshot* is a graph consisting of 8 nodes and 7 edges with feature size 8 for a node and 76 for an edge.

4.3. Analysis Pipeline

Dwivedi et al. [9] outlined an experimental pipeline for benchmarking of GNN, shown in Fig. 6, and it has been incorporated in this work. The graph and its features are given as input to the *Input Layer*, followed by convolutional computations in *GNN Layer* and graph classification task in the *Prediction Layer*.

Input Layer. The node features $\alpha_i \in \mathbb{R}^{a\times 1}$ for every node i and edge features $\beta_{ij} \in \mathbb{R}^{b\times 1}$ for every edge between nodes i and j are linearly projected to d-dimensional *hidden features* $h_i^{l=0}$ and $e_{ij}^{l=0}$, respectively:

$$h_i^0 = W_1^0\alpha_i\,;\; e_{ij}^0 = W_2^0\beta_{ij}\; where, W_1^0 \in \mathbb{R}^{d\times a},\;\; W_2 \in \mathbb{R}^{d\times b}$$

GNN Layer. Each GNN Layer computes hidden features for each node and edge of the graph through recursive message passing in the local neighbourhood of each node. $\mathcal{L}$ GNN layers are stacked for $\mathcal{L}$-hop neighbourhood message passing computations. For the *graph snapshots*, $\mathcal{L}$ is defined by the *depth* of *graph snapshot* to allow message diffusion throughout the network. In the presented work and the industrial system under consideration, we employ 2 GNN layers for 2-hop neighbourhood computation, refer Fig. 7.

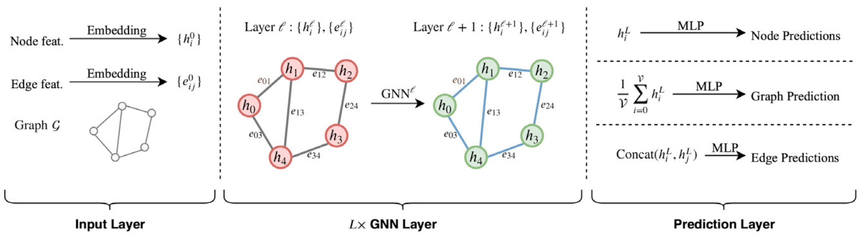

Node feat.
Embedding
$\{h_i^0\}$
Edge feat.
Embedding
$\{e_{ij}^0\}$
Graph $\mathcal{G}$
Layer ℓ : $\{h_i^\ell\}, \{e_{ij}^\ell\}$
Layer $\ell+1$: $\{h_i^{\ell+1}\}, \{e_{ij}^{\ell+1}\}$
GNN$^\ell$
h_i^L MLP Node Predictions
$\frac{1}{\mathcal{V}}\sum_{i=0}^{\mathcal{V}} h_i^L$ MLP Graph Prediction
Concat(h_i^L, h_j^L) MLP Edge Predictions
Input Layer
$L\times$ GNN Layer
Prediction Layer

Fig. 6: The GNN analysis pipeline adapted from Dwivedi et al. [9].

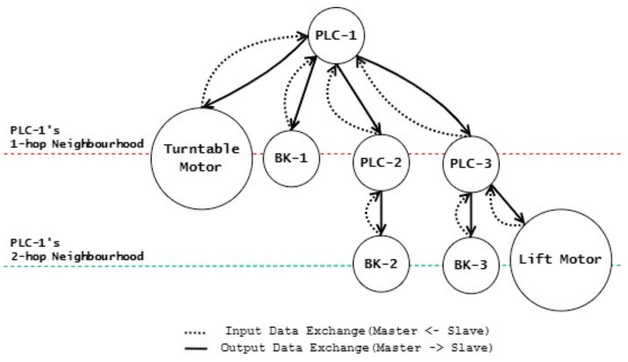

PLC-1
PLC-1's 1-hop Neighbourhood
Turntable Motor
BK-1
PLC-2
PLC-3
PLC-1's 2-hop Neighbourhood
BK-2
BK-3
Lift Motor
Input Data Exchange (Master <- Slave)
Output Data Exchange (Master -> Slave)

Fig. 7: 2-hop neighbourhood of the *Festo Demonstrator*'s logical network structure.

Prediction Layer. We pool final node embeddings of all the nodes to embed the graph's features, refer Section 2.3. A Multi-Layer Perceptron (MLP) is employed over the graph embeddings z_G for the graph classification task.

4.4. Implementation Frameworks

The analysis pipeline was realized in *Python* utilizing PyTorch Geometric (PYG) [11] and Deep Graph Library (DGL) [28] libraries for GNN architecture layers. Particularly, PyG implementations of *GCN*, *MPNN*, *GT* and DGL implementation for *GatedGCN* are utilized. Both the libraries require their own representation of datasets, hence, train and test snapshot datasets were converted into respective library formats. In addition, the libraries work over the *batches* of dataset for training and evaluation of GNN models.

In order to regularize the model parameters over different batches of dataset, *batch normalization* (BN) is augmented to each GNN layer. For every iteration of message passing,

$$h_u^{(k+1)} = UPDATE^{(k)}(BN(h_u^{(k)}, AGGREGATE^{(k)}(\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\}))$$

where, $UPDATE$ and $AGGREGATE$ are differentiable functions, $\mathcal{N}(u)$ is node u's graph neighbourhood and h_u, h_v are node embeddings.

Binary Cross Entropy (BCE) is utilized as the loss function to train GNN models for the graph classification task, which is defined as:

$$BCE = -t_1 \log(\sigma(s_1)) - (1 - t_1)\log(1 - \sigma(s_1))$$

where, t_1 and s_1 are ground truth and classification score of the sample for class C_1 of binary classes $\{C_1, C_2\}$, and σ is the sigmoid function.

Settings. *Specificity* is used to measure the GNN model's performance on detecting anomalies, whereas *Sensitivity* measures recall performance on normal *graph snapshots*. The GNN models are computed in the *Colab Pro* environment with a *Tesla P100-PCIE-16GB* GPU offered by Google Research[1].

4.5. Hyperparameter Optimization

The parameters of the learning process and model parameters for different configurations of every GNN architecture are evaluated with the Training Dataset, to get the best model settings for each GNN architecture. The Table 1 summarizes all the parameters which were optimized to get the best model configurations for each GNN architecture. For different configuration instances of each GNN architecture, the one with best performance on the Validation Dataset with high *Sensitivity* measure is chosen as the best model of the GNN architecture.

The number of *epochs* is the number of times the training dataset is passed forward and backward through the GNN. When the epochs are very low, the model is *underfitting* as the GNN has not learnt enough. On the other hand,

[1] https://colab.research.google.com/

Table 1: Summary of hyperparamter optimization of GNN models.

Parameters	**Value Options**	**Best Model Parameter Value**			
		GCN	**MPNN**	**GT**	**GatedGCN**
Epochs	[*300*]	60	200	100	150
Batch Size	[*16, 32, 64*]	16	32	16	32
Learning Rate	[*0.1, 0.01, 0.001, 0.0001*]	0.1	0.01	0.001	0.001

the model might *overfit* the learning for too high epochs. The *batch size* defines the number of sub-samples from training data as input to the GNN before the weights are updated. A bigger batch size slows the learning process, whereas the lower batch size fastens the learning process. The *learning rate* controls the frequency of updating the weights at the end of each batch. A very small learning rate results in slow converging of the model, and the too large learning rate would diverge the model.

4.6. Evaluation Performance

The GNN architecture models with optimized parameter are evaluated on the Test Dataset, for their performances on detecting *anomalies* and recalling the learnt *normal* behaviour of the industrial process. The GNN models are primarily evaluated for their ability to learn the representation of *spatio-temporal* characteristics as *graph snapshots* as described earlier and to discern abnormal from normal. The dataset contains 41 normal snapshots and 47 abnormal/anomalous snapshots, refer Section 4.2. The anomalous snapshots contains the faulty process payload byte transition encoded over the edge between *PLC-2* and *BK-2*. The time taken for each hyperparameter-optimized GNN architecture with number of model parameters, and performance of GNN architectures for anomaly detection are summarized in Table 2.

The isotropic GCN model couldn't distinguish between *normal* and *anomalous graph snapshots* as the information is encoded on the edge features. Since, the network structure remains constant in either evaluation case and is similar to learnt *normal* behavioural graph structure, GCN recalls all the *normal graph snapshots* successfully.

The anisotropic GNN models detected the anomalies completely with complete recall on *normal graph snapshots*. For the proposed spatio-temporal characteristics encoding on the edges (refer Section 3), the anisotropic GNN models could be incorporated for anomaly detection. The information on the edges of different connections could be enriched for analysis in more complex use cases than the evaluated process targeted attack, such as *Stealthy Attack* [10].

The evaluation result concludes the correctness of encoding mechanism outlined in Section 3 for modeling *spatio-temporal* graph representation. The *Sensitivity* score indicates that the *anisotropic* GNN are able to learn process behaviour of the industrial network based on the *spatio-temporal* encodings on the edges. The *Specificity* score conveys the applicability of Graph pooling for the *graph classification* task as process data anomaly detection method.

Table 2: Comparison of different GNN architectures' performance on anomaly detection and *spatio-temporal* modeling.

GNN Model	**Specificity**	**Sensitivity**	**Model Parameter Count**	**Computation Time per 10 Epoch (*s*)**
GCN (Baseline)	0.0	1.0	3,201	7.6
MPNN	1.0	1.0	75,361	4.2
GT	1.0	1.0	29,089	11.0
GatedGCN	1.0	1.0	187,521	6.9

The time taken to execute GNN architectures are low as the size of every evaluated *graph snapshot*, normal and anomalous, is small - 8 nodes and 7 edges. In larger industrial systems with increased process complexity, the number of nodes and edges grows accordingly. This leads to more model parameters and higher computation time for GNNs.

5. Related Work

In recent years, analogous to the presented work, applications of Graph Neural Networks (GNNs) for network cybersecurity based on features extracted from network traffic have gained prominence. These approaches leverage the ability of GNNs to model complex relationships within network data by representing it as a graph. For instance, Pujol-Perich et al. [22] explored GNNs for robust intrusion detection by constructing a heterogeneous graph of network assets and their connections, classifying flow-representing nodes to identify various attacks. GCN SCOPE [21] focused on identifying suspicious communications by building a multigraph from communication triplets and using R-GCNs for link prediction of anomalous connections. Furthermore, E-GraphSAGE [17] utilized a GNN inspired by GraphSAGE to capture both topological and edge features of network flows for edge classification in intrusion detection. Lastly, NF-GNN [5] employed a tailored GNN on network flow graphs for malware detection through graph-level classification. These works collectively demonstrate the versatility of GNNs in addressing diverse cybersecurity tasks by leveraging different graph representations and learning paradigms.

The presented work differs from the published literature in terms of graph representation of network traffic and the ML task. The industrial network communication is represented through a directed homogeneous graph (refer Section 3), which differs from the work of GCN SCOPE and Pujol-Perich et al.. E-GraphSAGE samples the neighbourhood information, whereas complete neighbourhood information is required for process data analysis (refer Section 4). The task of NF-GNN is similar to the proposed solution of utilizing GNN for process data analysis in industrial networks, hence, its usage for evaluation is scheduled as a component of future work.

6. Discussion and Conclusion

The presented work introduced a self-learning framework for anomaly detection in industrial systems by leveraging Graph Neural Networks (GNNs) to model and learn the *spatio-temporal* characteristics of process behavior directly from network communication traffic. Specifically, we addressed **(RQ-1)** how to represent spatio-temporal characteristics of industrial communication in graph form by encoding protocol-level temporal features — such as payload transitions, timing intervals, and cycle counter differences — on graph edges, and **(RQ-2)** how to learn process behavior from these representations for anomaly detection using various GNN architectures.

The approach was validated on a real-world miniaturized industrial system through a process targeted attack. The results showed that anisotropic GNNs (MPNN, Graph Transformer, GatedGCN) successfully detected anomalies, while isotropic GCN failed — underscoring the importance of edge-aware message passing for learning *spatio-temporal* characteristics. The use of negative sampling enabled training without labeled attack data, demonstrating the framework's effectiveness in self-supervised settings. In comparison to existing work in the cybersecurity domain, NF-GNN is most closely aligned in terms of architecture and task and is earmarked for future comparative evaluation.

It is also observed that the computational cost and scalability of GNN-based models are influenced by the size of the industrial network and the richness of encoded features. As the complexity of the system increases, so does the computational demand for learning and inference. Although the evaluation used PROFINET traffic, the methodology generalizes well to other deterministic industrial environments — such as manufacturing and process automation — where cyclic, structured communication patterns are common.

In conclusion, this paper demonstrates that GNNs, when applied to domain-aware spatio-temporal graph representations, can be successfully used for a self-learning anomaly detection approach in industrial communication networks. In future work, we will explore broader anomaly scenarios to further enhance detection robustness in operational technology environments.

Acknowledgements

The presented work is supported by topic Engineering Secure Systems of Helmholtz Association.

References

[1] Akoglu, L., Tong, H., Koutra, D., 2015. Graph based anomaly detection and description: a survey. Data mining and knowledge discovery 29, 626–688.

[2] Benczur, A.A., Csalogany, K., Sarlos, T., Uher, M., 2005. Spamrank–fully automatic link spam detection work in progress, in: Proceedings of the first international workshop on adversarial information retrieval on the web, pp. 1–14.

[3] Bresson, X., Laurent, T., 2017. Residual gated graph convnets. arXiv preprint arXiv:1711.07553 .

[4] Bruna, J., Zaremba, W., Szlam, A., LeCun, Y., 2013. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203 .

[5] Busch, J., Kocheturov, A., Tresp, V., Seidl, T., 2021. Nf-gnn: Network flow graph neural networks for malware detection and classification. arXiv preprint arXiv:2103.03939 .

[6] Castillo, C., Donato, D., Gionis, A., Murdock, V., Silvestri, F., 2007. Know your neighbors: Web spam detection using the web topology, in: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 423–430.

[7] Dai, H., Dai, B., Song, L., 2016. Discriminative embeddings of latent variable models for structured data, in: International conference on machine learning, PMLR. pp. 2702–2711.

[8] Daigavane, A., Ravindran, B., Aggarwal, G., 2021. Understanding convolutions on graphs. Distill doi:10.23915/distill.00032. https://distill.pub/2021/understanding-gnns.

[9] Dwivedi, V.P., Joshi, C.K., Laurent, T., Bengio, Y., Bresson, X., 2020. Benchmarking graph neural networks. arXiv preprint arXiv:2003.00982 .

[10] Ferrari, P., Sisinni, E., Bellagente, P., Rinaldi, S., Pasetti, M., de Sá, A.O., Machado, R.C., Carmo, L.F.d.C., Casimiro, A., 2020. Model-based stealth attack to networked control system based on real-time ethernet. IEEE Transactions on Industrial Electronics 68, 7672–7683.

[11] Fey, M., Lenssen, J.E., 2019. Fast graph representation learning with PyTorch Geometric, in: ICLR Workshop on Representation Learning on Graphs and Manifolds.

[12] Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E., 2017. Neural message passing for quantum chemistry, in: International conference on machine learning, PMLR. pp. 1263–1272.

[13] Hamilton, W.L., 2020. Graph representation learning. Synthesis Lectures on Artifical Intelligence and Machine Learning 14, 1–159.

[14] Hamilton, W.L., Ying, R., Leskovec, J., 2017. Inductive representation learning on large graphs, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 1025–1035.

[15] Iliofotou, M., Kim, H.c., Faloutsos, M., Mitzenmacher, M., Pappu, P., Varghese, G., 2011. Graption: A graph-based p2p traffic classification framework for the internet backbone. Computer Networks 55, 1909–1920.

[16] Kipf, T.N., Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 .

[17] Lo, W.W., Layeghy, S., Sarhan, M., Gallagher, M., Portmann, M., 2021. E-graphsage: A graph neural network based intrusion detection system. arXiv preprint arXiv:2103.16329 .

[18] Marcheggiani, D., Titov, I., 2017. Encoding sentences with graph convolutional networks for semantic role labeling. arXiv preprint arXiv:1703.04826 .

[19] Merkwirth, C., Lengauer, T., 2005. Automatic generation of complementary descriptors with molecular graph networks. Journal of chemical information and modeling 45, 1159–1168.

[20] Meshram, A., Karch, M., Haas, C., Beyerer, J., 2023. Towards self-learning industrial process behaviour from payload bytes for anomaly detection, in: 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–8. doi:10.1109/ETFA54631.2023.10275358.

[21] Oba, T., Taniguchi, T., 2020. Graph convolutional network-based suspicious communication pair estimation for industrial control systems. arXiv preprint arXiv:2007.10204 .

[22] Pujol-Perich, D., Suárez-Varela, J., Cabellos-Aparicio, A., Barlet-Ros, P., 2021. Unveiling the potential of graph neural networks for robust intrusion detection. arXiv preprint arXiv:2107.14756 .

[23] Rahman, M.S., Huang, T.K., Madhyastha, H.V., Faloutsos, M., 2012. Efficient and scalable socware detection in online social networks, in: 21st {USENIX} Security Symposium ({USENIX} Security 12), pp. 663–678.

[24] Rubio, J.E., Alcaraz, C., Roman, R., Lopez, J., 2017. Analysis of intrusion detection systems in industrial ecosystems., in: SECRYPT, pp. 116–128.

[25] Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., Sun, Y., 2020. Masked label prediction: Unified message passing model for semi-supervised classification. arXiv preprint arXiv:2009.03509 .

[26] Sukhbaatar, S., Fergus, R., et al., 2016. Learning multiagent communication with backpropagation. Advances in neural information processing systems 29, 2244–2252.

[27] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need, in: Advances in neural information processing systems, pp. 5998–6008.

[28] Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., Zhang, Z., 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 .

[29] Xiao, Q., Liu, J., Wang, Q., Jiang, Z., Wang, X., Yao, Y., 2020. Towards network anomaly detection using graph embedding, in: International Conference on Computational Science, Springer. pp. 156–169.

[30] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M., 2020a. Graph neural networks: A review of methods and applications. AI Open 1, 57–81.

[31] Zhou, J., Xu, Z., Rush, A.M., Yu, M., 2020b. Automating botnet detection with graph neural networks. arXiv preprint arXiv:2003.06344 .