Graph Neural Networks for Anomaly Detection in Spacecraft

Gamze Naz Kiprit [*1,3], Andreas Koch[1,2], Michael Petry[1,2], Martin Werner[2]

[1] *Telecom and Navigation Processing Germany, Airbus Defence and Space GmbH.*
[2] *Professorship Big Geospatial Data Management, Technical University of Munich.*
[3] *Department of Electrical Engineering, Technical University of Munich.*

Satellites play a crucial role in the global communications system, while being subjected to the harsh space environment that often causes component degradation. Early failure detection is important to ensure that the critical services they provide are not interrupted. This work proposes a forecasting-based anomaly detection method, where a Graph Convolutional Network (GCN) is leveraged to extract relevant information from time series. The proposed anomaly detection model reaches an F-Score of 89% on the open-source SMAP & MSL dataset, which is the best score achieved on this dataset to the best of the authors' knowledge. Furthermore, the proposed model is deployed on the space-grade AMD-Xilinx Versal AI Core series and its performance measures as well as occupied hardware resources are demonstrated.

1 Introduction

Today, numerous activities on Earth depend on satellites, such as radio signals, navigation and, in many countries, the internet access. These machines must execute critical tasks with exceptional reliability, even in the event of a failure. Failure Detection, Isolation and Recovery (FDIR) is a key strategy to fulfill the purpose of reliability and aims to identify and isolate the faults as early as possible [1]. The most common approach to identify anomalies is to detect out-of-limit (OOL) events. These events are identified by examining whether the design limits of any sensor, instrument, or subsystem are crossed. As data-driven Machine Learning (ML) methods advance, identifying anomalies months before OOL limits are crossed is becoming possible, as proposed in [2]. The ML-based anomaly detection methods also allow this process to run automatically, meaning that the incoming telemetry data is continuously processed to detect potential anomalies. These approaches include Support Vector Machines (SVMs) [3, 4], Autoencoders [5–7], Recurrent Neural Networks (RNNs) [7, 8], Generative Adversarial Networks (GANs) [9] and many others. Besides the variety of these architectures, the advanced development of hardware enables an easy acceleration of these models on space-ready hardware.

This work proposes a forecasting-based anomaly detection method using GCNs on the space-grade AMD-Xilinx Versal AI Core series. The next Section gives a brief background on Graph Neural Networks (GNNs). Then, Section 3 introduces the dataset and the target hardware and presents the proposed method including its implementation and deployment. Section 4 shows the achieved results with comparison to other state-of-the-art works and on-board measurements of the proposed method. Finally, Section 5 provides concluding remarks and suggestions for future work.

*This activity has been funded by DLR within the project Machine Learning for Telecom Satellites (MaLeTeSa). Corresponding author. E-Mail: gamzenaz.kiprit@tum.de

2 Graph Neural Networks

Graphs. Graphs are data structures suitable for modeling the interactions and relations between objects. Mathematically, a graph is a tuple $G = (V, E)$, where V denotes the list of objects, in this context nodes or vertices, and E is the list of edges. A graph can be represented using an *Adjacency Matrix* $\mathbf{A}$. The entries of $\mathbf{A}$ denote the existence of the edges.

The core concept in GNNs is the message-passing strategy. During each message passing iteration, each node in a graph aggregates information from its local neighborhood in order to update its node embedding $\mathbf{h}$. The more iterations are performed, the more information each node embedding contains from further nodes in the graph. A message passing iteration is mathematically expressed as:

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{self}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{neigh}^{(k)}\sum_{v\in N(u)}\mathbf{h}_v^{(k-1)} + b^k\right), \quad (1)$$

where $\mathbf{h}_u^{(k-1)}$ denotes the node embedding of the node u in the $(k-1)$-th iteration and the $\mathbf{h}_v^{(k-1)}$ denotes the node embeddings of the neighbors of the node u in

the $(k-1)$-th iteration. The $\mathbf{W}^{(k)}_{self}$ and the $\mathbf{W}^{(k)}_{neigh}$ are the trainable weight matrices and the b is the bias.

The Graph Convolutional Neural Networks (ConvGNNs) are the generalization of the Convolutional Neural Networks (CNNs) to the graph-structured data. They aim to capture and encode the structural information of a graph and allow nodes of the graph to consider the features of their neighbors. The graph convolution operation involves applying a convolution filter to all node features considered.

Graph Convolutional Network. The GCN is a message-passing based ConvGNN [10]. The basic GCN layer is defined as follows:

$$\mathbf{H}^{(k)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}\right), \tag{2}$$

where k is the iteration number, i.e. layer, $\mathbf{W}$ is the trainable parameter matrix and $\mathbf{H}$ is the embedded graph. In the equation above, $\tilde{\mathbf{A}}$ denotes the normalized variant of the adjacency matrix with self-loops and is equal to $(\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}$, where $\mathbf{D}$ is the degree matrix of $\mathbf{A}$.

3 Proposed Approach

The proposed anomaly detection approach consists of two components: forecasting and non-parametric thresholding, as seen in Figure 1. The forecasting component is a Deep Learning (DL) model, which aims to learn the normal behavior of the time-series and predicts the next time steps based on l previous steps. The non-parametric thresholding component compares the predicted value with the measured real value and decides whether there is an anomaly or not.

The DL model has three learnable layers: a GCN layer as the input layer and two dense layers. The input time series are treated as graph nodes in order to be fed into the GCN layer and an adjacency matrix is constructed to define the graph connections. Usually, there is domain-specific knowledge to construct the graph connections. For instance, in chemistry, the nodes of a graph can denote the molecules and the edges are chemical bonds. However, there is not any specific knowledge for the time series in this work. Moreover, since each connection of the graph defines the correlation or interaction between the nodes, different combinations of connections will lead to different results. For this reason, edge connections in time series are treated as hyperparameters. To the best of the authors' knowledge, this approach has not been seen in the literature so far.

In order to limit the hyperparameter space, the following restrictions are introduced:

- The current time steps are only connected to the future time steps. Therefore, the adjacency matrix is an upper triangular matrix.
- Each time step has a self connection, i.e., the main diagonal of the adjacency matrix is filled with ones.
- In order to maintain sequentiality, each time step is connected to the next. The line above the main diagonal is also filled with ones.

Thus, the hyperparameter space consists of the entries of the adjacency matrix that are above the first upper diagonal and are either 0 or 1. The hyperparameter exploration is done while training the network with the goal of maximizing the objective function. The prediction $\hat{y}$ of the most optimal configuration found is then handed to the second component.

In the second component, a threshold ϵ is calculated dynamically based on the error between the prediction $\hat{y}$ and the measured real value y. The method for calculating the threshold is proposed by [8]. All time steps where the error values are above this selected threshold ϵ, are marked as an anomaly. Based on the marked anomalies, the metrics True Positive (TP), False Positive (FP) and False Negative (FN) are calculated as follows:

- **True Positive:** One TP is counted, when a segment of a predicted sequence of anomalies matches a true labeled sequence. In the event of several predicted anomaly sequences that match a true labeled sequence, only one TP is recorded.
- **False Negative:** One FN is counted, if the prediction is false for a true sequence.
- **False Positive:** One FP is counted, if the prediction is true for a false sequence.

3.1 Dataset

The proposed method is evaluated on a labeled open-source anomaly detection dataset published by research scientists at NASA [8]. It consists of raw telemetry from two spacecraft: the Soil Moisture Active Passive (SMAP) satellite and the Mars Curiosity Rover. The provided original data is multi-variate, with multiple command channels and one telemetry channel. However, this work considers only the telemetry channel. The dataset contains point anomalies and contextual anomalies [11] and there are 105 anomaly sequences in total, 62 of which are point anomalies and 43 of which are contextual anomalies.

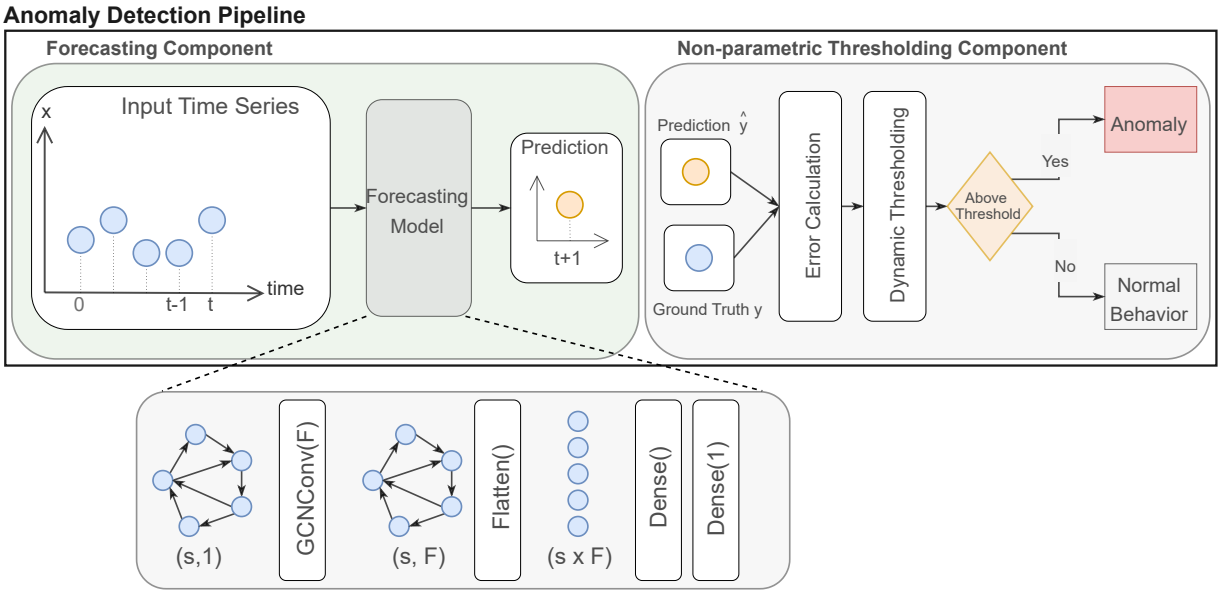

Figure 1: Proposed anomaly detection pipeline.

3.2 Implementation

The sliding window approach is used as a preprocessing step for modeling the prediction problem. The default window size is set to $l = 25$. The first 24 time steps of a sequence are used to train the network to predict the 25th time step. It is important to note that the training data only consists of the normal behavior of the sensor and the test data contains both anomalous time steps and time steps with normal behavior. The proposed network is implemented with TensorFlow and the training is done on Amazon Web Services (AWS). The hyperparameter optimization is done via the Amazon SageMaker Tuner, which applies a linear regression over all iterations in order to find an optimal hyperparameter configuration.

3.3 Hardware Deployment

The target hardware is the Versal Adaptive Compute Acceleration Platform (ACAP) AI Core Series from AMD-Xilinx, which is a space-ready hybrid compute platform with four main components: the Processing System (PS), the Programmable Logic (PL), 400 intelligent engines, also called AI engines, and the programmable Network-on-Chip (NoC) [12].

AMD-Xilinx offers a development environment called Vitis AI for AI inference on Xilinx edge devices. Specifically, it implements an AI inference for DL models on a programmable engine called the Deep Learning Processing Unit (DPU). For the Versal ACAP, the DPU is to be integrated into the hardware image, using both PL and AI engine resources, forming a unified IP block. The DPU is a generic ML inference accelerator and it dynamically loads pre-compiled DL models at run-time. For deployment, it is necessary to quantize pre-trained DL models, as the DPU is limited to quantized 8-bit values. Model quantization and compilation compatible with the target DPU can be performed by Vitis AI. The quantization and compilation of the GCN layer is not possible with the Vitis AI toolchain as the GCN layer is not supported by the DPU. As the GCN layer is the first layer of the network, executing the GCN layer is considered a preprocessing step and the trained GCN layer is implemented as a matrix multiplication of the input matrix, the adjacency matrix and the weight matrix of the layer using the NumPy-Matmul operation on the CPU. The bias is added to the result.

4 Results

This work considers three performance metrics: precision, recall and $F_{0.5}$ score. The reason for using the $F_{0.5}$ score is to assign more importance to FPs than FNs, as it should be avoided to terminate spacecraft operations by FPs. Table 1 shows the obtained results and ablation studies in comparison to other state-of-the-art methods on the same dataset. The first row, GCN-2Dense, shows the obtained results of the original proposed network with two dense layers. It has a precision of 94%, a recall of 70% and an $F_{0.5}$ score of 88%. The network in the second row, GCN-3Dense, has an additional dense layer and the best precision

with 96% and the best $F_{0.5}$ score with 89% among all networks. The third row shows the results of the two dense layers without the GCN layer. Here, a significant performance decrease is observed. Thus, the $F_{0.5}$ score of the proposed network overcomes the LSTM Network [8], by 3%, the Channel-Specific LSTM [13] by 9%, the TadGAN [9] by 34% and StackedPredictor [14] by 1%. Table 2 shows the $F_{0.5}$ scores of the

Method	Precision	Recall	F-Score
GCN-2Dense	0.94	0.70	0.88
GCN-3Dense	0.96	0.70	0.89
2Dense w/o GCN	0.72	0.74	0.72
LSTM Network [8]	0.87	0.80	0.85
Channel-Specific LSTM [13]	0.79	0.83	0.79
TadGAN [9]	0.51	0.78	0.54
StackedPredictor [14]	0.87	0.89	0.87

Table 1: Performance metrics of different models on the same dataset. The first three models are developed and implemented as part of this work.

proposed networks after quantization. Both models show a significant performance loss after quantization. However, a remarkable portion of the performance is recovered by applying smoothing to the error sequence, eliminating the peaks that occurred due to quantization. This is shown in the third column of the table. Both models are deployed on three different

Model	F-Score After Quantization	F-Score After Quantization & Smoothing
GCN-2Dense	0.71	0.83
GCN-3Dense	0.78	0.86

Table 2: Performance statistics after quantization.

processors: the CPU, the DPU with batch number one, i.e., DPU-B1, and the DPU with batch number five, i.e., DPU-B5. Figure 2 shows the runtime of the three processors with respect to different models with different sizes. On the x-axis, the proposed models are shown with gray dotted lines. It can be observed that executing the small models on the CPU is faster. However, as the model size increases, both the DPU-B1 and the DPU-B5 executions become significantly faster than the CPU. Figure 3 shows the comparison of the energy consumption per Bit of the processors in relation to the model size. The energy consumption of the CPU is extremely low for the smallest models and is the most efficient, but it drastically rises for the larger models. Using the DPU-B5 for both proposed models is the most efficient in terms of energy per Bit.

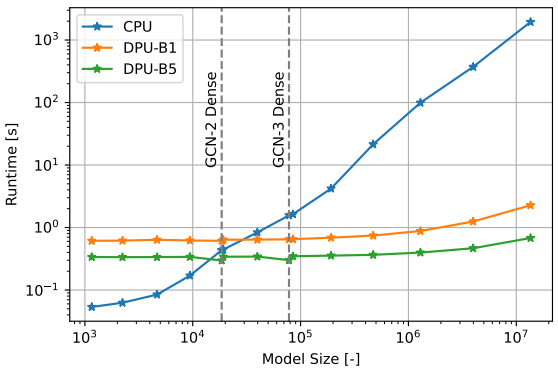

Figure 2: Runtime comparison of the three processors with respect to the model size.

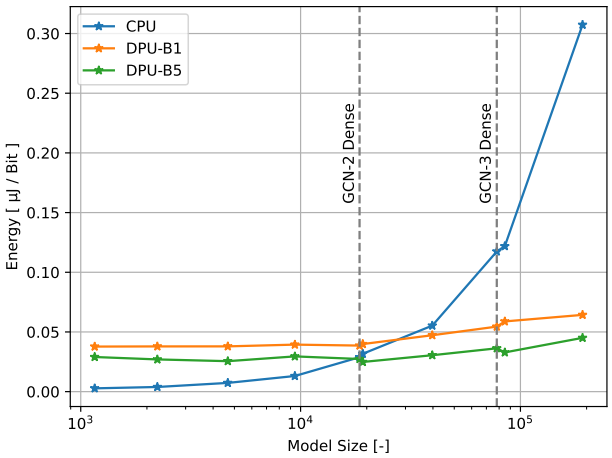

Figure 3: Energy consumption per Bit of the processors with respect to the model size.

5 Discussion

This work proposes a forecasting-based time series anomaly detection method based on GCNs. The best node connection combinations for the input graphs to the GCN are found in the hyperparameter optimization process during training. This method allows the GCN layer to extract the most relevant information. To the best of authors' knowledge, the obtained results are state-of-the-art on the used open-source dataset. These results support the model's capability of detecting anomalies in telemetry. However, a further evaluation on different datasets is needed to conclusively prove its ability to generalize. Moreover, future work may consider integrating the GCN layer on the FPGA. The authors hope that this work will incentivize the exploration of GCNs in the space community.

References

1. NASA. *Fault-Detection, Fault-Isolation and Recovery (FDIR) Techniques* https://llis.nasa.gov/lesson/839.
2. Andreas, K. *et al.* On-Board Anomaly Detection on a Flight-Ready System. https://www.bgd.ed.tum.de/pdf/2023_ADAP_Andreas.pdf (2023).
3. Wang, Y., Wong, J. & Miner, A. *Anomaly intrusion detection using one class SVM* in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.* (2004), 358–364.
4. Ma, J. & Perkins, S. *Time-series novelty detection using one-class support vector machines* in *Proceedings of the International Joint Conference on Neural Networks, 2003.* **3** (2003), 1741–1745 vol.3.
5. Sakurada, M. & Yairi, T. *Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction* in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis* (Association for Computing Machinery, Gold Coast, Australia QLD, Australia, 2014), 4–11. ISBN: 9781450331593. https://doi.org/10.1145/2689746.2689747.
6. Xu, H. *et al. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications* in *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18* (ACM Press, 2018). http://dx.doi.org/10.1145/3178876.3185996.
7. Park, D., Hoshi, Y. & Kemp, C. C. *A Multimodal Anomaly Detector for Robot-Assisted Feeding Using an LSTM-based Variational Autoencoder* 2017. arXiv: 1711.00614 [cs.RO].
8. Hundman, K., Constantinou, V., Laporte, C., Colwell, I. & Söderström, T. Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. *CoRR* **abs/1802.04431.** arXiv: 1802.04431. http://arxiv.org/abs/1802.04431 (2018).
9. Geiger, A., Liu, D., Alnegheimish, S., Cuesta-Infante, A. & Veeramachaneni, K. *TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks* 2020. arXiv: 2009.07769 [cs.LG].
10. Hamilton, W. L. Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* **14,** 1–159.
11. Chandola, V., Banerjee, A. & Kumar, V. Anomaly Detection: A Survey. *ACM Comput. Surv.* **41.** ISSN: 0360-0300. https://doi.org/10.1145/1541880.1541882 (July 2009).
12. Xilinx. *Versal ACAP System Software Developers Guide* https://docs.xilinx.com/r/2021.1-English/ug1304-versal-acap-ssdg/Development-Tools.
13. Baireddy, S. *et al. Spacecraft Time-Series Anomaly Detection Using Transfer Learning* in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2021), 1951–1960.
14. Li, T. *et al. A Stacked Predictor and Dynamic Thresholding Algorithm for Anomaly Detection in Spacecraft* in *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)* (2019), 165–170.