

# Autonomous Satellite Health Monitoring using EIRSAT-1 Telemetry

James Murphy<sup>1,2,\*,†</sup>, Muhammad Deedahwar Mazhar Qureshi<sup>1,†</sup>, Jake O'Brien<sup>1,†</sup> and Brian Mac Namee<sup>2</sup>

<sup>1</sup>*Réaltra Space Systems Engineering, The Realtime Building, Clonshaugh Business & Technology Park, Dublin, Ireland*

<sup>2</sup>*School of Computer Science, University College Dublin, Belfield, Dublin, Ireland*

## Abstract

The rapid expansion of satellite constellations, such as Starlink, has necessitated a shift from traditional human-based telemetry monitoring to more autonomous systems. With thousands of new satellites overwhelming existing operators, the development of automated satellite health monitoring has become essential. Current systems are largely statistical in nature, but AI-driven solutions offer the potential to significantly improve the accuracy and efficiency of anomaly detection. However, for AI systems to be adopted, satellite operators must not only trust the decisions made by these systems but also understand the reasoning behind them. This paper establishes a baseline autonomous health monitoring framework for satellite telemetry, with a focus on reliability and real-time anomaly detection. We leverage the EIRSAT-1 telemetry dataset to train and deploy our model and evaluate its performance across various Edge AI devices that either possess flight heritage or are scheduled for future space missions. These devices demonstrate the potential for real-time anomaly response, which is critical for mission success. To further enhance the accuracy of anomaly detection, we incorporate data denoising techniques as part of the postprocessing pipeline. Noise in reconstruction error can obscure subtle anomalies, leading to false positives or missed detections. By employing various denoising methods, we aim to create a clearer signal that allows for more accurate thresholding. We investigate the use of statistical approaches, such as moving averages and filtering, alongside machine learning models, such as clustering methods, for thresholding anomaly detection. These alternative approaches are compared in terms of computational efficiency, robustness, and suitability for real-time, onboard deployment in space environments.

## Keywords

Edge AI, Anomaly Detection, Satellite, Space

## 1. Introduction

In this paper, we propose a baseline autonomous health monitoring framework designed to operate on satellite telemetry data. Our system aims to reliably detect anomalies and respond in real time to support mission-critical decision-making. To demonstrate the viability of our approach, we utilize the EIRSAT-1 telemetry dataset[1] to train and deploy our models. Additionally, we evaluate the system's performance on a variety of Edge AI devices that are either flight-proven or expected to be deployed on upcoming missions. These devices enable real-time onboard anomaly detection, which is particularly valuable for reacting swiftly to unexpected events.

Satellite systems are highly complex hardware devices which generate substantial amounts of data. However, due to down-link bandwidth restrictions, only a fraction of that data makes it to ground. They are also limited when it comes to autonomy. If an error occurs on a satellite, it must go through the down-link cycle and be analyzed by satellite operators on ground, creating a large lag time between any action being taken to rectify the error. This can potentially lead to the loss of the satellite and the premature end of the mission. Our solution aims to be run on-board and in real-time in order to

---

*AICS'24: 32nd Irish Conference on Artificial Intelligence and Cognitive Science, December 09–10, 2024, Dublin, Ireland*

\*Corresponding author.

†These authors contributed equally.

✉ jmurphy@realtra.space (J. Murphy); mqureshi@realtra.space (M. D. M. Qureshi); jobrien@realtra.space (J. O'Brien); brian.macnamee@ucd.ie (B. M. Namee)

ORCID 0000-0002-3984-1371 (J. Murphy); 0009-0002-4878-9226 (M. D. M. Qureshi); 0009-0004-4447-7377 (J. O'Brien); 0000-0003-2518-0274 (B. M. Namee)



© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

remove these potential problems in the anomaly detection and rectification pipeline. However, this adds more complexity to the mission. Satellites are powered entirely by solar arrays, limiting the power available on-board. Space is also at a premium as mass to orbit costs are in the order of €30,000 per kg. Being in a vacuum environment also adds thermal management issues. Finally, a radiation intense environment is also an aspect that must be considered depending on the orbit of the mission. This paper aims to address these concerns and presents a potential alternative to the current ground based anomaly detection paradigm.

Our methodology includes training the model on the flight test dataset using only nominal (healthy) data. We then evaluate its performance by testing it on the remainder of the flight test dataset, where artificial anomalies have been injected to simulate potential issues. This allows us to assess the model's ability to detect anomalies under controlled conditions. Following this, we deploy the model on the thermal vacuum (TVAC) dataset, which contains real anomalies. Since the TVAC dataset is more representative of actual satellite conditions, it provides a more realistic test of the model's robustness and accuracy.

Based on the insights gained from both the labeled flight test data and the real anomalies in the TVAC dataset, we define our thresholding methods. This process is critical for optimizing the model's sensitivity and reducing false positives. Once optimized, we deploy the model on the unlabeled flight dataset to further evaluate its performance under more realistic, unlabeled conditions, demonstrating its ability to generalize to unseen data.

A key component of our approach is the preprocessing of telemetry data to remove noise, which can obscure meaningful anomalies. Denoising the data helps to enhance the accuracy of the anomaly detection process. In this paper, we explore various denoising techniques, including both statistical methods (e.g., moving averages, Kalman filtering) and machine learning models (e.g., autoencoders, isolation forests). These techniques are investigated for their ability to improve thresholding mechanisms in detecting anomalies. We compare the effectiveness of these approaches in terms of computational efficiency, robustness, and suitability for real-time, onboard deployment, ensuring that the selected method can operate reliably in the challenging conditions of space environments.

By establishing a baseline autonomous health monitoring system and investigating advanced denoising and thresholding methods, we aim to contribute to the growing field of AI-driven satellite health management, paving the way for more reliable and interpretable systems capable of supporting future space missions.

## **2. Dataset & Model Development**

This section describes the basic structure of our experiments and the machine-learning (ML) pipeline deployed for anomaly detection. Based on the challenging aspects of this problem, the ML pipeline can be broken into two major components: dataset preprocessing and model development. We analyze a multivariate time series and simplify it to a less complex feature space, which is then used to identify anomalous data.

### **2.1. Dataset and Pre-processing**

We use the EIRSAT-1 dataset as the data source for our experiments [1]. The EIRSAT-1 dataset consists of three sub-datasets:

- The Flight Test Dataset
- The Thermal Vacuum Test Dataset
- The Flight Dataset

These datasets represent data collected during different phases of the EIRSAT-1 project. The flight test and TVAC datasets are from the satellite's test campaigns, which include nominal and anomalous data, making them perfect for training and hyperparameter tuning. The flight dataset, on the other hand, is

entirely unlabelled and to date the satellite has had no reported anomalies. The data in each dataset is divided into four categories: Housekeeping, Telemetry-Enhanced-Diagnostics (TED), Attitude Control (ADCS), and Power. This paper focuses on the data from the Housekeeping category (Channel 15 in the dataset files). We drop variables that are missing data or ones with encoded or inaccessible information, leaving us with 85 usable variables. The dataset’s structure and usage details can be taken from our previous publication [1]. The EIRSAT-1 dataset’s high complexity requires a suitable pre-processing approach to make it usable for ML purposes.

## 2.2. Machine Learning Model

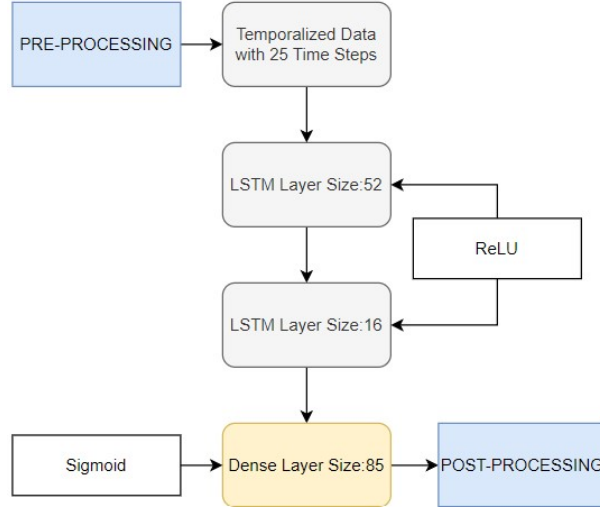
This section describes the different aspects of the ML model that convert our original multi-variate time series into a univariate time series that can be used to detect anomalies reliably.

### 2.2.1. AutoencoderNetwork Architecture

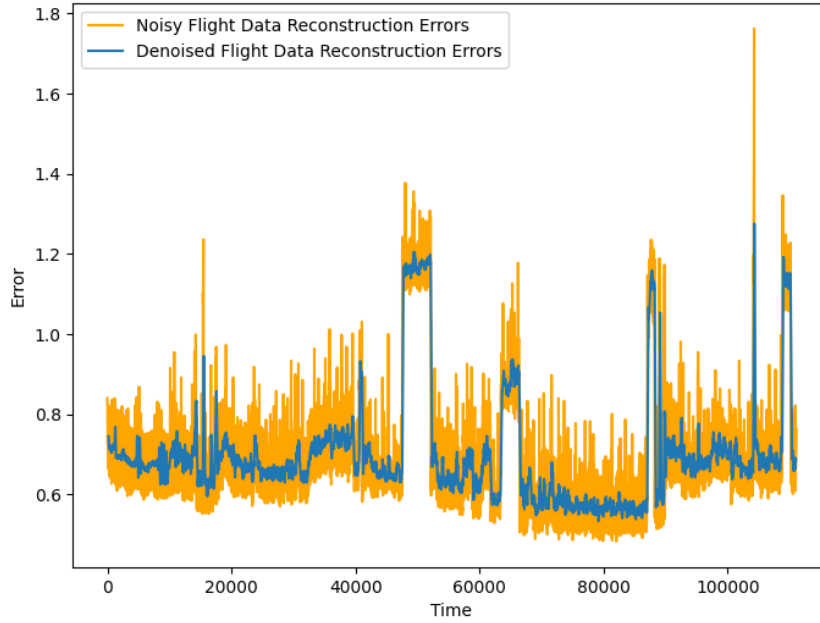
The first part of the ML pipeline consists of an autoencoder model. An autoencoder is a machine-learning model built on an encoder-decoder architecture that aims to learn a low-dimensional input data representation. The decoder then reconstructs the original input data from the low-dimensional abstraction. Our LSTM autoencoder model[2] is more focused on obtaining reconstruction errors than converting the data into low-dimensionality. The encoder network starts with an LSTM layer of size 52 and an LSTM layer of size 16. Both layers are activated with the ReLU activation function[3]. We also set the *return\_sequences* variable to *True*. This allows the subsequent layers to access the temporalized data from the previous layers. The decoder network consists of a single Dense layer the same size as our input (85) with the Sigmoid activation function. The sigmoid function regenerates the value of each variable between 0 and 1, corresponding to the scaled values from the Min-max Scaler. The architecture of the autoencoder can be seen in Figure 1. Since our model consists of a LSTM network, we need to temporalize our data to make it usable for our ML architecture. We set the number of time steps to 25 and also use this as a tunable parameter. This variable controls the limit of look-back for the LSTM network. A larger value would allow the LSTM network to capture more temporal information, however, given the hardware limitations the added complexity would make the model computationally too expensive. On the contrary, a smaller value would simplify the model but prevent it from capturing temporal trends. We then scale the data using sci-kit Learn’s Standard Scaler and Min-max Scaler to prevent scaling bias in the models[4]. We use the Keras Tuner module to fine-tune the neural network architectures of the LSTM autoencoders. This predefined module allows us to ascertain the optimum hyperparameter values, such as the number of layers and the size of each layer in the encoder and decoder model.

### 2.2.2. Model Training

The model is trained to optimize the Mean Absolute Error loss function with the Adam Optimizer. Early Stopping is used to obtain the optimum model based on the validation loss. The autoencoder is trained only on the nominal data in the Flight-Test dataset. This trains the model to reconstruct the nominal data well while amplifying the reconstruction loss for anomalous data. While the number of epochs while training is fixed at a relatively high number (e.g., 100 epochs), we trace the validation loss after each epoch, and model weights are updated only if the validation loss falls below the best performance during training (up to that point). The model gives an *ROC – AUC* value of 0.86 on the Flight-Test dataset and 0.98 on the TVAC dataset, thus demonstrating that the model is trained well. The reconstruction error over the whole dataset is integrated into a time series, and we use various thresholding techniques to identify anomalous data.



**Figure 1:** The network architecture of the autoencoder network.



**Figure 2:** The Noisy Reconstruction Error time series compared to the denoised time series.

### 2.2.3. Time Series Denoising

While the resultant time series can be used for classification, it is still noisy (especially in the case of the Flight data) and leads to false positives on anomalies. Therefore, we use a post-processing function to denoise the time series. This function uses a pre-defined filter  $X$  of alternating zeros and ones ( $[0, 1, 0, \dots]$ ) that repeats for a set window size  $W$ . In our experiment, we set the size as 20. A significant value for  $W$  leads to an oversimplified signal, while a small value retains noisy data. We convolve this filter over our time series to denoise it as shown in Equation 1.

$$T_{denoised} = (X \times W) * T_{noisy} \quad (1)$$

A comparison of the noisy and denoised signal on the Flight data is shown in Figure 2. The figure shows how significantly the denoising function improves the chances of fitting an appropriate threshold on the time series.

### 2.3. Thresholding and Anomaly Detection Methods

This section discusses the methods we applied to the obtained reconstruction errors from the autoencoder models. Thresholding here refers to establishing a linear or non-linear boundary on the y-axis, which effectively functions as a decision boundary for anomaly detection. Finally, we again used the standard scaler to rescale the signal.

#### 2.3.1. Percentile-based Linear Thresholding

Linear thresholding is a common method to identify anomalies in a time series dataset. We can then use a linear threshold to predict anomalies based on the reconstruction error. The high ROC score, as given in the previous sections, proves that the model is well-trained, and an optimum threshold would return a high F1-Score on Anomaly Detection. We generate an equidistant distribution of fifty points between the 50th and 100th percentile of the TVAC reconstruction error time series to determine this optimum threshold value. Based on the data, we determine the F1 score at each percentile and select the best-performing percentile as the static threshold for the data. This is then applied to the other datasets.

#### 2.3.2. Z-score based Linear Thresholding

We adopt another linear thresholding method to account for the shortcomings of previous methods, which will be discussed in greater detail in the next section. For this method, we calculate a Z-score that calculates the difference of each point  $E_i$  in the time series signal  $E$  from the mean value in terms of the number of standard deviations of the signal. The Z-score of a point in the time series is defined by Equation 2:

$$Z(E_i) = \frac{E_i - \mu(E)}{\sigma(E)} \quad (2)$$

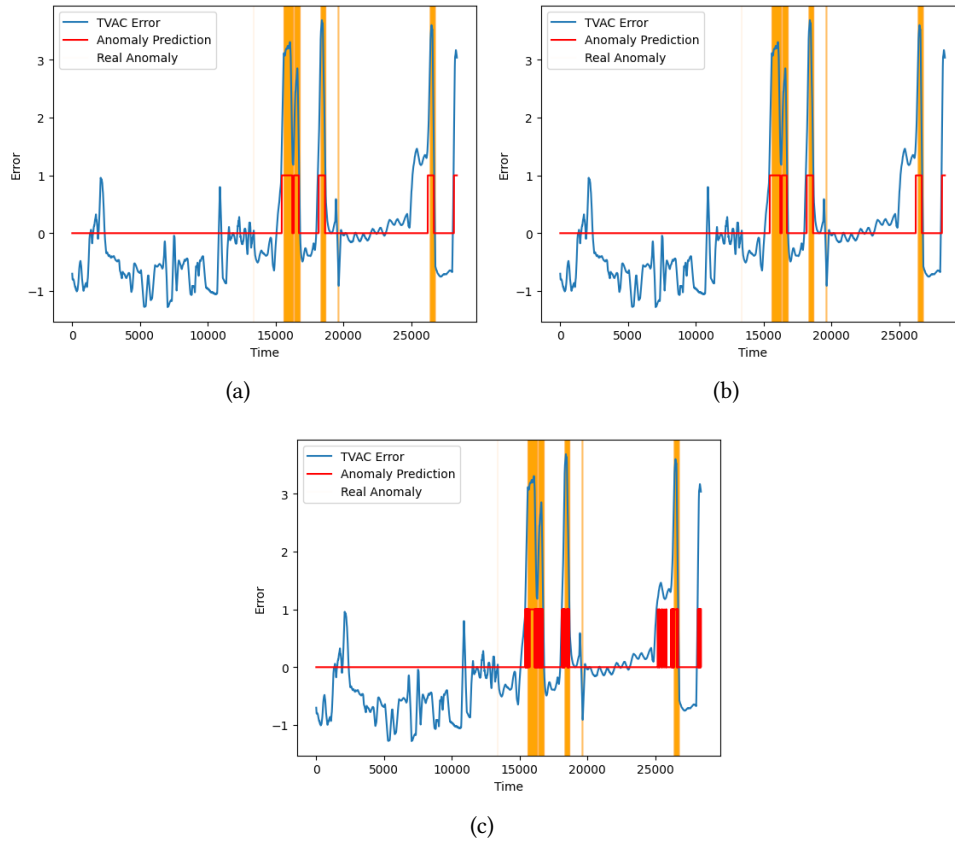
Here,  $\mu(X)$  refers to the mean of the signal while  $\sigma(E)$  represents the standard deviation value of the signal. With this equation, the mean of our signal is 0, and the magnitude of each sample is defined in terms of the signal standard deviation. To tune the threshold hyperparameter, we test a range of values between 1 and 2 on the TVAC dataset, giving an optimum value of 1.5.

#### 2.3.3. K-Nearest Neighbors

We also used a K-Nearest Neighbors Classifier as a supervised approach to anomaly detection on the problem. In this case, we used it as a classifier on the resultant reconstruction error from the LSTM autoencoder. The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier that uses proximity to make classifications or predictions about the grouping of an individual data point. The intuition behind using KNN here is that nominal data points should look structurally similar so that a KNN can classify anomalies successfully. The model performance in the TVAC dataset is shown in the table 1. KNN is helpful for anomaly detection, but it has a significantly higher inference time when compared to the other models used in this paper. Standard Scaler normalizes each feature before passing it into the model. The value of K is set at 3. We also use 5-fold cross-validation to validate the model performance on the training set. We train the KNN model on a split TVAC dataset and then use the model to make predictions on the Flight data to compare the similarity of predictions between all our detection approaches.

## 3. Results and Discussion

This section discusses the performance of our ML architecture on the TVAC and Flight datasets and the insights recovered from these datasets about the different intricacies of the classification/thresholding methods.



**Figure 3:** The figure shows the performance of the anomaly detection methods i.e., (a) Percentile-based Threshold (b) Z-Score based Threshold (c) K-Nearest Neighbors on the TVAC Dataset with the orange colored regions representing real anomalies.

### 3.1. Insights on the TVAC Data

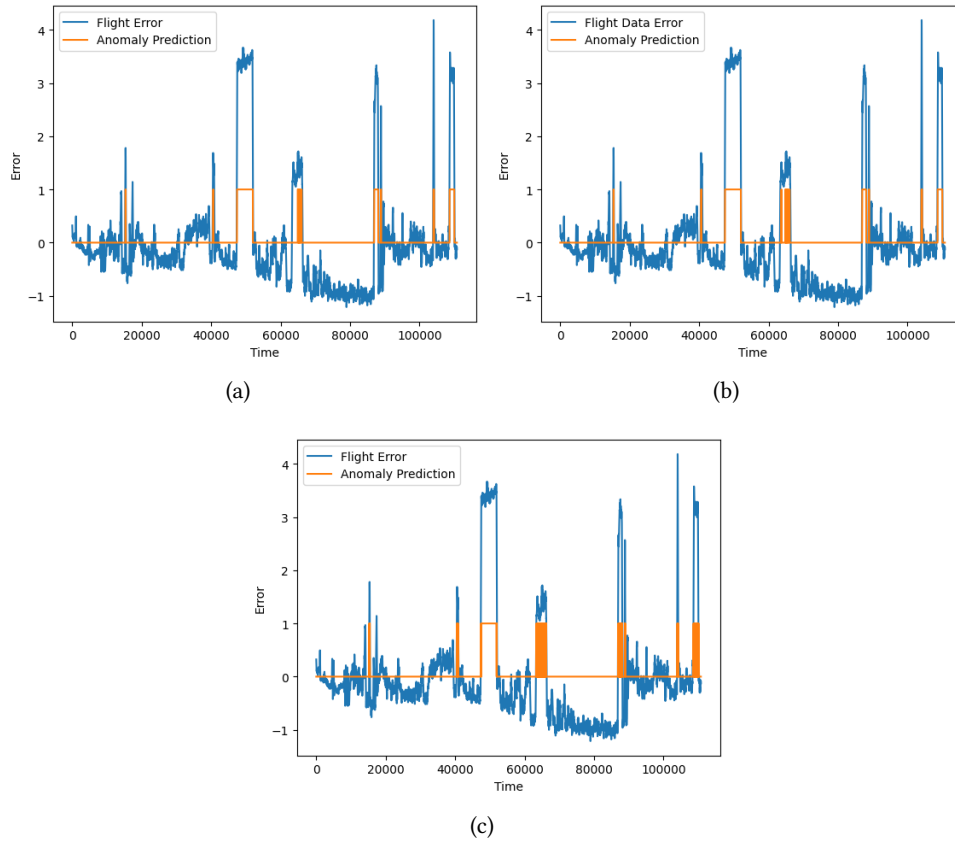
**Table 1**

The performance results for each method on the TVAC dataset.

Method	Precision	Recall	F-Score
Percentile Thresholding	0.84	<b>0.90</b>	0.87
Z-Score Based Thresholding	0.84	<b>0.90</b>	0.87
K-Nearest Neighbors	<b>0.88</b>	0.87	<b>0.88</b>

Applying our ML architecture to the TVAC data allows us to test the performance of the various thresholding or classification methods and draw a comparative analysis between them. Table 1 shows the performance of the different techniques on the TVAC dataset. Here, KNN is shown to do well across the three methods in terms of F-Score, but the other thresholding methods do better on the recall metric. We can further evaluate this by looking at Figure 3. This figure shows that we see a more consistent anomaly prediction on the two thresholding methods compared to KNN, where we notice that the anomalous areas are not consistently reported as anomalous with a small number of nominal predictions. This is reflected in the slightly lower recall value for KNN in Table 1. Additionally, the orange highlighted portion of the figure depicts the ground truth anomalies in the dataset. There are two types of anomalies on the TVAC dataset:

- During vibration testing, a damaged solder joint on the battery board and changing pressure in TVAC stopped the communication between the battery and the OnBoard Computer (OBC). The noticeable change in the data for this anomaly is that the Battery Voltage parameter (plat-



**Figure 4:** The figure shows the performance of the anomaly detection methods i.e., (a) Percentile-based Threshold (b) Z-Score based Threshold (c) K-Nearest Neighbors on the Flight Dataset.

form.BAT.batteryVoltage) drops to 0.

- A communication issue on the I2C Line between the onboard computer and the communications subsystem where the primary indicator for this anomaly is that the Temperature value shown in the parameter (platform.CMC.temperaturePA) abruptly goes to 0. It is important to note that this value can naturally go to 0 if the sensor is facing away from the sun in colder conditions. Therefore, it is also essential to notice the other CMC (Common Mode Current) parameters in the dataset.

Figure 3 depicts that the reconstruction error on the anomalous is significantly higher than the nominal regions, which shows that the autoencoder can reconstruct nominal data well.

### 3.2. Insights on the Flight Data

We tested the autoencoder-thresholding combination on the scaled flight dataset, revealing interesting information about the flight dataset. The Flight Dataset officially has no recorded anomalies. However, the model identified some anomalous areas where large chunks of anomalies were predicted. While the predicted chunks of anomalies might not be anomalous, it seemed an Out-of-Ordinary Operation (OOO). Upon investigation, we discovered that the model was identifying the switching states of the GMOD (Gamma Ray Module) and EMOD (Enbio Module) modules, which is not anomalous but not a regular operation that should happen. Table 2 shows a set of OOOs in the Flight Data with a static threshold. Similarly, the detected OOOs on the Flight data can be seen in Figure 4.

While there are three chunks of reported anomalies whose source has not been investigated yet, most chunks are labeled in the figure with their associated time stamps on the x-axis. One chunk of anomalies is not entirely reported as anomalous depending on the threshold value and the thresholding



**Table 2**

The identified OOOs in the flight dataset. It must be noted that this is not a complete list, and at other times, samples may contain the same or different OOOs.

OOO	Start Sequence	End Sequence	Component
1	15309	15507	GMOD
2	40633	41033	EMOD
3	47540	52119	GMOD
4	63200	66500	EMOD
5	87060	87060	GMOD
6	88971	88971	GMOD
7	89750	89750	GMOD
8	104125	104292	GMOD
9	108806	110144	GMOD

method used. A lower threshold with filtering based on consecutive anomalies solves this problem with some methods. Another interesting note is that the Z-Score predicts a consecutive chunk of anomalies in many cases. At the same time, KNN may give a tiny number of scattered nominal predictions in an OOO chunk. Since anomalies and OOOs do not occur as one-off isolated events in the time series but instead appear in a time chunk, both predictors should be considered valid.

Table 2 shows areas in the flight dataset that were labeled as OOO. After investigating why the model was producing such high reconstruction errors, we found we were detecting the powering on/off of the GMOD and EMOD payloads.

## 4. Deployment

For the targeted low-power small-footprint high-performance hardware to combat the constraints encountered in space-based missions, such as power, radiation tolerance, mass, etc. The developed LSTM model was deployed on two different flight-ready hardware devices.

- Nvidia Jetson Orin Nano 8GB [5]
- Raspberry Pi 4 [6]
- Intel Xeon CPU [7]

The LSTM model was deployed on all three EIRSAT-1 datasets, discussed in Section 2.1. The following results demonstrate the performance of the LSTM model on each hardware devices for each dataset. The Flight dataset is unlabelled and as a result the inferences per second (IPS) was the only metric obtained. The hardware selected for this dataset is based on previous work to investigate which edge AI devices with flight heritage and can effectively run anomaly detection architectures[8]. The experiment is designed where models are developed and tested on a CPU then deployed on hardware. This enables a comparison between the inferencing performance and any model performance differences due to optimisations that are sometimes required when deploying to edge devices [9].

### 4.1. Nvidia Jetson Orin Nano Results

As the Orin Nano can run at multiple power profiles, we explored the performance differences between the nominal 15W profile and the reduced power 7W profile.

Initially, we deployed the model and dataset on the Orin Nano at the nominal 15W profile. Table 3 shows interesting results on the deployment of the LSTM model on the Nvidia Jetson Orin Nano 8GB. As we are utilizing its GPU acceleration capabilities we expect slightly faster inferencing times. However, GPU acceleration tends not to affect inferencing times as much as training times. The Orin Nano is capable of 40 TOPS (Trillion Operations Per Second), specifically at the maximum power mode



**Table 3**

Model performance results from deployment on the Nvidia Jetson Orin Nano 8GB (15W & 7W) and Raspberry Pi 4 from all datasets

Device	Dataset	AUC Score	Precision	Recall	F1-Score	IPS
Flight Test	Orin Nano 15W	0.87	0.85	0.86	0.85	1661.54
Flight Test	Orin Nano 7W	0.87	0.85	0.86	0.85	1253.00
Flight Test	Raspberry Pi 4	0.87	0.85	0.86	0.85	1161.68
Flight Test	Xeon CPU	0.87	0.85	0.86	0.85	3477.45
TVAC	Orin Nano 15W	0.98	0.84	0.90	0.87	2226.29
TVAC	Orin Nano 7W	0.98	0.84	0.90	0.87	1270.12
TVAC	Raspberry Pi 4	0.98	0.84	0.90	0.87	133.75
TVAC	Xeon CPU	0.98	0.84	0.90	0.87	4407.47
Flight	Orin Nano 15W	-	-	-	-	1658.53
Flight	Orin Nano 7W	-	-	-	-	1206.96
Flight	Raspberry Pi 4	-	-	-	-	1058.96
Flight	Xeon CPU	-	-	-	-	2881.14

of 15W [5]. In this work, the model performs the fastest on the Thermal Vacuum dataset, where it is mostly out-performing the Flight Test and Flight datasets in terms of inferencing times.

Table 3 demonstrates further results on the deployment of the LSTM model on the Nvidia Jetson Orin Nano 8GB, which is set to a power mode of 7W and utilizing its GPU acceleration capabilities. There is a clear difference between the IPS in the 15W and 7W power modes. Most notably, the IPS difference on the TVAC dataset which went from 2226.29 predictions per second for the 15W power mode, to 1270.12 predictions per second while using the 7W power mode. The performance difference in the two power modes is primarily due to reduced power availability in the 7W mode, which limits the GPU's computational capability, reduces clock speeds, and thus lowers the rate at which predictions can be made by the model. Performance metrics such as AUC score, precision, recall and F1 score are not effected by the hardware power mode as they measure the effectiveness of the model's predictions, not the speed at which the predictions are made. It is important to highlight that, since the Jetson Nano operates on a full Ubuntu OS, the model's accuracy remains unaffected by factors such as floating-point precision variations. This is because no adjustments to floating-point computations are required in this environment, ensuring that the model maintains its performance consistency during deployment.

The deployment of the LSTM model on the Raspberry Pi 4 [6] produced the results seen in Table 3. The Pi has an average operational power usage of 5W which makes the IPS results comparable to the Orin Nano (7W). The Pi falls short of the Orin Nano performance but with a slightly higher power draw of 7W, this is to expected. One very poor result was the IPS for the Thermal Vacuum dataset. This is a result of memory issues with the Pi, producing a much lower IPS than the other two datasets.

Table 3 shows the highest inference per second values compared to the other hardware deployments. The Intel Xeon CPU [7] operates @ 2.20GHz with 8 cores. This is a naturally powerful unit and uses 61W, which is not suitable for the low-powered devices being targeted. However, it is a very interesting comparison between this CPU and the Orin Nano (7W) as it gives a good baseline to compare model performance on edge devices to PCs. Focusing on the Flight dataset, the difference in IPS is 1611.02 for a power difference of 54W.

## 5. Conclusions and Future Work

The Long Short-Term Memory (LSTM) model developed in this study is notably smaller in size compared to other publicly available models but maintains competitive classification accuracy. This compact architecture is well-suited for deployment on low-power devices, achieving a consistent inference rate exceeding 1000 inferences per second, except in cases where memory limitations are encountered, such as on the Raspberry Pi. The high inference rate is particularly advantageous for classifying analogue components like reaction wheels, whose housekeeping telemetry rates frequently surpass

100Hz. Additionally, the model's efficient performance suggests that there is substantial computational overhead available, providing the flexibility to either scale the model for enhanced accuracy or to execute it alongside other on-board artificial intelligence (AI) tasks, such as Earth Observation (EO) algorithms. This integration could further strengthen the robustness and reliability of EO algorithms. A similar approach could be applied to Autonomous Navigation Systems (ANS), potentially enhancing their reliability as well. Overall, this work has shown that a small deep learning model could in fact run on hardware capable of being flown on the same mission the dataset has come from, enabling a next generation of satellite health monitoring and reliability.

This work demonstrates the feasibility of running a small-scale deep learning model on hardware suitable for space missions, thus enabling next-generation satellite health monitoring and reliability. Given the clear distinction between nominal and anomalous events in the dataset, future efforts will focus on refining the model to better characterize specific activities aboard EIRSAT-1. This will involve developing a multiclass classification model to not only detect anomalies but also classify them into predefined categories such as GMOD, EMOD, or unknown classes. Deploying this model on live EIRSAT-1 telemetry at the ground station will enable real-time detection of Out-of-Ordinary Operations (OOOs). In the long term, we aim to test this model on a future nanosatellite mission to enable real-time, on-board anomaly classification, facilitating a fully autonomous satellite health monitoring system.

## Acknowledgments

This work was supported by Irish Research Council PhD Grant EBPPG/2020/11. This work was also supported by an European Space Agency General Science and Technology Program Grant. This work was further supported through the work done by ML Labs PhD program.

## References

- [1] J. Murphy, F. Wang, M. D. Mazhar Qureshi, B. M. Namee, From ground to orbit: Enhancing satellite autonomy with ai-powered anomaly detection (2024).
- [2] S. Hochreiter, Long short-term memory, Neural Computation MIT-Press (1997).
- [3] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.
- [4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, the Journal of machine Learning research 12 (2011) 2825–2830.
- [5] NVIDIA Jetson Orin Nano Series Modules, Technical Report, 2788 San Tomas Express Way, Santa Clara, CA 95051, USA, 2024.
- [6] Raspberry Pi 4 Datasheet, Technical Report, Raspberry Pi Foundation, 37 Hills Road, Cambridge, CB2 1NT, UK, 2024.
- [7] Intel Xeon CPU Product Brief, Technical Report, 2200 Mission College Blvd, Santa Clara, CA 95054, US, 2024.
- [8] J. Murphy, J. E. Ward, B. Mac Namee, Low-power boards enabling ml-based approaches to fdir in space-based applications, 2021.
- [9] J. Murphy, J. E. Ward, B. Mac Namee, An overview of machine learning techniques for onboard anomaly detection in satellite telemetry, in: 2023 European Data Handling & Data Processing Conference (EDHPC), IEEE, 2023, pp. 1–6.