Deploying Machine Learning Anomaly Detection Models to Flight Ready AI Boards

James Murphy
Réaltra Space Systems Engineering
jmurphy@realtra.space

Maria Buckley
Ubotica Technologies
maria.buckley@ubotica.com

Léonie Buckley
Ubotica Technologies
leonie.buckley@ubotica.com

Adam Taylor
Adiuvo Engineering & Training
adam@adiuvoengineering.com

Jake O'Brien
Réaltra Space Systems Engineering
jobrien@realtra.space

Brian Mac Namee
School of Computer Science, University College Dublin
brian.macnamee@ucd.ie

Abstract

This study explores the development and implementation of machine learning (ML) models on Edge-AI boards, aiming to identify the most effective solution for anomaly detection systems on space missions. We investigate ML anomaly detection techniques including Autoencoders, Long Short-Term Memory (LSTM) cells, Isolation Forests, and Transformers. These models were trained on a univariate dataset derived from real space missions and deployed on diverse hardware platforms engineered for space environments to comprehensively assess performance. Specifically, we explore space flight ready boards (Ubotica CogniSAT-XE1 and XE2, which incorporate Intel's Myriad 2 and X chips, respectively); commercial, non-space flight ready, edge-AI boards (NVIDIA's Jetson Nano and Google Coral); and Field Programmable Gate Array (FPGA) implementations (from Microchip, AMD, and NanoXplore). We compare the performance of anomaly detection models run on space flight ready and commercial boards (using CPU performance as a benchmark) to provide a thorough comparison of available platforms for onboard anomaly detection. This paper provides a detailed examination of both the optimal ML models and hardware platforms for deploying univariate anomaly detection systems in space flight contexts and draws conclusions about which ones are most suitable.

1. Introduction

Developments in Artificial Intelligence (AI) and Machine Learning (ML) technologies have heralded a new era in various fields, including aerospace engineering and space exploration. With the increasing complexity, number, and duration of space missions, the need for advanced onboard computational capabilities has never been more critical [11]. Anomaly detection, in particular, plays a pivotal role in ensuring the safety and success of these missions by identifying unexpected events or conditions that could lead to failures or mission-compromising scenarios. Anomaly detection implemented on ground-based monitoring systems faces significant challenges due to latency in communication and the vast amount of data generated by spacecraft, making onboard anomaly detection systems a primary area of interest. Recently developed advances in space-ready AI boards have opened up new possibilities for deploying sophisticated ML models directly onto spacecraft, enabling real-time data processing and anomaly detection without the need for constant communication with Earth [15]. This capability is crucial for deep-space missions where communication delays can span from minutes to hours. However, the deployment of such models in space environments poses unique challenges, including limited computational resources, stringent power consumption requirements, thermal issues, and the need to withstand a harsh radiation environment [10].

Given this context, our study focuses on evaluating various ML models for anomaly detection [20], and their suitability for implementation on different hardware platforms that are designed to operate in space conditions. By examining both deep learning and classical learning models, we aim to evaluate their performance, efficiency and compatibility for detecting anomalies in univariate data collected during space missions [16]. Moreover, this research extends

to an in-depth analysis of hardware platforms that are capable of supporting these ML models in space. This includes a comprehensive review of flight-ready boards, which are specifically designed for space applications, FPGA implementations, and commercial boards that offer potential for space deployment. This examination also highlights compatibility issues with certain models when deployed on these boards. Through this examination, we seek to identify the optimal combination of ML models and hardware platforms to enhance the reliability and autonomy of space missions by providing effective onboard anomaly detection capabilities, and therefore provide a valuable contribution to the community.

This paper proceeds as follows: Sec. 2 describes the dataset and anomaly detection approaches used in our experiments; Secs. 3 to 5 describe how models are deployed to commercial edge AI boards, space flight ready boards, and FPGA devices respectively; Sec. 6 presents and discusses the results of evaluating model performance on different devices; and Sec. 7 concludes the study and suggests directions for future work, addressing and offering solutions for compatibility challenges faced during the deployment process, ensuring a seamless integration of ML models with space-qualified hardware platforms.

2. Model Development

This study explores the performance of different hardware deployment options for ML models. To facilitate this research, a set of models was developed and evaluated when deployed on different hardware platforms. This section describes the dataset used to train and evaluate these models and the different ML algorithms used.

2.1. Dataset

The experiments described in this paper use a NASA dataset published by Hundman et al. [16] containing anonymised satellite telemetry from two sources: NASA's Soil Moisture Active Passive (SMAP) mission and NASA's Mars Science Laboratory (MSL). The dataset consists of 82 channels ranging from 1 500 to 9 000 data points. These channels contain a variety of point and contextual anomalies ideal for training and testing ML models. Due to the significant computation required for the experiments presented here, one channel was selected as a case study for our experiments. We use the F-5 channel as, based on [20], it is the most representative of the entire dataset.

2.2. Model Architectures Used

In order to investigate the feasibility of anomaly detection methods, a selection of state of the art ML approaches are used in our experiments. These are selected based on results presented be Murphy et al. [20], and are considered to be the current state of the art in anomaly detection. These models range from classical ML methods such as Isolation Forests to some of the newest deep learning architectures such as Transformers. The model architectures used in this study are:

- Basic Dense Autoencoder (BasicAE)
- Convolutional Autoencoder (CNNAE)
- Variational Autoencoder (VAE)
- Long Short-Term Memory Autoencoder (LSTMAE)
- Hybrid Autoencoder with CNN Encoder & LSTM Decoder (HybridAE)
- Transformer
- Isolation Forest

These models were trained using the Keras environment via TensorFlow 2 in Python [2]. The optimal network architectures were also chosen by using Keras-Tuner as a pseudo-hyperparamater search.

3. Deployment on Commercial Edge AI Boards

With the multitude of devices to run AI "on the Edge" available today, there has been a push in the space sector to qualify and fly commercial Edge-AI boards. In this study, we include representative examples from this category that are not currently recognised as space flight ready, but that have qualified flight ready examples from space engineering companies in different form factors. The Edge-AI boards used in this study, and described below, could be made flight ready, or are already undergoing certification for flight.

3.1. Raspberry Pi 4

The Raspberry Pi 4 offers enhanced processing power and memory capacity, making it a viable platform for deploying basic ML models. The Raspberry Pi 4 features a powerful Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC running at 1.5GHz [22]. The Raspberry Pi has already flown in previous space missions such as ESA's Astro Pi mission and has run on several CubeSat systems as testbeds. This makes the Raspberry Pi an interesting system to include in this study. Running models on a Raspberry Pi 4 is relatively straightforward thanks to the Linux based Operating System (OS), Raspbian. This only involves loading saved TensorFlow models into a Python script and running directly on the OS.

3.2. Google Coral

The Google Coral is powered by a quad Cortex-A53 and uses a Google Edge Tensor Processing Unit (TPU) [5]. It is also optimized for TensorFlow Lite (TFLite) [7], enabling it to run deep learning models with a smaller power footprint. This efficiency makes it ideal for Edge-AI, where processing power and energy availability are often limited. The Google Coral is currently in operation on-board the International Space Station (ISS) as part of a NASA mission to

explore Edge-AI devices in space. To deploy saved TensorFlow models onto a Google Coral, they must first be converted to TFLite format and then ported to the Coral. The Coral runs a Python script directly onboard through its OS and references the imported TFLite model files. Unfortunately, the requirement to convert models to TFLite has precluded several models in this study from being deployed to the Google Coral due to compatibility issues.

3.3. NVIDIA Jetson Nano

The NVIDIA Jetson Nano is a compact but powerful computing platform used by many terrestrial applications. Equipped with an NVIDIA Graphics Processing Unit (GPU), the Jetson Nano is capable of running parallel computations essential for deep learning models. This makes it ideal at processing the complex calculations required for ML tasks efficiently. The Jetson Nano is ideal for space applications where power availability is limited. Its energy efficiency does not come at the cost of performance, making it a balanced choice for demanding tasks [6]. The Jetson Nano is likely the most compatible system for deploying ML anomaly detection models as it runs its own version of Ubuntu on the development board. This allows all forms of machine learning frameworks to be used and deployed. The Nano runs a customised version of Ubuntu which has access to standard ML libraries such as TensorFlow, and has CUDA available natively [6]. This means that running TensorFlow models onboard is very straightforward and simply a matter of loading and running the TensorFlow model files as if on a standard computer system.

4. Myriad Based Systems Implementation

The Ubotica CogniSAT-XE1 and CogniSAT-XE2 On-Board AI Payload Processors bring Computer Vision (CV) and AI compute acceleration to a PC/104 form-factor for SmallSat and CubeSat missions. They are built around the Intel Movidius Myriad 2 and Intel Movidius Myriad X CV and AI Vision Processing Units (VPU), respectively, whose custom vector processors provide high-performance parallel and hardware accelerated compute within a low power envelope. Either Gigabit Ethernet or USB2.0/3.0 can be used as the primary control and data interface to the payload processors, enabling data rates sufficient to handle many CV and AI applications at near-streaming throughput. Both the Myriad 2 and Myriad X have flight heritage, and both have been subjected to radiation test campaigns and have been qualified for space flight [9, 10, 12].

The Myriad 2 flew as part of the Φ-Sat-1 mission which was the first Earth observation CubeSat to include a dedicated AI accelerator on board to run a neural network. The CogniSAT-XE1 will fly on the Φ-Sat-2 mission, a 6U Cubesat due to launch in 2024. This will allow developers to run neural networks on the XE1 during flight [15, 19]. Since

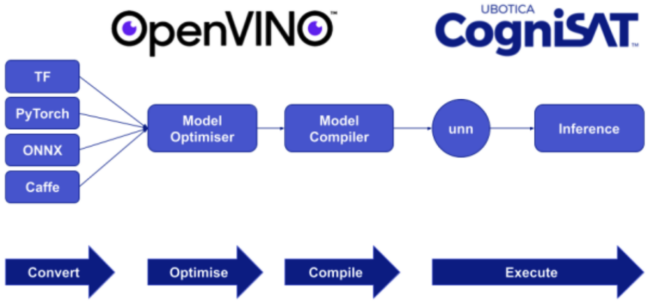


Figure 1. Complete conversion flow from the CogniSAT training environment

March 2024, the XE2 has flown as part of the CogniSAT-6 mission, a Live Earth Intelligence satellite developed by Ubotica and Open Cosmos.

4.1. Network Compilation and Deployment

Common ML frameworks (e.g., TensorFlow, PyTorch, Caffe) can be used for neural network (NN) model development and training, with the model subsequently compiled to target the Myriad device using Intel's OpenVINO toolkit. To compile an NN model to a Myriad device, each layer of the NN must be supported by OpenVINO [3]. Additional layer support has been added with each release of OpenVINO. As OpenVINO 2020.3 is the latest version that supports the Myriad 2, it can happen that layers that are supported by OpenVINO 2022 and the Myriad X, are not supported by OpenVINO 2020, and therefore certain network architectures cannot be compiled to the Myriad 2.

In order to run a network on the CogniSAT hardware, a Myriad-specific version must be compiled. This is a two step process, *optimise* and *compile*, and is shown in Fig. 1.

The Model Optimiser is used to improve final model performance by applying optimisation methods such as quantisation and pruning [3]. Common pre-processing operations can be integrated into the network using the Model Optimiser, such as normalisation using scale and mean values, and input channel order swapping. The network format generated using the Model Optimiser is platform agnostic and is known as an Intermediate Representation (IR). The IR can be deployed to Intel CPUs and GPUs as well as the Myriad using OpenVINO [4]. Compilation of the IR to a Myriad specific format is performed using a compilation tool provided by OpenVINO [4]. Parameters such as the input and output precision (e.g., FP32, FP16) can be specified using the compile tool.

Ubotica's CogniSatApp software allows for the deployment of NNs to CogniSAT hardware. The use of JavaScript Object Notation (JSON) configuration files allows for easy deployment of new applications without the need to write/alter any code. The design of CogniSatApp

Table 1. Results for compiling and running the models on the XE1 and XE2

Model Architecture	Compatibility	
	XE1	XE2
BasicAE	Yes	Yes
CNNAE	Yes	Yes
HybridAE	No	Yes
LSTMAE	No	Yes
Transformer	No	Yes
VAE	No	No
Isolation Forest	No	No

addresses pre- and post-processing in a generic manner, allowing for no or multiple pre- and post-processing operations to be deployed.

4.2. Model Compilation

The models used in this study are exported from the Keras framework [2], and are provided in the .h5 format. The NN optimisation and compilation flow described in the previous section is followed for each of these networks. It was found that not all networks could be compiled for the XE1, and mitigating steps were required to compile some of the NNs for the XE2. This is summarised in Tab. 1, where "Yes" means that the model is supported and "No" means that the model is not supported.

The BasicAE and CNNAE networks compiled and ran on both the XE1 and XE2 successfully. The Transformer network compiled and ran on the XE2, but could not be compiled to run on the XE1 due to layers not being supported. Both the HybridAE and LSTMAE networks were compiled to run on the XE2, but required a conversion to ONNX (Open Neural Network Exchange) due to layer incompatibility (Tensorflow Loop v5) when converting directly from the provided .h5 network format. These networks could not be compiled to run on the XE1 due to specific layers not being supported. Finally, the VAE could not be successfully compiled to run on either board due to the limitations of OpenVINO with custom layers and loss functions required by variational autoencoders. Isolation forests were also found to be incompatible with both the Myriad 2 and Myriad X architectures. However, this was a layer issue and in general, should not preclude other classical learning architectures from being deployed.

4.3. Deployment and Model Assist Pipeline

When deploying models to the Myriad, comparison against a test and validation dataset is required, as well as comparison against the performance obtained when running on the CPU using Keras. For this purpose, the Model Assist application was developed.

Figure 2. Model Assist application flow diagram

The Model Assist application allows for easy deployment of the networks on CPU (using Keras) and on the XE1 or XE2, and to allow for comparison of inference metrics and benchmarking parameters. The application performs pre- and post-processing according to the specified network architecture. A flow diagram of the application can be seen in Fig. 2. Specification of the model architecture, pre- and post-processing, and data directory structure are provided in a JSON configuration file, and the runmodes are specified by the user with runtime arguments. The application allows for each network to be run on: CPU using Keras, and XE1 and XE2 using CogniSatApp.

Batch processing was implemented for some of the models when running inference on the XE1 and XE2. Batch processing is particularly advantageous when the duration

required to transfer the input data from the host machine to the inference engine is significantly greater than the inference time itself. The batch size that resulted in the highest throughput was selected for each model.

The XE1 and XE2 were deployed to the XE1 and XE2 over USB.

5. Deployment onto FPGAs

Satellites intended for use in geostationary orbit, or on deep space exploration missions, utilise radiation qualified Field Programmable Gate Arrays (FPGA), typically QML V (Qualified Manufacturer List) due to higher radiation environments. These devices are provided from a range of manufacturers including Microchip, AMD and NanoXplore [13], and each mission typically uses a different FPGA technology depending upon the performance and size requirements of the mission. In this study we evaluate models deployed onto a range of FPGA devices.

The most objective solution is to create a device independent system, i.e., can use any underlying FPGA technology. This rules out vendor solutions which tie the system to a particular FPGA technology. Some vendors such as NanoXplore do not provide ML frameworks for their technology to be used in this scenario [13].

5.1. Implementing Neural Networks for FPGA

Implementing a generic ML solution for FPGA can be achieved in several different ways. The first would be to implement the network using a Hardware Description Language (HDL). Such an approach, however, would bring with it complexity of conversion from the model to the Register-Transfer Level (RTL) implementation and prohibit the scaling of the solution and easy modification. There would also be the traditional FPGA challenges of verification of the design prior to implementation [13].

A second approach would be to leverage commercial tools such as MATLAB / Simulink which enable the generation of HDL from models. However, this approach requires the use of high performance external memory such as DDR3/4. External memory may not be available for all deployed applications.

The final potential approach is to implement the neural network using TinyML [8, 21] deployed on a small softcore processor that can be implemented within the FPGA fabric. The advent of several open source softcore RISC-V processors ensures this can be a technology independent solution. This enables the implementation of a RISC-V core within a target FPGA, while the network is compiled using the industry standard framework of TensorFlow Lite for Microcontrollers to implement the solution [7]. In this RISC-V based approach the application executes from the tightly coupled block RAM memories of the target FPGA. This means the FPGA programming file can be generated with the application included. It also means that a deployment framework which can enable on-the-bench and in-orbit reconfiguration can be created, as the tool flow and object generation is very straightforward (summarised in Fig. 3). There is no need for the FPGA design to be changed to update the network.

For this study, several RISC-V cores were considered, including the offerings from NIOSV, MicroBlaze V, and MiV32 along with Open Source RISC-V Processors including VexRISCV, NEORV32, IBEX and SERV. The processor selected was the NEORV32 as this provided the most complete package of configurable system-on-chip (SoC), built tools and an advanced extensible interface (AXI) for interfacing with several different peripherals should it be required [1].

5.2. Toolchains and Development Flow

One additional key consideration in this work has been the ease of re-targeting between FPGA vendors for deployment. As such, vendor tool chains are only used to perform the implementation and bit stream generation. To avoid the added complexity of creation of multiple build scripts required for different vendors, it was decided to use the open source build tool FuseSoC [18]. FuseSoC is a package manager and build system for HDL code, and allows easy definition of several targets. At the heart of FuseSoC is the concept of a core, this is a reusable element of IP. With the IP Cores defined, the FuseSoC package manager and build system is able to organise the design we intend to create. EDALize is then called from within FuseSoC to create the specific electronic design automation (EDA) tool projects. Using FuseSoC creates a flexible build chain which is able to target any of the target devices [18].

In addition to FuseSoC, to enable portability and scalability of the solution the NEORV32 processors was provided with a generic wrapper which enables its integration with FPGA Architect SysML to structural Register Transfer Level (RTL) Framework. This enables the integration of the machine learning processor within larger FPGA designs.

FPGA Architect is a graphical FPGA architectural tool which enables developers to capture the architecture of the design graphically. Once captured graphically the diagram can be generated as VHDL. This implementation is interface aware and creates all of the AXI networks to ease development. The source created is also vendor independent.

As we are targeting FPGA architectures, we need to consider the deployment requirements of the AMD Kintex KU60 as it is an industry standard for space missions [14]. When targeting the AMD Kintex KU60 using FuseSoC to build the design results in a small, compact implementation, which uses under 3% of the available resources of the FPGA. For bench testing the NEORV32 is deployed with its standard boot loader application which allows new exe-

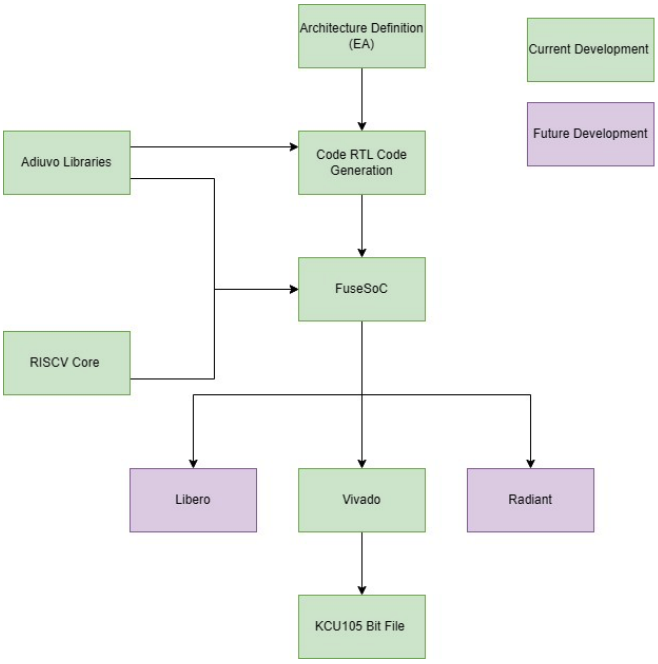


Figure 3. FPGA Development Flow Diagram

cutables to be uploaded and run over the serial port.

Deploying models onto the FPGA at the initial stage is achieved by taking the Tensorflow models and converting them to TFLite models, which are then converted into C header files for use with TFLite for Microcontrollers.

6. Evaluating Deployment Options

To identify the optimal combination of ML models and hardware platforms for reliability and autonomy of space missions through effective onboard anomaly detection capabilities, we have performed an evaluation experiment to compare the performance of the models described in Sec. 2 when deployed on the commercial and space flight ready hardware solutions described in Secs. 3 and 4. Performance on a CPU (Intel i9 14^{th} generation) is also included for refenrece. After deploying these models to each of the hardware solutions, we evaluate the performance using the test dataset described in Sec. 2.1. Anomaly detection performance is measured using the Area Under the Receiver Operating Characteristic Curve (AUC ROC) and the F1 Score measure [17]. The throughput of models during deployment, measured in Inferences Per Second (IPS), and their power consumption, measured in Watts, are also monitored. In order to consider a solution suitable for space flight use, we set the minimum inferencing throughput at 100Hz and the maximum power draw to be 5W as these are good baselines for small satellite missions due to the power restrictions and average on-board data-rates. This will give us a baseline for the feasibility of operating these models on Small Satellite missions.

6.1. Results

The results from this experiment are displayed in Tab. 2 and Fig. 4. The table combines all measures used in this study to understand the relationships between the models themselves and the boards they are deployed on. The heatmaps in Fig. 4 are used to visualise the relationships between the key measures in order to help choose a device and model architecture for future anomaly detection missions.

6.2. Discussion

The comparison of inferencing performance in Tab. 2 yields interesting results. Although the same models were deployed onto all boards, there are slight differences in some AUC and F1 scores between inferencing directly on the model and inferencing on a loaded TensorFlow .h5 file for some model architectures. There were also some differences caused by differences in floating point accuracies present in some systems compared to running on a CPU capable of FP64. Some other aspects that may have affected results could be not running ML models via CUDA on the Nano and the conversion process to TFLite required for inferencing on the Coral. The most surprising result is how well the Raspberry Pi performed, given it is not optimised for ML models.

Another interesting outcome is the relatively high throughput of the Isolation Forest. This is likely due to its computationally simplicity compared to the neural network models. However, it was higher in power draw than almost all other models and lower in overall accuracy.

Generally, these results are inline with what is expected from the datasheets for each device, especially when it comes to throughput. One surprising outcome is how close all of the devices are in power draw with the most power hungry system, the Raspberry Pi, only just going above 5W of total power during inferencing. The idle power consumption for the XE1 device is 0.73W and for the XE2 is 1.22W. Idle power for the Nano is 3.7W, the Coral is 3.45W and the Pi is 3.5W. Relative power consumption is the total power consumption minus the idle power consumption.

7. Conclusion

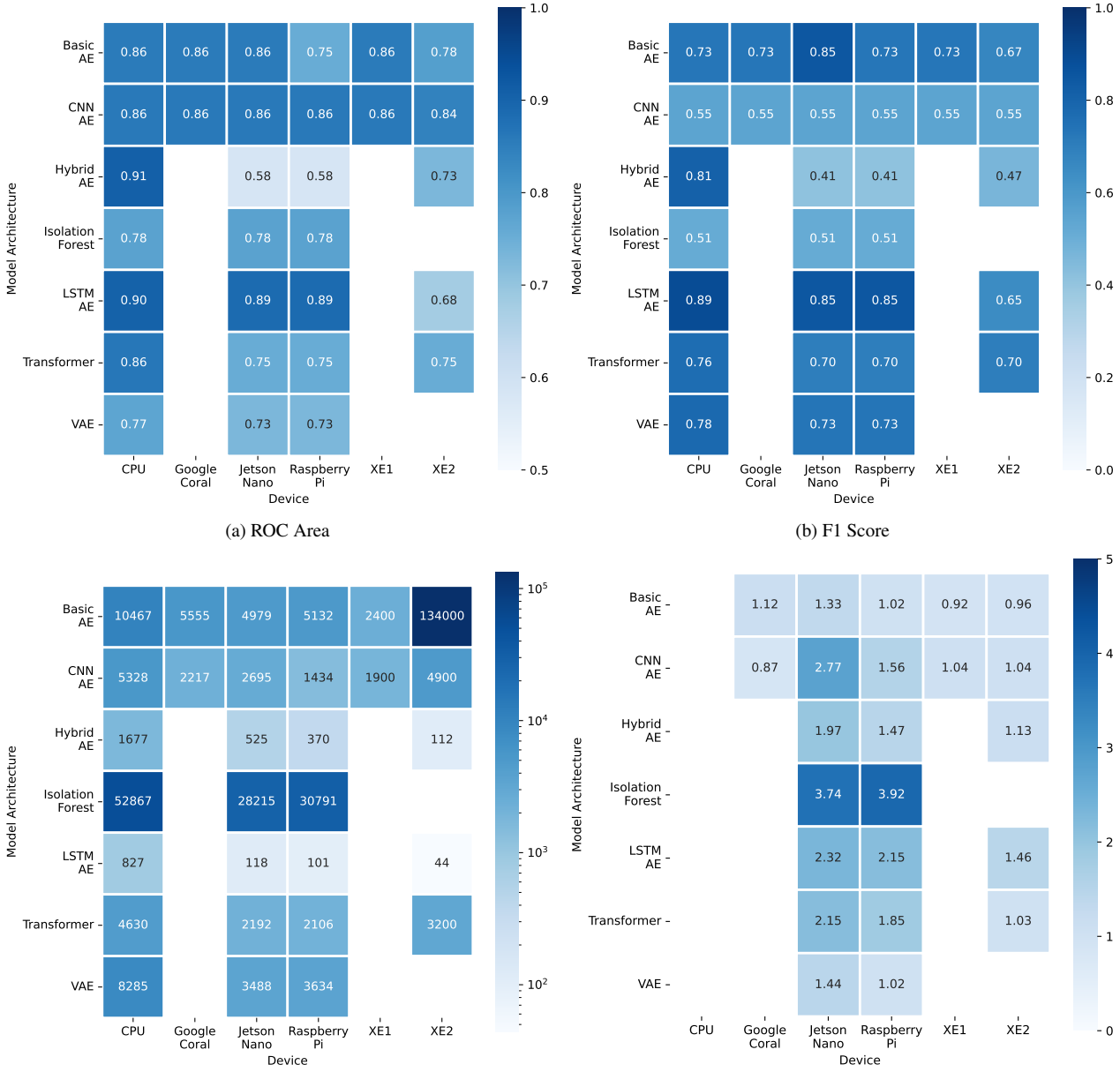
The ideal model to use for anomaly detection is highly dependent on the hardware available. This study shows that, like in previous work [20], the LSTM Autoencoder is the best performing model when it comes to model accuracy, but pays for this with slow inferencing times and compatibility issues. However, if we consider how newer boards and frameworks are dealing with these compatibility issues, it is likely that in future iterations of these devices this will no longer be an issue.

Table 2. Performance Metrics for models running on all devices.

Model Architecture	Device	ROC Area	F1 Score	Inferences Per Second (IPS)	Relative Power Consumption (W)
Basic AE	CPU	0.857	0.730	10 467	-
	XE1	0.857	0.730	2 400[1]	0.92
	XE2	0.784	0.669	134 000[2]	0.96
	Jetson Nano	0.857	0.848	4 979	1.33
	Raspberry Pi	0.754	0.730	5 132	1.02
	Google Coral	0.857	0.730	5 555	1.12
CNN AE	CPU	0.860	0.545	5 328	-
	XE1	0.861	0.545	1 900[1]	1.04
	XE2	0.842	0.545	4 900[1]	1.04
	Jetson Nano	0.860	0.545	2 695	2.77
	Raspberry Pi	0.860	0.545	1 434	1.56
	Google Coral	0.860	0.545	2 217	0.87
LSTM AE	CPU	0.903	0.892	827	-
	XE1	-	-	-	-
	XE2	0.679	0.646	44	1.46
	Jetson Nano	0.885	0.848	118	2.32
	Raspberry Pi	0.885	0.848	101	2.15
	Google Coral	-	-	-	-
Hybrid AE	CPU	0.907	0.810	1 677	-
	XE1	-	-	-	-
	XE2	0.729	0.465	112	1.13
	Jetson Nano	0.585	0.413	525	1.97
	Raspberry Pi	0.585	0.413	370	1.47
	Google Coral	-	-	-	-
Transformer	CPU	0.863	0.764	4 630	-
	XE1	-	-	-	-
	XE2	0.754	0.703	3 200[1]	1.03
	Jetson Nano	0.754	0.703	2 192	2.15
	Raspberry Pi	0.754	0.703	2 106	1.85
	Google Coral	-	-	-	-
VAE	CPU	0.770	0.779	8 285	-
	XE1	-	-	-	-
	XE2	-	-	-	-
	Jetson Nano	0.729	0.733	3 488	1.44
	Raspberry Pi	0.729	0.733	3 634	1.02
	Google Coral	-	-	-	-
Isolation Forest	CPU	0.776	0.514	52 867	-
	XE1	-	-	-	-
	XE2	-	-	-	-
	Jetson Nano	0.776	0.514	28 215	3.74
	Raspberry Pi	0.776	0.514	30 791	3.92
	Google Coral	-	-	-	-

[1] Batch size 100

[2] Batch size 1000



(a) ROC Area

(b) F1 Score

(c) Inferences Per Second (IPS) (note log scale)

(d) Relative Power Consumption (W)

Figure 4. Heatmaps of performances by device and models.

As we set out to gauge the capability of these devices to run anomaly detection models in Small Satellite missions with the given targets of 100Hz inferencing throughput and a maximum power draw of 5W, almost all of these models and devices have shown that they are effective choices for deployment in space missions. As all of these devices have space heritage, albeit in some cases in different form factors, the decision of which device to use will be determined by the mission profile and power available.

It is clear that more work is needed by the community on compatibility problems, particularly with newer model architectures. However, it is also clear how much potential these models have within the space domain. The planned next steps include expanding the solutions compared (including FPGA) and developing multivariate models to expand anomaly detection capabilities to the entire satellite rather than individual channels. The far future goal is to fly these models in satellite systems and prove how valuable they can be.

References

[1] https://stnolting.github.io/neorv32/ug/. Accessed: 2024-03-08. 5

[2] https://keras.io/. Accessed: 2024-03-08. 2, 4

[3] https://docs.openvino.ai/2022.3/openvino_docs_model_optimization_guide.html. Accessed: 2024-03-08. 3

[4] https://docs.openvino.ai/2022.3/openvino_docs_model_optimization_guide.html. Accessed: 2024-03-08. 3

[5] https://coral.ai/products/dev-board/. Accessed: 2024-03-08. 2

[6] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano-developer-kit/. Accessed: 2024-03-08. 3

[7] https://www.tensorflow.org/lite. Accessed: 2024-03-08. 2, 5

[8] N. N. Alajlan and D. M. Ibrahim. Tinyml: Enabling of inference deep learning models on ultra-low-power iot edge devices for ai applications. *Micromachines*, 13(6):851, 2022. Line 341. 5

[9] L. Buckley, A. Dunne, G. Furano, and M. Tali. Radiation test and in orbit performance of mpsoc ai accelerator. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–9, Big Sky, MT, USA, 2022. IEEE. 3

[10] Megan Casey, Ed Wyrwas, and Rebekah Austin. Recent radiation test results on cots ai edge processing asics. In *NEPP Electronics Technology Workshop (ETW)*, 2022. 1, 3

[11] I. Del Portillo, B. G. Cameron, and E. F. Crawley. A technical comparison of three low earth orbit satellite constellation systems to provide global broadband. *Acta astronautica*, 159:123–135, 2019. 1

[12] E. Dunkel, J. Swope, Z. Towfic, S. Chien, D. Russell, J. Sauvageau, D. Sheldon, J. L. Romero-Canas, and L. Buckley et al. Espinosa-Aranda. Benchmarking deep learning inference of remote sensing imagery on the qualcomm snapdragon and intel movidius myriad x processors onboard the international space station. In *IGARSS 2022-2022 IEEE International Geoscience and Remote Sensing Symposium*, pages 5301–5304. IEEE, 2022. 3

[13] H. Foster. The 2022 wilson research group functional verification study. Siemens. com, 2023. 5

[14] C. M. Fuchs, P. Chou, X. Wen, N. M. Murillo, G. Furano, S. Holst, and K. Marinis. A fault-tolerant mpsoc for cubesats. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2019. 5

[15] G. Giuffrida, L. Fanucci, G. Meoni, M. Batic, L. Buckley, A. Dunne, Č. van Dijk, M. Esposito, J. Hefele, and N. et al. Vercruyssen. The φ-sat-1 mission: The first on-board deep neural network demonstrator for satellite earth observation. *IEEE Transactions*. 1, 3

[16] 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*, pages 387–395, 2018. 1, 2

[17] John D Kelleher, Brian Mac Namee, and Aoife D'arcy. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press, 2020. 6

[18] O. Kindgren. A scalable approach to ip management with fusesoc. In *1st Workshop on Open-Source Design Automation (OSDA)*, 2019. 5

[19] N. Longepé, N. Melega, V. Marchese, A. Paskeviciute, I. Babkina, I. Petrelli, M. Casaburi, D. Peressutti, and N. O. Kadunc. φsat-2 mission overview for the orbitalai challenge, version 1.0. Tech. rep., European Space Agency, 2023. 3

[20] J. Murphy, J. E. Ward, and 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)*, pages 1–6. IEEE, 2023. 1, 2, 6

[21] P. P. Ray. A review on tinyml: State-of-the-art and prospects. *Journal of King Saud University-Computer and Information Sciences*, 34(4):1595–1623, 2022. 5

[22] E. Upton and G. Halfacree. *Raspberry Pi user guide*. John Wiley Sons, 2016. 2