Towards On-Orbit Anomaly Detection using LSTM RNN

An AMD VE2302 Adaptive SoC based solution

A. C. McCormick (Alpha Data), A. Adetomi (Alpha Data) and K. O'Neill (AMD)

Abstract

In modern satellites, telemetry data is collected from the multiple instruments and subsystems on board. Parameters such as voltage, current, temperature, pressure, flow rate, magnetic field strength, electrical field strength, and mechanical strain are measured. Small satellites may have dozens or hundreds of telemetry channels, while a large, Class-A government satellite may have as many as 4,000 channels of telemetry data. This data is collected at each instrument and passed to the Telemetry Tracking and Control system on board the satellite, for transmission to the satellite's mission control center on the ground. Analysts review the telemetry data, looking for out-of-character or out-of-specification data which could indicate a potential or developing fault condition, possibly requiring a change in operation mode of the satellite or even the preparation of another satellite to tak over duties of a satellite which will imminently fail. Human analysis activities are time consuming, repetitive and error-prone, and hinder the goal of timely preparation for imminent fault conditions. Creating AI-based telemetry anomaly detection systems to fly in space would enable real-time detection of possible or developing fault conditions and would buy valuable time for satellite operators to prepare alternative resources, minimizing loss of mission data.

In this paper we show the development of an AI-based telemetry processing and analysis system which uses long short-term memory (LSTM) recurrent neural networks (RNN), implemented on an AMD Versal TM Edge adaptive SoC to monitor 40 channels of telemetry data. System performance and device utilization is shown. The implementation of the anomaly detection system can be viably integrated into AMD's Versal Edge VE2302 adaptive SoC, which will be offered in radiation-tolerant version qualified for space flight, allowing for autonomous detection of telemetry anomalies in real time on orbit.

Introduction

The objective of the work reported in this paper is to investigate a potential Space-deployable hardware implementation of a representative LSTM-based anomaly detection solution for Spacecraft Telemetry already tested in ground-based data center hardware [**[1]**]. While GPU and CPU based computing clusters and data centers, can provide very flexible, easily programmable, high-performance processing for machine learning batch inference and model training, these solutions are often much less suitable for on-orbit deployment where power efficiency, long-term product lifetime, radiation tolerance and low-latency inference are priorities. This work will investigate the deployment of an inference solution in more suitable hardware provided by AMD, namely the Versal Edge VE2302 device, which is available as a flight-qualified radiation-tolerant component, includes the AMD XDNA (AIE-ML) architecture suitable for very power-efficient, low-latency inference implementation, and has an expected device availability of over 15 years.

This paper will only report the implementation of one representative anomaly detection solution. There are many similar approaches that are potentially equally valid and applied to anomaly detection in spacecraft [2] and many other domains **[2]**, **[3]**, **[4]**, **[5]**, **[6]**; however, comparison between these on data science criteria will not be made. Only where features of the representative detection solution impact on the implementation will the data science be touched on, specifically the data type and model sizes. Specifically, this project has avoided any requirement to train models, and all results are based on models and the open-source data set available from the reference solution.

In this paper, the high-level architecture of the anomaly detection solution will be described including how different stages map into different areas of the Versal Edge device. The instantiation in detail of a single thread of execution will be described, and simulation results presented, comparing LSTM behavior with the reference data, at a range of data types. The scaling up of this solution will also be discussed, taking two approaches: first a Time Division Multiplexing approach, which shares the hardware between multiple threads of telemetry data, and multiple models; secondly the parallel nature of the AIE-ML hardware will be exploited to run many instances of the TDM multi-threaded solution in parallel, scaling up to a solution that can implement around 40 models in parallel.

High Level Architecture

The Versal Edge architecture is an Adaptive System on Chip with several distinct regions with different processing capabilities. At the scalar side, there is a multi-core based CPU system that can provide a powerful level of control over the device as well as significant sequential processing capability, with two ARM Cortex(R)-72 Application Cores, two ARM Cortex-R5 Real-Time cores, and also a pair of smaller Triple-Mode Redundant hardened Microblaze CPUs dedicated to chip management functions and tasks such as radiation mitigation scrubbing.

In addition to this, the chip includes a sizeable area of Programmable Logic which is fully reconfigurable to allow the implementation of a large range of processing circuits, and access to very flexible I/O, as one might expect within an FPGA-related device. This includes a significant number of DSP tiles, which may be very useful for sensor pre-processing.

While there are a few other key features within the device, especially some powerful hardware IP cores and a Network on Chip (NoC), the third main area that is of significance to this application is the array of AIE-ML engines sometimes referred to as the XDNA architecture. This is an array of VLIW DSP processor cores, each capable of performing SIMD vector arithmetic and having their own memory buffers arranged in a grid array highly suited to streaming data between the cores. This architecture has demonstrated significant performance per watt and latency advantages for Machine Learning applications compared with competing architectures.

The reference AI system here being used as a source of the models and data involves three stages. The first stage, collecting the telemetry signal and combining it with the input command signals into a single feature vector for the trained model, is performed prior to the application of the system. The reference data for testing is this pre-processed input and can effectively be made available in memory for early development and prototype testing. For a deployable system, this data may be collected directly into programmable logic pins, or interfaces implemented in that part of the device.

The second stage of the reference system is a Long Short-Term Memory (LSTM) Recurrent Neural Network. This is a trained machine learning model that aims to capture the dynamics of a physical system by predicting future values from the current input and also its memory state, captured in each LSTM layer. This memory state is updated every cycle based on the new inputs and the previous values, with some weights setting a level of forgetting, determining how many iterations an input value will influence the memory. The model behaves as a multi-dimensional non-linear auto-regressive (similar to an IIR/ARMA) filter with the influence of each input sample vector decreasing over time. The output of this model is a prediction of the future behavior of the telemetry channel.

Then the third stage of the reference system compares the predicted values with the real values once they are available. Usually, the determination of the presence of an anomaly is the computationally relatively simple task of thresholding. The determination of such threshold levels is, however, non-trivial, and beyond the scope of this paper. The reference system paper [1] proposed a dynamic unsupervised learning approach and compared this with more conventional statistical approaches assuming a Gaussian distribution of error values. In terms of implementation, these approaches and others are relatively computationally light in comparison with the LSTM stage and should all be comfortably computable by the scalar processing ARM cores within the Versal Edge device.

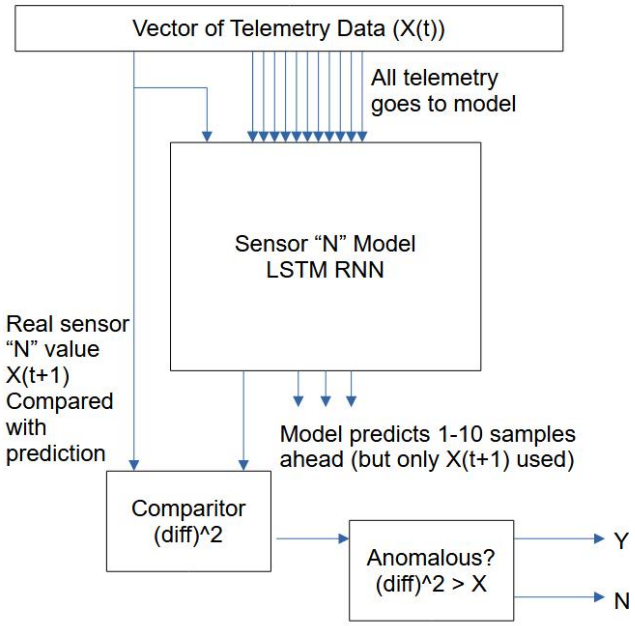

Figure 1 : Reference System Implementation using GPU/CPU hardware

The reference paper reported on the processing of downloaded telemetry data, giving a long-term analysis of the overall spacecraft health, cataloguing potential anomalies and possibly allowing decisions to be made to ensure the long-term viability of the mission. Figure 1 shows the structure of this use case. This paper however aims to look at how real-time anomaly detection, which could allow for almost instant decision making, perhaps useful in the case of some anomalies, might be implemented to allow a degree of autonomy in the spacecraft's health maintenance.

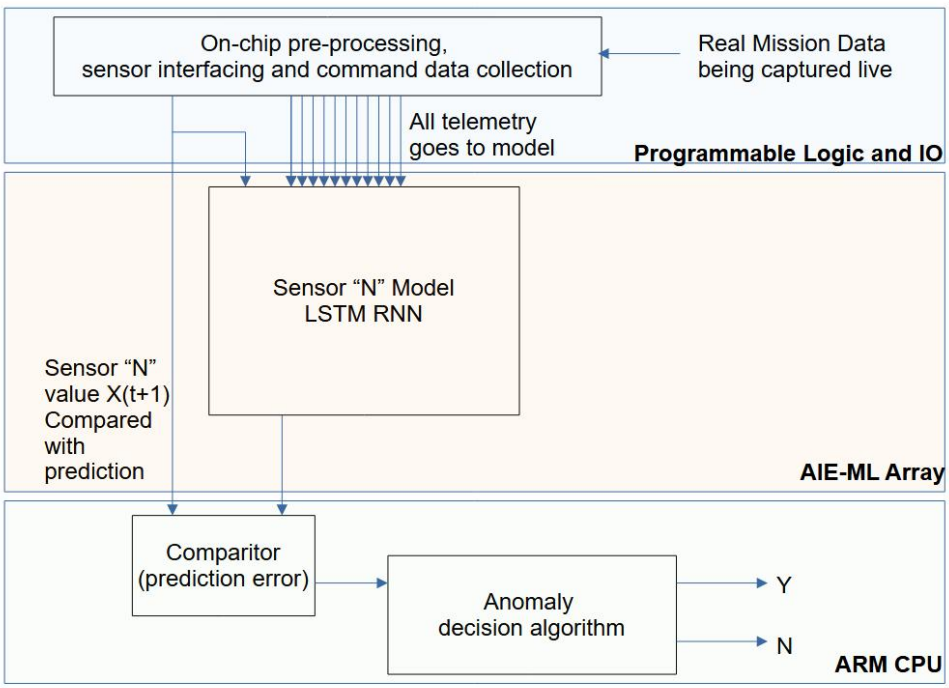

Figure 2 : Proposed flight system, with real-time anomaly detection

A deployable solution, therefore, requires a structure closer to the one shown in Figure 2. In this case, sensor and command data is captured directly by the programmable logic IO and fed into the LSTM model. As a part of the development of such a solution for this paper, and since testing it relies on captured data, the architecture shown in Figure 3 is used for the proof of concept.

The proof-of-concept implementation targets the VEK280 development board from AMD, which contains a Versal Edge chip with similar capabilities as the XQRVE2302 device that has been released for Space applications. The reference data and models are loaded onto the off-chip DDR4 memory banks for processing. At configuration and startup time, the models are loaded into the on-chip AIE-ML shared memory as this data is re-read for every data sample, and therefore caching the data here will minimize memory bandwidth limitations. The predictions of the LTSM, running in the AIE-ML part of the device are output to be used by the embedded ARM core performing the control function and this processor core can also be used to perform any anomaly detection thresholding comparisons required.

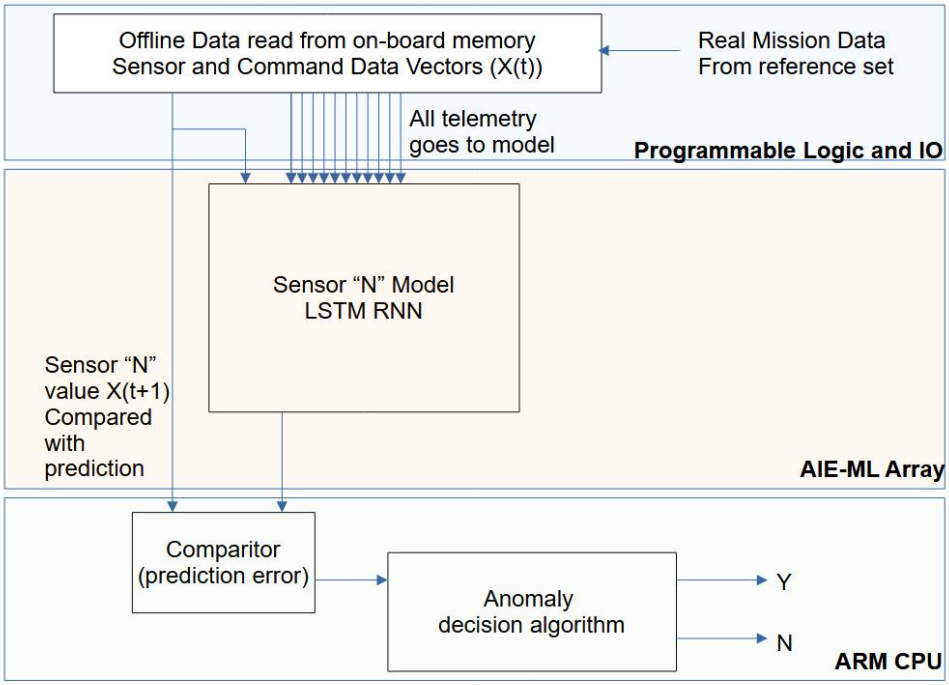

Figure 3 : Prototype system developed as a reference and proof of concept

LSTM Implementation using the AMD XDNA AIE-ML Architecture

Figure 4 shows the general structure of the LSTM model, which consists of three stages. The first two stages are recurrent LSTM layers which feedback their outputs to use in the calculation of the next vector of outputs. The final layer is labeled the dense layer and is simply a more conventional feed forward neural network layer, with each neuron used to predict the future value of the input time series a number of time steps in the future.

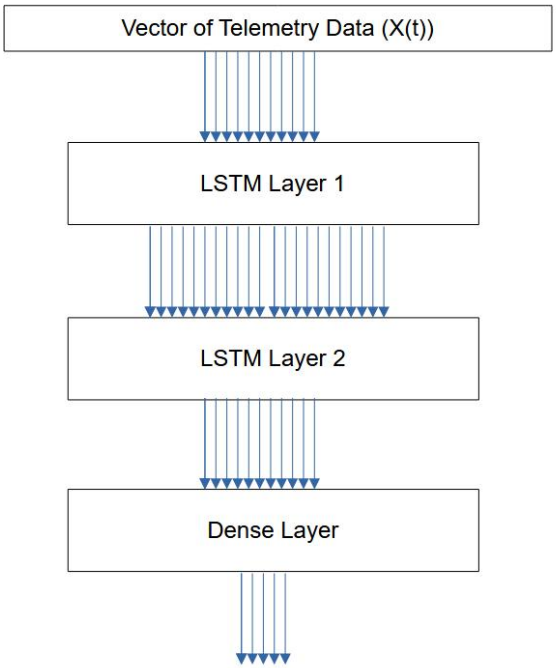

Figure 4 : 3-Layer LSTM RNN Model

The core of an LSTM layer is shown in figure 5. There are 4 intermediate matrix products computed based on current inputs and layer memory. These are then combined in a non-linear way to generate the next layer of outputs. The choice of 80 internal neurons does not match perfectly to the AIE-ML architecture, however since all 4 intermediate matrix products can be computed together as a 106 (layer 1) or 161 (layer 2) element vector multiplied by a 106x320 (or 161x320) matrix, the ability for the AIE-ML to perform 64 Bfloat16 multiply accumulates per clock cycle, maps very efficiently. The reduction of the 320 intermediate outputs down to 80 new values is less efficient; however, since this is a relatively small part of the computation, it does not have a major effect on performance.

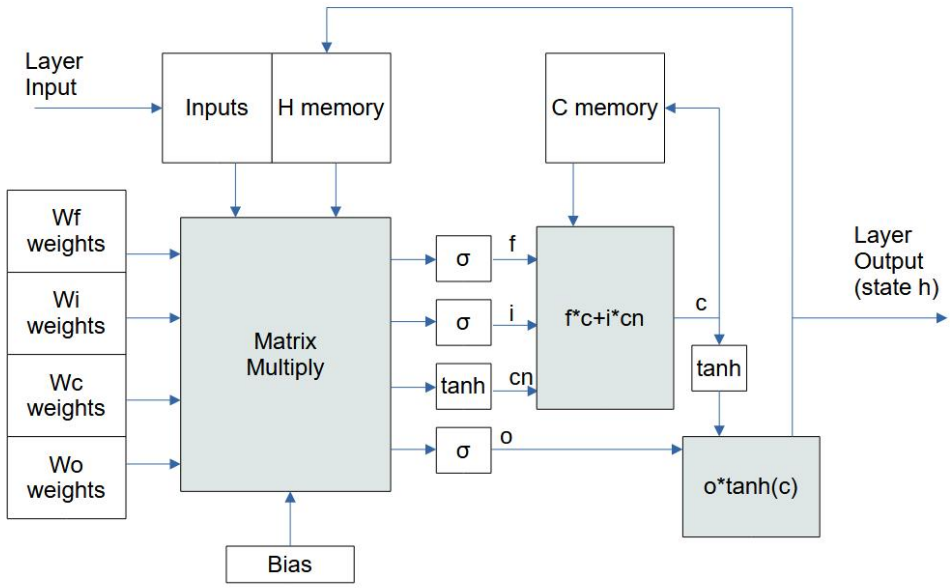

Figure 5 : Operation of an LTSM RNN Neuron Layer

Implementation of designs in the AIE-ML uses C++ libraries which allow a structural description of the processing kernels, input data resources and shared memory as a graph, with the core processing kernels coded as a C++ template class with a run-time method to step a cycle of processing. It is easiest to understand designs by assuming that each kernel maps to a single AIE-ML unit, however, when designs get big, multiple kernel threads can share the processing resources.

The following code shows the template class interface used for the main LSTM processing kernel.

```
template<size_t NUM_NEURONS, size_t NUM_FEATURES, typename ELEMENT_TYPE, typename ELEMENT_ACC_TYPE, size_t VSIZE, size_t VSIZEH, size_t TDMUX>
    void LSTM_KERNEL<NUM_NEURONS, NUM_FEATURES, ELEMENT_TYPE, ELEMENT_ACC_TYPE,VSIZE, VSIZEH, TDMUX>::lstm_func(
  input_async_buffer<ELEMENT_TYPE,extents<NUM_NEURONS*4*(NUM_FEATURES+NUM_NEURONS+1)*TDMUX>> &modell_in,
  input_stream<float> *x_in,
  output_stream<float> *h_out)
```

The following interface for the DENSE processing kernel has a similar template.

```
template<size_t NUM_NEURONS, size_t NUM_FEATURES, typename ELEMENT_TYPE, typename ELEMENT_ACC_TYPE, size_t VSIZE, size_t TDMUX>
    void DENSE_KERNEL<NUM_NEURONS, NUM_FEATURES, ELEMENT_TYPE, ELEMENT_ACC_TYPE,VSIZE,TDMUX>::dense_func(
  input_async_buffer<ELEMENT_TYPE,extents<NUM_NEURONS*(NUM_FEATURES+1)*TDMUX>> &model_in,
  input_stream<float> *x_in,
  output_stream<float> *y_out)
```

To build the network it is necessary to create two LSTM kernels and one DENSE kernel and connect them together in the graph header file.

```
...
 k1 = kernel::create_object<LSTM_KERNEL<NNEURONS1,NSENSORS,bfloat16,accfloat,64,16,NTDM>>();
...
 k2 = kernel::create_object<LSTM_KERNEL<NNEURONS2,NNEURONS1,bfloat16,accfloat,64,16,NTDM>>();
...
 k3 = kernel::create_object<DENSE_KERNEL<NDENSE,NNEURONS2,bfloat16,accfloat,16,NTDM>>();
...
 connect<stream> net2 (in2.out[0], k1.in[1]);
 connect<stream> net3 (k1.out[0], k2.in[1]);
 connect<stream> net5 (k2.out[0], k3.in[1]);
...
```

Figure 6 shows the graph output by the compiler, showing the input data streams, the connections to the shared memory holding model coefficients, and the passing of data between kernels and out of the model. Shared memory is only used for the LSTM layer coefficients, the dense layer coefficients take up a lot less space and can fit in the local buffers of the AIEs.

The shared memory blocks are 512kB in size and include their own programmable DMA engines to stream out the data to the AIE-ML cores. These can be configured to be written once and then repeatedly output the data, which maps perfectly to the use case for LSTM model coefficients, as the local buffer memory in the AIE-ML

cores is not sufficient for these layers.

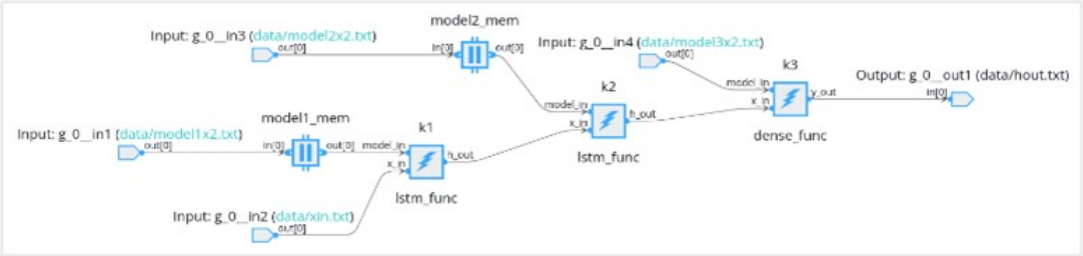

Figure 6 : Compiled LTSM network Graph

Simulation Results

To verify the operation of the C++ code used to implement the LSTM, the data sets from the reference paper [1], handily available in the public domain were used for comparison. Very helpfully the results did not only have the input signals, and model coefficients, but also pre-generated model output results. For a full comparison, the LSTM algorithm was run in double precision floating point in MATLAB, in C++ with bfloat16 arithmetic and these outputs for one step ahead are compared with the actual input signal for that time step and the results from the reference paper.

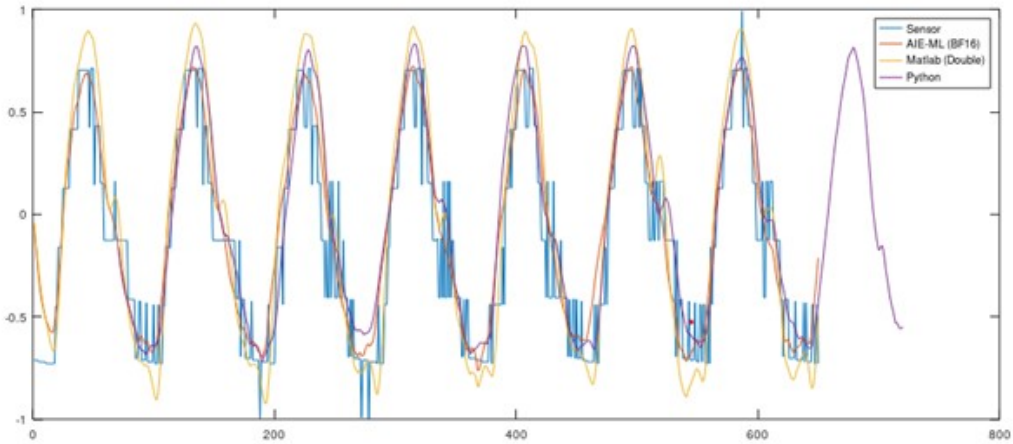

Figure 7 : Comparison of modelled telemetry and different model implementations

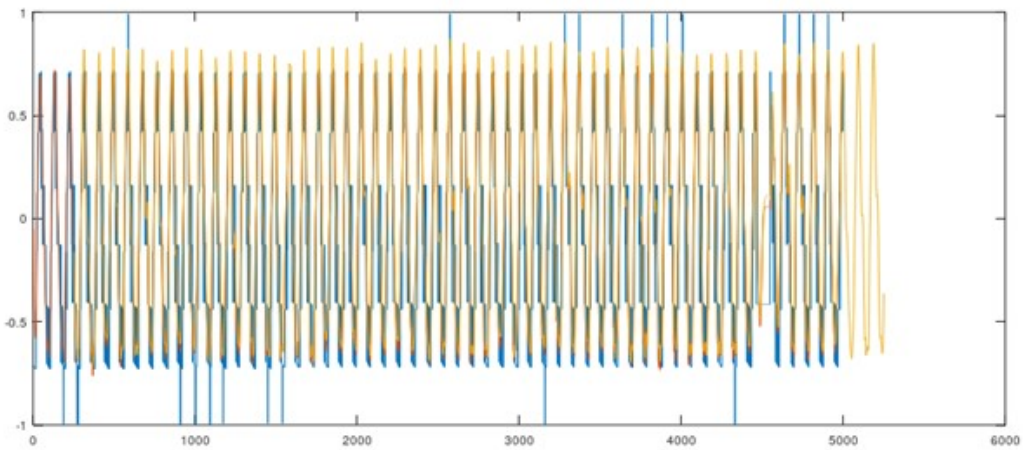

Figure 8 : Complete telemetry data set simulation for one channel showing anomaly, comparing reference result with BFLOAT16 tracking.

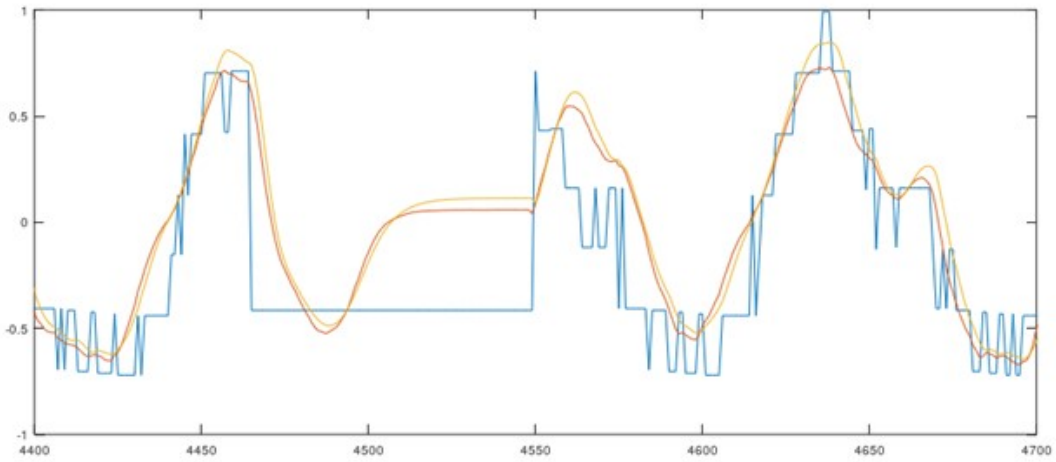

Figure 9 : Magnified view of the anomaly section

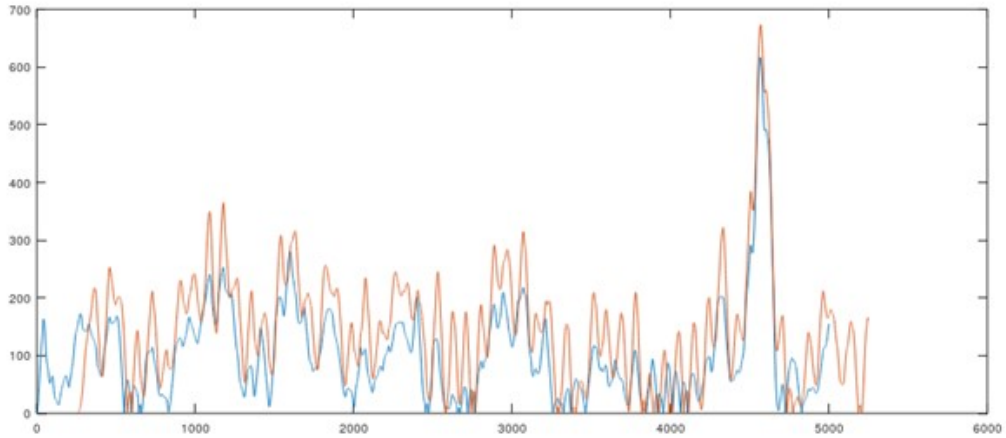

Figure 10 : Smoothed prediction error, showing spike around the anomaly

The results shown in figures 7 through 10 show the match between the model outputs. While not an exact match, all 3 model output signals follow a smooth curve which may match some underlying dynamic of the sensor monitored signal. The sensor monitored signal is obviously quantized at a relatively low resolution, and the models are predicting values at a much higher resolution in between. The exact cause of the very small differences between the model outputs is not fully understood, although the difference between the MATLAB and C++ results is due to the use of the lower resolution bfloat16 data type, as using the same C++ code and a 32-bit floating-point type results in an exact match. The implementation of the non-linear sigmoid functions may also play a part, as in the AIE-ML, some approximation is applied.

Figure 7 shows comparison of all 4 signals for the first 600 data points. Figure 8 captures the same data set (now excluding the MATLAB implemented double precision results) for long enough to show the anomaly that is in the data, at just after 4400 samples. Figure 9 zooms in on this anomaly and shows both the reference data and the AIE-ML simulation similarly failing to track the signal when the anomaly occurs.

Figure 10 plots a low pass filtered version of the absolute difference between the predicted next value and the actual sensor value for both the reference data and the AIE-ML simulation. This highlights that there is a peak in this occurring around the time of the anomaly, which is the basis of any anomaly detection approach. These simulation results did not extend to the predication error processing algorithms to determine when an anomaly has occurred, such as the unsupervised approach taken in the reference paper.

The reference telemetry data and LSTM models can be downloaded from the GitHub site set up by the authors of the reference paper: *https://github.com/khundman/telemanom*

To extract the training and test data it is necessary to use python, which can export the files as text suitable for Vitis simulations, or for import into Matlab (or Octave):

```
import numpy as np
x=np.load('P-1.npy')
np.savetxt('P-1.txt',x)
```

The models are in *.h5* format and therefore can be read directly by Matlab (or Octave). This creates some complex objects in Matlab with multiple fields and therefore conversion functions to extract the relevant Matrix data from these objects for the different model layers are provided. *convert_model.m* and some simlar functions supporting different vector sizes and data types are provided to covert the model data into files suitable for parsing by the Vitis simulator. *sim_model.m* and some similar functions are provided to run a Matlab code simulation of the LSTM model for comparison. The archive of Matlab functions is available alongside the Versal AIE-ML project code in the archive available from Alpha Data and/or AMD.

Hardware Implementation on the VEK280 Platform

GMIOs replace the PLIOs from the simulation for loading the model weights from the external memory to the AIE-MLs. A total of 3 GMIO ports are required, one for each of the kernels (two LSTM kernels and one dense kernel). HLS-based data mover kernels are used to bridge between memory and the AXI4-Stream interface to supply input data from memory and retrieve output data to memory. An mm2s kernel is used to stream input sequences to the AIE-ML arrays, similar to how the solution would be deployed in real-life applications. An s2mm kernel replaces the output PLIO for streaming the predicted output from the AIE-ML arrays to memory.

A configuration file (system.cfg) is used to specify the interconnections between the data movers and the AIE-ML engine as follows, where DataIn1 and DataOut1 are the respective labels of the input and output PLIOs defined in the graph.h file:

```
[connectivity]
nk=mm2s:1:mm2s
nk=s2mm:1:s2mm
sc=mm2s.s:ai_engine_0.DataIn1
sc=ai_engine_0.DataOut1:s2mm.s
```

A host software code (host.cpp) is used to manage the loading of model data and the streaming of input and output data. In this implementation, the model and spacecraft sensor data from the Telemanom GitHub repository are converted to text files and stored on the microSD card. These files are then read into memory and streamed to the AIE-MLs using the XRT control interface.

The following are the main functions implemented in host.cpp:

1. Load the xclbin file.

2. Prepare data - read from the microSD card and convert the floating-point numbers to bfloat16. Since there is no intrinsic support for bfloat16 on the embedded Arm CPU, the float models are converted to bfloat16 by truncating the mantissa of the floating-point number to fit into the 16-bit bfloat16 format.

3. Allocate model and input memory and map in data read from step 2.

4. Transfer the model data using the GMIOs.

5. Start the mm2s and s2mm kernels.

6. Start the graph, setting the number of iterations to match the amount of input data presented to avoid lock-ups and stalls.

7. Wait for the graph to finish execution.

8. Wait for the s2mm kernel to finish streaming data.

9. Save predictions as txt files on teh microSD card.

Results Generated on the VEK280 Platform

The single model code was re-targetted for implementation at the VEK280. A number of minor changes to the implementation have been made to best process the data. At this stage, the Shared Memorry buffers of the AIE-ML have not been exploited, however this creates a slight problem in that the full models for the LSTM layers occupy more memory than a single AIE-ML tile's memory. However splitting the model in two and allocating a second data buffer that maps to a neighbouring AIE-ML tile provides a solution that works for a single model, and allows the model data to be accessed at maximum rate.

The data upload was also re-written for deployment on the hardware. The simulation used the very easy to use trivial approximation of streaming all data into the memory and through the models from PLIO inputs, simulated using text files. For deployment, it is likely that the telemetry data may be available within the PL part of the device and using data movers withing the PL to push the data into the AIE array seems a valid solution. For the Model Data, this is likley to be coming from storage attached to the PS system, and therefore using GMIO to transfer this across from the attached DDR4 banks was used as a more representative use case.

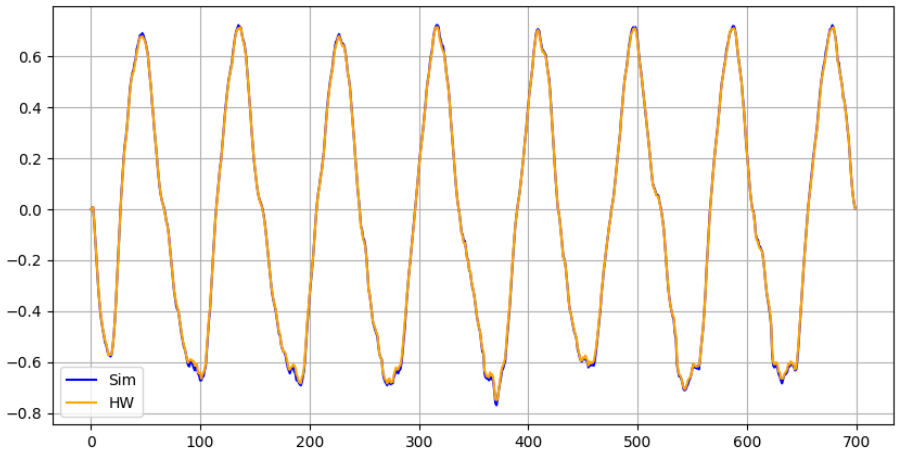

Figure 11 : Comparison of simulated and actual VEK280 generated results

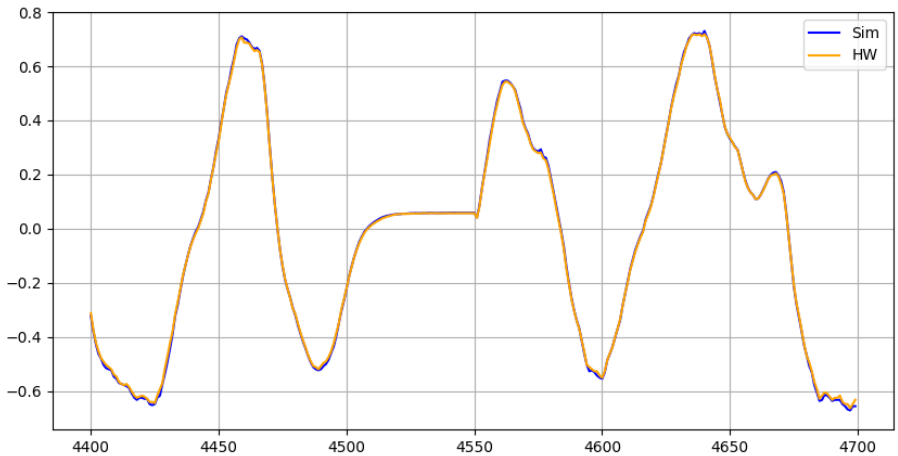

Figure 12 : Magnification of simulated and actual VEK280 generated results

The results of the VEK280 processed real data are shown in comparison with the simulation results in figures 11

and 12. These show that the switch to real hardware even with the changes to the data flow and scheduling do not make any functional difference to the output.

The source code for the simulation and implementation of this solution are available in the Versal AIE-ML projects archive, provided alongside this document from Alpha Data and/or AMD.

Scaling up and out

The above solution only implements a single model looking for anomalies on a single sensor. Most modern spacecraft have very large numbers of sensors and a lot of telemetry to track. The AMD XQRVE2302 Adaptive SoC has 30 AIE-ML cores and can compute far more than the single model shown so far.

Two strategies are investigated to increase the number of models supported. The first approach is to Time-Division Multiplex (TDM) the use of each kernel. Effectively, this means that for each successive time step the kernel runs, it processes a different sensor. The cost of this in resources is relatively low. The LTSM model memory must be increased by the number of TDM threads. Additionally, since it is assumed that each sensor will need an independent model, the Shared Memory buffer in the AIE-ML array storing the model coefficient will also increase in size. For the models used here with 80 neurons, the LTSM model memory using AIE-ML local buffers, does limit the number of TDM threads to 16 if the bfloat16 type is used. More limiting is the Shared Memory capacity. As these blocks are limited to 512kB, if the model for that layer is stored in a single memory, this limits the number of TDM threads to just 5 for the 80 neuron size layers, with up to 80 inputs. This could however be doubled potentially by storing the model data packed as 8 bits. Equally, reducing the model size and re-training could similarly reduce the memory footprint.

The second strategy is the rather more obvious solution of simply scaling out the design and running multiple instances of the network in parallel. With its 34 AIE-ML cores the obvious degree of parallelization to target is 10 instances which should assign 1 kernel to each AIE-ML, although as previously mentioned kernels may share AIE-ML units, as especially the Dense Layer may require far less computation in comparison to the other layers. Again, the Shared Memory capacity also creates a limit and the VE2302 device has 17 buffers. With 2 required per instance, the limit appears to be 8 instances.

An additional restriction when scaling up is the GMIO bandwidth for transferring data from the off-chip memory into the Shared Memory, the number of parallel routes is limited, however as this transfer is only performed once at the start, the GMIO input channel can be shared between multiple Shared Memory buffers using a lightweight AIE-ML kernel performing the data copy.

Considering all these approaches, with the model size referenced, it appears that 40 models should be able to run in parallel on the XQRVE2302 hardware, without further optimization. If the model coefficients can be stored as 8-bit data and converted to bfloat16 dynamically then it is trivial to scale to 80 models running in the device.

Scaling up and scaling out resource utilization results

To test out the feasibility of the deployment of this solution with 40 models, initially, the design is being targeted at the AMD VEK280 development board. This board has the VE2802 device which is from the same device family and features the AIE-ML and Shared Memory Tiles in its AIE-ML array. It is, however, significantly larger in resources than the VE2302 device that will be available in a radiation-tolerant qualified version. The testing will, however, aim to ensure that the resources required do not exceed the capacity of the smaller device.

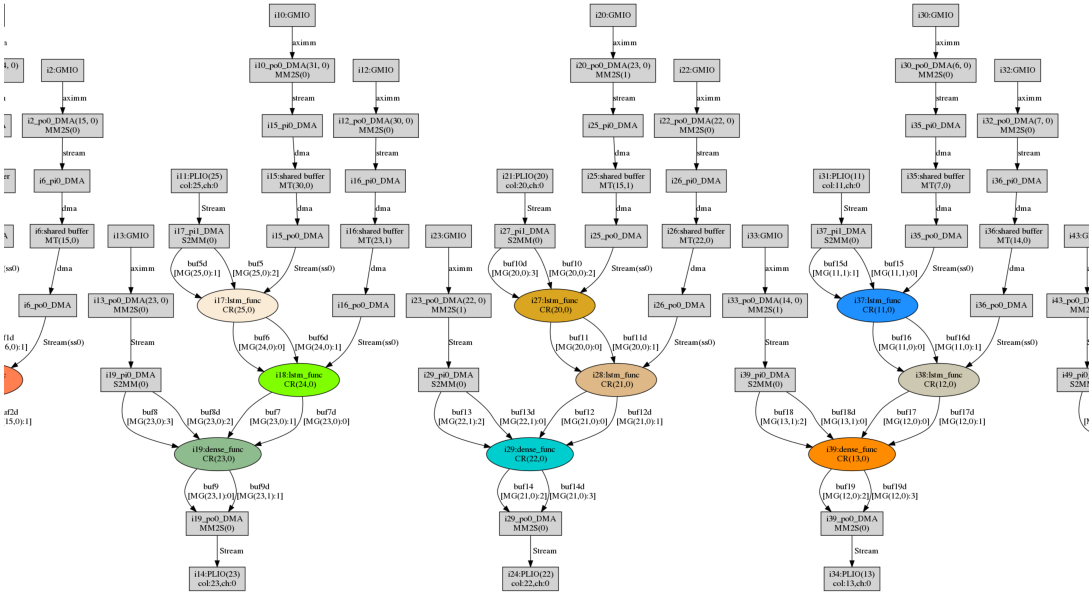

Figure 13 : Model implementation showing AIE-ML and Memory and IO use of 3 of the 8 parallel implemented models.

A small extract of the model graph from the design built is shown in Figure 13. This graph shows the mapping of the processing kernels along with all the memory buffers and inputs and outputs of the AEI-ML array system. It would be extremely difficult to show the full network, and so a snapshot capturing just 3 of the 8 parallel processing streams (each running 5 concurrent models as TDM streams) is shown to give a general overall view of the structure.

Resource utilization results are shown in table 1, following this. This shows the potential capacity of both the VE2802 and VE2302 devices and compares these with the actual implementation. In terms of AIE-ML processing tiles, the load is relatively light with only 24 out of 34 tiles used in the VE2302. This could potentially allow a different application to run on those resources in parallel with the Anomaly detection.

Resource	VE2802	VE2302	40 LSTM Model Implementation
AIE-ML Tiles	304	34	24
AIE-ML Memory	152Mb	17Mb	440kb
Shared Memory	Buffers (512kB)	76	17
Shared Memory	304Mb	68Mb	54.7Mb
GMIO Inputs	4x12	4x6	24
PLIO Inputs	6x28	6x12	8
PLIO Outputs	8x28	8x12	8

Table 1 : Resource Utilization

The Shared Memory buffer size does appear to be the critical limitation here, with 16 out of 17 buffers used (54.7Mb out of 68Mb) leaving little room for additional models to be stored.

The other resource that is close to being fully utilized is the GMIO input connections. The initialization of the Shared memory buffers holding the LSTM components occupies all the connections to the external DDR memory via the NoC in this implementation. However, as previously mentioned, as this operation is only performed once at initialization it should be possible to share these GMIO inputs, if necessary, using an additional AIE-ML kernel or other method, as this is not the performance critical part of the design.

Principal performance will be tied to the memory bandwidth between the Shared Memory and AIE-ML tiles. In the current design this is limited to a single stream from each shared memory, so 16x4GB/s in total; however, it should be possible in the next stages of development to increase this.

The current development of this approach has however been stalled as although it looks like the hardware can support the use of the Shared Memory tiles to send model data to multiple models, the current tool chain, does not have a user friendly high level API solution for achieving this. It looks like improvements in the programming libraries and API will address this in future releases of the Vitis toolflow.

Therefore this development has been paused, but this allows for some exploration and examples covering other means to increase the number of models and therefore telemetary channels that can be monitored within the VE2302 devices.

Pruning

The choice of model size used in the reference paper was somewhat arbitrary, and most likely derived from previous work on LSTM models in different domians. The authors do indicate that no investigation of optimal model size was performed but that they felt that the models used were larger than they needed to be. This is generally not an issue in ground based systems, or ground based offline processing, where near infinite compute resource can be available from data centres, running massive batched computations. However for deployable edge systems, requiring near real time responses, and especially in space, processing and memory resources can be scarce and selecting smaller models should certaing be more seriously investigated.

In this work, a conscious decision to do no re-training has been taken, and so the performance of any pruning will be limited. However the objective here is to confirm that the required numerical processing of the sensor data model is possible in the embededed platform, rather than ensure minimum numerical difference. The output layer is actually significantly redundant at the inference stage as for the anomaly detection only the one neuron predicting the immediate next sample is used. The other neurons will have been useful in helping the model capture more of the dynamic behaviour in training, but once deployed, they do not provide useful computation. This does allow a very agressive pruning from 10 neurons down to 1.

The 80 neuron size of the LSTM layers may have fitted well with a particular GPU architecture used in early research on LSTMs some time in the past, but otherwise is quite arbitrary. For the AIE-ML architecture, it is clear that computationally a power of 2 sized network of 16,32 or 64 neurons will work out as the most efficient, due to the vector sizes available for arithmetic. Therefore for proof of concept a choice of 32 neurons was selected, to give a significant reduction in memory use, while still remaining a significant parallel processing workload.

The trivial pruning algorithm reduces the middle layer to the 32 neurons that had the most influence on the output layer single neuron (largest weights). The second stage of pruning appreared less accurate as the total influences of the input layer on the middle layer of each neuron were less variable, but still the sum square of the input weights to the 32 remaining middle layer neurons was used to select the 32 most influential input layer neurons.

After applying these prunings to the network, and scaling the output to compenstae for the loss of energy in the weights (scaling up by 80^2/32^2), the model still appears to track the telemetry data in a similar way to the full model. So while full re-training would be recommended for a real deployment, for this exercise in evaluating the processing performance, this pruned model appears to be good enough in function.

A Single Model Pruned Design

To demonstrate the operation of the pruned model, a single model project was generated. This is simplified compared with even the initial single model project in that the memory requirements for the models are now lowered and the LSTM layers can fit in a single AIE-ML memory. The dense layer has not been further optimised, and uses a relativley inefficient implementation, however the contribution of this to the overall workload is minimal, and possibly could even be moved to an ARM CPU. This model is also available as a separate project in the projects archive.

One additional topic that this project covers is performance optimization. For ease of readability, and to match a streamed flow of feature data, the main code implemented the general Tensor Product as a sequence of 64 word

wide parallel scalar/vector multiply accumulates, with the input scalar constant. As in the full code, the data width is 320 words wide, and even in the pruned code, the data width is 128 words wide, this requires each of the 5 (or 2) 64 word accumulators to be stored and reloaded after each vector multiply accummulate. The input scalar values occupy a very small amount of memory, and so a more efficient solution is possible. In the optimized code, the processing is reordered to perform the scalar/vector multiply accumulate only on 64 words width of the data at a time. This allows the accumulator to keep its contents in the AIE-ML accumulator register and thus avoid these memory accesses between multiply-accumulate operations. There is a small penalty in performance in that the input features and recurrent memory need to be re-read multiple times, but in total these extra memory operations are significantly outweighed by the other improvements, and the overall performance is a lot faster.

Scaling out the pruned model designs

The main benefit of the pruning is the much smaller memory footprint. This allows a far higher number of models to be mapped into both individual AIE-ML tiles and also across the array, without having to use the Shared Memory. Apart from the current tool limitation, that is due to be resolved, there is a performance benefit to just using the AIE-ML memory directly, as it has a far higher bandwidth than the stream from the Shared Memory. For cases where this matters, it is useful to look at a scaled out version of the pruned design. This is also provided as a project in the archive.

The memory footprint for the largest layer in the pruned models is 12240 bytes, allowing 5 TDM threads to be interleaved in each core. Assuming 2 AIE-ML tiles are needed per model, (this assumes that the dense layer can either share a tile, or can be implemented on an ARM CPU, as it is computationally very lightweight), then we might expect to be able to fit in 16 parallel streams into a VE2302 device.

Resource	VE2802	VE2302	80 LSTM Model Implementation
AIE-ML Tiles	304	34	32
AIE-ML Memory	152Mb	17Mb	16Mb
Shared Memory	Buffers (512kB)	76	17
Shared Memory	304Mb	68Mb	0
GMIO Inputs	4x12	4x6	16
PLIO Inputs	6x28	6x12	16
PLIO Outputs	8x28	8x12	16

Table 2 : Resource Utilization

Conclusions

This work has investigated the implementation of a potential AI-based anomaly detection scheme for spacecraft telemetry on hardware that is suitable for flight. Adaptive SoC devices such as the Versal Edge XQRVE2302 provide the mix of resources to implement such a scheme. Flexible I/O and programmable logic can be utilized to directly capture the sensor and command telemetry data. The powerful AIE-ML processing cores within the device can provide a very high performance, low-power solution for implementing the considerable computational load required for LSTM recurrent neural network implementation. The multi-core ARM application and real-time processors in the device provide the flexibility to process the LSTM network outputs and make real-time decisions and classification based on this. The parallel nature of the AIE-ML array allows multiple instantiations of the algorithm to run multi-threaded and in parallel across the array. The primary limitation of this is on-chip memory, limiting the solution based on the reference paper data set to monitoring around 40 sensors. The model size in the reference paper is, however, somewhat arbitrary, and the memory footprint varies with the square of the number of neurons. Thus, if the underlying dynamics of the sensor allowed and models can be reliably retrained to a 64-neuron size (or even 32 neuron), then a potential increase in the number of sensors monitored

by a factor of 2 (or 8) could be achieved.

References

This section provides supplimental information for this document.

[1] K. Hundman, V. Constantinou, C. Laporte, I. Colwell and T. Soderstrom *Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding,* KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2018, London, Aug 19-23, 2018

[2] L. Gunn, P. Smet, E. Arbon and M. D. McDonnel *Anomaly Detection in Satellite Communications Systems using LSTM Networks,* Military Communications and Information Systems Conference (MilCIS), Canberra, ACT, Australia, 2018

[3] M. Cheng, Q. Xu, J. L.V., W. Liu, Q. Li and J. Wang *MS-LSTM: A multi-scale LSTM model for BGP anomaly detection,* IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 2016

[4] S. Ahmad, K. Styp-Rekowski, S. Nedelkoski and O. Kao *Autoencoder-based condition monitoring and anomaly detection method for rotating machines,* IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020

[5] A. McCormick and A. Nandi *Neural network autoregressive modeling of vibrations for condition monitoring of rotating shafts,* Proceedings of International Conference on Neural Networks (ICNN'97) Houston, TX, USA, 1997

[6] S. Chauhan and L. Vig *Anomaly detection in ECG time signals via deep long short-term memory networks,* IEEE International Conference on Data Science and Advanced Analytics (DSAA), Paris, France, 2015

Revision History

Date	Revision	Nature of Change
19/08/24	1.0	First draft

Address: Suite L4A, 160 Dundee Street, Edinburgh, EH11 1DQ, UK
Telephone: +44 131 558 2600
Fax: +44 131 558 2700
email: sales@alpha-data.com
website: http://www.alpha-data.com

Address: 10822 West Toller Drive, Suite 250 Littleton, CO 80127
Telephone: (303) 954 8768
Fax: (866) 820 9956 - toll free
email: sales@alpha-data.com
website: http://www.alpha-data.com