DOI 10.5162/ETTC2024/A2.2

Towards Edge AI in Flight Test Data Acquisition: A Proof of Concept for Anomaly Detection in Aircraft Electrical Networks

Rémy Pelluault[1],Ghislain Guerrero[1], Sangaran Sivakumaran[1]
[1] Safran Data Systems, 5 Avenue des Andes, 91940 Les Ulis, France

remy.pelluault@safrangroup.com
ghislain.guerrero@safrangroup.com
sangaran.sivakumaran@safrangroup.com

Abstract:

In recent years, the proliferation of artificial intelligence (AI) technologies and their associated use cases has been observed across various industrial and consumer domains. Within the field of instrumentation, the longstanding trend of acquiring and digitizing physical measurements captured by sensors has been further bolstered by the rise of edge computing, enabling early data-to-information transformation. Consequently, the emergence of edge AI becomes a natural extension of this evolution.

However, the deployment of edge AI in the realm of embedded instrumentation introduces new possibilities while also challenging fundamental aspects of the field, including metrology, determinism, miniaturization, and power saving. In light of these considerations, Safran Data Systems has developed a Proof of Concept aimed at integrating an edge AI capability into a flight test data acquisition unit. The primary objective is to leverage this technology for anomaly detection in time series data pertaining to aircraft electrical networks monitoring.

This paper presents the technological approach employed, highlighting its unique features and advantages. Furthermore, the outcomes and results obtained through experimentation are shared, shedding light on the effectiveness and potential of the implemented edge AI solution. Additionally, the paper explores future use cases and potential enhancements, paving the way for continued research and development in this area.

Key words: Machine learning, anomaly detection, time series analysis, on board processing

Introduction

More Electrical Aircraft (MAE) represents a significant advance in the aeronautical sector, aimed at reducing CO2 emissions and promoting decarbonization. This innovation is transforming aircraft design by replacing combustion engines and traditional mechanical systems with electrical technologies.

The transition to MAE poses several major challenges for aircraft manufacturers, and by extension, for Safran Data Systems. A disruptive innovation in the aeronautical industry requires special care during certification to ensure compliance with organizations such as the EASA or FAA, which translates into a massive multiplication of measurement points and data, to ensure the aircraft's nominal operation. However, current on-board-to-ground data transmission technologies are often limited in data rate, exceeding just a few tens of Mbits, which may prove insufficient to efficiently manage the increasing volumes of data generated as required by aircraft manufacturers.

And this trend doesn't just apply to future electric aircraft. Generally speaking, in flight testing, the doctrine is to record as much as possible, to oversize the acquisition, even if it means exploiting only a tiny fraction of what we

collect. The price of an unsuccessful test flight is extremely dissuasive, which leads to two statements: “Prevention is always better than cure”. In addition, when a problem occurs: “track down the root cause of the disturbing event”. This leads to an oversizing of the aircraft measurement plan: more sensors, more storage capacity, increased throughput, etc... If normality is expected, anomaly is unacceptable.

So how can we manage these constantly growing quantities of data? From a technical point of view, we can always improve throughput, storage, computing power, telemetry... However, in terms of human resources? How can we effectively leverage this huge amount of data, which will often just be recorded without ever being processed due to a lack of human resources? Data is more and more the heart of our industries and we still need to be able to extract information from data in an efficient and relevant way.

One innovative solution is the use of Artificial Intelligence (AI). Such algorithms could be used cleverly to process part, or even all, of the data, which would otherwise probably just be recorded without ever being considered. The use of AI in this context comes up against a number of constraints. Traditionally, the flight test domain requires deterministic approaches to ensure the reliability and predictability driven by certification and safety. The somewhat 'black-box' aspect of AI, where decision-making processes are not entirely transparent, poses a problem in an environment where every component must be fully certified and justifiable. Nevertheless, there remains significant potential for AI to assist test operators. In our example, we would like to propose an algorithm for real-time anomaly detection, helping engineers to identify potential problems more quickly and accurately. Of course, the role of such an AI must remain that of an assistant, not a decision-maker. AI, and anomaly detection, is not entirely new in the world of instrumentation [1], but there is currently very little embedded and real-time application, which could remains relevant for specific use-cases

Anomaly detection

The projet implies the creation of an algorithm integrated into an airborne platform. This algorithm will be specifically designed to supervise an aircraft's electrical networks in order to detect any anomalies.

Today, it is of course possible to monitor these electrical networks using "conventional" methods such as Total Harmonic Distortion (FFT), but these on-board techniques have certain limitations, particularly when it comes to detecting complex anomalies.

Indeed, the application of these methods requires manual definition of normality thresholds. For example, an operator might judge an electrical AC signal to be nominal if it is between 110 and 120 V with a THD lower than 5%, no carrier current present, etc...

However, this approach only detects predefined events, potentially allowing signals to slip through which might subjectively be judged as abnormal, but which do not correspond to the established criteria. This restriction represents a significant drawback, as it limits the system's ability to identify only a finite set of anomalies, thus increasing the risk of non-detection of new anomalies or variants of previously uncharacterized anomalies. The idea would be that, using AI, we could train it on signal considered valid and warn us if the monitored signal diverges from this normality.

AI creation

The project has the significant advantage of generating easily large-scale artificial datasets.

It is possible to generate sinusoidal signals considered to be valid (for example, at 400 Hz and 115V) as well as signals with various anomalies such as overvoltages, spikes, noise, carrier current or modulations. Subsequently, we will be able to validate the model with real data, extracted from an arbitrary generator, in order to test its robustness and accuracy under a variety of conditions.

Initially, our architecture was based on an artificial intelligence system with supervised learning. In this approach, the model was fed with both anomaly-free data and data containing one or more categorized anomalies.

The objective of the algorithm was to classify the inputs as "nominal" or "abnormal". However, this approach was quickly discarded. The main drawback of this method lies in its intrinsic limitation: it could only detect a finite number of predefined anomaly categories. As a result, the model could only identify anomaly types for which it had been explicitly trained for, making it inefficient to detect new anomalies or previously uncharacterized variations. In the end, our algorithm was not much of an evolution compared with more "classic" methods, since here again, the algorithm was not able to adapt itself to new situations.

The real revolution is unsupervised learning: a program that learns by observing "normal" or typical data. Later, when used, this program can alert us if something unusual happens,

something that does not look like what it initially learned. To develop such a system, we will explore different methods and architectures.

Let us start with the basics: the neuron. The simplest type of neuron is called a **perceptron**. It is often the first model used for simple tasks in machine learning. It works as follows: it takes an input, multiplies it by a weight, adds a bias (a fixed adjustment), then applies what is known as an activation function to obtain a final result. Despite its simplicity, the perceptron is widely used in machine learning

However, the perceptron is very limited especially for time series analysis. In a time series, each value shall be considered in its context, because information is not just a series of independent, isolated numbers. A value at a given moment (x(t)) has a given meaning because it is part of a sequence of values. The standard perceptron, which process isolated inputs as single values, is therefore not at all suited to time series and lead us to consider an improved version of the perceptron: the **RNN** (Recurrent Neural Network). This type of network is designed to take into account the time or order of the data.

To achieve this, in addition to the usual input, weights, bias and activation function, the RNN uses a kind of feedback loop. This means that information is not just processed and forgotten; instead, it circulates back through the network. For example, information from two previous time steps x(t-2) influences information from previous time step x(t-1), which in turn affects current information x(t).

It is similar to an IIR (Infinite Impulse Response) digital filtering system, where the current output y(t) depends not only on the current input x(t) but also on all previous inputs. In this way, RNNs are able to "remember" past data, which is crucial for processing time series. However, RNNs have a limitation: although they are affected by recent inputs, their ability to remember older information is limited. This means they tend to have a "short-term memory", which can be a problem when learning or recalling details of older data in a long sequence.

To overcome the short-term memory limitations of RNNs, **LSTM** (Long Short-Term Memory) networks were developed. These networks are an advanced variant of RNNs, designed specifically to avoid the problem of memory loss of old information.

LSTMs operate with a more complex structure, enabling them to retain information for long periods. They incorporate so-called "gates" - mechanisms that decide which information to keep and which to discard.

Thanks to these gates, an LSTM can efficiently recall important events in a sequence, even if they occurred a long time ago. This makes it particularly useful for tasks such as time series prediction. We choose this neural network architecture for our project.

A quick note about transformers: there are very promising for anomaly detection and time series analysis thanks to their attention mechanism. However, they require a large number of parameters to perform correctly, and are still relatively recent, which may be incompatible with embedded architecture. Their operation is completely different from perceptrons, RNNs and LSTMs. However, these architectures are very interesting when computing power is not a limiting factor, such as for ground stations and post-flight processing.

In addition to LSTMs, we will be using an auto-encoder system.

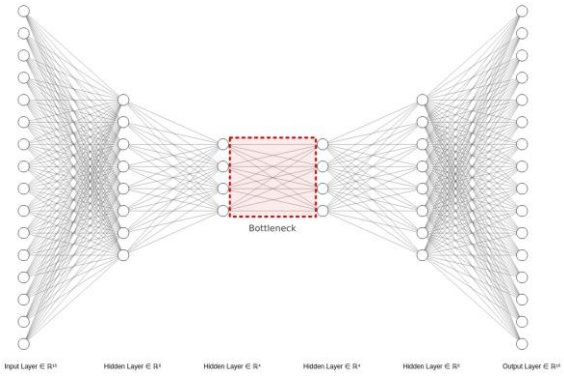
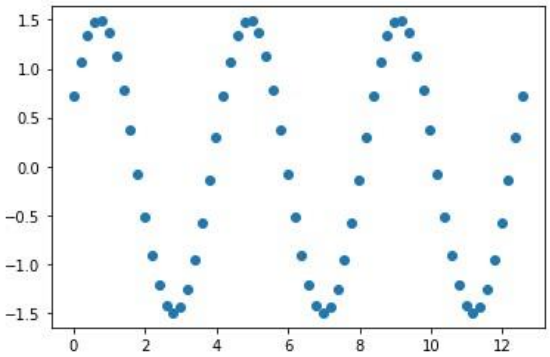


Fig1: An example of autoencoder architecture

The aim of an auto-encoder is to reproduce as output what it receives as input. Assume we feed it a sine wave curve, with the intention that the output replicates this same sine wave. This may seem uninteresting at first glance, but the interest of the auto-encoder lies in its double-funnel structure. In the center, the feature space (called latent space) is smaller than at the ends. This means that, to successfully reproduce the input sine, our model must find a simplified way of representing it, with very few parameters.

We can draw an analogy with human reasoning: it is a bit like a person having to draw a sine wave.

If we give an individual the following vector:



And ask him to remember it, he could learn all 64 values that make it up. But more likely he will recognize a sine wave, and so, using a simple formula like *A*sin(x*ω+φ)*, he will memorize just 3 values: amplitude, frequency and phase shift. Therefore, instead of storing 64 separate values for our sine wave, we would only need 3 parameters, which is a encoded (or compressed) version of the original signal.

The same applies to our algorithm. The model is trained ingesting signals that the user considers nominal. Once the training is completed, the parameters are set, and the auto-encoder can reproduce the learned signal. An auto-encoder trained on a sine wave is fed by such signal, it will output a sine wave with similar characteristics (amplitude, phase shift and frequency). But if it is fed by a different signal, such as a square wave, the output will be very different from the input, as it is not possible to represent a square waveform adjusting A, ω and φ of the function *A*sin(x*ω+φ)*. An output radically different from the input indicates an anomaly.

The denser and larger a neural network is, with many layers and connections, the more it can learn to recognize complex functions. The monitoring of simple sine waves does not really need very large neural networks. However, this type of system is also capable of processing signals far more complex than simple sine waves, such as detecting events in signals coming from vibro-acoustic, dynamic strain or pressure measurements.

Learning these tasks is surprisingly simple: the system listen to a signal considered nominal to learn and recognize it. Once it has done this, it will be able to detect anomalies on its own. In short, by listening to and memorizing what is normal, the neural network learns to identify what is not, signaling any deviation as an anomaly. This makes these systems extremely efficient for monitoring and detecting problems without continuous human supervision.

Testing the PC model

We use a simple method to decide whether a signal is normal or abnormal: we compare the model prediction with the actual input. If our model and the signal are perfect, the result of this comparison (i.e. the difference between prediction and input) should be a zero vector. If the values of the output vector deviate from zero, this indicates that the incoming signal could be abnormal.

To decide whether a signal is normal or abnormal, we have several options:

1. **Sum of absolute values**: We add up the absolute values of the points on the output vector. If the total exceeds a certain threshold, which we call ε. If the total exceeds a certain threshold (ε), then the signal is considered abnormal.

2. **Least squares method**: This method gives greater weight to values that deviate significantly from zero. If the sum of the squares of the deviations exceeds the threshold ε, the signal is considered abnormal.

3. **Examining each value**: We examine each point of the output vector individually. If any of these values exceeds the ε threshold, we classify the signal as abnormal.

In our case, we prefer to use the third method as it gave better results for our application. But the important thing to remember is that whatever method we choose, it is crucial to define a value for ε. This ε value is essential as it helps maximizing the accuracy of our model by allowing us to effectively distinguish between normal and abnormal signals. We will soon delve deeper into how to choose this value optimally.

Our first computer tests are very encouraging. We use synthetic data to train our model, then test it on real data taken by a data acquisition unit fed by an arbitrary function generator. The results show that our model is effective.

We evaluate the quality of a model by creating a data set consisting of real signals recorded by a data acquisition unit. This dataset is divided in two: one half contains nominal data and the other, "abnormal" data. Anomalies can be caused by a variety of factors, such as the addition of even and odd harmonics, frequency

or amplitude shifts, or the presence of carrier currents.

When we compare different models to determine the most effective, it is essential to harmonize our approach to the ε threshold. If ε is too small, even slight noise could cause a signal to be classified as "abnormal", resulting in many false positives (the model reports an anomaly when there is none) and most likely no false negatives. Conversely, a ε too large could cause our model to consider all signals as healthy, increasing false negatives and eliminating false positives.

So, how do we choose the right ε for each of the models we want to test? We decided to set ε at a level where there are no false positives. This approach standardizes the basis of comparison for all the models we will evaluate in the future, enabling a fair and consistent evaluation between them.

It is important to note that according to the concept of operation (CONOPS), we would probably prefer to have zero false negatives, even if this implies accepting some false positives. False positives can be re-evaluated by a human operator to confirm whether they really represent an anomaly, while a false negative could remain permanently undetected. The choice of the ε threshold we use to compare our models is based on ease of comparison between them. However, nothing prevents us from adjusting ε later on to modify the sensitivity of our model according to the specific needs of final user.

Our first auto-encoder model was of the following size and architecture:

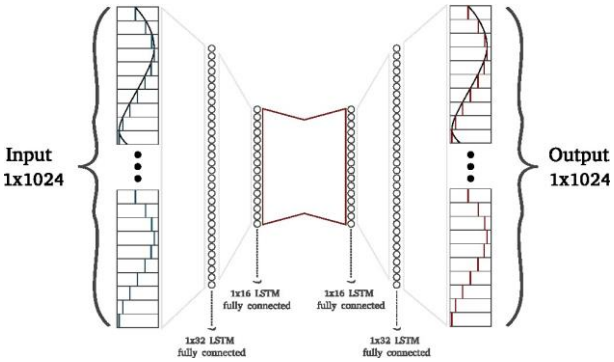


Fig2: First autoencoder+LSTM model

Relatively frugal with its 15,000 parameters, our model had the following already promising performances

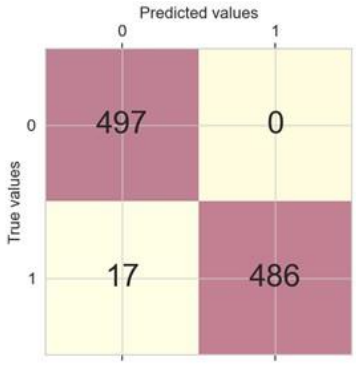


Fig3: Confusion matrix of first model

The next step is to port this architecture to the host target.

Embedding software

We are now going to integrate our current model into the XMA systems, one of the most advanced modular DAU. We will leverage the OBP, or On-Board Processing module. OBP is designed to run C-language programs. It is specifically used to process data, whether periodically sampled or aperiodically produced, in an XMA system. This module also offers local serial ports and discrete I/Os, as well as a Gigabit Ethernet interface.

The OBP is particularly customizable, making it ideal for quick prototyping needs. It is equipped with a Zynq-7020, a System On chip (SOC) that combines a FPGA and a processing unit running an optimized Linux to be more responsive (Prempt-rt). Users can develop their own applications [2] thanks to an SDK that provides the necessary tools and guidance.

To begin with, we are going to try and implement our algorithm in the software part of the SOC. The OBP, being an optimized embedded system, does not directly support complex frameworks like Keras. Therefore, we need to compile the code in C or C++ to run our algorithm on the target. To convert models created in Keras to C code, we used a tool called Keras2C.

We have successfully integrated our model into our OBP module and can now read data from a sensor acquired by a generic analogue module, like the ANA module. This system captures signals at a frequency of 32 768 Hz. The model works well, and can correctly identify anomalies in the signal when artifacts are artificially added with a generator. The performance obtained in terms of accuracy is comparable to the PC version algorithm. However, this first non-optimized implementation does not allow real time signal processing.

Our model processes a 1024-point window as input. To keep pace in real time, this means processing 32,768 points per second, or 32 windows per second, giving an ideal processing time (known as "inference time") of 31 milliseconds. At this stage, without optimization, our model requires 5 seconds to infer a single window

To improve this, we are considering two main optimizations:

1. Quantization: the original model weights are stored in 32-bit format (FP32). We could convert them to smaller formats, such as FP16 or especially INT8. Calculating with integers rather than floating-point numbers could dramatically reduce computation time and potentially double inference speed. It would also reduce the memory footprint; although this is not the main concern (the non-optimized model size is just about 256 KB).

2. Pruning: This technique consists of removing non-significant neural connections to lighten the model. Fewer connections means fewer calculations and therefore faster inference, which is crucial for real-time applications and resource-constrained devices. This approach can maintain high accuracy while improving speed, with the potential to double inference speed, albeit at the cost of some effort and perhaps a slight drop in quality.

Even with the optimizations envisaged, achieving an inference time of 31 milliseconds remains a considerable challenge. We will probably have to explore more radical options to get there.

A long-term solution could be to consider a different type of hardware. Traditional processors (CPUs) are not particularly efficient for matrix computation, which are at the heart of machine learning algorithms. The GPU (Graphics Processing Unit), known for its ability to handle such computations efficiently, offers a better balance between performance enhancement and development simplicity. However, neither OBP nor any other XMA module currently incorporates a GPU as of today. The GPUs have a large footprint, consume and dissipate a lot, making them difficult to embed in compact DAU modules like the XMA. However, other flight test instrumentation systems, such as the MDR (Modular Data Recorder), have GPU-capable platforms. Another option would be to use an FPGA (Field-Programmable Gate Array), designed for parallel computing and promising excellent performance in terms of inference time and energy efficiency. However, development and deployment on FPGAs are more complex. It is an attractive solution, but for this proof-of-concept, we decided to explore the full potential of the CPU implementation. We will consider this solution for future developments.

Coming back to our inference time problem, another option would be to modify the auto-encoder model itself, reducing the number of parameters. By progressively reducing the size of our auto-encoder, we have achieved promising results and similar accuracy, with an even more resource-efficient model.

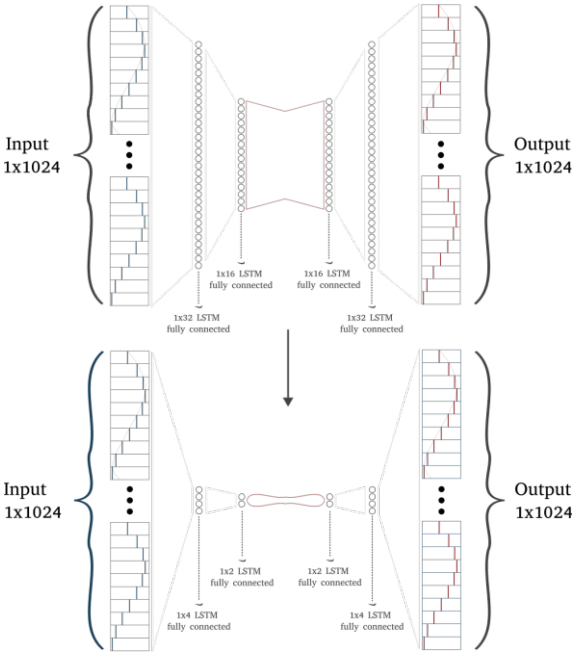


Fig4: shrinking down the precedent model

Just as deep, but 10 times narrower, this model has only 300 parameters, 50 times less than before, with limited impact on accuracy.

Parameters	Accuracy	TPR
15 000	98,8	97,7
300	96,9	93,9

The inference time for this new model on our target system is 80 times faster, or about 60

milliseconds. We are not yet fast enough for real time processing, but we are getting close. In fact, with a little more effort, it would be quite feasible to analyze a signal in real time using our on-board CPU. The performance we have achieved is considered well enough for a demonstrator. For the future, it does not seem wise to continue fine-tuning our current model for this CPU configuration. It would be more appropriate to develop a solution based on specific hardware acceleration.

What's next?

We are now concentrating on hardware acceleration, in particular using an FPGA. The next logical step would be to develop a dedicated hardware acceleration to perform the necessary computation directly on the FPGA. The expected performance of the FPGA is very promising, and should greatly surpass the performance currently achieved with the CPU.

Here is a comparison between the current performance of our implementation and the performance we hope to achieve in the near future with an FPGA, maintaining the same accuracy:

Type	Inference (ms)	Architecture	Parameters
CPU	60	16x4x4x16	0.3 k
FPGA	1	4x2x2x4	15 k

Using the FPGA of the same platform as before, but with a little further development, we anticipate offering the same real-time accuracy, process several signals in parallel, and use larger models.

Model size is a crucial aspect for the future. We have already shown that a simple network can deliver comparable performance. So why consider a larger network? In our current case, which involves monitoring AC electrical networks, the small size of our model is not a problem. However, for the future, it would be interesting to detect anomalies or event of interest in much more complex time series, such as those arising from vibro-acoustic phenomena. For these cases, a larger model, capable of modelling complex phenomena almost perfectly, would be required. While a 300-parameter model may be enough for predictable periodic signals such as sine waves, initial results shows that a larger model will be able to monitor much more sophisticated signals.

Another option would be to use a GPU, which offers many advantages. GPUs, especially those from NVIDIA with CUDA, make it much easier to deploy neural networks, offering excellent performance. It is almost certain that if we deploy our model on a GPU-equipped platform, we will achieve real-time performance without difficulty. While it is entirely possible to integrate neural networks on FPGAs, GPUs, with their ability to perform matrix calculations and store large amounts of information in their VRAM, are often preferred to FPGAs

The MDR (Modular Data Recorder), designed by Safran Data Systems, is a modular flight test recorder with internal media for storing data from the aircraft.

The MDR is larger and more powerful than the XMA, and can be used for computationally intensive applications, such as video encoding, which may require the use of GPUs. Like the XMA-OBP, the MDR provides a user processing module and embeds a more complex platform, including a CUDA-compatible NVIDIA GPU. It would therefore be entirely feasible to integrate our temporal analysis algorithms. This has already been done for object detection algorithms in the past [3]. In addition to its function as a multimedia recorder and telemetry gateway, the MDR is also designed to capture all the data streams generated by the flight test installation of a test vehicle, making it a strategic choice for the implementation of anomaly detection algorithm, allowing the user to incorporate edge-computing activities into his measurement plan.

On the other hand, we could also consider positioning our algorithms even closer to the source of the measurements than our current XMA.

Given that models as frugal as 300 parameters offer satisfactory performance, we might consider deploying these nano-models, such as the one for anomaly detection in power grids, in an IoT-like context. In particular, this could be achieved using µMA [4], a miniaturized and distributed data acquisition unit, enabling physical deployment as close as possible to the measurements.

Throughout this document, we have discussed a machine learning algorithm for power network analysis. However, the main objective is to demonstrate the ability of our equipment to integrate machine learning algorithms. One of the target would be to offer a software package (SDK, framework) enabling the instrumentation engineers to integrate their own algorithms as easily as possible. Currently, it is possible to integrate algorithms on the CPU of an OBP, as we have done. However, for custom ML algorithms, the hardware acceleration offered by the GPU (in the MDR) or FPGA (in the XMA) would be preferred.

Conclusion

In the near future, we anticipate a considerable increase in the amount of data to be transferred, stored and analyzed during flight testing campaigns. To maximize the support that Safran Data Systems can offer to aircraft manufacturers, it will become necessary to go beyond the simple acquisition, recording and uploading of signals. Each link in the chain should become an active agent, leveraging AI to facilitate and accelerate test set-up and data processing.

We have already explored anomaly detection, but time series analysis offers far more possibilities.

With the right algorithms, we could, for example, consider the automatic detection of sensors when configuring data acquisition units, or the automatic labeling of data to identify different phases of flight. We can easily imagine greatly simplifying "preflight-check", where each agent (XMA/MDR/µMA) checks the behavior of each signals before a flight to ensure that everything runs smoothly.

Generally speaking, multiplying small algorithms would save operators a considerable amount of time. By increasing efficiency, we are convinced that we could save many flight or engineering hours, whether by offering our own algorithms or by allowing users to implement their own models. This can be done via a centralized computing approach, or by deploying at the edge of the system more frugal nano-models.

Glossary

AI: Artificial Intelligence

CPU: Central Processing Unit

DAU: Data Acquisition Unit

EASA: European Union Aviation Safety Agency

FAA: Federal Aviation Administration

FFT: Fast Fourier Transform

FPGA: Field Programmable Gate Array

FTI: Flight Test Instrumentation

GPU: Graphical Processing Unit

IoT: Internet of Things

LSTM: Long Short-Term Memory

MDR: Modular Data Recorder

MAE: More Electrical Aircraft

ML: Machine Learning

MOE : More electrical aircraft

OBP: On-Board Processing

PoC: Proof of Concept

RNN: Recurrent Neural Network

SDK: Software Development Kit

SDS: Safran Data Systems

SOC: System On a Chip

Sps: samples per second

THD: Total Harmonic Distortion

XMA: eXtra Modular Acquisition unit

References

[1] F. Morel, C. Barreyre, A. Charcosset, O. Coustie, J-E. Denis, L. Hammoud, N. Hans, B. Rouzier, Anomaly detection: a first AI based application for OASIS, *ETTC 2023, Toulouse, 2023.*

[2] V. Belaud, G. Guerrero, R. Pelluault, Q. Lecoq, On-Board Processing: A Decade of Experience Opening to new Applications, *ETTC 2023, Toulouse, 2023.*

[3] Hardt, S. & Faber, M. (2023). Vlab – Enhanced Video and Data Pre- Processing. International Telemetering Conference Proceedings, 58.

[4] G. Guerrero, V. Chomel, F. Monteil, O.Pinto. A new modular electronics approach applied to instrumentation units, *ETTC 2020 (Virtual).*