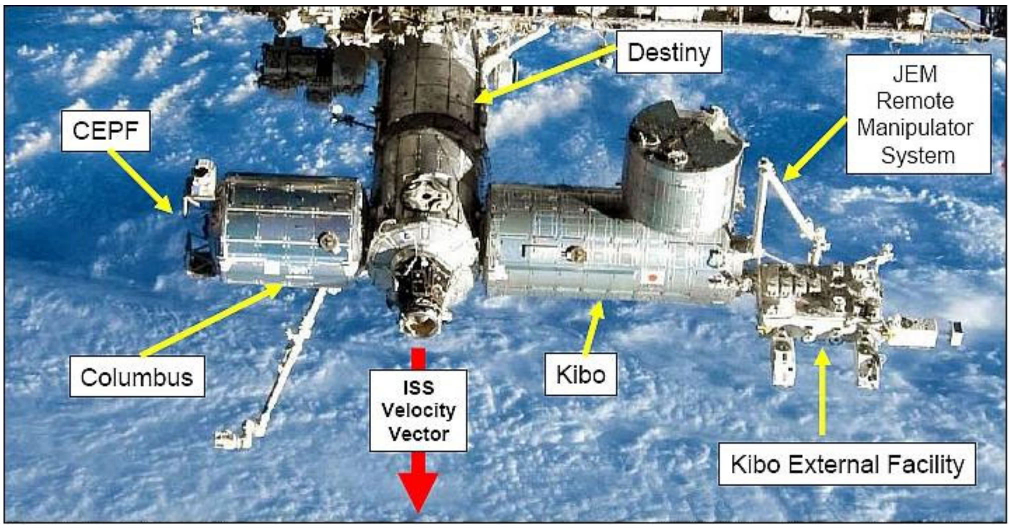
How Airbus Detects Anomalies in ISS Telemetry Data Using TFX

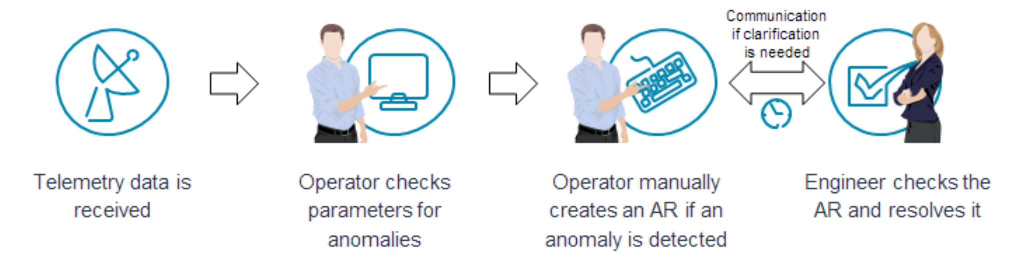
A guest post by Philipp Grashorn, Jonas Hansen and Marcel Rummens from Airbus



The International Space Station and it's different modules. Airbus designed and built the Columbus module in 2008.

Airbus provides several services for the operation of the Columbus module and its payloads on the International Space Station (ISS). Columbus was launched in 2008 and is one of the main laboratories onboard the ISS. To ensure the health of the crew as well as hundreds of systems onboard the Columbus module, engineers have to keep track of many telemetry datastreams, which are constantly beamed to earth.

The operations team at the Columbus Control Center, in collaboration with Airbus, keeps track of thousands of parameters, monitored in 24/7 shifts. If an operator detects an anomaly, he or she creates an anomaly report which is resolved by Airbus system experts. The team at Airbus created the ISS Analytics project to automate part of the workflow of detecting anomalies.


Communication if clarification is needed
Telemetry data is received
Operator checks parameters for anomalies
Operator manually creates an AR if an anomaly is detected
Engineer checks the AR and resolves it

Previous, manual workflow

Detecting Anomalies

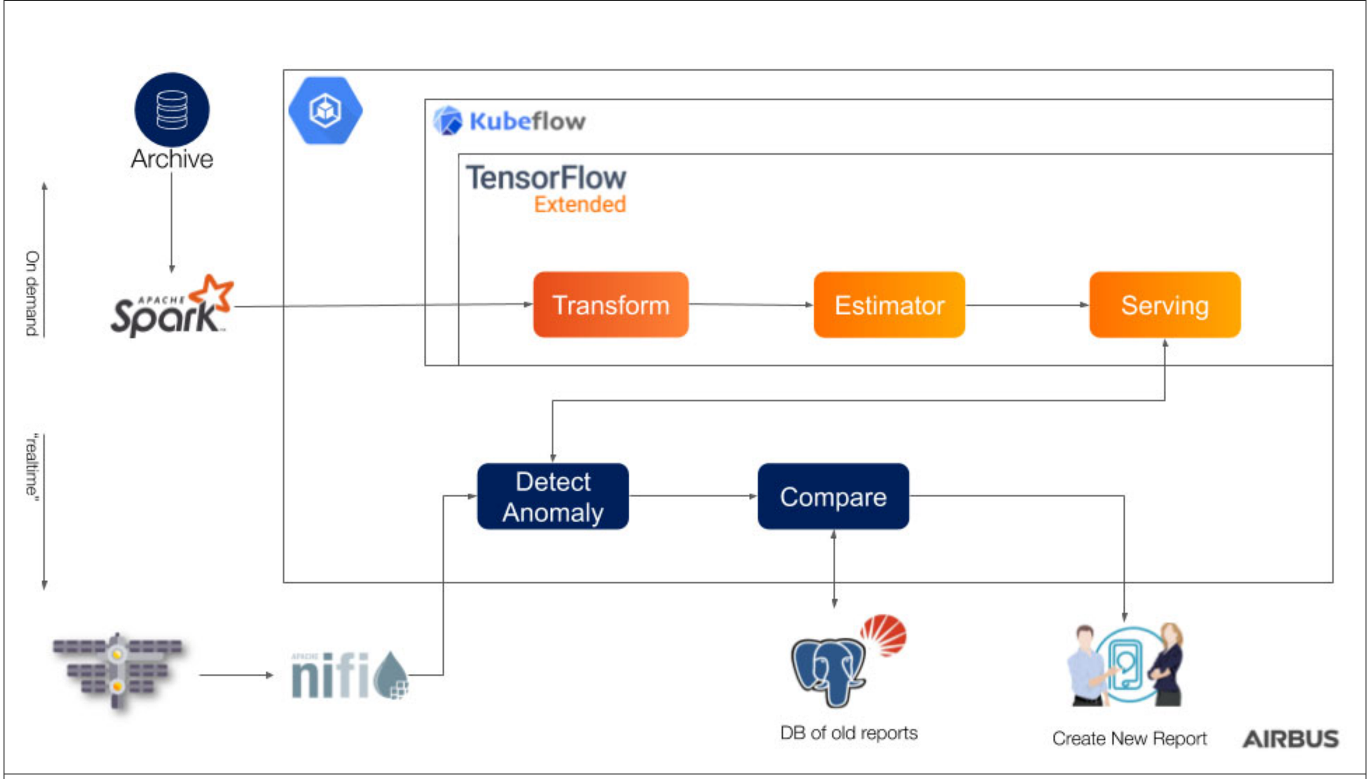
The Columbus module consists of several subsystems, each of which is composed of multiple components, resulting in about 17,000 unique telemetry parameters. As each subsystem is highly specialized, it made sense to train a separate model for each subsystem.

Lambda Architecture

In order to detect anomalies within the real time telemetry data stream, the models are trained on about 10 years worth of historical data, which is constantly streamed to earth and stored in a specialized database. On average, the data is streamed in a frequency of one hertz. Simply looking at the data of the last 10 years results in over 5 trillion data points, (10y * 365d * 24h * 60min * 60s * 17K params).

A problem of this magnitude requires big data technologies and a level of computational power which is typically only found in the cloud. As of now a public cloud was adopted, however as more sensitive systems are integrated in the future, the project has to be migrated to the Airbus Private Cloud for security purposes.

To tackle this anomaly detection problem, a lambda architecture was designed which is composed of two parts: the speed and the batch layer.



High Level architecture of ISS Analytics

The batch layer consists only of the learning pipeline, fed with historical time series data which is queried from an on-premise database. Using an on-premise Spark cluster, the data is sanitized and prepared for the upload to GCP. TFX on Kubeflow is used to train an LSTM Autoencoder (details in the next section) and deploy it using TF-Serving.

The speed layer is responsible for monitoring the real-time telemetry stream, which is received using multiple ground stations on earth. The monitoring process uses the deployed TensorFlow model to detect anomalies and compare them against a database of previously detected anomalies, simplifying the root cause analysis and decreasing the time to resolution. In case the neural network detects an anomaly, a reporting service is triggered which consolidates all important information related to the potential anomaly. A notification service then creates an abstract and informs the responsible experts.

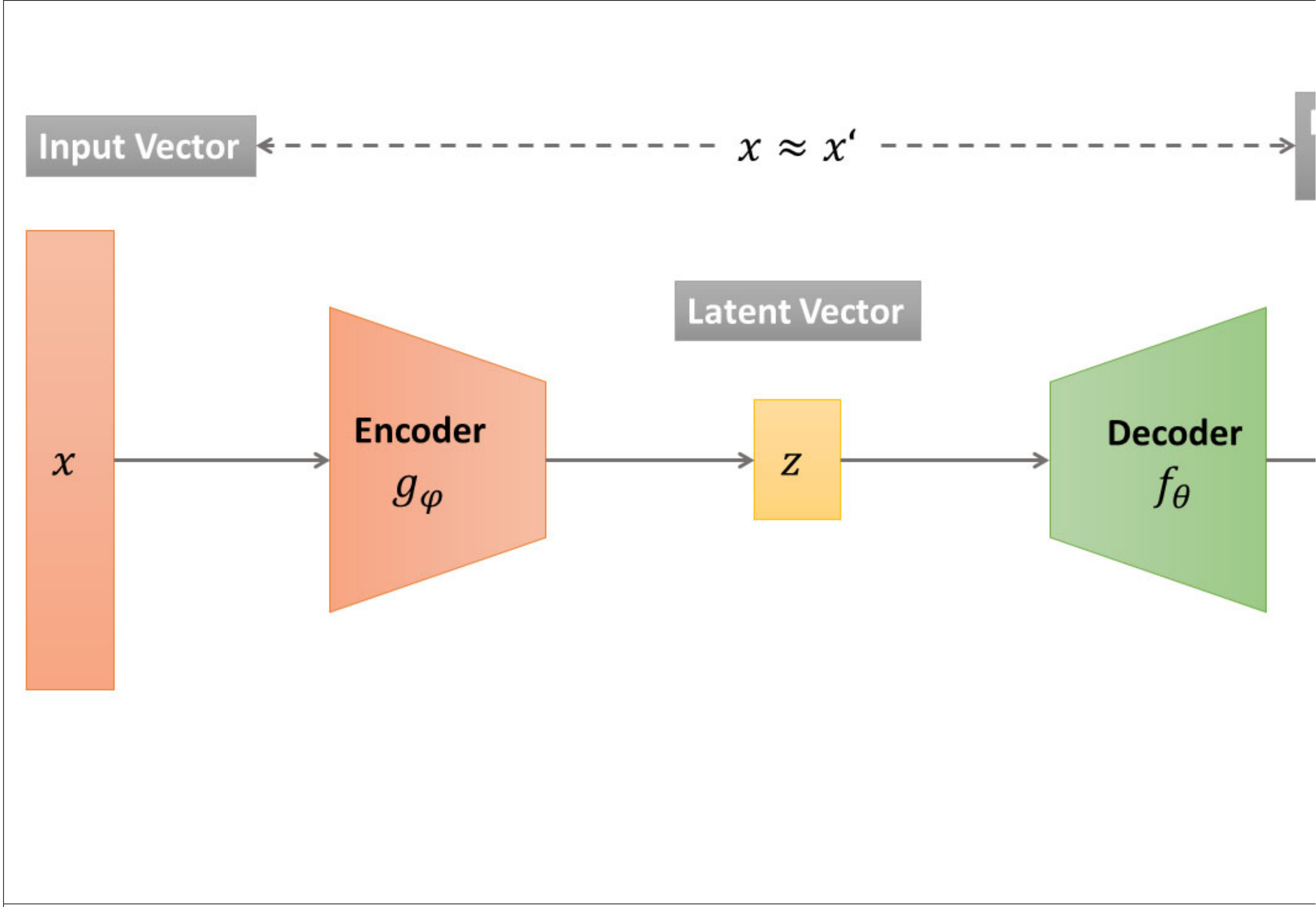
Training an Autoencoder to Detect Anomalies

As mentioned above, each model is trained on a subset of telemetry parameters. The objective of the model is to represent the nominal state of the subsystem. If the model is able to reconstruct observations of nominal states with a high accuracy, it will have difficulties reconstructing observations of states which deviate from the nominal state. Thus, the reconstruction error of the model is used as an indicator for anomalies during inference, as well as part of the cost function in training. Details of this practice can be found here and here.

The anomaly detection approach outlined above was implemented using a special type of artificial neural network called an Autoencoder. An Autoencoder can be divided into two parts: the encoder and the decoder. The encoder is a mapping from the input space into a lower dimensional latent space. The decoder is a mapping from the latent space into the reconstruction space with a dimensionality equal to the input space.

While the encoder generates a compressed representation of the input, the decoder generates a representation as close as possible to the original input, using the latent vector from the encoder. Dimensionality reduction acts as a funnel which enables the autoencoder to ignore signal noise.

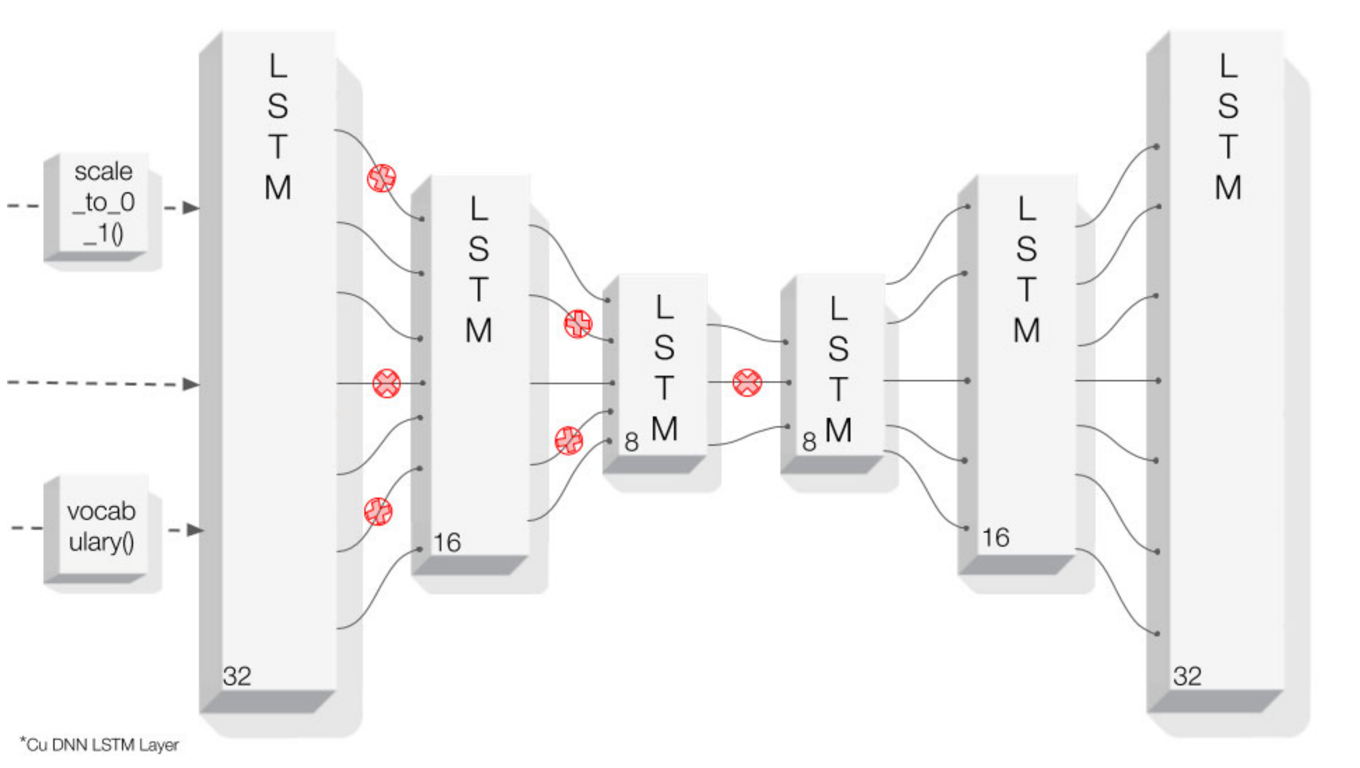
The difference between the input and the reconstruction is called *reconstruction error* and is calculated as the root-mean-square error. The reconstruction error, as mentioned above, is minimized in the training step and acts as an indicator for anomalies during inference (e.g., an anomaly would have high reconstruction error).



Example Architecture of an Autoencoder

LSTM for sequences

The Autoencoder uses LSTMs to process sequences and capture temporal information. Each observation is represented as a tensor with shape `[number_of_features,number_of_timesteps_per_sequence]`. The data is prepared using TFT's `scale_to_0_1` and `vocabulary` functions. Each LSTM layer of the encoder is followed by an instance of tf.keras.layers.Dropout to increase the robustness against noise.



Model Architecture of ISS Analytics (Red circles represent dropout)

Using TFX

The developed solution contains many but not all of the TensorFlow Extended (TFX) components. However it is planned to research and integrate additional components included with the TFX suite in the future.

The library that is most used in this solution is tf.Transform, which processes the raw telemetry data and converts it into a format compatible with the Autoencoder model. The preprocessing steps are defined in the `preprocessing_fn()` function and executed on Apache Beam. The resulting transformation graph is stored hermetically within the graph of the trained model. This ensures that the raw data is always processed using the same function, independent of the environment it is deployed in. This way the data fed into the model is consistent.

The sequence-based approach which was outlined in an earlier section posed some challenges. The `input_fn()` of model training reads the data, preprocessed in the preceding tf.Transform step and applies a windowing function to create sequences. This step is necessary because the data is stored as time steps without any sequence information. Afterwards, it creates batches of size `sequence_length * batch_size` and converts the whole dataset into a sparse tensor for the input layer of the Autoencoder (`tf.contrib.feature_column.sequence_input_layer()`expects sparse tensors).

The `serving_input_fn()` on the other hand receives already sequenced data from upstream systems (data-stream from the ISS). But this data is not yet preprocessed and therefore the `tf.Transform` step has to be applied. This step is preceded and followed by reshaping calls, in order to temporarily remove the sequence-dimension of the tensor for the `preprocessing_fn()`.

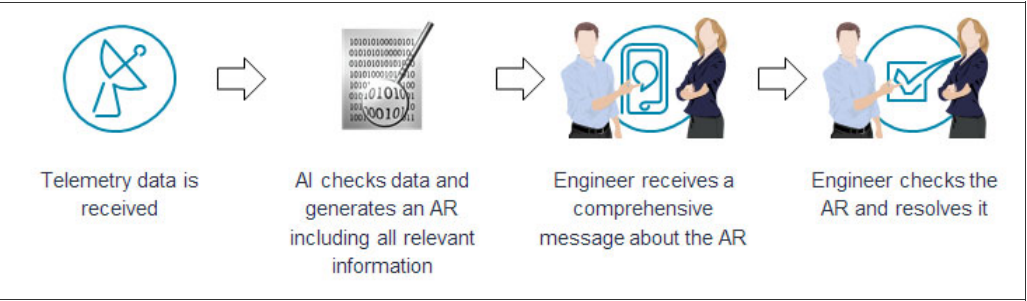
Orchestration for all parts of the machine learning pipeline (transform, train, evaluate) was done with Kubeflow Pipelines. This toolkit simplifies and accelerates the process of training models, experimenting with different architectures and optimizing hyperparameters. By leveraging the benefits of Kubernetes on GCP, it is very convenient to run multiple experiments in parallel. In combination with the Kubeflow UI, one can analyze the hyperparameters and results of these runs in a well-structured form. For a more detailed analysis of specific models and runs, TensorBoard was used to examine learning curves and neural network topologies.

The last step in this TFX use case is to connect the batch and the speed layer by deploying the trained model with TensorFlow Serving. This turned out to be the most important component of TFX, actually bringing the whole machine learning system into production. Its support for features like basic monitoring, a standardized API, effortless rollover and A/B testing, have been crucial for this project.

With the modular design of TFX pipelines, it was possible to train separate models for many subsystems of the Columbus module, without any major modifications. Serving these models as independent services on Kubernetes allows scaling the solution, in order to apply anomaly detection to multiple subsystems in parallel.

Utilizing TFX on Kubeflow brought many benefits to the project. Its flexible nature allows a seamless transition between different environments and will help the upcoming migration to the Airbus Private Cloud. In addition, the work done by this project can be repurposed to other products without any major rework, utilizing the development of generic and reusable TFX components.

Combining all these features the system is now capable of analysing large amounts of telemetry parameters, detecting anomalies and triggering the required steps for a faster and smarter resolution.


Telemetry data is received
AI checks data and generates an AR including all relevant information
Engineer receives a comprehensive message about the AR
Engineer checks the AR and resolves it

The partially automated workflow after the ISS Analytics project

To learn more about Airbus checkout out the Airbus website or dive deeper into the Airbus Space Infrastructure. To learn more about TFX check out the TFX website, join the TFX discussion group, dive into other posts in the TFX blog, or watch the TFX playlist on YouTube.