18th International Conference on Space Operations, Montreal, Canada, 26 - 30 May 2025.

SpaceOps-2025, ID #416

Leveraging Artificial Intelligence and Automatization to Improve Service and System Monitoring at EUMETSAT

Niccolò Pilloni[1], Victor Sierra Urueña[2]

[1] Thorn SDS, Hilpertstraße 27, 64295 Darmstadt, Germany
[2] European Organisation for the Exploitation of Meteorological Satellites (EUMETSAT)
Eumetsat-Allee 1, 64295 Darmstadt, Germany.

Niccolo.Pilloni@external.eumetsat.int, Victor.Sierra@eumetsat.int

Abstract

EUMETSAT's service and system monitoring tools are vital for ensuring the completeness, timeliness, and quality of meteorological satellite data, as well as for maintaining the health of the supporting infrastructure. With the increasing volume and complexity of data, the task of maintaining these monitoring tools has become exponentially more demanding, rendering traditional manual processes insufficient. Technological advancements present opportunities to reduce costs by minimizing the need for active monitoring, automating various processes, reducing manual labour, increasing efficiency in error management, and preventing failures. Considering these challenges, we are building automation tools and integrating Artificial Intelligence (AI) to enhance two critical functions: the prediction of multiple system-level parameters to proactively identify potential issues and the generation of configuration files for our Service Monitoring and Reporting Tool (SMART).
To proactively address potential issues before they arise in mission control centres and to automate basic contingency actions such as server restarts, we need to predict system-level metrics (e.g., , disk usage). To achieve this, we use Machine Learning (ML) algorithms to perform complex time series forecasting.
Due to the breadth of operational products and imagery that EUMETSAT disseminates to our end users, we find ourselves in a situation where changes to the baseline of services is a daily business. Traditional methods of manual updating the configuration for the associated monitoring changes therefore becomes a full-time task. The manual generation of configuration files can also be challenging due to unexpected side-effects. For example, a minor typo which might go unnoticed until the validation stage, can lead to unexpected errors or the system becoming unstable, with the inevitable time loss of troubleshooting issues. To address this, we introduce our innovative automation software for generating configuration files for SMART. We also explore ML models and present prototype examples that can further enhance our tool. This automation is vital for streamlining the configuration process, improving efficiency and speed, reducing human error during development, and ensuring our monitoring setups are consistently optimised to meet the dynamic demands of real-time service operations Finally, we discuss the cost and time-saving benefits of our AI-driven solutions while evaluating the associated risks of partially or fully automating processes in operational systems.

Keywords: EUMETSAT, Artificial Intelligence, Monitoring, Time series forecast, Machine Learning.

Acronyms/Abbreviations

AI	Artificial Intelligence
CNN	Convolutional Neural Network
GRU	Gated Recurrent unit
GEMS	Generic Event Monitoring System
LLM	Large Language Model
LSTM	Long-Short Term Memory networks
ML	Machine Learning
MLP	Multilayer Perceptrons
MSE	Mean squared error
RMSE	Root mean squared error
RNN	Recurrent Neural Network
SMART	System Monitoring and Reporting Tool

1. Introduction to EUMETSAT

EUMETSAT, headquartered in Darmstadt, Germany, is an intergovernmental organization responsible for operating Europe's meteorological satellites. These satellites provide continuous observation of Earth's atmosphere, oceans, and land surfaces, delivering essential data 24/7 throughout the year. The collected data is disseminated widely to national meteorological services across Europe and a global user community, supporting critical environmental monitoring and forecasting activities
.
EUMETSAT currently manages a diverse fleet of operational satellites:

- **MTG-I1 (Meteosat Third Generation – Imaging 1):** A geostationary satellite launched in December 2022. It enhances weather forecasting through advanced imaging capabilities focused on rapid and detailed atmospheric observations.

- **Meteosat Second Generation (MSG):** Geostationary satellites including Meteosat-10 and Meteosat-11 covering Europe and Africa, alongside Meteosat-9 monitoring the Indian Ocean region.

- **Metop Series:** Polar-orbiting satellites (Metop-B and Metop-C), part of the Initial Joint Polar System (IJPS), operated collaboratively with NOAA, providing comprehensive data for weather forecasting, climate monitoring, and environmental research.

- **Jason-CS/Sentinel-6:** A mission dedicated to precise global sea level measurements, contributing significantly to climate monitoring, oceanographic studies, and seasonal weather predictions. While Jason-3 is part of this international collaborative framework, it is not directly operated by EUMETSAT.

- **Sentinel-3 Satellites (S3A and S3B):** Offering critical data on global ocean color, sea surface temperatures, and sea surface heights, these satellites support extensive oceanographic and environmental applications.

Future satellites and programmes which will be launched from 2025 - 2028, and operated at EUMETSAT, are:

- Two new satellites from the Meteosat programme, **MTG-S1** and **MTG-I2**
- A second satellite from the Sentinel-6 programme, **Sentinel-6B**.
- Three new satellites from the EUMETSAT Polar System programme, **EPSSG-A1** , **EPSSG-B2** and **EPS Sterna**
- Two new satellites from the Sentinel-3 program, **S3C** and **S3D**.
- Three new satellites to monitor the global anthropogenic CO2 and CH4, **CO2M- A, CO2M-B, CO2M- C**.

2. Challenges and Strategic Innovations

With an anticipated significant expansion in satellite missions over the next two decades, EUMETSAT faces increasing operational complexities. Managing an expanding satellite fleet and the exponential growth of data volumes demands strategic ground segment infrastructure scalability and innovative system architecture redesign.
A primary challenge for EUMETSAT is efficiently monitoring extensive data transfers while minimizing associated financial and human resource expenditures. This paper examines current solutions, emerging technologies, and strategic initiatives developed by EUMETSAT to overcome these operational hurdles, ensuring long-term sustainability, enhanced efficiency, and continuous advancement in satellite operations and data management.

3. Mission performance Monitoring and Reporting at EUMETSAT

EUMETSAT provides critical satellite-based data and products continuously, operating 24 hours a day, seven days a week. To maintain uninterrupted and reliable operations, it is essential to continuously monitor and manage the underlying infrastructure and operational processes. Effective monitoring facilitates prompt detection of anomalies, rapid issue characterization, and identification of affected components, such as antenna failures or configuration errors in processing systems.
To achieve comprehensive monitoring and reporting [1] capabilities, two proprietary solutions and one COTS are employed. These proprietary tools have evolved significantly over the past decade, each fulfilling distinct yet complementary roles. One tool is designed primarily to collect event logs in a centralized manner (GEMS), while the other ensures that actual data flows match predefined user requirements and expectations (SMART). Additionally, EUMETSAT incorporates OP5, a monitoring platform derived from Nagios.
These tools are briefly introduced in the following subsections to provide an overview of their functionality, setting the stage for a discussion on potential improvements.

3.1 Generic Event Monitoring System (GEMS)

The Generic Event Monitoring System (GEMS) serves as EUMETSAT's centralized solution for systematically aggregating log data from diverse operational and mission-specific systems. GEMS supports real-time event monitoring through a dedicated user interface, enabling immediate intervention by mission operator, and also allows for retrospective analysis through historical data queries. Operators can access GEMS through the graphical user interface (GUI), shown in Fig. 1.

Architecturally, GEMS consists of two primary components:

- GEMS Client: A lightweight Java application deployed on operational servers, responsible for capturing relevant event logs either directly through an embedded local agent or by interfacing with system APIs. The client transfers collected data securely and efficiently to the GEMS server using standard communication protocols, including FTP, HTTP (RESTful API), or JMS message queues. The client is platform-independent and can run effectively across various UNIX/Linux and Windows environments.

- GEMS Server: Receives, processes, and stores incoming log data into a centralized database. It offers operators and analysts a user-friendly interface for real-time monitoring, event correlation, and extensive querying capabilities to explore historical events and perform trend analysis.

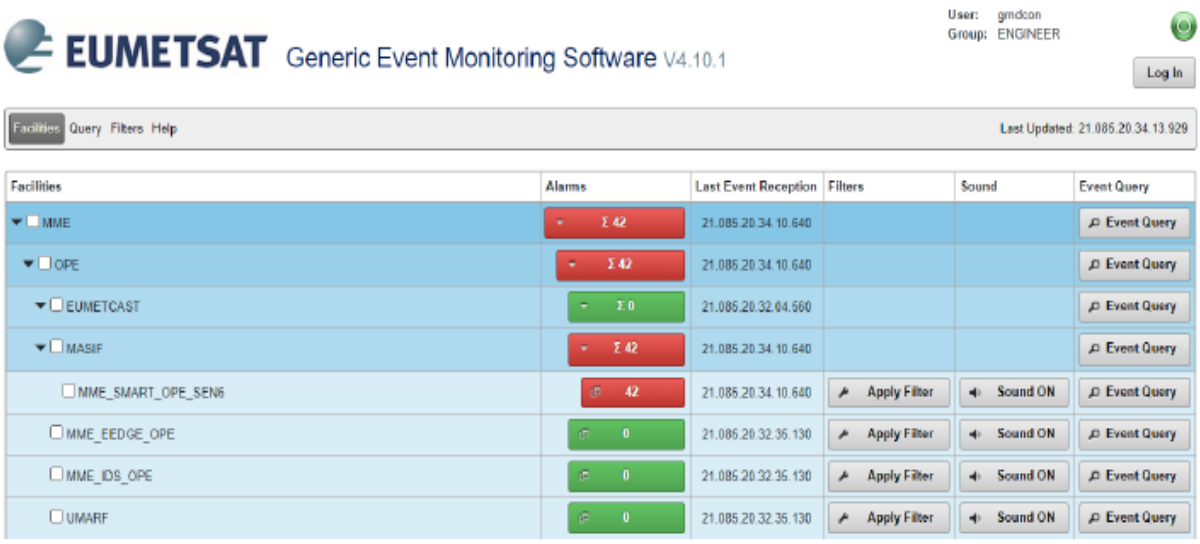


Fig. 1. GEMS graphical user interface

3.2 System Monitoring and Reporting Tool (SMART)

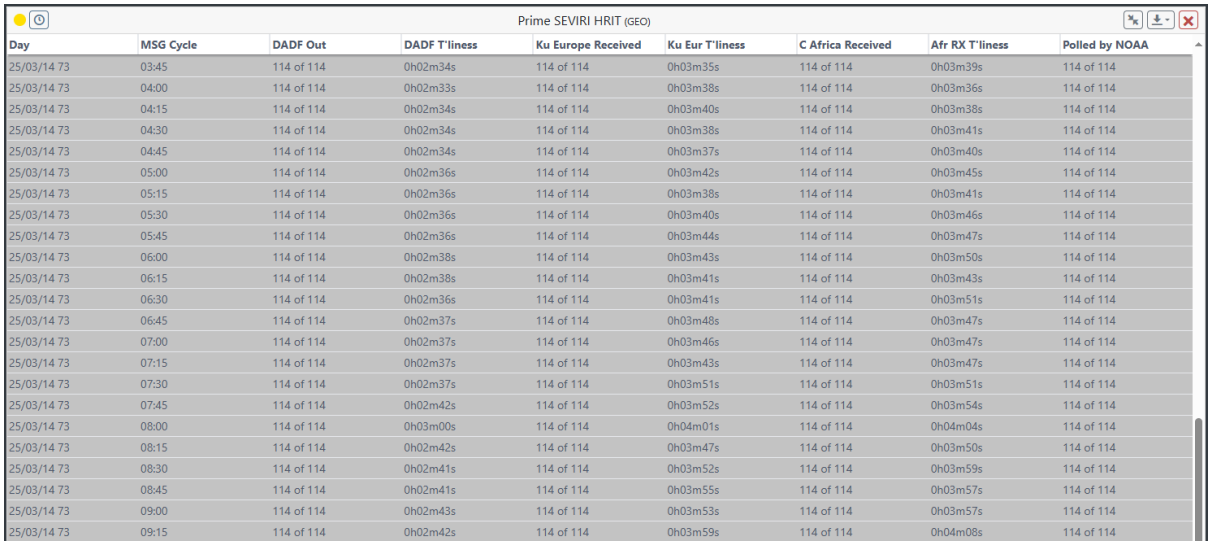
The System Monitoring and Reporting Tool (SMART) delivers dynamic and precise monitoring capabilities by integrating operational schedules, mission planning files, and other dynamic inputs to establish accurate expectations. It evaluates the completeness, timeliness of products delivered by EUMETSAT services.

Like GEMS, SMART is accessed by mission operators through dedicated viewers, as shown in Fig. 2. It provides continuous monitoring of data flows across mission facilities, giving operators a clear and immediate visual overview, ensuring uninterrupted 24/7 operational oversight.

3.2.1 SMART Architecture

SMART architecture is composed of two primary components:

- SMART Relay: Contains the web-based client interface used for accessing viewers and generating reports.
- SMART Instances: Independent deployments aligned with specific satellite missions, including MSG, MTG, S3, and S6, among others.



Day	MSG Cycle	DADF Out	DADF T'liness	Ku Europe Received	Ku Eur T'liness	C Africa Received	Afr RX T'liness	Polled by NOAA
25/03/14 73	03:45	114 of 114	0h02m34s	114 of 114	0h03m35s	114 of 114	0h03m39s	114 of 114
25/03/14 73	04:00	114 of 114	0h02m33s	114 of 114	0h03m38s	114 of 114	0h03m36s	114 of 114
25/03/14 73	04:15	114 of 114	0h02m34s	114 of 114	0h03m40s	114 of 114	0h03m38s	114 of 114
25/03/14 73	04:30	114 of 114	0h02m34s	114 of 114	0h03m38s	114 of 114	0h03m41s	114 of 114
25/03/14 73	04:45	114 of 114	0h02m34s	114 of 114	0h03m37s	114 of 114	0h03m40s	114 of 114
25/03/14 73	05:00	114 of 114	0h02m36s	114 of 114	0h03m42s	114 of 114	0h03m45s	114 of 114
25/03/14 73	05:15	114 of 114	0h02m36s	114 of 114	0h03m38s	114 of 114	0h03m41s	114 of 114
25/03/14 73	05:30	114 of 114	0h02m36s	114 of 114	0h03m40s	114 of 114	0h03m46s	114 of 114
25/03/14 73	05:45	114 of 114	0h02m36s	114 of 114	0h03m44s	114 of 114	0h03m47s	114 of 114
25/03/14 73	06:00	114 of 114	0h02m38s	114 of 114	0h03m43s	114 of 114	0h03m50s	114 of 114
25/03/14 73	06:15	114 of 114	0h02m38s	114 of 114	0h03m41s	114 of 114	0h03m43s	114 of 114
25/03/14 73	06:30	114 of 114	0h02m36s	114 of 114	0h03m41s	114 of 114	0h03m51s	114 of 114
25/03/14 73	06:45	114 of 114	0h02m37s	114 of 114	0h03m48s	114 of 114	0h03m47s	114 of 114
25/03/14 73	07:00	114 of 114	0h02m37s	114 of 114	0h03m46s	114 of 114	0h03m47s	114 of 114
25/03/14 73	07:15	114 of 114	0h02m37s	114 of 114	0h03m43s	114 of 114	0h03m47s	114 of 114
25/03/14 73	07:30	114 of 114	0h02m37s	114 of 114	0h03m51s	114 of 114	0h03m51s	114 of 114
25/03/14 73	07:45	114 of 114	0h02m42s	114 of 114	0h03m52s	114 of 114	0h03m54s	114 of 114
25/03/14 73	08:00	114 of 114	0h03m00s	114 of 114	0h04m01s	114 of 114	0h04m04s	114 of 114
25/03/14 73	08:15	114 of 114	0h02m42s	114 of 114	0h03m47s	114 of 114	0h03m50s	114 of 114
25/03/14 73	08:30	114 of 114	0h02m41s	114 of 114	0h03m52s	114 of 114	0h03m59s	114 of 114
25/03/14 73	08:45	114 of 114	0h02m41s	114 of 114	0h03m55s	114 of 114	0h03m57s	114 of 114
25/03/14 73	09:00	114 of 114	0h02m43s	114 of 114	0h03m53s	114 of 114	0h03m57s	114 of 114
25/03/14 73	09:15	114 of 114	0h02m42s	114 of 114	0h03m59s	114 of 114	0h04m08s	114 of 114

Fig. 2. SMART Viewer example: MSG, a mission's prime instrument, facilitates SEVIRI data dissemination.

3.2.2 SMART Configuration Files

SMART configuration files define the structure of services by dividing them into monitoring checkpoints known as 'trackers'. Typical examples include product reception, generation of Level 0 and Level 2 products, product archiving, and delivery.

3.2.3 SMART Configuration Challenges

A significant challenge associated with SMART involves the considerable effort required to update and manage its configuration files. Continuous updates to product parameters, quality expectations, and viewer configurations result in labor-intensive maintenance. Given the anticipated increase in data flows, additional dedicated engineering resources might become necessary.

3.3. OP5 Periodic Monitoring

OP5 is an enterprise version of the third party Nagios product, which is widely used in industry for periodic monitoring of systems. It performs basic checks on servers, applications and COTS processes (e.g. CPU usage, physical memory, system load, availability status) once every five minutes (on average) and is highly configurable in terms of metrics and their frequency.

When one of the OP5 requests results in a failure, the tool will increase the frequency of the poll so that the system is evaluated more accurately. After a specified failure threshold, OP5 will raise an alarm via UI, email or any other configurable mean (such as GEMS).

OP5 currently monitors in the region of 4000 operational servers providing approximately 70,000 metrics regarding the state of the operational system. In addition, the storage capacity used for holding the performance data is efficient - OP5 is relying on RRD files that can keep for each hosts/service statistics for many years in a less than 1 MB file. As a result, the system is using less than 30 GB disk space for performance data over a 2 year period.

4. Purpose of the Study

The rapid evolution of satellite services has significantly increased the need for efficient system maintenance and proactive monitoring within the EUMETSAT suite of system monitoring tools.
Traditional manual approaches are no longer sufficient to address emerging issues, such as sudden spikes in resource usage and unexpected system errors. Furthermore, the frequent updates required for service configurations present an increased risk of human errors, such as typos or misconfigurations, potentially compromising system stability.
The integration of advanced technologies, particularly automation and artificial intelligence (AI), provides several strategic benefits:

- **Proactive Issue Detection:** Machine learning models can predict system metrics, enabling early identification and resolution of potential issues.
- **Automated Configuration Management:** Automating the creation and updating of SMART configuration files significantly reduces the occurrence of errors typically associated with manual processes.
- **Resource Optimization:** Automation allows for dynamic and efficient resource management, ensuring timely responses to system demands and minimizing service disruptions.

Ensuring consistent and reliable real-time service is essential, as operations are inherently time-sensitive and must run without interruption, 24/7.
The following sections will focus on further enhancing reliability and efficiency. Methodologies for predicting critical system metrics, identifying additional metrics for future monitoring, and leveraging automation to streamline XML configuration file management, thus further enhancing reliability and efficiency. Additionally, we will explore how AI particularly through the use of Large Language Models (LLMs) can further automate the generation of XML configuration files, significantly simplifying and improving our management processes.

5. Time series forecast - Predictive Maintenance through ML

To effectively tackle proactive system maintenance challenges, we utilize advanced machine learning models trained on historical time series data gathered from EUMETSAT's monitoring systems in this study. These datasets include critical metrics such as disk space usage. Leveraging this historical data, our models forecast system behaviours and proactively identify anomalies indicative of potential issues.
Various deep learning neural network architectures have demonstrated significant efficacy in time series forecasting. Among these, Multilayer Perceptrons (MLPs) [2], Convolutional Neural Networks (CNNs) [3], and Recurrent Neural Networks (RNNs) [4] stand out. These networks excel in capturing complex patterns within sequential data, surpassing traditional statistical forecasting methods. Particularly notable are Gated Recurrent Units (GRUs) [5], a sophisticated variant of RNNs explicitly designed to address the vanishing gradient problem, enabling efficient learning of long-term temporal dependencies.
GRUs utilizes two gating mechanisms the update gate and the reset gate. This streamlined design allows GRUs to effectively manage prolonged sequential relationships while significantly reducing computational complexity.

On the other hand, CNNs are typically used for classification and image recognition tasks. However, they have the advantage of excelling in identifying local patterns and capturing short-term variations.

5.1 **Data Extraction and preparation**

The server is equipped with Munin [6], a monitoring tool that allows us to extract the metrics we aim to predict. Munin is an open-source networked resource monitoring tool that provides comprehensive insights into the performance and status of IT infrastructure components. It falls into the category of computer system, network, and infrastructure monitoring software.
Munin retrieves data points every five minutes, offering a granular view of system behaviour over time. By continuously collecting time-series data from various monitored components, Munin ensures a reliable historical dataset, stored in its backend, which can be exported for advanced machine learning analysis.
Initially, we worked directly with the extracted measurements data, attempting to predict the absolute values of the metrics as they were recorded. However, we later realized that our primary interest lay in understanding how these values evolved - whether they increased or decreased over time.
To capture this dynamic behaviour, we calculated the difference between consecutive timestamps, effectively transforming the data into a first-order difference series, ccalculated by subtracting each value in a time series from its preceding value:

$$Xt' = Xt - Xt - 1$$

This transformation is commonly used to analyse trends and convert a non-stationary time series into a stationary one, facilitating more effective statistical analysis and forecasting.

This transformation had an important side effect: it eliminated the need for explicit data normalization, as the differences inherently adjusted for scale variations. In the performance evaluation, we will compare the results obtained from both the raw and normalized data to assess the impact of this approach on predictive accuracy.
To ensure that the model focuses on meaningful trends and does not learn from potentially misleading data, we applied a 20% drop filtering on the training values. This decision was motivated by the fact that we are primarily interested in increasing memory usage trends over time. A sudden and significant drop in memory usage may indicate an active maintenance operation, such as a system reboot or memory cleanup, rather than a natural system behaviour.
By removing these abrupt declines, the model is better equipped to capture the gradual accumulation of memory consumption, leading to more reliable prediction in a real-world deployment scenario.

5.2 **Sliding window technique**

In time series forecasting using neural networks, the primary goal is to provide the input data and corresponding target values in such a way that the network can learn temporal patterns, allowing for accurate predictions. An effective technique for constructing these input-target pairs is the "sliding window" approach.
The operation of the sliding window is based on a fixed time window, referred to as the window size, which determines how many past time steps will be used as input for the neural network. The target value is obtained by shifting the window forward by the number of time steps to be predicted. At each step, the window moves forward, generating a new input-target pair. This process continues until the window covers the entire available time series.
The sliding window is particularly advantageous for long time series, as it reduces computational complexity and optimizes the use of available data for training the network.

Consider a time-series $\{x_1, x_2, \ldots, x\_T\}$, where T represents the total number of observations. The sliding window technique with a window length n and forecasting horizon m can be formalized as follows:

- Input: For each time step t, we define a time window of length n $X_t = \{x_t, x_{t-1}, \ldots, x_{t-n+1}\}$ where X_t represents the input vector containing the values of the time series at the n preceding time steps.

- Target y_t is the value of the time series that we wish to predict after a forecasting horizon of m steps $y_t = x_{t+m}$

- Generation of Input-Target Pairs: the window shifts at each time step t, generating a new input-target pair (X_t, y_t). This process repeats until the entire time series is covered: (X_{t1}, y_{t1}), (X_{t2}, y_{t2}), …, (X_{tk}, y_{tk}), where $k = T - n - m + 1$ represents the total number of generated pairs.

For this project, sequences of 12 points were selected, with the objective of predicting the subsequent 12 points. Given the data granularity of 5 minutes per time step, this setup provides at least one hour of predictive insight. Future work will explore different sequence lengths and constraints to refine forecasting performance further.

5.3 **Data split in training, validation and test sets**

In machine learning, datasets are typically divided into three subsets: **training**, **validation**, and **test sets**. This split ensures that the model is trained effectively, optimized properly, and evaluated fairly.

- The training set (70% in this project) is used to teach the model patterns from the data, allowing it to adjust its parameters.
- The validation set (20%) is adopted to fine-tune hyperparameters and prevent overfitting by assessing the model's performance on unseen data during training.
- The test set (10%) provides a final unbiased evaluation of how well the model generalizes to completely new data.

The chosen split of 70-20-10 is a well-balanced approach, ensuring that the model has enough data to learn effectively while also maintaining a sufficient portion for validation and testing. This distribution minimizes overfitting risks and enhances the model's ability to perform accurately in real-world scenarios.

5.4 **Normalization**

Normalization [7] rescales the data to a specific range or ensures a standard scale (mean of 0 and standard deviation of 1). This helps prevent issues caused by features with different scales dominating the learning process, ensuring that all features contribute equally to the model's training. This makes the optimization process more stable and efficient.

We tested several normalization techniques, let X' be the normalized value and X the raw value:

- **StandardScaler:** Standardizes features by removing the mean and scaling to unit variance, suitable for normally or nearly normally distributed data.
  $$X' = (X - \mu) / \sigma$$
  where μ is the mean and σ is the standard deviation.

- **RobustScaler:** Designed for datasets with outliers; scales features based on the median ($\tilde{X}$) and the interquartile range (IQR), enhancing robustness against extreme values.
  $$X' = (X - \tilde{X}) / IQR$$
  where $IQR = Q_3 - Q_1$, representing the difference between the third and first quartiles.

- **MaxAbsScaler:** Scales each feature by its maximum absolute value, resulting in values within the range [-1, 1]. Particularly useful for sparse datasets or constrained feature ranges.
  $$X' = X / |X\_max|$$

- **MinMaxScaler:** Rescales data into a specified range, commonly [0,1] or [-1,1]. This method preserves relative distances but is sensitive to outliers.

$$X' = (X - X\_min) / (X\_max - X\_min)$$

We found that the most effective normalization technique was the **MinMaxScaler.** This approach allows us to constrain the values between [0,1], preserving their relative scale while ensuring compatibility with machine learning models. Despite its sensitivity to outliers, this choice was optimal for our dataset, as it retains interpretability while improving model performance.

However, in our case, as already previously mentioned it was reasonable to experiment without normalization, since memory percentage usage values are inherently low and exhibit a limited range. This reduces the risk of scale-related instability, making normalization less critical compared to other system metrics.

5.5 **Optimizer**

The model was trained using the Adam optimizer [8], a popular optimization algorithm in deep learning due to its ability to adapt learning rates dynamically during training. Adam utilizes estimates of the first and second moments of the gradients to update the model parameters efficiently. The specific update rules of Adam are described by the following equations:

$$m\_t = \beta_1 m_{(t-1)} + (1 - \beta_1) \nabla L_t$$
$$v\_t = \beta_2 v_{(t-1)} + (1 - \beta_2) (\nabla L_t)^2$$
$$\hat{m}_t = m_t / (1 - \beta_1^t), \quad \hat{v}_t = v_t / (1 - \beta_2^t)$$
$$\theta_t = \theta_{(t-1)} - \alpha \hat{m}_t / (\sqrt{(\hat{v}_t)} + \varepsilon)$$

Where:

- θ is the set of model parameters at iteration t.
- α is the learning rate, a hyperparameter determining the step size during parameter updates.
- ∇_t represents the gradient of the loss function with respect to the parameters at iteration t.
- m_t is the exponential moving average of the gradient (first-moment estimate).
- v is the exponential moving average of the squared gradient (second-moment estimate).
- β_1, β_2 are hyperparameters controlling the exponential decay rates for the moving averages of the first and second moments, respectively.
- $\hat{m}_t$ $\hat{v}$ denote the bias-corrected versions of m_t and v_t.
- ε is a small constant added to prevent division by zero, ensuring numerical stability.

5.6 **Loss function**

The loss function employed is the Mean Squared Error (MSE), which quantifies the average squared difference between the predicted values and the observed target values. The MSE effectively penalizes larger prediction errors more significantly than smaller ones, providing a robust metric for regression tasks. Due to its differentiability and sensitivity to larger deviations, MSE is particularly suitable for optimization using gradient-based methods, facilitating efficient convergence during training.

$$MSE = (1/N)\ \Sigma(y_i - \hat{y}_i)^2$$

where:
y_i is the true memory usage at time step i.
$\hat{y}_i$ is the predicted memory usage at time step i.
N is the number of time steps.

5.7 **Validation Metric**

The Root Mean Squared Error (RMSE) metric is used to evaluate the model's performance. RMSE is calculated as the square root of the mean squared differences between predicted and observed values, providing an interpretable measure of predictive accuracy expressed in the original units of the target variable. Due to its intuitive interpretation, RMSE facilitates meaningful comparison of model performance across different datasets and scenarios.

$$RMSE = \sqrt{[(1/N)\ \Sigma(y_i - \hat{y}_i)^2]}$$

often includes ongoing activities such as satellite launches, software upgrades, and routine operational tasks. Failure to account for these concurrent responsibilities can lead to resource conflicts, delays, and priority shifts that affect the overall migration schedule. In addition to resource constraints, the introduction of new

5.8 **Model Architecture for Memory Usage Prediction**

Classical Machine Learning models linear and polynomial regression and several neural network architectures were evaluated. Each model was designed to capture different aspects of time series data, and the comparison of their performance provided valuable insights into the strengths and weaknesses of each approach.
Among all the models tested, to predict memory percentage usage, we designed a deep learning model that leverages a combination of 1D convolutional layers (Conv1D) and Gated Recurrent Units (GRU). This hybrid architecture is particularly well-suited for time series forecasting, as it efficiently captures both local dependencies (short-term patterns) and long-term dependencies (temporal correlations over extended periods). The architecture consists of the following key components:

5.8.1 1D Convolutional Layer (Conv1D)

The first layer of the model is a 1D convolutional layer with 64 filters and a kernel size of 5, using the ReLU activation function. This layer plays a crucial role in extracting localized temporal features from the input time series, Identifying short-term fluctuations in memory usage patterns and reducing the input dimensionality while preserving relevant information.

Mathematically, the convolution operation for a single filter is defined as:

$$h_t = \sigma(\Sigma_{i=0}^{k-1} w_i x_{t+i} + b)$$

where:
x_{t+i} represents input time series values within the kernel window of size k.
w_i are learnable filter weights.
b is the bias term.
σ is the ReLU activation function, defined as ReLU(x) = max(0, x).

5.8.2 Flattening Layer

Following the convolutional layer, we apply a Flatten layer, transforming the multi-dimensional feature map into a one-dimensional vector:

$$X\_flattened = Flatten(X\_conv)$$

5.8.3 Repeat Vector Layer

We use a RepeatVector layer to replicate the flattened feature representation across all future prediction steps:

$$X\_repeated = RepeatVector(n\_output, X\_flattened)$$

where n_output is the number of predicted time steps.

5.8.4 Stacked GRU Layers

The core consists of four stacked GRU layers, each with 100 hidden units and ReLU activation, learning short-term variations and long-term dependencies:

$$z_t = \sigma(W\_z\, x_t + U\_z\, h_{t-1} + b\_z)$$
$$r_t = \sigma(W\_r\, x_t + U\_r\, h_{t-1} + b\_r)$$
$$\tilde{h}_t = tanh(W\_h\, x_t + U\_h\, (r_t \odot h_{t-1}) + b\_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

where:
z_t update gate
h_t hidden state (output)
$\tilde{h}_t$ candidate hidden state
W, U, b weight matrices and biases

5.8.5 Time-Distributed Dense Layer

Finally, a Time-Distributed Dense layer produces the predictions:
$y_t = W_o\, h_t + b_o$
where:
W_o, b_o weights and biases of the dense layer $\mathbf{h_t}$ hidden state from GRU layer.

5.9 Model Tuning

The hyperparameters considered during the tuning phase include:

- **Number of Units** in the GRU Layers. Each GRU layer was tested with 50, 100, and 200 units. Increasing the number of units allows the network to capture more complex temporal dependencies in memory usage trends but also increases computational cost.

- **Batch size** which refers to the number of training samples processed together before updating the model's parameters was varied among 8, 16, 32, and 64.

- The **kernel size** of the Conv1D layer was varied among 3, 5, and 7, smaller kernels (k=3) emphasize local short-term variations, whether Larger kernels (k = 7) capture broader temporal dependencies in memory usage fluctuations.

- **Early stopping** was implemented to prevent overfitting by halting training when the model's performance on the validation set no longer improved. It monitored the loss function and stopped training if no progress was observed for five consecutive epochs (where one epoch represents a full pass through the entire training dataset), reducing computational cost and enhancing generalization.

5.10 Result and Discussion

The training and evaluation of the Conv1D-GRU model for memory usage prediction were conducted on a high-performance computing cluster consisting of both CPU-based and GPU-based nodes. The CPU nodes were equipped with 24 virtual CPUs (vCPUs), 128 GB of RAM, and 500 GB of disk storage, with a total of six such nodes utilized during training. Additionally, two GPU-based nodes, each featuring an NVIDIA A40 GPU with 48 GB of dedicated memory, were employed primarily to optimize the training process by leveraging parallel computations, significantly accelerating model convergence. The combination of multi-core CPUs and high-memory GPUs enabled efficient processing of extensive time-series datasets and greatly reduced the training duration.
As previously mentioned, the results include a comparative analysis between normalized and non-normalized data. The rationale behind this comparison is to assess whether a model trained on raw (non-normalized) data might better reflect real-world performance compared to one trained on processed (normalized) data.
From the obtained results, it is clear that normalization yields a lower loss function value, indicating that the model successfully identifies a lower local minimum during training (Fig 3). Correspondingly, the Root Mean Squared Error (RMSE), calculated in the training (Fig 4.), validation (Fig 5.) and test (Fig 6.) set is also lower for the normalized dataset. However, a detailed examination of the graphs reveals that RMSE values from the non-normalized data are already effective from the very first epoch, consistently maintaining values close to

zero. Conversely, the normalized data's RMSE decreases more gradually, eventually converging to even lower values than the raw data.
Nonetheless, when evaluating predictions using actual raw values, predictions derived from non-normalized data closely approximate the actual observed values more consistently ("raw prediction" and "normalized prediction" in Fig 7). In contrast, normalized data predictions, although statistically closer on average, occasionally deviate significantly from the true values. This implies that normalization generally leads to predictions that are more accurate on average but less consistently close to the actual observed intervals compared to raw data (Fig 7.).
Given these observations, further validation could involve cross-validation using different subsets of the dataset or applying additional statistical tests such as residual analysis to verify model performance and robustness. For these reasons, we have chosen to prioritize training the model using non-normalized data, as it provides predictions that are consistently closer to realistic operational intervals, despite a slightly higher average error.
At present, we have fully developed the tool, which is under validation and has been integrated thanks to collaboration with another team at EUMETSAT known as the Machine Learning Framework [10]. Specifically, our neural network runs within their system on a daily basis, providing predictions every 5 minutes (due to the granularity of Munin) for the upcoming hour. To ensure the neural network remains continuously updated, the data is trained automatically on a weekly basis. During each training session, the model is updated using the most recent month's data, following a sliding window approach where the oldest week of data is discarded, and the latest week's data is incorporated. This systematic process maintains an up-to-date and effective predictive model.

These results are already valuable for real operations, as they enable accurate prediction of future values with minimal error. Furthermore, by optimizing the model to filter out large variations (as discussed in Section 5.3), the likelihood of triggering unnecessary alerts during maintenance activities is significantly reduced.

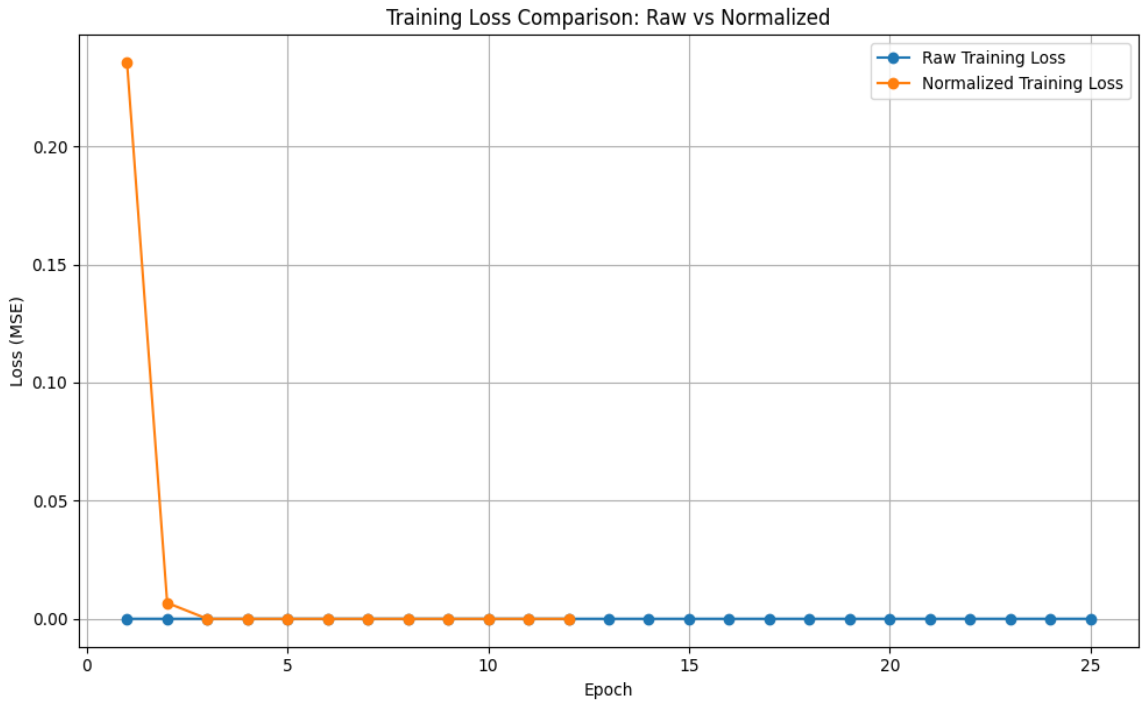


Fig. 3. Training Loss Comparison Between Raw and Normalized Data, emphasizing the influence of data preprocessing on model performance and convergence speed

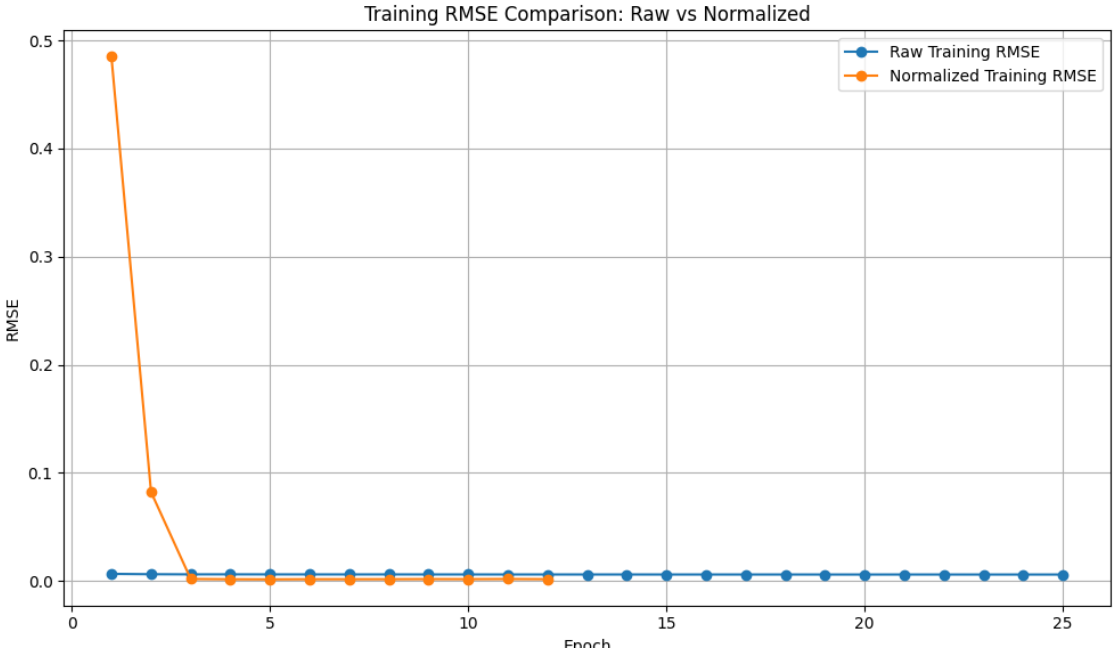


Fig. 4. Training RMSE Comparison Between Raw and Normalized Data

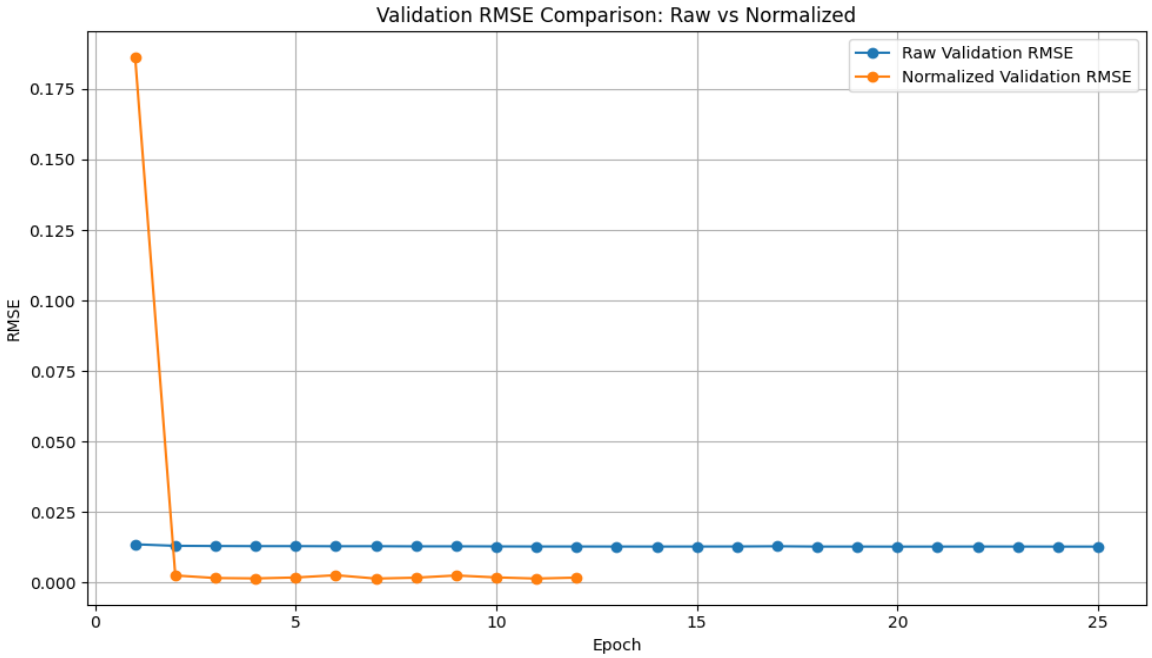


Fig. 5. Validation RMSE Comparison Between Raw and Normalized Data

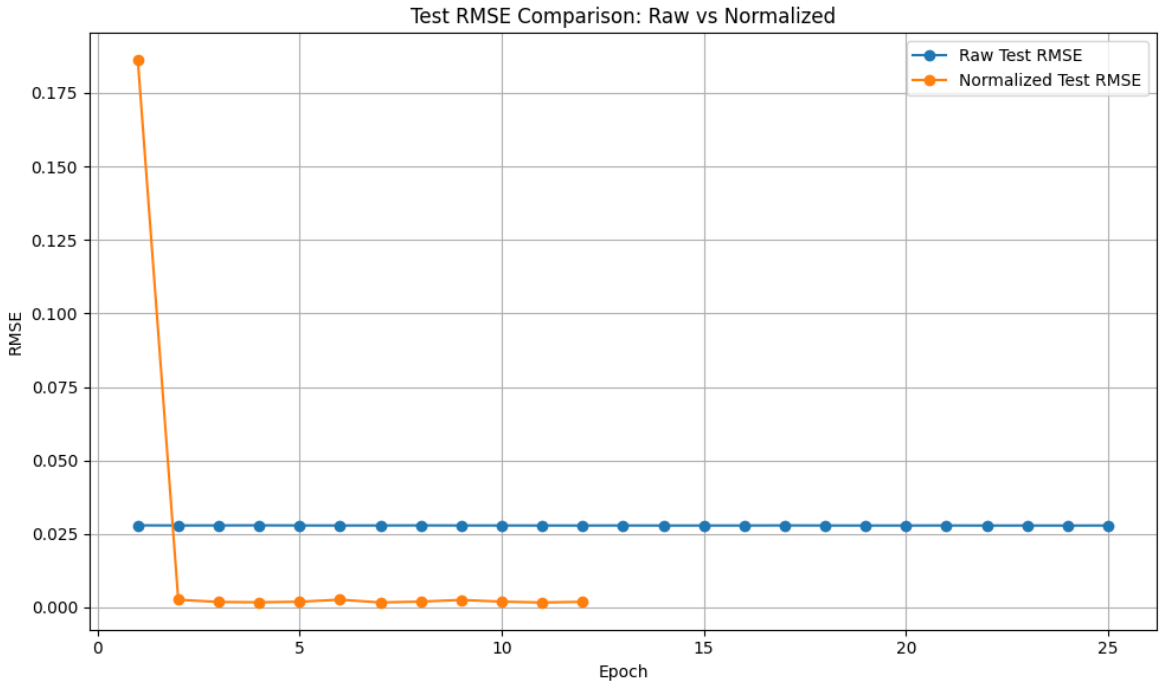


Fig. 6. Test RMSE Comparison Between Raw and Normalized Data

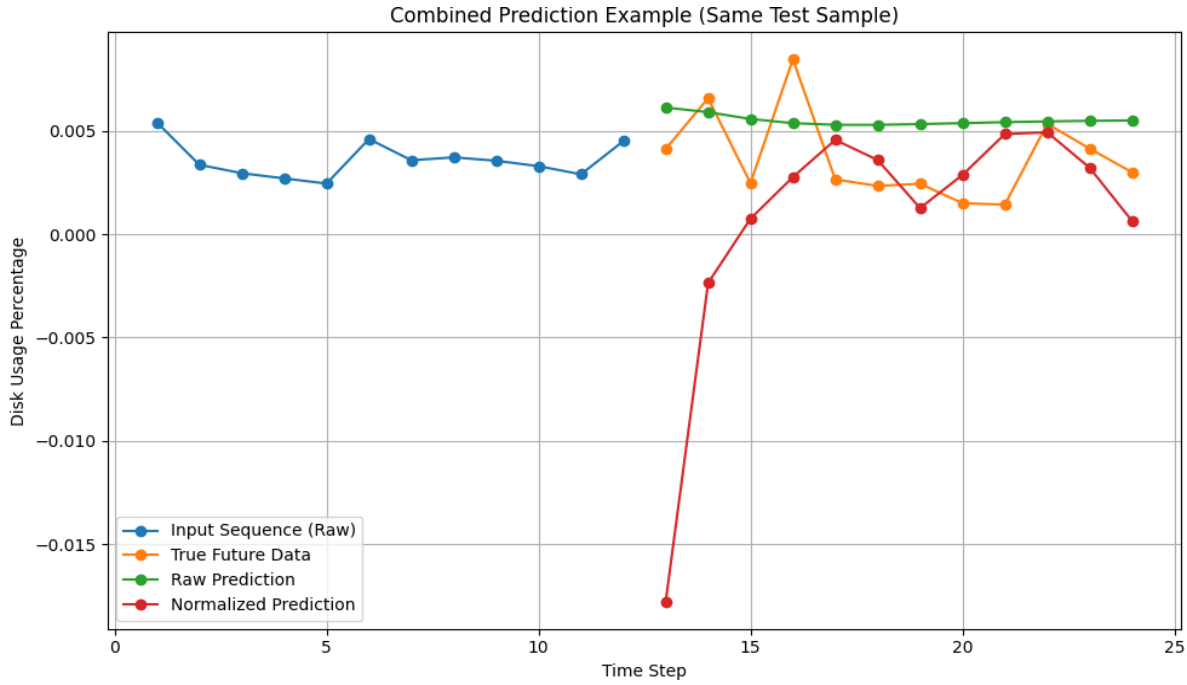


Fig. 7. Example of a comparison between predictions made using raw and normalized data.

6 Automation of Configuration File Generation

The software developed for SMART employs a Python-based graphical user interface (GUI), built using the PyQt5 framework, to automate the generation and management of XML configuration files. XML file configurations are central to SMART's operation, defining critical aspects of system monitoring. The primary goal of this automation is to significantly streamline the process of managing these XML configurations, thus reducing manual workload, minimizing human-induced errors, and addressing configuration complexity.
To achieve efficient management and rapid adaptability, the software uses JSON-based configuration files. This approach facilitates straightforward customization and flexibility in defining various XML structures required by SMART.
Initially, an attempt was made to automate log extraction and event monitoring using logs generated by the GEMS system. However, this method proved overly complex and prone to inaccuracies. As a result, the current implementation focuses instead on providing an intuitive graphical interface, enabling users to efficiently manage and validate XML configurations directly, ensuring accuracy, reliability, and simplicity.
Consequently, we shifted our approach towards leveraging the constant attributes consistently present in GMES logs, such as the facility identifier, event reception timestamp, satellite identifier, and similar metadata. Particularly relevant is the adherence to the WMO convention [9], a standardized naming format for file identification, which ensures consistency and facilitates accurate extraction of critical information for subsequent mapping by SMART.

6.1 System Architecture

The software consists of two interconnected components:

- Graphical User Interface (GUI) (Fig 8.): Implemented with PyQt5, it handles user interaction, data entry, input validation, dynamic updates, and real-time XML previews.

- Configuration Management Module: Manages JSON-defined templates specifying XML parameters, mappings, tokenizer configurations, and attribute details.

These components collaborate seamlessly, ensuring efficient, accurate, and user-friendly XML generation.

6.2 XML Generation Tab, configuration tab and XML Generation workflow

This tab, implemented via the GenerateXMLTab class, provides several input fields for defining parameters essential for XML generation. Users can input parameters to define Facility, Tracker, and Event configurations. When requesting an XML preview, the tool validates all inputs using QMessageBox, alerting users about any missing or inconsistent fields. Once validated, Python functions leverage the JSON configuration to dynamically generate XML elements.
The ConfigTab enables direct editing of the JSON configuration used for XML generation. Users can modify parameters directly, adjusting configurations to meet specific needs (Fig. 9). Changes to the JSON are validated, ensuring that configurations remain consistent and error-free.

The XML generation employs Python libraries, specifically xml.etree.ElementTree for XML element creation and xml.dom.minidom for formatting and indentation .

The workflow includes:

- Input Validation:
  Verifies all required fields, identifying missing or inconsistent inputs to ensure configuration integrity.
- Dynamic XML Creation:
  Constructs XML elements dynamically based on JSON configurations, including:
    - *Event parameters* and *Global parameters*, each defined with specific attributes.
    - *Mappings*, implementing parameter-specific rules using substrings and tokenizers, defined by delimiters and positions specified in JSON.

- XML Formatting (Fig. 9):
  Uses Python's XML libraries (xml.etree.ElementTree for element creation and xml.dom.minidom for formatting and indentation), ensuring readability and structured XML outputs.

- Preview Generation (Fig. 10):
  Provides an XML preview allowing users to review and confirm the correctness of configurations prior to deployment.

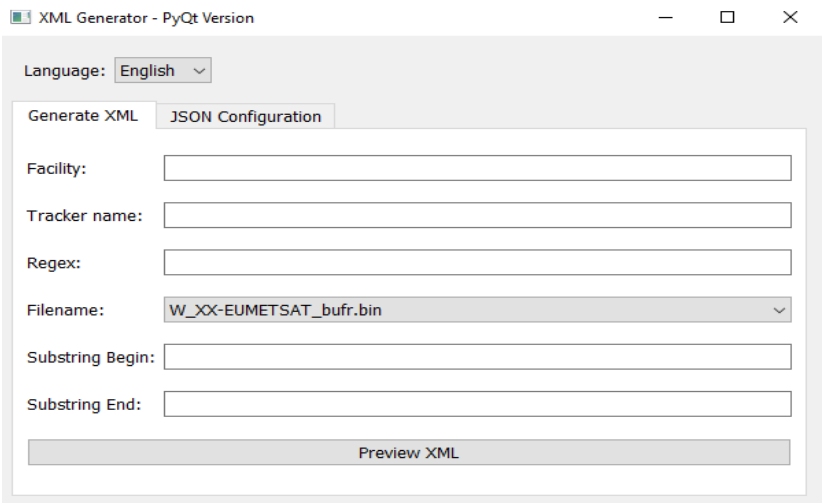


Fig. 8. Graphical User Interface (GUI) of the configuration XML automation software

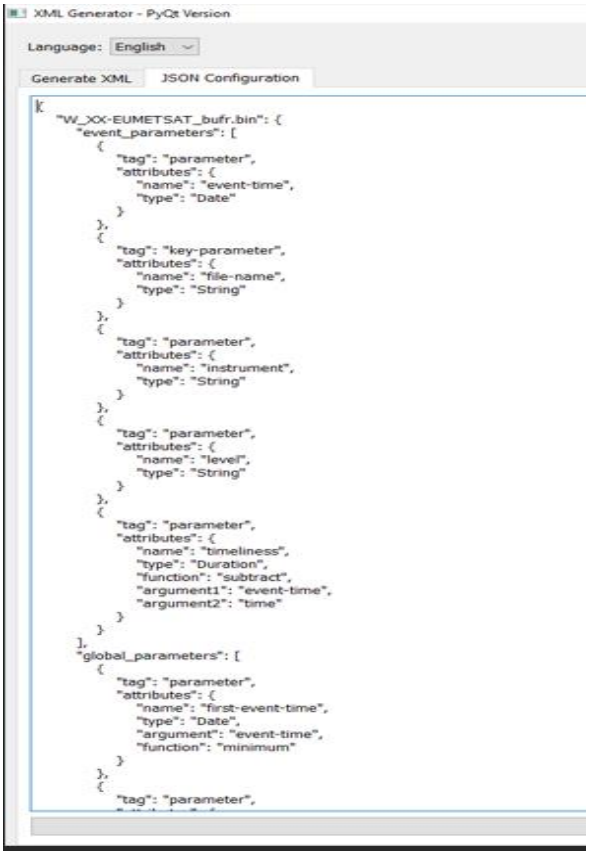


Fig. 9. JSON modification directly via software

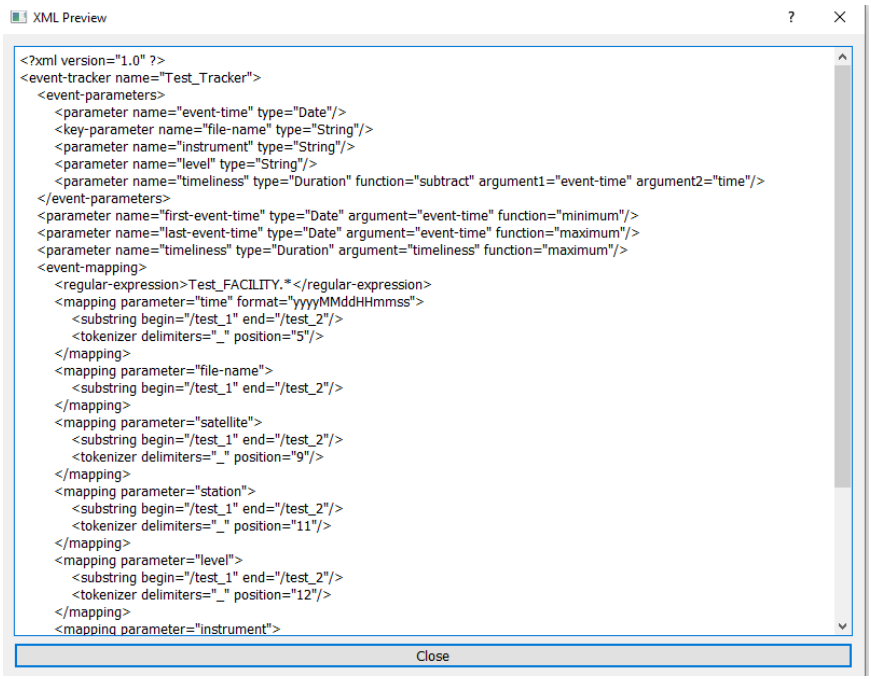


Fig. 10. XML output preview

6.3 **Artificial Intelligence Integration**

The AI-driven module leverages innovative prompt engineering techniques with large language models (LLMs) to automatically parse textual logs pasted directly by the user. Without manual parameter identification or configuration, the AI module intelligently extracts necessary parameters from the raw log input and constructs a complete, structured XML file.

A concrete example of such a prompt engineering approach is as follows:

Given the following log entry (example):

"24.193.10.00.00.000 RDS_EARS_VAL earsvalserver.eumetsat.int DID INFO lmc.log:[2024-07-11 11:00:00] I :/EARS_PDP/NOAA19_AVHRR_TO_EUMETCAST_PreProc.xml File download/upload completed : dataFlow=NOAA19_AVHRR_TO_EUMETCAST file=W_XX-EUMETSAT_avhrr_20240711_xxxxx_noaa19.hrp.bz2 startTime=2024-07-11 11:00:00 endTime=2024-07-11 11:40:00 timeElapsed=4",

Extract the following parameters and generate a structured XML file:
- Event Time
- Data Flow Identifier
- Filename
- Start Time
- End Time
- Elapsed Time

The LLM efficiently interprets this log entry, accurately extracting all listed parameters and automatically generating the corresponding XML file. This capability dramatically simplifies the workflow, significantly reducing user effort and potential human errors.

6.4 **Risks and Limitation**

The integration of AI into technical processes such as XML data management and predictive analytics for time series cam significantly enhanced operational efficiency. However, it simultaneously introduces new vulnerabilities that require careful consideration. In XML automation, minor coding or configuration errors can rapidly proliferate unintended and potentially harmful modifications, reducing data integrity and operational reliability. Furthermore, excessive reliance on automation may diminish necessary human oversight, leading to unaddressed errors becoming systemic. Security risks related to system accessibility and permissions may also expose organizations to external cyber threats.
In time series forecasting, predictive models face risks such as uncertainty in forecasts, especially when anomalies or previously unobserved events occur, potentially resulting in flawed decision-making. Overfitting to historical data can severely limit a model's generalization capabilities, compromising its effectiveness in new scenarios. Moreover, biases arising from incomplete or unrepresentative training data can systematically distort predictions. Finally, significant data gaps in time series, if inadequately managed, can considerably reduce forecast accuracy, particularly during critical events.
To mitigate these risks, implementing routine human validation of automated outputs, continuously monitoring AI tool performance, and periodically updating predictive models with new data are recommended practices.

7 Future work

We have several upcoming projects aimed at developing and integrating the tools presented in this paper. Future work includes the integration of additional metrics that are currently being tested, such as CPU usage. The goal of these tests is twofold: firstly, to predict multiple variables concurrently to enhance system efficiency and performance, and secondly, to understand the correlations between these variables. By identifying these correlations, we aim to leverage multiple inputs to predict the behaviour of individual variables more accurately. Additionally, we plan to expand our research to include reinforcement learning methods, which will help us uncover previously unknown correlations between variables. Our intention is to integrate the AI system directly into our operating environment, where predictive thresholds can trigger automated alarms or notifications when forecasts surpass predefined limits.
Future developments will further enhance AI capabilities through advanced techniques such as fine-tuning transformer-based models and integrating natural language processing (NLP) for improved log interpretation. Additional planned enhancements include integration with version control systems (e.g., Git), robust schema validation, and comprehensive logging mechanisms to ensure better traceability and easier maintenance.

8 Conclusion

In conclusion, the automated XML management and predictive forecasting tools developed for EUMETSAT operations have substantially enhanced monitoring capabilities. The XML automation software streamlines configuration management, minimizing manual effort and mitigating errors while aligning with recognized standards (e.g., WMO). This approach ensures consistent file identification and reliable data mapping throughout the system.
Meanwhile, the forecasting tool -driven by artificial intelligence - delivers valuable insights through precise predictions of operational metrics such as disk space consumption, thereby enabling proactive resource allocation and bolstering overall system stability. Continuously retraining the predictive model with up-to-date data preserves both accuracy and relevance over time.
Ongoing improvements aim to incorporate reinforcement learning techniques for deeper analysis of variable correlations, transformer-based model fine-tuning, and NLP functionalities for advanced log analysis. In addition, the planned integration of version control systems, extended schema validation, and comprehensive logging will further enhance maintainability, traceability, and operational resilience.
We, in the mission performance team, strongly believe that investing in AI will yield more efficient real-time operations, reducing both turnaround time and personnel demands, ultimately fostering greater productivity and effectiveness at EUMETSAT.

References

[1] Edwards, Tristan, & Sierra Urueña, V. (2023): Integration of Multi-Mission Services into Operations – The Appeal and Reality. *SpaceOps-2023, ID #432* [Online]
[2] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.
[3] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a Convolutional Neural Network. In 2017 International Conference on Engineering and Technology (ICET) (pp. 1-6). IEEE.
[4] Idelbayev, Y. (2018). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. arXiv preprint arXiv:1808.03314.
[5] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv preprint arXiv:1412.3555.
[6] Jung, P. (2009). Munin—the Raven Reports. https://www.linuxjournal.com/article/10248

[7] Huang, L., Qin, J., Zhou, Y., Zhu, F., Liu, L., & Shao, L. (2020). Normalization Techniques in Training DNNs: Methodology, Analysis and Application. *arXiv preprint arXiv:2009.12836*.
[8] Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1412.6980
[9] National Weather Service. (n.d.). *File Push Service*. Retrieved March 17, 2025, from https://www.weather.gov/tg/filpush
[10] Gianni Casonato, Ruth Britton, Luca Garegnani, EUMESTAT Machine Learning Framework System and AI/ML Applications, SpaceOps-2025, ID # 80 [Online]