RESEARCH PROJECT REPORT
Anomaly Detection on Flight Simulator

Supervisors:
Mr. Joël Potin
Mr. Florian Montels

Students:
Phuc Luan Nguyen
Thibault Lauilhé

Department:
Mathematical Modeling and AI

Academic Year: 2024 - 2025

Contents

Introduction

Anomaly detection is a fundamental field in data science, aimed at identifying unusual behavior or events within a dataset. Whether it involves preventing banking fraud, detecting failures in an industrial production line, or identifying signs of illness from medical data, this discipline plays a central role in key sectors. Today, anomaly detection methods and models can handle high-dimensional data, often in real-time, to identify any irregularities.

In the aviation sector, anomaly detection has become a major challenge for ensuring the safety and efficiency of operations. During every flight, from preparation to piloting and maintenance, a significant amount of data is generated. When properly analyzed, this information not only helps anticipate technical failures but also detects abnormal behavior that could compromise flight safety and smooth operations. Flight simulators, which have become essential tools for pilot training and testing new procedures, provide a realistic framework for observing, recording, and analyzing "aircraft" behavior in response to given inputs.

The objective of this project is to develop a robust method for detecting anomalies within a flight simulator. The goal is to distinguish normal behavior from potentially dangerous or faulty situations, thereby enhancing the safety and reliability of simulation systems.

This work is being conducted in partnership with SII Toulouse, a company specializing in advanced engineering systems. This collaboration combines technical expertise with industry knowledge.

In this report, we will cover the various stages of the study, from data collection and preparation to the evaluation of detection methods, while addressing the specific challenges posed by the aviation context.

1 State of the art

1.1 State of the art in anomaly detection

As mentioned above, anomaly detection is very important in the aeronautical sector, particularly with the introduction of flight simulators to ensure the safety and smooth running of real flights. In this section, we will give a state-of-the-art overview of existing Data Science models for anomaly detection. This will enable us to choose one or more models to implement and train for our study, depending on the constraints imposed by our dataset.

In the context of a flight simulator, a multivariate time series is a set of data collected over time that includes several variables measured simultaneously. Each variable represents a relevant characteristic or parameter of the flight, such as position, speed, altitude, orientation, and various pilot commands.

In a flight simulator, data are not necessarily independent. For instance, pilot commands can interact with each other (e.g., a change in speed might lead to a change in altitude). Moreover, multivariate time series can have very high dimensions, with more than 50 pilot commands being captured at every instant.

1.1.1 Major Categories of Anomaly Detection Approaches

Anomaly detection approaches for time series data can be grouped into several major categories [4]:

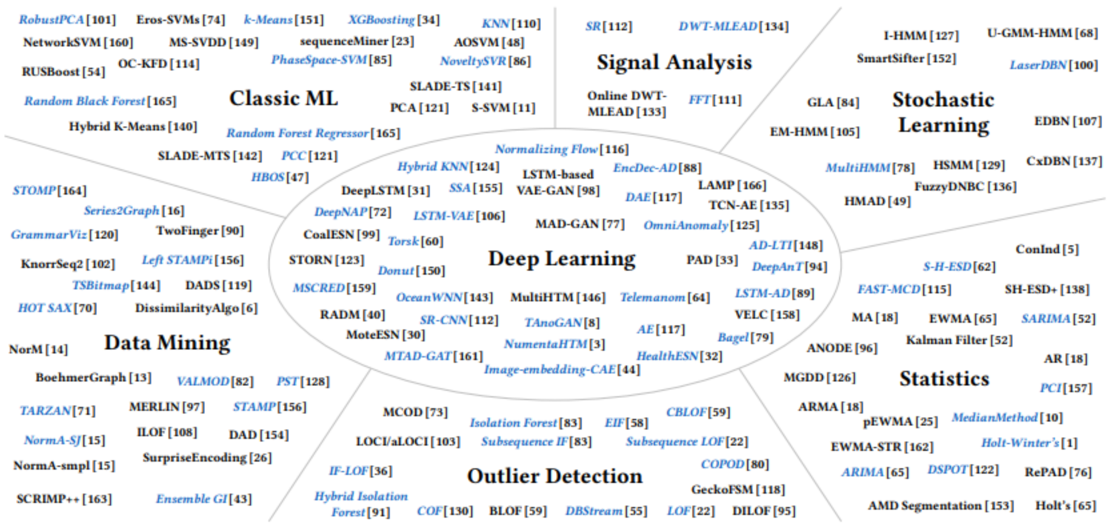


Figure 1: Overview of different approaches for anomaly detection

- **Prediction-Based Methods:** These methods rely on predictive models capable of forecasting future values of a time series. An anomaly is detected when the difference between observed and predicted values exceeds a threshold. Techniques like ARIMA and Long Short-Term Memory (LSTM) networks are commonly used in this category.

- **Reconstruction-Based Methods:** These methods build a model of normal behavior by learning to reconstruct subsequences of a normal time series. Anomalies are identified by high reconstruction errors. Autoencoders, recurrent neural networks (RNNs), and LSTM-Encoder-Decoder models fall into this category [1].

- **Distance-Based Methods:** These techniques calculate similarity measures between subsequences of the time series. Anomalies are characterized by abnormally high distances compared to normal subsequences. Methods such as LOF (Local Outlier Factor) or K-Nearest Neighbors (KNN) are often used.

- **Distribution-Based Methods:** These methods estimate the statistical distribution of the data. Points located at the extremes of the distribution are considered anomalies. Approaches such as Isolation Forest and the S-H-ESD (Seasonal Hybrid Extreme Studentized Deviate) method belong to this category.
- **Encoding-Based Methods:** These methods use encoding techniques to reduce the dimensionality of subsequences of the series. Anomalies are identified directly in the encoded space based on anomaly scores derived from latent representations.

1.2 The Interest of Autoencoders for Anomaly Detection in Flight Simulators

In the context of flight simulators, multivariate time series exhibit high complexity due to the numerous parameters and the variability of commands given by the pilot. Autoencoders, particularly LSTM-Encoder-Decoder models, are well-suited for such data as they can capture and reconstruct the complex temporal dynamics observed during normal situations.

Using an autoencoder allows us to model only normal behavior without requiring specific anomaly data [1]. Thus, when an abnormal sequence occurs, it results in a high reconstruction error, which can be detected as an anomaly. This approach is ideal when anomalies are rare or difficult to represent in the training data, as is often the case in flight simulations.

However, we have noticed that anomalies appear in a very localized way around a certain time step. LSTM auto-encoders will therefore have to respond to this punctual problem. At this stage of the study, we don't know how the LSTM model will react to the reconstruction of this abnormal peak.

So, the LSTM AutoEncoder will be a relatively simple first approach, which we'll be forcing ourselves to make more complex in the rest of the study by crossing this model with other architectures in order to add value to training and reconstruction. Models such as UNet and Transformers could then be studied.

1.3 Relevance of UNet-LSTM models for anomaly detection

The UNet-LSTM model combines the strengths of the UNet and LSTM architectures. Thanks to its skip-type connections, it limits information loss during sequence reconstruction [3]. These features are particularly useful for analyzing multivariate time series, where anomalies manifest themselves as subtle or localized deviations. Its application in the context of flight simulators makes it possible to efficiently capture temporal dynamics while preserving local details, thus improving anomaly detection.

1.4 Relevance of Transformers for anomaly detection

Transformer models, thanks to their attention mechanism, can capture long-term dependencies while facilitating parallel data processing [5]. This capability is essential for the long and complex time series generated by flight simulators. In addition, Transformers are recognized for their efficiency in handling large datasets and extracting global trends, making them a promising choice for reconstruction and anomaly identification in this field.

2 Methodology

2.1 Data Preparation

Our dataset consists of simulations in the form of multivariate time series, including 949 normal simulations and 36 abnormal simulations. However, the first challenge we encountered was the slight variation in the number of time steps across simulations. To address this, we adjusted all time series to have the same length, ensuring they could be used as input to the model.

Next, we observed that normal simulations contain 135 variables, whereas abnormal simulations have 137 variables. This difference arises because abnormal simulations include two additional ground truth variables that describe events expected to occur. To ensure consistency, we removed these two variables from the abnormal simulations, so the anomaly data also contains 135 variables and can be used as input for the model. Additionally, we retained the two ground truth variables to compare the reconstructed values, anomalous values, and ground truth values, allowing us to evaluate whether the reconstructed values for anomalous simulations align with the ground truth.

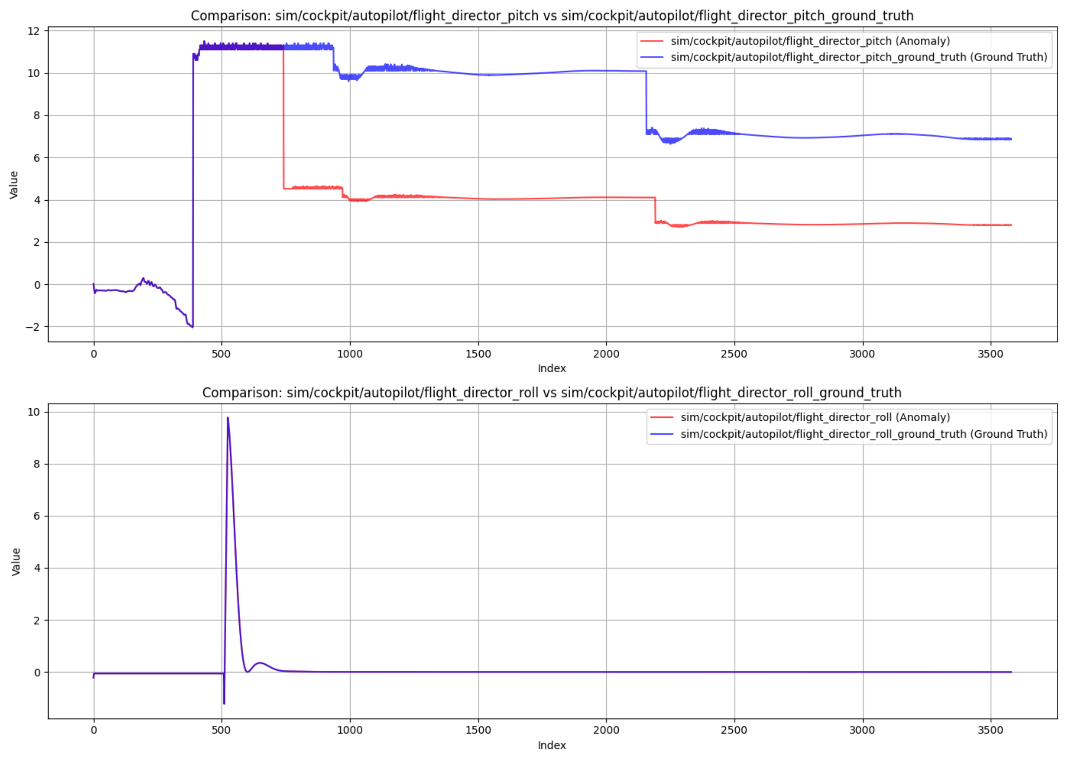
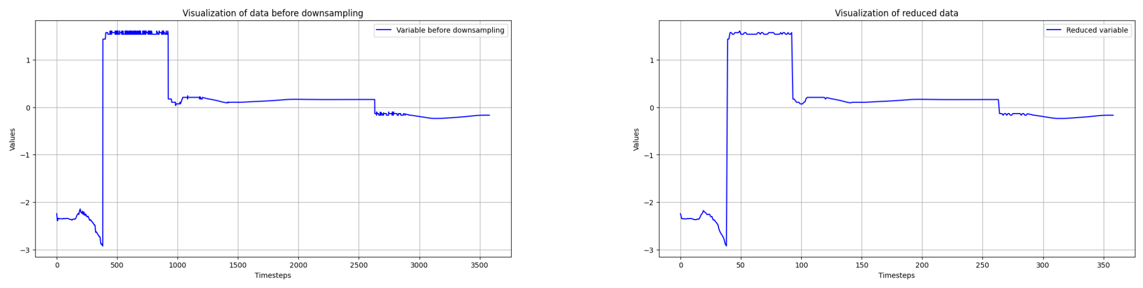


Figure 2: The difference between ground truth variables and anomalous variables

Furthermore, since the autoencoder model we used is designed to train only on normal data, its purpose is to capture the characteristics of normal simulations. This enables it to produce a significant reconstruction error when presented with anomalous simulations. To this end, we split the dataset of normal simulations into two parts: 80% for training (`train_normal`) and 20% for testing (`test_normal`). The anomalous simulations were used exclusively for testing.

Then, we normalized the data using `StandardScaler`. This choice was made because, in the presence of anomalies, some variables may exhibit abnormally high values, causing the maximum value to become excessively large. For this reason, `MinMaxScaler` was deemed unsuitable for this problem.

Finally, because our original dataset consists of multivariate time series with 3,581 time steps and 135 variables, the input data size was substantial, making training computationally expensive. To address this, we reduced the time steps by a factor of 10, resulting in 359 time steps per simulation. Importantly, this reduction preserved the general characteristics of the variables over time.



(a) Before downsampling (b) After downsampling

Figure 3: Comparison of time series data before and after downsampling.

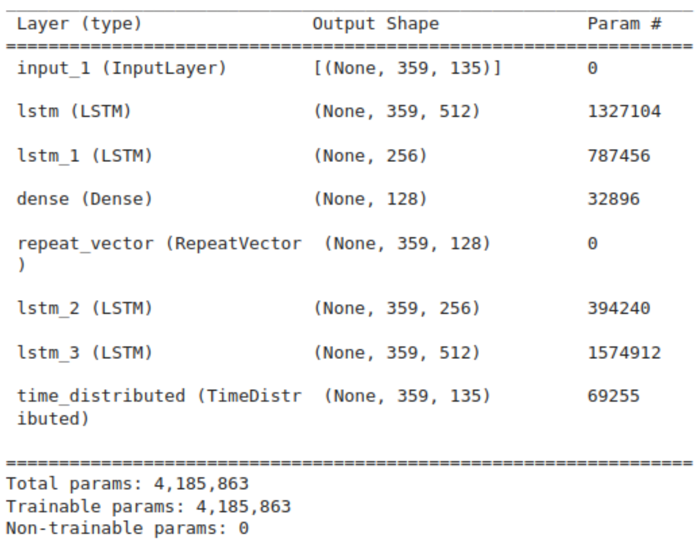
Our final dataset consisted of:

- **Training data**: 759 simulations with 359 time steps and 135 variables
- **Normal test data**: 190 simulations with 359 time steps and 135 variables
- **Anomaly data**: 36 simulations with 359 time steps and 135 variables

2.2 LSTM Autoencoder

2.2.1 Architecture and Parameters

Initially, based on the literature we reviewed and discussions with our tutors, we decided to experiment with an LSTM Autoencoder model structured as follows:



```
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 359, 135)]        0

 lstm (LSTM)                 (None, 359, 512)          1327104

 lstm_1 (LSTM)               (None, 256)               787456

 dense (Dense)               (None, 128)               32896

 repeat_vector (RepeatVector  (None, 359, 128)         0
 )

 lstm_2 (LSTM)               (None, 359, 256)          394240

 lstm_3 (LSTM)               (None, 359, 512)          1574912

 time_distributed (TimeDistr  (None, 359, 135)         69255
 ibuted)

=================================================================
Total params: 4,185,863
Trainable params: 4,185,863
Non-trainable params: 0
```

Figure 4: Architecture of the LSTM Autoencoder model

Input layer: The input data consists of multivariate time series with dimensions corresponding to the number of time steps and features.

Encoder:

- The first LSTM layer contains 512 units with `tanh` activation and `return_sequences=True`, allowing it to extract temporal features across the sequence.
- The second LSTM layer contains 256 units with `tanh` activation and `return_sequences=False`, which compresses the temporal information into a single vector.

Bottleneck (latent space): A dense layer with 128 units and `ReLU` activation is used to capture a compressed latent representation of the input data.

Decoder:

- The `RepeatVector` layer replicates the latent representation for each time step in the sequence.
- The first LSTM layer in the decoder contains 256 units with `tanh` activation and `return_sequences=True`.
- The second LSTM layer contains 512 units with `tanh` activation and `return_sequences=True`, fully reconstructing the sequence.

Output layer: A layer reconstructs the original sequence with the same number of features for each time step.

The activation functions used in the encoder, bottleneck, and decoder layers of the LSTM Autoencoder were chosen to balance effective feature extraction, non-linearity, and stability:

- Encoder (tanh): The `tanh` activation is used in the encoder layers because it captures temporal dependencies effectively while mapping the input to a range of $(-1, 1)$. This zero-centered output aids in faster convergence and better gradient flow when processing sequential data.
- Bottleneck (ReLU): The `ReLU` activation is employed in the bottleneck layer as it provides efficient and sparse representations by outputting zero for negative inputs. This is particularly useful for reducing redundancy in the latent space and emphasizing critical features.
- Decoder (tanh): The `tanh` activation is also used in the decoder layers to ensure the reconstructed sequence remains bounded and stable within the $(-1, 1)$ range, aligning well with the output of the encoder.
- Output Layer (Linear): The final layer does not use an explicit activation function to directly reconstruct the input features without additional scaling or transformations.

This combination of activation functions ensures effective encoding of temporal features, meaningful latent representations, and stable reconstruction of sequential data.

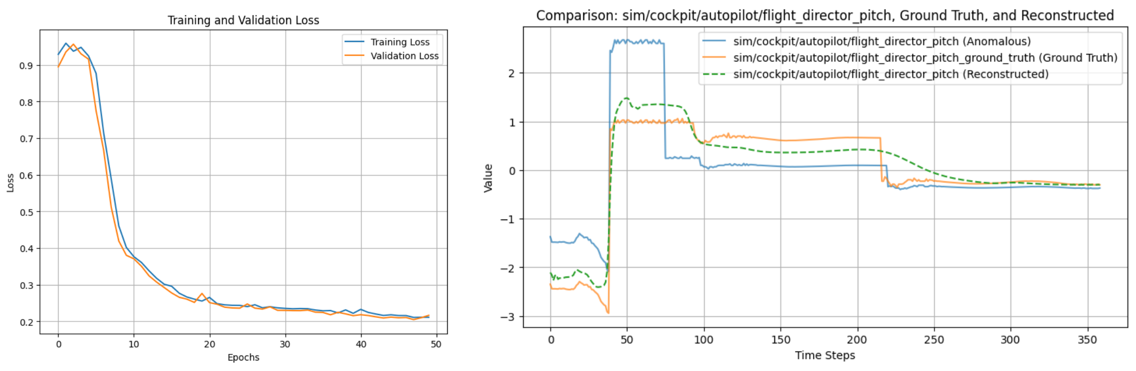
2.2.2 Model Training

The model was trained using only normal simulation data to learn its reconstruction. The reconstruction error is calculated as the difference between the original and reconstructed sequences. Higher reconstruction errors during testing indicate that the input data likely contains anomalies. The following configuration was used during training:

- **Loss function**: Mean Squared Error (MSE)
- **Optimizer**: Adam with a learning rate of 0.001
- **Batch size**: 32
- **Number of epochs**: 50

The loss function used for training is the Mean Squared Error (MSE). This metric measures the mean squared error between actual and predicted values. It heavily penalises large errors, making it a suitable metric for problems such as sequence reconstruction.

After the training process, we obtained the loss chart and the reconstructed data for the 15th anomalous simulation, as shown in the figure below.



(a) Training and validation loss

(b) Compare the reconstructed data with other datasets.

Figure 5: LSTM Autoencoder.

We observed that the loss value converged, and the reconstructed data described the ground truth relatively well. This indicates that the combination of LSTM and Autoencoder is capable of reconstructing multivariate time series data.

However, despite this relatively good performance, the reconstruction still appears somewhat simplistic, resulting in noticeable discrepancies between the reconstructed data and the ground truth. To address this limitation, we drew inspiration from the combination of Autoencoder, CNN, and the U-Net architecture, which we had previously implemented for image processing in our class. Building on this idea, we experimented with integrating Autoencoder, LSTM, and the U-Net architecture to reconstruct multivariate time series data for our project.

2.3 LSTM UNet Autoencoder

2.3.1 Principle of U-Net and its relevance to our study

The U-Net model is built on a symmetrical encoder-decoder architecture with *skip connections* that link corresponding layers of the encoder and decoder. These connections enable effective communication between equivalent layers in the encoder and decoder. Consequently, the decoder is guided by the encoder during reconstruction, enhancing its precision and ensuring detailed recovery of the data, following the U-Net principle.

The integration of LSTM and U-Net combines the strengths of both approaches: capturing temporal dynamics with LSTM and preserving essential information through U-Net's skip connections. This synergy minimizes information loss and facilitates anomaly detection by highlighting discrepancies between the reconstructed and actual data. Thus, this architecture is particularly well-suited for analyzing complex datasets, such as multivariate time series.

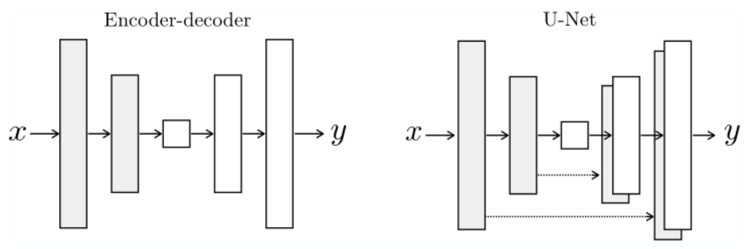
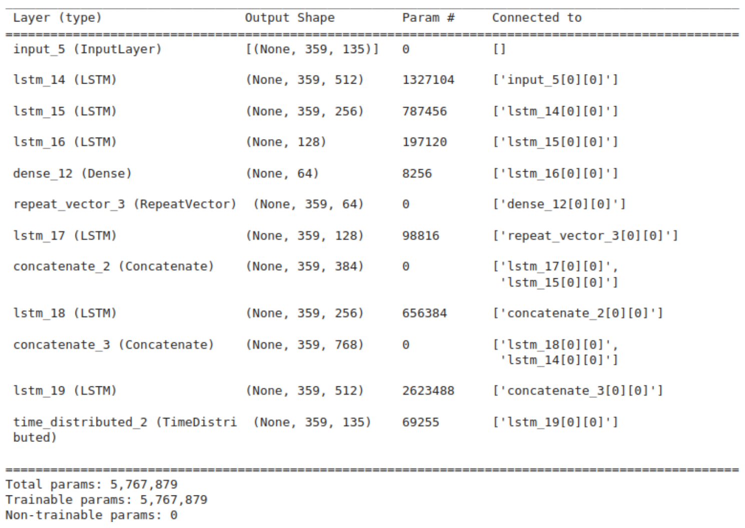


Figure 6: Autoencoder model and Unet Autoencoder model

2.3.2 Model Architecture and Training

We trained an LSTM-UNet model, which combines the principles of LSTM-based networks with the U-Net approach. The following are the key steps of the architecture used:



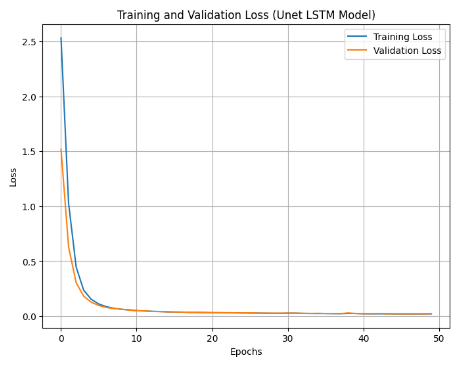
```
Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)           [(None, 359, 135)]   0           []

lstm_14 (LSTM)                 (None, 359, 512)     1327104     ['input_5[0][0]']

lstm_15 (LSTM)                 (None, 359, 256)     787456      ['lstm_14[0][0]']

lstm_16 (LSTM)                 (None, 128)          197120      ['lstm_15[0][0]']

dense_12 (Dense)               (None, 64)           8256        ['lstm_16[0][0]']

repeat_vector_3 (RepeatVector)  (None, 359, 64)     0           ['dense_12[0][0]']

lstm_17 (LSTM)                 (None, 359, 128)     98816       ['repeat_vector_3[0][0]']

concatenate_2 (Concatenate)    (None, 359, 384)     0           ['lstm_17[0][0]',
                                                                 'lstm_15[0][0]']

lstm_18 (LSTM)                 (None, 359, 256)     656384      ['concatenate_2[0][0]']

concatenate_3 (Concatenate)    (None, 359, 768)     0           ['lstm_18[0][0]',
                                                                 'lstm_14[0][0]']

lstm_19 (LSTM)                 (None, 359, 512)     2623488     ['concatenate_3[0][0]']

time_distributed_2 (TimeDistri  (None, 359, 135)    69255       ['lstm_19[0][0]']
buted)

==================================================================================================
Total params: 5,767,879
Trainable params: 5,767,879
Non-trainable params: 0
```

Figure 7: Architecture of the LSTM-UNet Autoencoder Model

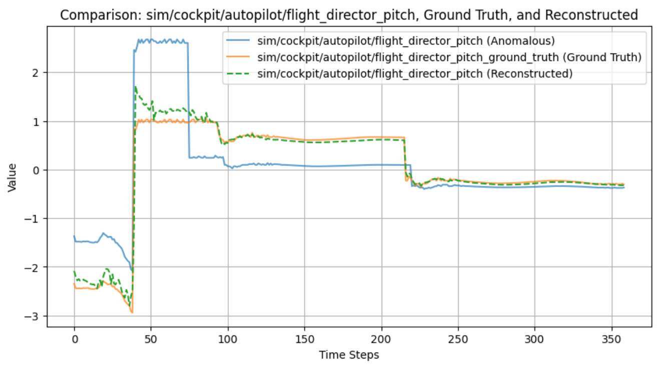
In this model, the architecture fundamentally follows the autoencoder structure, similar to the previous model. However, the unique aspect lies in the decoder layers, which, in addition to being connected sequentially to the preceding layers, also incorporate skip connections with their corresponding encoder layers. These skip connections enhance the reconstruction process by increasing the level of detail.

Regarding activation functions and training hyperparameters, we used the same configuration as in the LSTM Autoencoder model described earlier.

After the training process, we obtained the loss chart and the reconstructed data for the 15th anomalous simulation, as shown in the figure below.



(a) Training and validation loss



(b) Compare the reconstructed data with other datasets.

Figure 8: LSTM Unet Autoencoder.

We observed that the loss function has converged, and the reconstruction in this model is nearly identical to the ground truth data. A detailed comparison between the models will be conducted in the subsequent sections.

During experimentation, we reflected on the increasing use of Transformer models for processing sequential data, particularly in large language models. Since our data is in the form of time series, which is also a type of sequential data, it raises the question: can a Transformer combined with an autoencoder be utilized to reconstruct time series data effectively?

2.4 Transformer Autoencoder

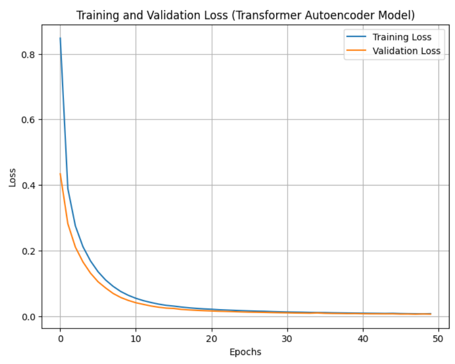
2.4.1 Principle of Transformer and its relevance to our study

Since our data is sequential, we considered whether it would be feasible to use a Transformer to address the problem. If applicable, we could leverage the strengths of the Transformer model, particularly its *self-attention* mechanism. For example, Transformers enable parallel processing, which accelerates model training and is advantageous for handling large datasets. Additionally, Transformers effectively capture long-range dependencies, a feature that LSTM models also aim to address through the use of various gates. However, the *self-attention* mechanism allows Transformers to handle this challenge more efficiently.

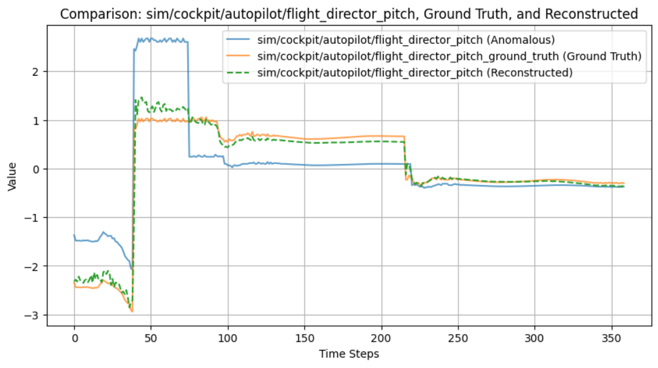
2.4.2 Model Architecture and Training

Due to the time constraints of experimenting with this model, we did not conduct an in-depth analysis. Therefore, for this model, we primarily focused on testing whether training could be successfully performed and whether the reconstruction results were satisfactory.

After the training process, we obtained the loss chart and the reconstructed data for the 15th anomalous simulation, as shown in the figure below.



(a) Training and validation loss



(b) Compare the reconstructed data with other datasets.

Figure 9: Transformer Autoencoder.

We observed that the loss function converged, and the reconstructed data accurately described the ground truth. Additionally, during training, we found that the Transformer model was significantly faster compared to the previous models when using the same hyperparameters. A detailed analysis of the results will be provided in the subsequent sections.

3 Results

To compare the three models, we first evaluate the quality of data reconstruction from the anomalous data. Among the two variables representing the anomalous data, *roll* and *pitch*, we chose the *pitch* variable for presentation purposes to illustrate the reconstruction characteristics of the models.

Next, we compare the three models based on the average MSE error calculated from the normal simulation data used for testing. Finally, we analyze the training time required for each model to assess their computational efficiency.

3.1 Results for LSTM Autoencoder

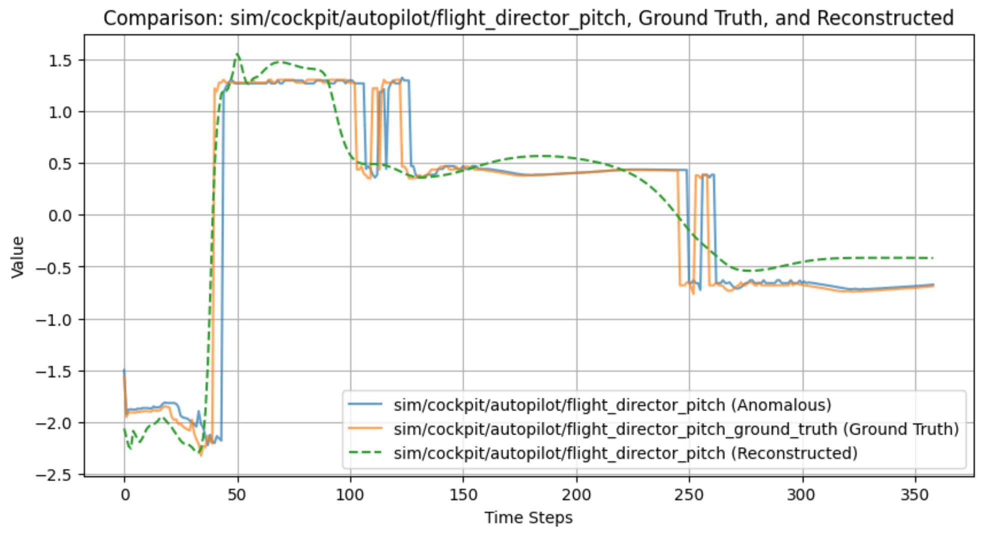


Figure 10: Reconstruction by LSTM Autoencoder

3.1.1 Key Phase Analysis

As we can see from Figure 10, we can analyze the model's ability to reconstruct in several phases (over timesteps)

Initial phase ($t < 50$) The reconstruction (green line) broadly follows the ground truth (orange line) with a certain phase shift that is not really significant.

Critical phase around $t \approx 50$ A sharp transition is visible in the ground truth. Although the anomalous data seem to follow this transition, they show significant deviations, highlighting the presence of anomalies. The reconstruction attempts to approximate the ground truth, but discrepancies appear, which could indicate a difficulty in the model's ability to capture these complex dynamics.

Perdition of mode ($t > 50$) After timestep 50 (critical phase), the LSTM model seems to get lost in the reconstruction. As a result, the decoder part of LSTM has difficulty reconstructing the sequence due to a lack of information. We notice that it reconstructs an oscillating sequence with a decreasing character. But at no point does it capture the local ground-truth dynamics at timesteps 120 and 250.

3.1.2 Discussion

This initial model demonstrated that using an LSTM Autoencoder could partially reconstruct our data. However, a more complex and optimized model is required to achieve more accurate data reconstruction.

3.1.3 Prospects for Improvement

To enhance the model's performance, we could have considered the following avenues:

- Optimizing hyperparameters: Due to the limited computational power of personal computer that we use, it was not feasible to perform extensive hyperparameter optimization. With access to more powerful hardware, we could use grid search to identify the optimal hyperparameters, such as LSTM layer size and learning rate.
- Experimenting with scaling methods: While we justified the choice between MinMaxScaler and StandardScaler based on theoretical considerations, it would be worthwhile to experiment with other scaling methods. This could help determine whether alternative preprocessing techniques improve the quality of input data and lead to more accurate data reconstruction.

3.2 Results for LSTM-UNet Autoencoder

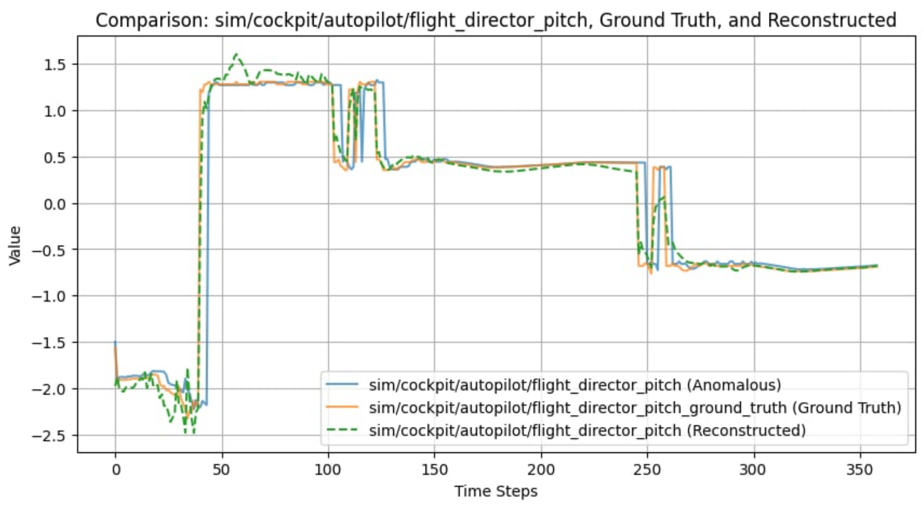


Figure 11: Reconstruction by LSTM-UNet Autoencoder

3.2.1 Key Phase Analysis

As shown in Figure 11, we can analyze the model's ability to reconstruct in several phases (over the timesteps)

Initial phase ($t < 50$) The reconstruction (green line) follows the ground truth (orange line) and captures subtle dynamics. The model is able to reconstruct very short variations, even if it causes the sequence to oscillate.

Critical phase around $t \approx 50$ This time, the abrupt ground-truth transition is better handled by UNet-LSTM. We can see that UNet has succeeded in attenuating the oscillating effects of the LSTM model, as the reconstructed curve stabilizes and gradually comes back into phase with the ground truth.

Stable phase ($t > 50$) After timestep 50 (the critical phase), the sequence reconstructed by the UNet-LSTM model remains broadly in phase with the ground truth, although the anomalous sequence appears slightly out of phase. Note also that for $120 < t < 250$ the reconstructed sequence follows the curvature of the ground truth, although this is a minor detail, demonstrating the model's ability to capture fine, local dynamics.

3.2.2 Discussion

Figure 11 confirms our choice of combining a UNet architecture to guide the LSTM model in reconstruction. The detailed sequence is much more controlled and focuses on precise, local variations.

3.2.3 Prospects for Improvement

Similar to the methods for LSTM Autoencoder

3.3 Results for Transformers

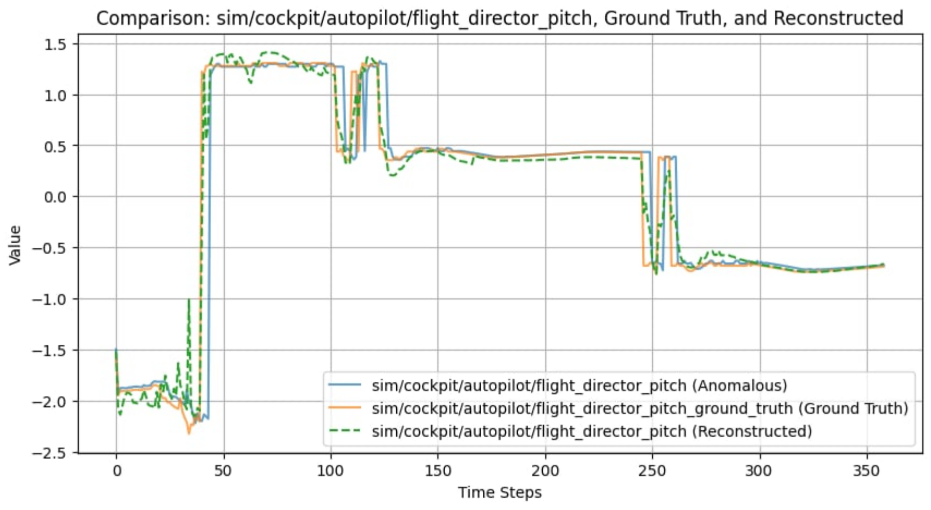


Figure 12: Reconstruction by Transformers

3.3.1 Key Phase Analysis

Initial phase ($t < 50$) During this phase, the Transformers model captures the general pattern of ground truth. However, it amplifies small variations, bringing out noticeable oscillations. Note that the model amplifies these variations all the more when they occur over a short period. However, unlike the simple LSTM model, the Transformers model is able to stabilize and bring itself back into line with the ground truth as soon as a "plateau" appears.

Critical phase around $t \approx 50$ The study around the critical phase remains the same: the model captures the general trend but does not dwell on local details, and makes the sequence slightly oscillating, although always more or less in phase with the ground truth.

Stable phase ($t > 50$) After the critical phase, the analysis leads us to the same conclusions as for the previous stages.

3.3.2 Discussion

The results obtained show that Transformers are effective for sequence reconstruction. The model gives a very reliable overall ground-truth reconstruction, although oscillating in places, particularly in critical areas. This raises the question of how to modulate the Transformers architecture to better manage these complex and dynamic sequences of variations.

3.3.3 Prospects for Improvement

To enhance the model's performance, we could have considered the following avenues:

- Fine-tuning the model: Adjust Transformer hyperparameters, notably the size of attention layers and regularization mechanisms.
- Training data enrichment: Since this model can be trained quickly and can handle large datasets, we can augment it with additional data that we collect to develop a more accurate and robust model.

3.4 Comparison of Average MSE on Normal Simulation Test Data

We observed that the reconstruction error progressively decreases from the LSTM Autoencoder to the LSTM-Unet Autoencoder and finally to the Transformer Autoencoder. This indicates that the Transformer Autoencoder provides the best reconstruction quality among the three models. Furthermore, the low errors in the latter two models demonstrate their ability to reconstruct the data effectively.

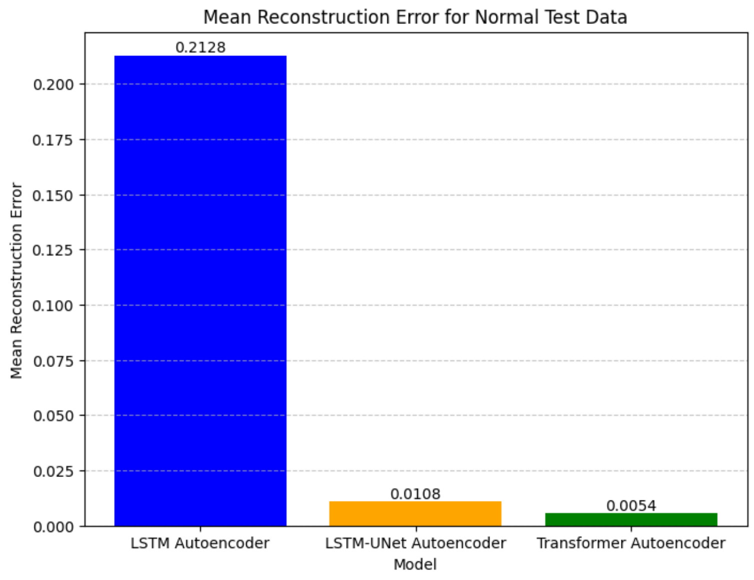


Figure 13: Mean Reconstruction Error for Normal Test Data

3.5 Comparison of Training Time

We observed that the Transformer model trains faster than the other models. This is expected because, with the self-attention mechanism, the Transformer can perform parallel computations efficiently. In contrast, for LSTM-based models, each node depends on the output of the previous node, requiring sequential processing. Additionally, the LSTM-Unet model, with its integration of the Unet structure, is more complex, resulting in longer training times compared to the initial LSTM Autoencoder model.

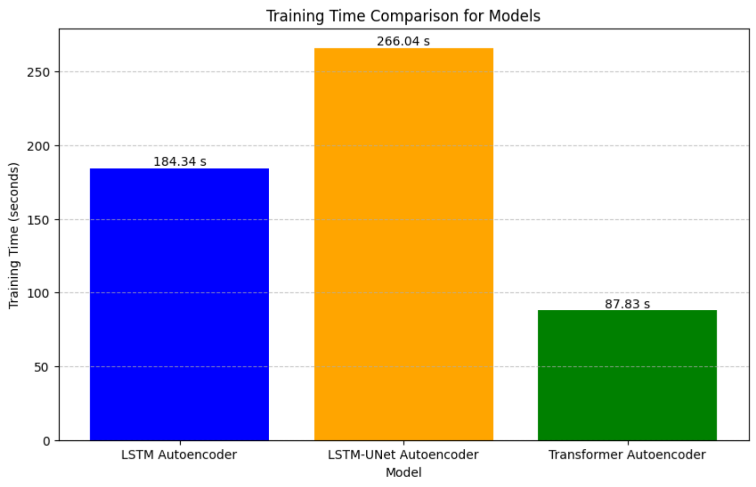


Figure 14: Training Time Comparison for Models

3.6 Detecting Simulation Anomalies

The final step is to use the reconstructed data from the models to determine whether an input simulation is anomalous. However, at this stage, we have not explored the problem in depth and have only attempted an initial, commonly used method. Specifically, we attempted a threshold-based approach, setting the threshold at the 95th percentile of the MSE error on the training data (accepting a 5% false positive rate). We then compared the reconstruction errors of anomalous simulations against this threshold to evaluate its effectiveness in detecting anomalies.

Unfortunately, the results of this method were not effective. For example, even with the Transformer model, which exhibited the smallest reconstruction error, only 2 out of 36 anomalous simulations were detected. When we adjusted the threshold to the 80th percentile (accepting a 20% false positive rate), the results improved, with 21 out of 36 anomalous simulations being detected. The figure below illustrates the MSE errors of the anomalous simulations compared to the 80th percentile.

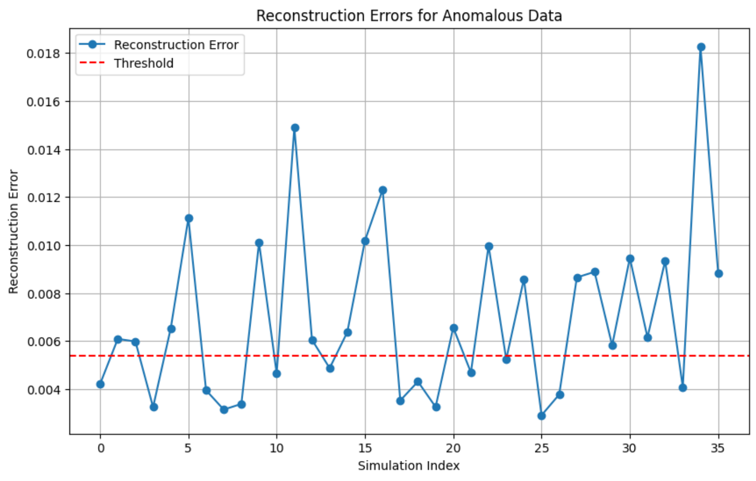
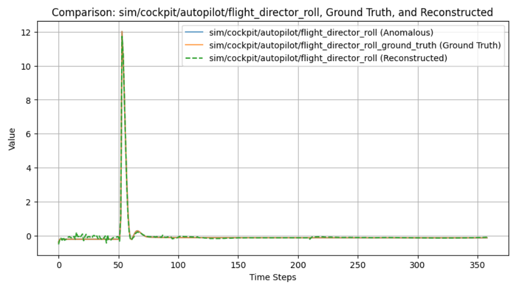


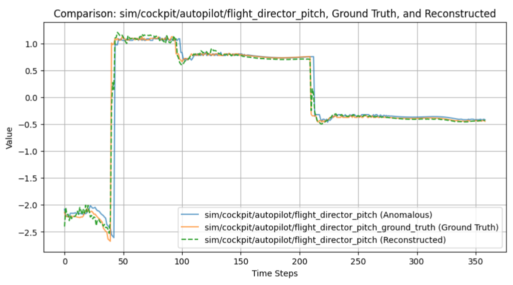
Figure 15: Reconstruction Errors for Normal Data

However, our objective is to maximize detection and identify all anomalous simulations. This is crucial in the context of flight simulations because any anomaly in an aircraft system could lead to severe consequences. Therefore, we must find an alternative approach to replace this method and ensure all anomalies are detected.

The reason why the MSE-based method is not entirely suitable for detecting anomalies lies in the nature of certain anomalous simulations. In some cases, the discrepancies between the anomalous data and ground truth occur over very short periods. As a result, averaging the squared errors over many time steps yields a small value. For example, in the anomalous simulation with index 7, the discrepancies in both anomalous variables are minimal compared to the ground truth. Consequently, the reconstructed data does not differ significantly from the anomalous data, leading to low MSE values.



(a) Compare *roll* variable in datasets



(b) Compare *pitch* variable in datasets

Figure 16: Compare the reconstructed data with other datasets.

Currently, after observing all anomalous simulations, we note that for simulations where the differences between the anomalous data and ground truth persist over a relatively long period (greater than 25 time steps), visual inspection of the anomalous and reconstructed data curves can easily reveal significant deviations. However, this approach is impractical for simulations with short-term deviations, as such differences are more challenging to detect.

The ideas we discussed with our tutors to address this issue will be presented in the *Proposed Methods for Future Work* section.

4 Conclusion and perspectives

4.1 Conclusion

At this point, we have not fully achieved the initial objective of this project, which was to develop a model capable of determining whether a simulation is an anomaly. However, we have accomplished several important steps toward solving this problem:

- Analyzing the dataset and preparing it for model development.
- Successfully reconstructing data from anomalous simulations to closely resemble the ground truth values. Among the models, the Transformer Autoencoder achieved the smallest reconstruction error on normal simulation test data and required the shortest training time. Therefore, the Transformer Autoencoder is the most effective model for data reconstruction in our study.

The remaining step, which we have not yet explored in depth, is determining how to use the reconstructed data to identify whether a given simulation is an anomaly.

4.2 Proposed Methods for Future Work

In discussions with our tutors, we identified several methods to address the anomaly detection problem:

- Using the ROC curve to determine a more suitable threshold for detecting anomalies.
- Evaluating the average error on the two anomalous variables and the remaining 133 variables. For normal simulations, these two values may not differ significantly, whereas for anomalous simulations, the discrepancy between these two metrics could be substantial.

These methods represent potential directions for future research to improve the effectiveness of anomaly detection in this context.

References

[1] Pankaj Malhotra et al. “LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection”. In: *Proceedings of the ICML 2016 Anomaly Detection Workshop*. Tata Consultancy Services Ltd. New York, NY, USA, 2016.

[2] Joël Potin and Florian Montels. *Anomalie détection pour simulateur*. Tech. rep. France: Groupe SII, 2024.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention* (2015), pp. 234–241. DOI: `10.1007/978-3-319-24574-4_28`. URL: `https://doi.org/10.1007/978-3-319-24574-4_28`.

[4] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. “Anomaly Detection in Time Series: A Comprehensive Evaluation”. In: *Proceedings of the VLDB Endowment* 15.9 (2022), pp. 1779–1797. DOI: `10.14778/3538598.3538602`.

[5] Ashish Vaswani et al. “Attention is All You Need”. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2017). DOI: `10.48550/arXiv.1706.03762`. URL: `https://doi.org/10.48550/arXiv.1706.03762`.

Annexes

The detailed code for this project is stored at the following GitHub repository:

`https://github.com/NaUl51/Projet-RI-5A`