CONTINUOUS ALERT LEVEL DETECTION IN FLIGHT DATA TIME SERIES USING MACHINE LEARNING

Gabriel de Champeaux[1,2], **Arnaud Polette**[1], **Jean-Philippe Pernot**[1], **Ammar Mechouche**[2], **Ali Yatsou**[2]

[1]Arts et Métiers Institute of Technology, LISPEN, F-13617 Aix-en-Provence, France.
[2]Airbus Helicopters, F-13725 Marignane, France.

ABSTRACT

Situational Awareness and Alerting Systems in rotorcraft and other aircraft are typically designed with discrete triggering thresholds. While these rule-based systems fulfill strict certification requirements, they lack the ability to gauge proximity to critical thresholds in a continuous manner. As a result, they often pose a trade-off between excessive false alarms and delayed warnings. In this paper, a data-driven methodology is proposed to create a continuous surrogate model of such alerting systems using machine learning, specifically time-series regression models. The approach includes data preprocessing, a Multilayer Perceptron model architecture trained to predict a postprocessed continuous metric that reflects the proximity of flight data to an alert threshold. The methodology is illustrated on real flight data, showing how it can build a new kind of distance of alarm occurrence. A discussion concludes with challenges posed by low sampling rates, lessons learned, and directions for future research, including the possibility of leveraging previous predictions to enhance model performance and identifying patterns of approaches to alert triggering.

INTRODUCTION

Ensuring flight safety remains an absolute priority in the aerospace industry, particularly for Airbus Helicopters. Despite significant advancements in aircraft design and operational training, statistical reports (Refs. 1–4) from relevant aviation regulatory and safety authorities indicate that over 80% of undesired events are attributable to operational factors rather than mechanical failures. To mitigate these risks, industry and regulators have introduced a variety of Situational Awareness and Alerting Systems (AS) — such as Helicopter Terrain Awareness and Warning System (HTAWS) (Ref. 5), Traffic Alert and Collision Avoidance System (TCAS) (Ref. 6) — and have also experimented with flight envelope protection mechanisms (Refs. 7,8). These systems provide pilot warnings whenever incoming flight data crosses predefined thresholds.

However, the rule-based nature mandated by the certification standards (Ref. 9) of most certified AS technology imposes significant limitations. Thresholds must balance early detection with the risk of frequent false alarms—too conservative, and they overwhelm pilots with warnings that may not be critical; too lenient, and they fail to provide timely alerts. Moreover, these alerts are inherently discrete: the system use dedicated logic for each alert level and abruptly transitions from "no alert" to a specified alert level, leaving no indication of *how close* the aircraft might be to triggering the next level. This behaviour is illustrated at time t, using $n+1$ input parameters noted p_i in Figure 1, representing an AS having 2 levels of alert: *AMBER* and *RED*.

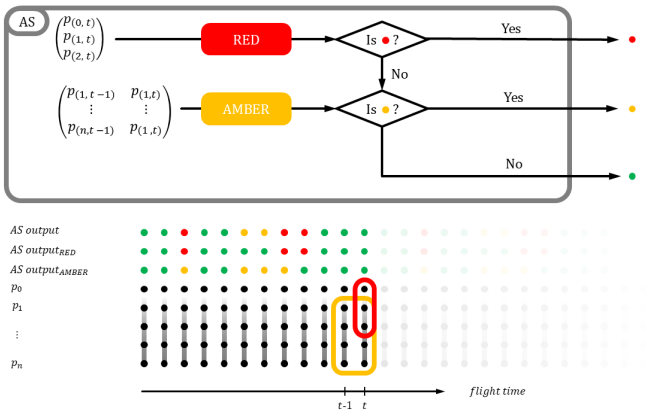

Figure 1. Illustration of a discrete AS with two alert levels.

To address this gap, the contribution of the paper is a continuous surrogate model for alert-level detection using machine learning. The term *surrogate model* introduced by N. V. Queipo et al (Ref. 10), designs a data-driven approach that

approximates the behavior of a complex system, in this case, an AS. By combining time-series regression with domain-specific preprocessing, here the approach approximates the behavior of existing AS logic while assigning a *continuous score* that reflects the proximity to a given alert threshold. This surrogate model has two potential benefits:

1. Enhanced Situational Awareness – The continuous metric can be used to inform pilots or operators *how close* they are to crossing a threshold, enabling smoother post-flight analyses.

2. Improved Alert Tuning and Optimization – Because the model is differentiable and data-driven, designers can perform more refined trade-offs and run large-scale simulations to determine optimal threshold settings.

Figure 2 depicts the surrogate model of the *AMBER* part of the AS represented in Figure 1. Its behaviour is modelized at time t, using n input parameters noted p_i on the two last frames.

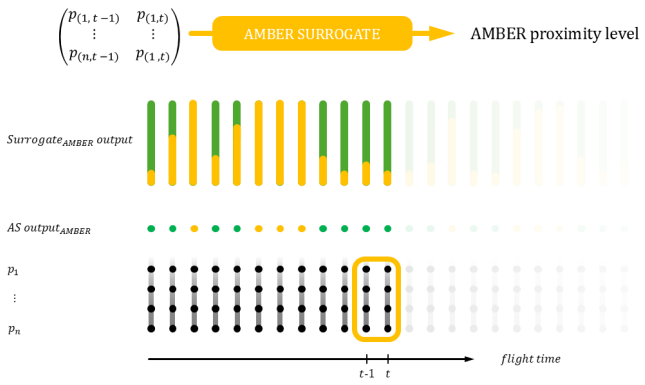

Figure 2. Surrogate model estimating the *AMBER* alert level of the AS in Figure 1.

In the following, prior statistical and machine learning methods for time-series modeling, analysis and forecasting (Section PRIOR WORK) are discussed. Next, the proposed approach is detailed (Section PROPOSED APPROACH), including the generic data preprocessing, machine learning architecture, and a proof-of-concept example illustrating how the model outputs a continuous alert metric. Finally, results are discussed in Section RESULTS AND DISCUSSION, focusing on model performance, the challenges of low sampling rates in flight data, and how the continuous alert measure can be interpreted.

PRIOR WORK

Classical Statistical Foundations

Early time-series work in flight-data analysis relied on linear models such as ARIMA and Box-Jenkins methodology (Ref. 11) and state-space Kalman filters (Ref. 12). These methods offer optimality guarantees under Gaussian noise but assume strictly defined decision boundaries and low-dimensional settings.

Rule-Based Continuous Methods: Fuzzy Logic

Zadeh's theory of fuzzy sets formalised reasoning with degrees of truth (Ref. 13), making it a natural fit for "how-close-to-the-limit" aviation tasks. Fuzzy inference systems have been fielded for terrain-following guidance (Ref. 14), dynamic envelope protection (Ref. 15), anti-icing control on the DA42 (Ref. 16), and proactive flight-risk scoring (FORAS) (Ref. 17). Neuro-fuzzy hybrids further extend rule bases with learning for fault detection in engines and sensors (Ref. 18). While fuzzy systems are highly interpretable, their scalability is severely limited by combinatorial rule explosion: the number of rules grows as $O(k^n)$, where k is the number of fuzzy sets per input and n is the number of input variables, typically making standard fuzzy inference impractical beyond ≈ 10 inputs (Refs. 19–21).

Deep Learning Architectures for Time-Series Forecasting

Convolutional Neural Network (CNN) (Ref. 22) has been adapted for time-series analysis by applying 1D convolutions across sequential data (Ref. 23). This model is effective at detecting local patterns and short-term dependencies. However, pattern recognition is not a requirement for AS logic, which follows predefined deterministic rules rather than learning implicit temporal structures.

Moreover, CNNs extract temporally invariant features, which is not relevant for AS, as each flight parameter has a specific and independent role in alert-level determination. As a result, CNNs do not align well with the structured rule-based logic required in certified AS.

Recurrent Neural Network (RNN) (Ref. 24) and Long Short-Term Memory (LSTM) (Ref. 25) are widely used for modeling sequential dependencies. LSTMs, in particular, improve upon RNNs by introducing gating mechanisms that retain relevant historical information while discarding less useful features. These architectures have proven effective for long-range dependency modeling in speech recognition, anomaly detection, and financial forecasting.

However, LSTMs rely on dynamic internal memory states and probabilistic weight updates, making them unsuitable for deterministic AS modeling. Since an AS operates on fixed logic with periodic updates, there is no need to maintain a long memory state across time steps. Furthermore, LSTMs do not provide an explicit windowed representation of inputs, making their internal state transitions difficult to interpret in an AS-like setting.

Transformers and Attention Mechanisms (Ref. 26) have gained popularity in time-series forecasting due to their ability to assign dynamic importance weights to different time steps.

These methods have shown strong results in cases where long-term dependencies and complex variable interactions are critical.

However, in the context of AS, every input parameter has a fixed and predefined contribution to the alert level at any instant. There is no need for selective attention or global context awareness, as the AS updates its state using the most recent flight data without dynamically adjusting input weighting. Thus, attention-based models introduce unnecessary complexity and do not align with the structured logic of AS.

Multilayer Perceptron (MLP) with Sliding Window (Ref. 27) is an old way to forecast time-series (Ref. 23). Given that AS logic updates periodically at a fixed frequency, the model must be able to approximate its behavior without relying on long memory states or dynamic temporal dependencies. Instead of using recurrent or attention-based architectures, usage of an MLP trained on a fixed-size sliding window is proposed.

Rather than modeling temporal dependencies explicitly as in RNNs or Transformers, the sliding window method processes a fixed number of past time steps simultaneously. Specifically, at each update step t, the model takes as input:

$$\mathbf{X}_t = (\mathbf{x}_t, \mathbf{x}_{t-1}, ..., \mathbf{x}_{t-\iota}) \tag{1}$$

where ι is a small fixed value (typically $1 \leq \iota \leq 10$), ensuring that only recent history is used. The MLP then learns a direct mapping:

$$\mathbf{Y}_t = f(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-\iota}) \tag{2}$$

The sliding window approach with an MLP offers several advantages. Unlike LSTMs or Transformers, it does not require recurrent memory states, eliminating the need to maintain hidden states across time steps. The MLP architecture has reduced complexity, making it simpler to train and deploy while requiring fewer hyperparameters compared to recurrent or attention-based models. Furthermore, by explicitly encoding a small window of past states, the model effectively learns local trends without relying on dynamic weighting mechanisms.

AS Operational Constraints and Choice of MLP

Certified AS must adhere to strict deterministic and traceable rules, as defined in DO-178C (Ref. 9). However, it is important to clarify that only the AS logic must comply with certification requirements—there is no obligation to certify the internal network components used to approximate AS behavior.

Key AS constraints include fixed rule-based logic, where each input parameter contributes in a predefined and deterministic manner to the alert level decision. The system operates with periodic updates, computing outputs at fixed intervals based on the latest available flight data. Additionally, it exhibits short-term dependency, as some past states may be retained for deriving features such as velocity and acceleration, enabling deterministic short-term forecasting. However, the logic does not rely on long-term memory.

This approach aligns with the AS update frequency by considering a fixed number of recent time steps, mirroring how AS processes flight data in real time. Additionally, it ensures a deterministic output, where each input consistently produces the same alert level prediction. Furthermore, the periodic nature of AS logic eliminates the need for complex temporal models, rendering recurrent or attention-based mechanisms unnecessary.

Handling Missing Values in Time Series Data

A key consideration when using an MLP with a sliding window is the potential presence of missing values in the input multivariate time series. Since the model processes a fixed-size window of past observations, missing values within this window could disrupt the learned mapping and lead to unreliable predictions.

To mitigate this issue, several strategies can be applied. One common approach is imputation, where missing values are replaced using interpolation, previous values (forward fill), or statistical methods such as mean or median imputation. Another alternative is to introduce a masking mechanism, where missing values are explicitly flagged as additional input features, allowing the model to adjust its predictions accordingly. However, given that the AS operates under strict deterministic constraints, a more robust strategy would be to ensure data completeness through preprocessing steps before feeding the inputs to the model.

Furthermore, since the MLP does not maintain recurrent memory states, the impact of a missing value is constrained to the specific time step where it occurs, rather than propagating through future predictions as in RNN-based architectures. This makes the model more resilient to localized gaps in the data, provided appropriate preprocessing steps are applied.

PROPOSED APPROACH

General Approach and Case Study

The primary objective of this study is to develop a generic methodology for constructing a continuous surrogate model for AS based on flight data. The proposed framework is designed to be applicable to various AS models, ensuring its adaptability to different alerting mechanisms.

To validate this methodology, it is applied to a specific flight envelope protection system. To introduce a minimal yet informative temporal component, the last 6 frames are considered as input for each prediction. In this case study, the surrogate model is trained using the same 10 flight parameters as inputs that the real AS uses for alert-level determination. Since an AS can incorporate timing mechanisms—allowing, for instance, an alert level to be maintained at a lower state for a

fixed duration after exiting a higher alert state—the previous outputs of the *AMBER* and *ALERT* levels are considered as part of the model's inputs. These outputs are thus added to the ten parameters defined over the five input frames. However, *AMBER* and *ALERT* levels at time t are excluded from the inputs, as the first is directly used as the model's output and the second can't interact with the first at same timestep, as needed by basics of software design architecture (see Figure 1).

Given that the available dataset has a sampling rate of 2 Hz, while real AS data streams typically operate at 10–100 Hz, it is important to acknowledge that the learned model is trained on a lower-resolution version of reality. However, since the surrogate model is designed to replicate AS logic rather than precise physical behavior, this limitation does not affect its conceptual validity.

Furthermore, the studied alert level (the lowest level of the system) presents a significant class imbalance, with raw data containing over 95% of non-alert samples. Addressing this imbalance is crucial to ensure that the model does not simply learn to predict the majority class while ignoring alert occurrences. To mitigate this issue, class-based undersampling and specific preprocessing strategies are employed, as described in the next section.

Preprocessing

The preprocessing stage aims to sample the dataset, ensure data partitioning avoids data leakage, and format the inputs for optimal training.

Class Sampling A pronounced class imbalance —approximately 95% of the samples were originally labelled *AMBER-false*— caused the network to converge to a trivial solution in which every input was predicted as *AMBER-false*. To counteract this tendency, undersampling of the majority class was adopted.

A naive remedy would involve random undersampling until perfect class balance is achieved. Although straightforward, such an approach discards a substantial amount of informative data and fails to reflect the natural prevalence of non-alert states, often leading to poor generalisation at inference time. Alternative strategies, including weighted loss functions (Ref. 28), were also investigated; however, the extreme imbalance rendered the resulting training dynamics unstable and difficult to reproduce.

The adopted procedure maintains a moderate over-representation of the non-alert class: *AMBER-true* samples constitute at least 20% of every training epoch, while the remaining 80% is filled with randomly drawn *AMBER-false* samples. This configuration provides the model with a more realistic distribution of alert events than a perfectly balanced dataset, yet still supplies sufficient minority-class examples for meaningful learning. Empirically, the network escapes the undesirable all-negative local minimum in roughly 30% of initialisations under these proportions. Continuous monitoring during the first training epochs —specifically, tracking accuracy (a plateau near 80% is indicative of the degenerate solution) and inspecting the confusion matrix at regular intervals— enables early detection and re-initialisation when necessary. In practice, this constitutes a reasonable trade-off between dataset representativeness and training stability.

Clustered Data Splitting To prevent data leakage and ensure proper generalization, a clustered split strategy is used to divide the dataset into training, validation, and test sets. The dataset is randomly split into clusters using a fixed seed to maintain reproducibility, with the distribution set at 65% for training, 5% for validation, and 30% for testing. Since flight data sessions may contain temporally overlapping sequences, all data from the same flight session is assigned to a single partition. This approach prevents data leakage by ensuring that the model does not encounter slightly modified versions of the same flight sequence across different splits, thereby enhancing the reliability of performance evaluation.

Input Formatting Once the dataset is sampled and partitioned, the input features are flattened into a single vector, where each sample comprises the last five frames of the ten flight parameters and the nine outputs of other/previous states of the AS (*RED*-state on five frames, *AMBER*-state on four frames), resulting in a vector of size 70. The output remains binary, indicating whether the AS is in the studied alert state or not. In Figure 3, at instant t, $\mathbf{X}$ represents the input data flattened vector, and $\mathbf{Y}$ the output data generated from the time-organized data.

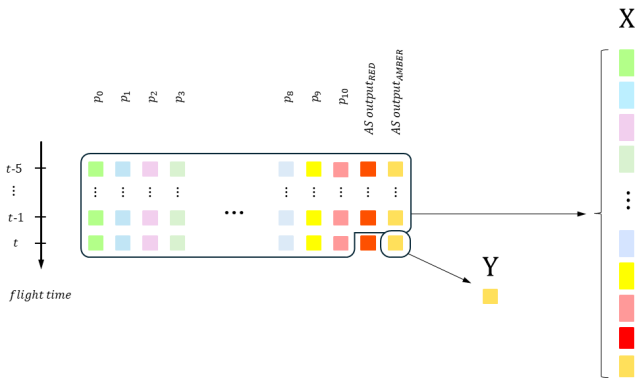

Figure 3. Representation of the input and output formatting.

The model is trained using an MLP architecture, optimized for binary regression/classification of the *AMBER*-proximity/state at frame t.

Model Training

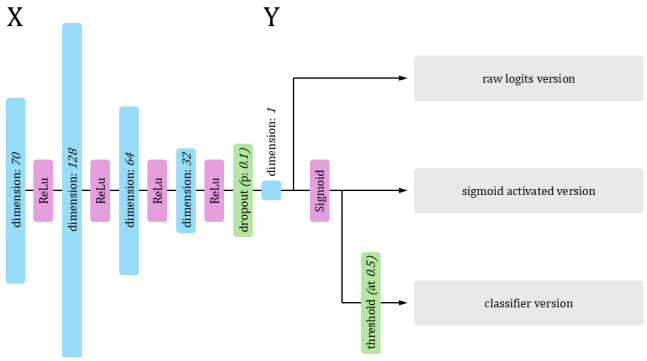

Figure 4. Representation of the model architecture.

Model Architecture Figure 4 illustrates the triplet representation of the saved neural network architecture. In this diagram, blue layers denote fully connected neuron layers, purple layers indicate activation functions, and green layers represent additional operations applied to the outputs of preceding layers. Layers annotated with **X** and **Y** correspond to the input and output layers, respectively, as defined in Figure 3. To enhance training stability and performance, the model is trained directly on its raw logits using PyTorch's Sigmoid-activated Binary Cross-Entropy Loss (Ref. 29), which internally combines a sigmoid activation with binary cross-entropy and employs the log-sum-exp trick for numerical stability (Refs. 30, 31), followed by a manual thresholding step.

The model is stored in this raw form, without its `dropout` layer to keep numerical determinism at inference time. In alternative versions, a sigmoid layer is appended to the output to constrain regression results within the interval $[0,1]$. One such variant further includes a manually defined threshold to directly produce binary classification outputs. A `softmax` activation is not required, as the task involves binary classification with a single output neuron.

Training Details To address class imbalance, a proportionally weighted loss function is employed to prevent the model from favoring the majority class. An `Adam` optimizer and a batch size of 4096 are used. The learning rate decays exponentially from $\eta_0 = 10^{-3}$ to $\eta_N = 10^{-5}$, with the value at epoch e given by:

$$\eta_e = \eta_0 \left(\frac{\eta_N}{\eta_0}\right)^{e/N} \tag{3}$$

where:

- $\eta_0 = 10^{-3}$: initial learning rate,
- $\eta_N = 10^{-5}$: final learning rate,
- $N = 10,000$: total number of training epochs,
- e: current epoch index.

Training is conducted for up to $10,000$ epochs, with early stopping based on validation accuracy stagnation to avoid convergence to suboptimal local minima. On a single NVIDIA GeForce RTX 4070 Laptop GPU, paired with a 13th Gen Intel(R) Core(TM) i9-13900H CPU and 64 GB of DDR5 RAM, the full training and testing process completes in approximately 1 hour and 14 minutes.

Logit Rectification into a Continuous Proximity Score

Post-hoc calibration techniques transform raw model outputs into calibrated scores or probabilities, improving interpretability and reliability.

Classical methods include Platt scaling (Ref. 32), temperature scaling (Ref. 33), and isotonic regression (Ref. 34), often used for binary classification. In regression contexts, empirical quantile mapping has been employed to adjust predictive distributions to match target distributions (Refs. 35, 36).

In climate science, semi-parametric quantile mapping techniques (Ref. 37) combine Empirical Cumulative Distribution Functions (ECDF) with chosen target laws (e.g., normal, gamma, uniform) to correct model bias while preserving physical interpretability.

The method proposed in this work adopts a similar philosophy: transforming raw logits into interpretable continuous scores by mapping the ECDF of validation data into a target distribution. Unlike typical classifier calibration, the objective is not probability estimation but producing a smooth, interpretable score reflecting proximity to an alert threshold.

Although the network is trained with Sigmoid-activated Binary Cross-Entropy Loss, the model is stored *without* the final sigmoid so that its single output neuron produces an unbounded raw logit $\ell_t \in \mathbb{R}$. This choice avoids heavy saturation of values at the $[0,1]$ boundaries (see Figure 7) and preserves the fine-grained information needed for continuous alert-level assessment.

To convert ℓ_t into an interpretable proximity score $S_t \in [0,1]$, a rectification mapping is applied as follows:

Step 1 - Empirical normalisation. All validation logits associated with the *non-alert* class are collected to build their ECDF: $F_0(\ell) = \Pr[\ell^{\text{val}} \leq \ell \mid \text{no alert}]$. Let F_0^- denote the ECDF value immediately below the decision threshold $\ell = 0$.

Step 2 - Target distribution. A monotone target Cumulative Distribution Function (CDF) G is chosen on $[0,1)$; in this work $G(u) = u$ (uniform law) is used, but a Beta(α,β) or any strictly increasing G could be substituted.

Step 3 - Rectification formula. For an incoming logit ℓ_t:

$$S_t = \begin{cases} 1, & \ell_t \geq 0, \\ G^{-1}\left(\dfrac{F_0(\ell_t)}{F_0^-}\right), & \ell_t < 0. \end{cases} \tag{4}$$

This strictly monotone mapping guarantees $S_t \in [0,1]$ and produces the desired distribution of S in the negative-logit domain.

Because $S_t = 1$ for all logits with $\ell_t \geq 0$, the rectified score S_t is a *mixed random variable*—having both a continuous component on $[0, 1)$ and a discrete mass at $S_t = 1$ (Ref. 38).

In implementation, Equation (4) is evaluated in real time via cubic-spline interpolation of the empirical $(\ell, F_0(\ell))$ pairs (approximately 1 µs per evaluation on CPU), followed by application of G^{-1} (uniform, Beta, etc.—in the Beta case using percent point function).

The calibrated mapping object is saved alongside the network weights and re-loaded at inference, adding negligible overhead.

RESULTS AND DISCUSSION

Neural Network Convergence

With the adopted configuration, the model achieves raw accuracies of 99.00% on the training set, 98.99% on the validation set, and 98.92% on the test set. These results indicate a high level of performance across all datasets, suggesting that the model has effectively captured the underlying data patterns and demonstrates strong generalization capabilities.

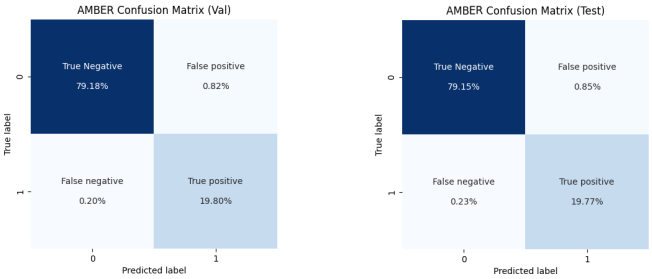

Figure 5. Confusion matrices for the validation and test sets.

Prediction errors can be grouped into two types: false positives and false negatives. The confusion matrices in Figure 5 illustrate the distribution of these errors across the validation and test sets.

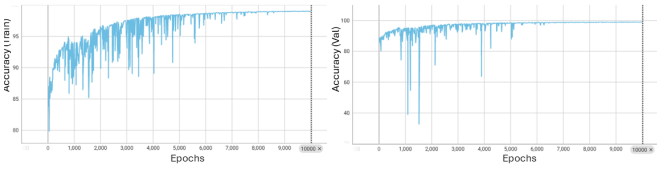

Figure 6. Training accuracies over epochs on training and validation sets.

As shown in Figure 6, both training and validation accuracies increase rapidly and stabilize towards the end of the training process. The curves exhibit near-convergence with a horizontal asymptote, indicating that the number of training epochs is sufficient.

About Probability Density Functions (PDF) of the versions of the model

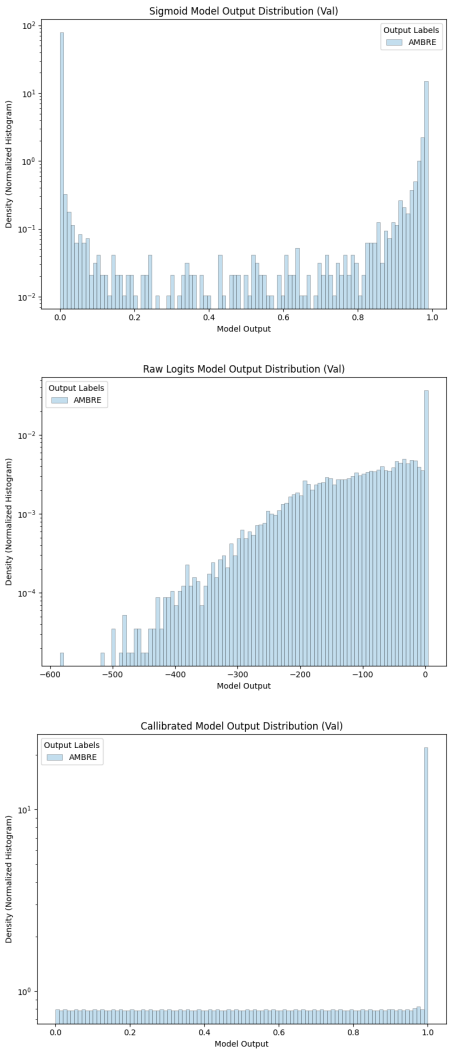

Figure 7. Normalized PDF plots of the outputs upon the test dataset for the sigmoid, the raw logit and the rectified versions of the model.

Figure 7 presents the PDF of three versions of the model's output evaluated on the test set: the version with a sigmoid activation applied to the final layer, the unbounded raw logits, and the rectified outputs as described in Section Logit Rectification into a Continuous Proximity Score.

The sigmoid-transformed output exhibits a highly saturated distribution, with a large concentration of values at the boundaries of the $[0, 1]$ interval and relatively few values in the intermediate range. This behavior is typical of sigmoid functions applied to post-training processes and indicates poor resolution for intermediate alert levels. As such, this output format is not well suited for tasks requiring fine-grained continuous scoring.

In contrast, the raw logits produce a more naturally distributed density, without artificial compression near the endpoints.

This untransformed output retains richer information about the model's internal confidence and serves as a better candidate for continuous alert-level estimation.

Finally, the rectified version, obtained by applying the PDF transformation defined in Section Logit Rectification into a Continuous Proximity Score, demonstrates a more uniform distribution over the $[0, 1]$ interval, with a notable concentration near 1.0, reflecting the 20% proportion of *AMBER-positive* samples in the validation set. This rectification step enables direct interpretability of scores as alert-level proximities and enhances the model's utility in continuous monitoring scenarios.

Sequence scoring examples

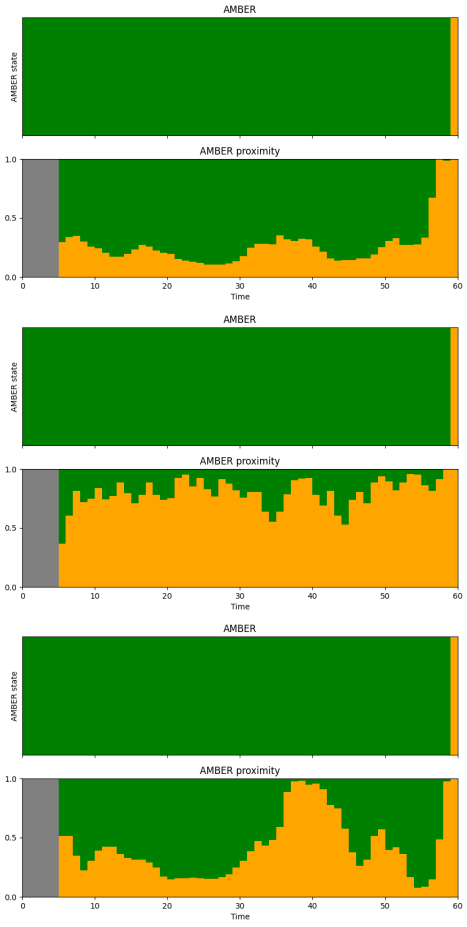

Figure 8. Three examples of flight sequence, for each one, on top are raw binary data, and below a recoloring in continuous *AMBER* levels using the surrogate model presented in Section PROPOSED APPROACH.

Figure 8 illustrates three distinct flight sequences, each annotated with the proximity scores produced by the surrogate model described in Section PROPOSED APPROACH. While all sequences ultimately lead to the triggering of an *AMBER* alert, the surrogate model provides a continuous scoring profile that reflects the evolving proximity to the alert threshold. The grey area at the beginning of each plot corresponds to the model's blind zone, where predictions are not available due to the requirement of a minimum temporal context (five preceding frames).

These three sequences offer illustrative insights into different alert-triggering dynamics:

In the first sequence, the proximity score remains low (below 0.5) for most of the approach before rising sharply to reach the alert threshold. This suggests that the conditions leading to the alert were established abruptly.

In the second sequence, the proximity score is high from the outset and increases steadily before crossing the threshold. This reflects a progressive deterioration of flight conditions, indicating a more gradual transition into the alert state.

The third sequence presents a more erratic profile. The proximity score starts at a moderate level, dips briefly, then rises to exceed 0.9, decreases again, and finally surges to trigger the alert. This variability may indicate unstable flight conditions or corrective attempts by the pilot. The observed fluctuations suggest difficulty in maintaining a stable regime and may prompt further investigation into the pilot's awareness and the necessity of the maneuver.

Although the proximity score alone does not provide direct information about the underlying causes, contributing parameters, or possible preventive actions, it serves as a valuable diagnostic indicator. From a qualitative perspective, the smooth temporal transitions in the predicted score demonstrate local consistency across frames, supporting the plausibility of the model's behavior. While the presented results are encouraging, further validation in operational contexts is necessary to fully establish the model's practical applicability.

CONCLUSION AND FUTURE WORK

The main contributions of this paper are:

1. A generic, data-driven workflow that transforms discrete certified Alerting Systems into a *continuous* surrogate model, formulated as a binary regression on short sliding windows of flight data (cf. Section PROPOSED APPROACH).

2. A lightweight MLP achieves 98.9 % test accuracy in reproducing certified AS logic, with post-training logit calibration producing an interpretable $[0, 1]$ proximity score for alert thresholds.

3. We demonstrate that (i) an MLP with fixed-length context preserves the rule-based nature of certified AS, (ii) our preprocessing pipeline effectively addresses extreme class imbalance, and (iii) logit calibration via an empirical PDF transform maintains confidence semantics.

4. Qualitative evaluations on real flight logs show the surrogate delivers smooth, frame-consistent proximity scores across diverse alert dynamics.

Building on these results, the next steps are:

- Re-train and benchmark on higher-frequency streams and additional aircraft types to assess generalizability.
- Integrate the continuous proximity score into post-flight monitoring dashboards for operator evaluation.
- Combine proximity scores with feature-attribution methods (e.g. SHAP, Integrated Gradients) to highlight the drivers of alert triggers.

Author contact:

Gabriel de Champeaux,
gabriel.de_champeaux_de_la_boulaye@ensam.eu,
gabriel.de-champeaux@airbus.com
Arnaud Polette, arnaud.polette@ensam.eu
Jean-Philippe Pernot,
jean-philippe.pernot@ensam.eu
Ammar Mechouche,
ammar.mechouche@airbus.com
Ali Yatsou, ali.yatsou@airbus.com

ACKNOWLEDGMENTS

The first author would like to express his deepest gratitude to his co-authors for their invaluable guidance, constructive feedback, and continuous support throughout the development of this work. Their expertise and mentorship have been instrumental in shaping both the research and the writing process of this first publication.

This work has been carried out as part of a CIFRE PhD thesis in collaboration between Airbus Helicopters and the LISPEN laboratory of the Arts et Métiers Institute of Technology. The authors sincerely thank Airbus Helicopters for its financial support and for providing access to the necessary data and computational resources.

Additionally, the authors acknowledge the contributions of their colleagues at Airbus Helicopters and LISPEN, whose insightful discussions and technical input have helped refine this research.

REFERENCES

1. European Union Aviation Safety Agency (EASA), "Appendix 3: Advanced Statistics for Helicopters," Technical report, 2023.
2. Bureau d'Enquêtes et d'Analyses pour la Sécurité de l'Aviation Civile (BEA), "Hélicoptères 2023 : Enseignements de sécurité de l'aviation légère," Technical report, 2023.
3. Federal Aviation Administration (FAA), *Helicopter Flying Handbook*, 2019, pp. 13–1 – 13–25.
4. Baumgartner, H. M., Durham, J., DiDomenica, R., and Hu, P. T., "Human Factors Analysis of Helicopter Air Ambulance Accidents, Incidents, and Events (2013-2023)," Technical report, Federal Aviation Administration (FAA), 2024.
5. "Certification Memorandum CM-FT-004 Issue 01: Helicopter Terrain Awareness and Warning System and Ground Proximity Warning System Alerting Functions for Offshore Operations," Certification Memorandum CM-FT-004 Issue 01, European Union Aviation Safety Agency (EASA), April 2019.
6. Federal Aviation Administration (FAA), "Introduction to TCAS II Version 7.1," , 2011.
7. Glad, J., "Helicopter obstacle detection and information system," US Patent App. 13/695,557, May 23 2013.
8. Nikolajevic, K., Belanger, N., Damiani, N., and Violette, A., "Safety system, a helicopter fitted with such a system, and a safety method seeking to avoid an undesirable event," US Patent 10,062,293, August 28 2018.
9. EUROCAE & RTCA, *Software Considerations in Airborne Systems and Equipment Certification*, EUROCAE, ed-12c edition, January 2012.
10. Nestor V. Queipo, Raphael T. Haftka, Wei Shyy, Tushar Goel, Raj Vaidyanathan and P. Kevin Tucker, "Surrogate-based Analysis and Optimization," Technical report, NASA Marshall Space Flight Center, 2005.
11. Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M., *Time series analysis: forecasting and control*, John Wiley & Sons, 2015.
12. Kalman, R. E., "A new approach to linear filtering and prediction problems," Vol. 82, (1), 1960, pp. 35–45. DOI: 10.1115/1.3662552.
13. Zadeh, L. A., *Fuzzy Sets*, Vol. 8, 1965, pp. 338–353. DOI: 10.1016/S0019-9958(65)90241-X.
14. Rahim, M., and Móhammad-Bagher Malaek, S., "Aircraft terrain following flights based on fuzzy logic," *Aircraft Engineering and Aerospace Technology*, Vol. 83, (2), 2011, pp. 94–104. DOI: 10.1108/00022661111120980.
15. Duerksen, N., "Fuzzy Logic Decoupled Longitudinal Control for General Aviation Airplanes," NASA Contractor Report CR–201639, NASA Langley Research Center, 1996.
16. Gliwa, W., Grzesik, N., and Kuźma, K., "Fuzzy logic controller design for the anti-icing system in the DA42 diamond aircraft," AIP Conference Proceedings, Vol. 2029, 2018. DOI: 10.1063/1.5066479.
17. Hadjimichael, M., "A fuzzy expert system for aviation risk assessment," *Expert systems with applications*, Vol. 36, (3), 2009, pp. 6512–6519. DOI: 10.1016/j.eswa.2008.07.081.
18. Vladov, S., Yakovliev, R., Hubachov, O., Mykolenko, K., Drozdova, S., and Rud, J., "Modified Neuro-Fuzzy Failure Classifier of Helicopters Turboshaft Engines," 2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT), 2023. DOI: 10.1109/C-SIT61576.2023.10324287.
19. Yager, R. R., and Filev, D. P., "Essentials of fuzzy modeling and control," *New York*, Vol. 388, 1994, pp. 22–23.
20. Casillas, J., Cordón, O., Triguero, F. H., and Magdalena, L., *Interpretability issues in fuzzy modeling*, Vol. 128, Studies in Fuzziness and Soft Computing, Springer, 2003.
21. Driankov, D., Hellendoorn, H., and Reinfrank, M., *An introduction to fuzzy control*, Springer Science & Business Media, 2013.

22. Fukushima, K., "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, Vol. 36, (4), 1980, pp. 193–202.

23. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J., "Phoneme recognition using time-delay neural networks," *Backpropagation*, Psychology Press, 1987, pp. 35–61. DOI: 10.1109/29.21701.

24. Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning internal representations by error propagation," Technical report, 1985.

25. Hochreiter, S., and Schmidhuber, J., "Long Short-Term Memory," *Neural Computation*, Vol. 9, (8), 1997, pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

26. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Łukasz Kaiser, and Polosukhin, I., "Attention Is All You Need," Advances in Neural Information Processing Systems, Vol. 30, 2017.

27. Lapedes, A., and Farber, R., "How Neural Nets Work," Neural Information Processing Systems, edited by D. Anderson, Vol. 0, 1987.

28. Cui, Y., Jia, M., Lin, T.-Y., Song, Y., and Belongie, S., "Class-balanced loss based on effective number of samples," Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019.

29. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, Vol. 32, 2019, pp. 8024–8035.

30. "BCEWithLogitsLoss — PyTorch Documentation," Online, "This loss combines a Sigmoid layer and the BCELoss ... take advantage of the log-sum-exp trick for numerical stability.", 2025.

31. Blanchard, P., Higham, D. J., and Higham, N. J., "Accurately computing the log-sum-exp and softmax functions," *IMA Journal of Numerical Analysis*, Vol. 41, (4), 2021, pp. 2311–2330. DOI: 10.1093/imanum/draa038.

32. Platt, J., *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, Vol. 10, (3), 1999, pp. 61–74.

33. Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q., "On calibration of modern neural networks," International conference on machine learning, 2017.

34. Zadrozny, B., and Elkan, C., "Transforming classifier scores into accurate multiclass probability estimates," Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, 2002. DOI: 10.1145/775047.775151.

35. Song, H., Diethe, T., Kull, M., and Flach, P., "Distribution calibration for regression," International Conference on Machine Learning, 2019.

36. Kuleshov, V., Fenner, N., and Ermon, S., "Accurate uncertainties for deep learning using calibrated regression," International conference on machine learning (ICML), 2018.

37. Rajulapati, C. R., and Papalexiou, S. M., "Precipitation bias correction: a novel semi-parametric quantile mapping method," *Earth and Space Science*, Vol. 10, (4), 2023, pp. e2023EA002823. DOI: 10.1029/2023EA002823.

38. Siegrist, K., "3.3: Mixed Distributions," Statistics LibreTexts, "The probability measure P really is a mixture of a discrete distribution and a continuous distribution.", 2022.