Machine Learning Toolbox for Anomaly Detection in Low-Flying Aircraft Surveillance

1st Melvyn Pirolley
Université de Franche-Comté
FEMTO-ST Institute, CNRS
Belfort, France
melvyn.pirolley@univ-fcomte.fr

2nd Raphaël Couturier
Université de Franche-Comté
FEMTO-ST Institute, CNRS
Belfort, France
raphael.couturier@univ-fcomte.fr

3rd Aymeric Cretin
Smartesting
Besançon, France
aymeric.cretin@smartesting.com

4th Antoine Chevrot
Smartesting
Besançon, France
antoine.chevrot@smartesting.com

5th Thomas Dubot
Universite de Toulouse
ONERA, DTIS
Toulouse, France
thomas.dubot@onera.fr

***Abstract*—The increasing density of low-flying aircraft and the development of new technologies for light aviation bring new challenges for air controllers. However, an important part of today's air traffic control relies on the insecure Automatic Dependent Surveillance-Broadcast (ADS-B) protocol. It cannot be trusted because anyone can read, emit, or modify ADS-B messages with a little equipment. This weakness can give rise to various attacks and a new security system should be developed for ADS-B to avoid a complete stop of the traffic.**

This work will present two attack scenarios and provide multiple methods based on machine learning to face those attacks automatically. The two scenarios cover spoofing attacks and ghost detection. This article also presents a few additional tools, developed in the context of our research project, for anomaly detection and ADS-B visualization. All those algorithms have the objective of constituting a complete toolbox for low-altitude ADS-B anomaly detection. Overall, all attacks can be detected with an accuracy of over 96%.

***Index Terms*—Cybersecurity, Machine learning, ADS-B, low-altitude air traffic**

I. Introduction

With the regular arrival of new low-altitude technologies, the density of low-altitude traffic increases. More and more projects are attempting to develop new lightweight aerial vehicles for transporting people or goods. In recent years a lot has been written about parcel delivery by drones [1]. More recently, during the Paris 2024 Olympics Games, the city has expressed its ambition to equip itself with an electric vertical take-off and landing aircraft (eVTOL) to provide rapid transport between a few points of interest. All these examples testify to the enthusiasm for the development of light transport systems. It will not be long before air traffic control systems are heavily impacted. These solutions are lighter than conventional air traffic, and will probably rely on protocols such as ADS-B (Automatic Dependent Surveillance-Broadcast) due to the very low altitude of these aircraft.

However, as those protocols are completely open, they are highly likely to be attacked [1]. These light transportation systems could be used to lead attacks with potentially dangerous consequences for the population. For example, eVTOL could be targeted by saturation attacks. As they act like taxis, they always follow the same route, and saturation of an eVTOL route by ghost aircraft would lead to the impossibility of ensuring air traffic control in the area, which could result in the suspension of eVTOL service.

That is why this study focuses on securing low-altitude air traffic control. Numerous attacks are possible, such as injecting phantom aircraft to flood the network, distorting trajectories, emitting false alarm signals, making an aircraft disappear, impersonating another aircraft, etc. The aim is therefore to provide a complete architecture based on deep learning and other technologies for detecting and fixing anomalies in low-altitude ADS-B data. This work is part of a research project, founded by the DGA (French defense procurement agency), named DApIA. It is the culmination of earlier work carried out as part of the GeLeaD (see Acknowledgements on the bottom part) [2, 3]. These three publications deal with ADS-B anomaly detection, but their main limitation is that they concentrate on commercial traffic, whereas the present study focuses on low-level traffic.

This article begins by presenting some other works on ADS-B anomaly detection. It then details the dataset's format used for training and evaluating our models. It continues with an explanation of the methods developed to cope with spoofing and flooding attacks. The next section presents the results obtained by our models. It ends with a conclusion and

This work was supported by the DGA (French defense procurement agency) in the context of the DApIA project (project number ANR-22-ASM2-0001) related to the ANR ASTRID Maturation program (specific support for follow-up research works of projects that have received a grant from the French Ministry of Armed Forces - GeLeaD project number ANR-18-ASTR-0011). It was also partially supported by the EIPHI Graduate School (contract ANR-17-EURE-0002). Computations have been performed on the supercomputer facilities of the "Mésocentre de Franche-Comté".

[1] https://www.bbc.com/news/business-67132527

prospects for improvement.

II. Related works

In the literature, some other works have presented models dealing with anomaly detection in ADS-B time series.

In [4], the authors present an unsupervised LSTM Encoder-Decoder to detect abnormal ADS-B messages with less than 4.5% of false positive detection. The model works by encoding windows of ADS-B messages in a vector and then reconstructing the trace with a decoder model. The reconstructed trace is compared with the original trace to assess model errors. Anomaly detection is based on the fact that an abnormal trace would be incorrectly reconstructed and amplified, producing a peak error. This reconstruction error is due to the fact that the model has only been trained on normal trajectories and would therefore not react correctly to modified trajectories, since it has never seen any during training. The implementation of this type of encoder-decoder model is part of an idea that we have not yet experimented with. A similar model could be useful for confirming our model's prediction for saturation attacks. This model also has similarities with [5].

In [6], the author presents another anomaly detection method. In this situation, the next ADS-B message is predicted based on a history of n messages. When the next message is received, the algorithms compare it with the model's prediction. When the difference between the prediction and the actual message does not exceed a certain threshold, the message is normal. This study is relevant to our work because it is very similar to our model for saturation attacks. The only difference is that it predicts all ADS-B features, whereas our model only predicts latitude and longitude for saturation detection. The advantage of predicting all features is that their model will be able to detect a wide variety of attacks. To achieve that, they have defined different error thresholds for each ADS-B feature to adapt to each anomaly. On the other hand, the advantage of our model is that it specializes in fixing saturation attacks.

III. Dataset

The dataset used in this study was mainly built up from the OpenSky network history database [7]. It contains several years of ADS-B message history worldwide, making it an inexhaustible source of ADS-B data for our algorithms. In addition, other data sources were used, such as OpenStreetMap tiles and an airport dataset to obtain airport coordinates. [8].

From this database, flight data was collected around Toulouse (from 0.72561 latitudes and 43.11581 longitudes to 2.16344 latitudes and 44.07449 longitudes) at altitudes of less than 10,000 feet, to retain only low-level traffic. Messages were collected between 2022 and 2023. Finally, messages are grouped by registry code and split between landing and take-off to form flights. Each flight is saved in one CSV file containing the following rows:

- **timestamp**: Date of the message
- **icao24**: Transponder identifier
- **latitude**: Coordinates
- **longitude**: Coordinates
- **groundspeed**: Horizontal speed
- **track**: Orientation (0° is north)
- **vertical rate**: Ascensional speed
- **callsign**: Registry code of the aircraft
- **onground**: True if on ground
- **alert**: True if in alarm state
- **spi**: Special position indicator
- **squawk**: A status code
- **altitude**: Barometric altitude
- **geoaltitude**: GNSS altitude

Low-quality flights were removed from the dataset based on criteria such as duration, amount of missing values, total flight length ... Each flight lasts a minimum of 15 minutes. As the aircraft transmits ADS-B every second, flights generally contain one message per second, but due to coverage problems, some messages may be missing. The final data set contains 10,158 flights for training (12% of the training set is used for testing) and 819 flights for evaluation.

IV. Method

Anomaly detection in low-level air traffic presents many challenges compared with anomaly detection in commercial aviation. The main difference lies in the great variety of aircraft and their respective trajectories. Therefore, to address this variety, multiple attack scenarios were established, and related anomaly detection models were developed.

A. Scenario A: Spoofing attacks

Our first scenario takes place in the context of the 2023 Rugby World Cup in Toulouse or a similarly large event. Toulouse has very dense low-altitude traffic, with a lot of SAMU (French emergency) helicopters, two heliports, low-altitude flight clubs, and military areas. In this traffic, an attacker could use a drone to target the stadium. In flight, the drone would emit false ADS-B messages, to impersonate a SAMU helicopter and make air surveillance believe it to be friendly. As a result, air traffic could be severely disrupted, preventing aircraft from taking off for safety reasons.

To face spoofing attacks, a deep Convolutional Neural Network (CNN) was developed. The model is a classifier that determines the aircraft type based on its trajectory only. Then, by comparing the prediction of the model and the registry code of the aircraft, it is possible to check if the aircraft is not spoofing the identity of another kind of aircraft. For example, if a plane uses the identity of a helicopter, it will be unmasked because the model will label its trajectory as a plane.

The model architecture is described in Figure 1. To make predictions, it combines four different inputs: an ADS-B window, a take-off context, a geographic context, and the distance between the aircraft and near airports. As output, the model gives three probabilities, one for each of the defined classes:

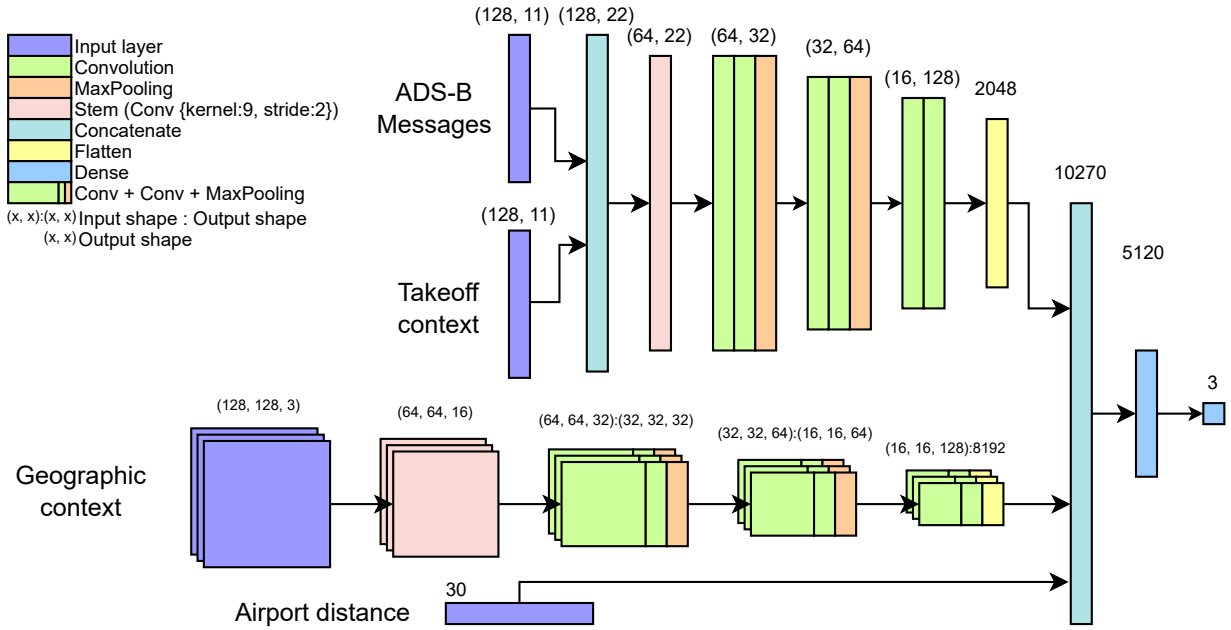


Fig. 1: CNN model architecture for Spoofing detection

- **Commercial aircraft**: at take-off and landing, when they cross the low altitude traffic.
- **Light aircraft**: regroup small aircraft, used by aeronautic clubs, tourism, transportation ...
- **Helicopter**: mainly SAMU helicopters, can also be police helicopters.

The ADS-B window corresponds to the last 128 timestamps, i.e. about 2 minutes of flight. The features used are timestamp, latitude, longitude, altitude, geoaltitude, ground speed, track, vertical speed, ground, alert, and spi. The trajectory latitude and longitude are transformed to relative coordinates on (0°, 0°). This normalization greatly helps the model as it will get all trajectories in the same format. The timestamp field is set at 0 for the current message and with negative values for the historical messages. Finally, all features are scaled between 0 and 1 using MinMaxScalers. Among the secondary data, the takeoff context corresponds to the first 128 time steps of the trajectory. This contextual data remains the same throughout the flight. It helps the model to remember the shape of the take-off so that it can continue to make good predictions in monotonous areas such as high-altitude flights. The geographic context is a small tile from the OpenStreetMap. It gives the model information about the environment (city, river, forests, ...) around the aircraft. This information is very important because aircraft regularly follow some geographic structures. For example, on take-off, SAMU helicopters follow the Garonne River to reduce noise in the city center. Finally, the distance from the airport helps the model to understand in which zone the aircraft is flying. It gives a sort of absolute coordinate location as the latitude and longitude of the aircraft are set to relative coordinates.

B. Scenario B: Saturation attacks

Our second scenario is about detecting and fixing flooding attacks. It involves the injection of ghost aircraft into the system. As there are multiple ways to realize a flooding attack this scenario is divided into two main categories: saturation and replays.

Those two types of attacks could have a large impact on dense air-lines. This is the case between Nice, Monaco, and Cannes, where many private helicopters act as taxis between these three destinations. The helicopters follow very similar trajectories, making them easier to target. With the advent of eVTOL technologies, these helicopters could be transformed into eVTOLs, for economic and noise reasons. In this case, traffic would increase drastically, making a saturation attack even more dangerous.

This first section discusses the methodology used to address saturation attacks. Saturation attacks refer to any flooding attack that injects false ADS-B messages around an aircraft's position to disrupt its tracking. These attacks can resemble those shown in Figure 2, and the goal is to identify ghost signals without filtering out real aircraft.

To achieve this, messages that are not coherent with the rest of the trajectory must be identified. A residual LSTM model was developed (Figure 3) to predict the next position of the aircraft based on its trajectory. Trained on normal trajectories, this model detects anomalies when the prediction error increases significantly.

For example, in Figure 4, the model's predictions were compared for a ghost diverging 30° from the original aircraft, another diverging 10°, and the real aircraft. The results show that the more the trajectory is modified, the farther the predictions are, leading to a higher model error.

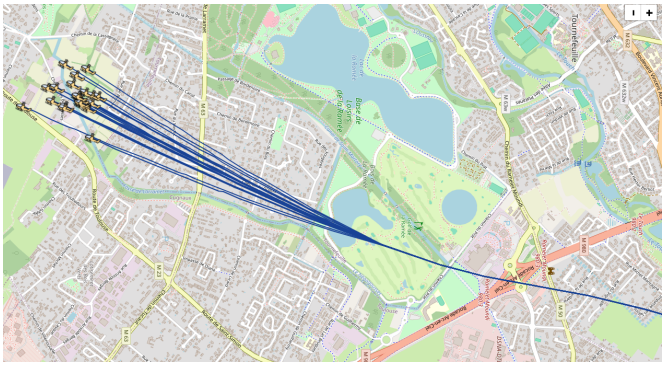


Fig. 2: Saturation exemple

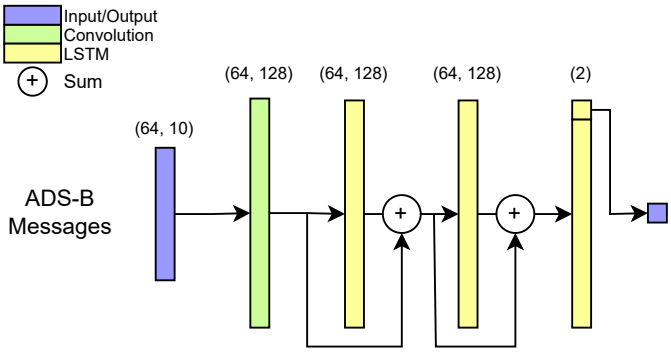
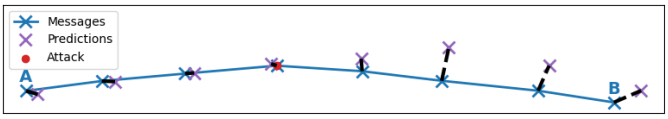
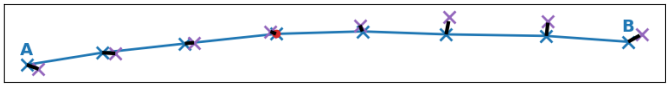


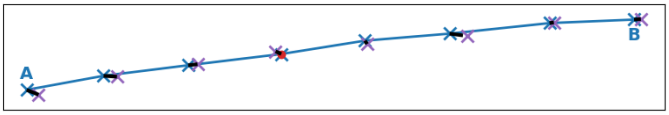
Fig. 3: Residual LSTM architecture for Saturation detection



(a) Divergence=30°



(b) Divergence=15°



(c) Real aircraft

Fig. 4: Comparison between model predictions and truth values

Based on this approach, it is possible to define an error threshold that, once exceeded, triggers an anomaly. This threshold must be fine-tuned to detect the maximum number of anomalies possible, without too many false positives.

C. Scenario B: Replay attacks

The second type of flooding attack is replay. The main difficulty in replaying trajectories lies in the fact that they are real trajectories that have occurred in the past. Replay attacks could disrupt the air traffic control of an entire airline, as an attacker could regularly inject replays. In this case, air traffic controllers would not be able to distinguish between real and fake aircraft, preventing them from ensuring the safety of real aircraft.

To detect if a flight is a replay from the past, the only reliable possibility is to compare it with a historical database and check if this precise flight already happened. Since a historical database represents a vast amount of data, pair-wise comparisons between flights would be too slow. Therefore, research focused on developing a hash table system specifically for storing trajectories. The approach was inspired by the Shazam app, which addresses a similar task by recognizing short extracts of time series in very large databases [9]. However, the Shazam hash method differs significantly because it deals with audio data, whereas our focus is on trajectories.

The hash algorithm must meet certain constraints. Firstly, the algorithms must be able to detect within a short window of a few minutes whether the trajectory is a trade-in or not. Indeed, it would be too late to know whether a trajectory is a trade-in once it has been completed. So, trajectories are split into small sub-windows of 32 timesteps. Secondly, as the attacker may use replays from a trajectory that happened somewhere else, our hash function should be resilient to basic transformations such as translation, rotation, scaling, and symmetries. To follow this constraint, every trajectory window is converted into a fingerprint before being hashed. The fingerprint is a series of micro right and left turns. The benefit of this transformation is that it makes trajectories invariant to nearly every transformation except symmetries. Additionally, it simplifies the trajectory significantly while maintaining its uniqueness. Finally, even though fingerprints are affected by symmetries, they can still be managed. Mirroring a trajectory will have the effect of inverting every right and left turn in its fingerprint. Hence, the hash function should be defined to generate the same hash value for opposite fingerprints. This can be achieved by associating right turns with the value 1 and left turns with 0, allowing the computation of a number based on the binary value of the fingerprint. This number is then reversed using the XOR operation if its value is greater than the median. The XOR operation ensures that opposite fingerprints produce the same hash value.

One weakness of the first version of this algorithm was its sensitivity to straight flights. When three points were almost aligned, the algorithm started to confuse right and left turns due to floating-point precision issues. To address this, a straight label was added to the fingerprint, acting as a wildcard. The idea is that when it is unclear whether the current message forms a right or left turn, it is ignored. The straight label generates a match whether compared with a right or left turn. This behavior is achieved by generating every sub-combination

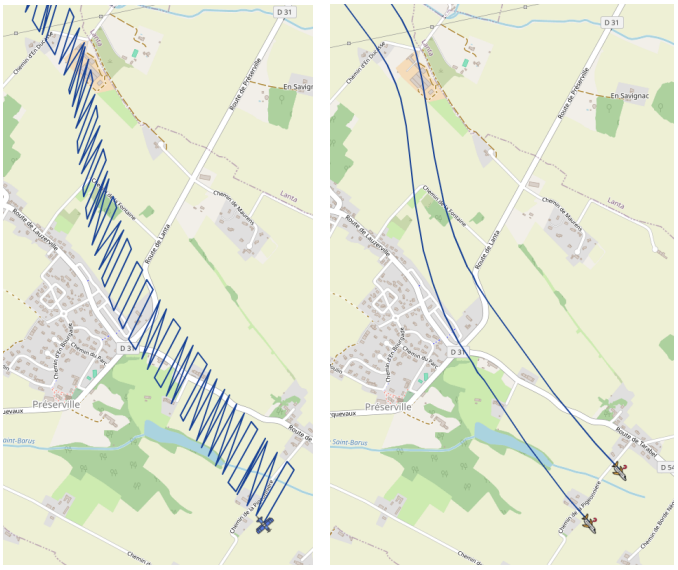
of a fingerprint by replacing wildcards with right and left and then comparing each sub-fingerprint with the hash table.

D. Additional tools

During the realization of the project, some secondary tools were developed to ease the development of the project or to fix some practical problems.

1) Trajectory Separator: has been developed to separate messages for flights having the same registry code. This can happen when ghost messages use the same registry code of a plane while it flight. Without a system to separate duplicated registry code, the flight could look like Figure 5a. The screen displays messages as if they were part of the same trajectory.

In such a situation, when several messages are received with the same register code, the algorithm creates sub-trajectories for each conflicting message. It aims to assign each of these messages to a sub-trajectory, trying to keep the trajectory as smooth and coherent as possible. To achieve this, the algorithm predicts for each sub-trajectory the position where the next message should be. Then it assigns the message to the trajectory that has the nearest predicted position. As a result, the algorithm will reconstruct trajectories as in Figure 5b.



(a) Before (b) after

Fig. 5: Before and after duplicated registry code separation

To predict the next position the algorithm applies the speed vector of the aircraft from its position (using spherical calculation). The prediction could be more accurate with a machine-learning model however it would be more time-consuming.

2) ADS-B visualizer: is a tool for visualizing ADS-B files (Figure 6). Its goal is to ease the visualization of our ADS-B flight's content. It can be used to understand the specificity of some attacks. It is also used to demonstrate the project by allowing the visualization of model predictions.

It provides a large range of functionalities such as play, pause, increasing or decreasing the time speed, going backward, and jumping to a specific time. It can manage multiple aircraft at once, and add the ability to filter aircraft by type or name. It can display all trajectories at once, to highlight the busiest airline. It can graphically display speed, altitude, and other aircraft profiles.

An online version of the visualizer is available at https://adsb-visualizer.web.app/ and the source of the project can be found in GitHub at: https://github.com/DApIA-Project/ADSB-Visualizer

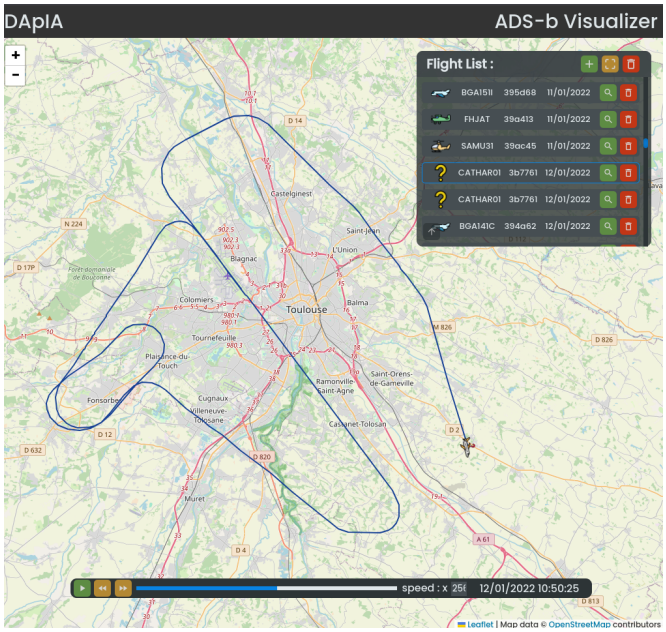


Fig. 6: ADS-B visualiser interface

3) A Python library for anomaly detection: The following text should be kept in mind:

"This tool has been developed to ensure reproducibility and simplify the use of our detection tool. It has been designed for easy integration and consists of only one function called "predict()". This function takes all ADS-B messages received in the last second as input." The input format of "predict()" is a list of dictionaries, each dictionary represents one ADS-B message and should contain the following keys: timestamp, icao24, latitude, longitude, groundspeed, track, vertical_rate, callsign, on-ground, alert, spi, squawk, altitude, geoaltitude. The function will output the same ADS-B messages, annotated with the following flags:

- **Spoofing**: True if spoofing anomaly has been detected
- **Flooding**: True if the aircraft is a ghost
- **Replay**: True the flight has been detected as a replay

To make predictions the library stores in a buffer the messages from the last call to be able to access the historical data needed by the models.

Hence, it is easy to predict every source of ADS-B. The code in figure 7 is an example of predicting an ADS-B record saved as CSV.

The library code is available in our GitHub: https://github.com/DApIA-Project/Anomaly-Detection and can

be installed using the command: `pip install AdsbAnomalyDetector`

```python
from AdsbAnomalyDetector import predict
import pandas as pd

data = pd.read_csv("./record.csv")
start = data["timestamp"].iloc[0]
end = data["timestamp"].iloc[-1]

out_df = pd.DataFrame()
for t in range(start, end):
    messages = data[data["timestamp"] == t]
        .to_dict("records")

    messages_out = predict(messages)
    for message in messages_out:
        out_df.loc[len(out_df)] = message

out_df.to_csv("output.csv")
```

Fig. 7: Code example to check anomalies on an ADS-B record saved as CSV

V. Results

A. Scenario A: Spoofing attacks

To verify the efficiency of the model, it was tested under real conditions by streaming actual flight messages and checking the model's classification. Since the model makes predictions on flight windows, it provides one prediction for each message. To aggregate predictions, the most confident prediction of the model is used to determine the aircraft type. This method yields the following confusion matrix (Figure 8).

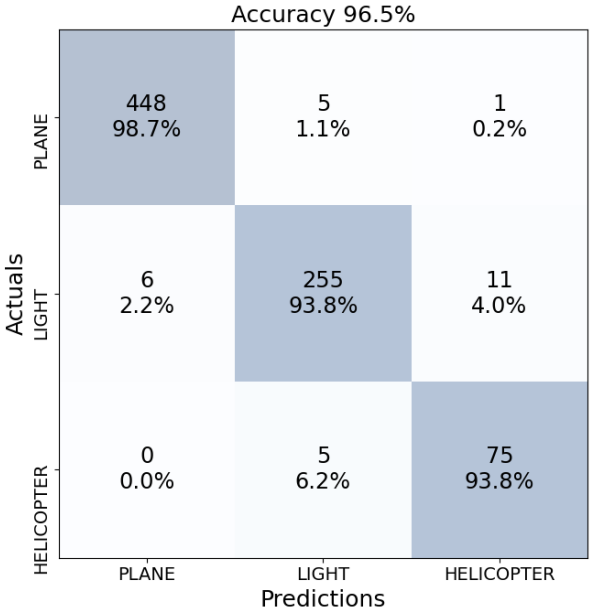


Fig. 8: Confusion matrix of the CNN model for Spoofing detection

The majority of the model's mistakes occur between light planes and helicopters. This confusion is mainly because the trajectory of a flight at a certain altitude can sometimes be very linear and exhibit similar patterns between helicopters and light planes. Given that commercial planes are heavier, it is not surprising that the model easily recognizes them.

To compare with other models, multiple experiments were conducted using other classical machine learning techniques.

Model	MSE	Accuracy (%)
Transformer	0.0373	95.6%
LSTM	0.0323	96.0%
CNN	**0.0181**	**96.5%**

TABLE I: Spoofing models comparison

Surprisingly, time series models such as LSTM and Transformers perform less effectively. This may be due to the fact that the problem is a classification task, while LSTM and Transformers are generally more suited for regression tasks. An ablation study of this model was conducted previously to demonstrate the necessity of each sub-module. As shown in Table I the best approach is to use CNN models. This model requires approximately 45 minutes of training on an Intel i7 CPU over 150 epochs. Other models utilize the same architecture as the CNN, but the Take-Off and ADS-B modules are replaced with LSTM or Transformers. The Transformers model is inspired by [10].

B. Scenario B: Saturation attacks

Detecting the correct airplane among a mass of ghost aircraft can be challenging under saturation. That is why our model focuses on detecting ghosts rather than finding the right aircraft. In an ideal situation, all the ghosts would be detected, leaving only the correct aircraft.

To evaluate the efficiency of a model, its accuracy was computed based on the percentage of ghosts that incur a higher loss than the real aircraft. The metrics used depend significantly on the difficulty of the attack and were derived from a dataset of saturation attacks generated by ourselves. Under actual saturation attacks, the model's performance could vary. Multiple models are compared in Table II.

Model	MSE	Distance error (m)	Accuracy (%)
Transformer	0.0058	22.46m	82.6%
CNN	0.0061	17.21m	89.8%
Reservoir	0.0044	13.99m	98.5%
Residual-LSTM	**0.0034**	**13.60m**	**98.5%**

TABLE II: Saturation models comparison

The residual LSTM architecture gives the best result with an accuracy of 98.5

The reservoir computing model performs surprisingly well, despite being simpler than classical neural network techniques. It achieves the same accuracy as the LSTM, with a slightly higher error distance. Its simple architecture allows for quick

predictions due to its short execution time. The reservoir architecture consists of 1000 units connected to a fully connected layer of neurons as readout [11].

While the Transformer models have good generalization, they do not outperform the LSTM or the Reservoir. The Transformer model used here comes from [12], using the classical variant from their GitHub. Its underperformance could be due to the short forecasting horizon, as transformers are typically effective for long-term forecasting. Gradient boosting algorithms were also tested but did not perform better.

C. Scenario B: Replay attacks

The hash table developed to solve replay attacks is one of our best models. It can generally determine if a trajectory is a replay with a one-minute flight window. Depending on the quality of the ADS-B coverage, it can need a bit more time to make its prediction. Thanks to the characteristics of hash tables, false positive detections are never generated. Hash collisions are possible, as the number of different hashes is $2^{31} \approx 2B$. It seems to be small as common hash systems use larger ranges like 2^{128} combination for MD5 however, it provides better reactivity of detection. Furthermore, our model's predictions are not based solely on a single match. For reliability reasons, the model only qualifies a flight as a replay if several messages correspond consecutively to the same flight in the past. So, even if it is possible to have a collision for a message, it is impossible to have several consecutive collisions with the same flight, and the model therefore never generates a false-positive detection.

In terms of detection capacity, the system is very accurate. It can determine if a trajectory is a replay even when deformed by common matrix transformations. Providing a precise accuracy score is challenging as it depends on the strength of the attack. However, with raw replay attacks, the system achieves 100% correct predictions, and on the test dataset, it correctly detects ghosts with 98% accuracy.

Finally, the algorithm is really fast, thanks to the hash table which has $0(1)$ complexity. Hence, even if the database is really large the algorithms will be able to process a flight of 15 minutes in 134ms.

One limitation of our system is that the hash table is stored on the computer's RAM. As it can become very large, it would be better to store it on the disk so as not to overflow the computer's RAM and to allow memory space for our neural network algorithms. However, this change would slow down the detection.

VI. Conclusion and perspectives

In this paper, several models for detecting anomalies in ADS-B time series are proposed. These models can work together to provide reliable detection of ghosts using ADS-B reruns or ghosts with abnormal trajectories. It also includes a tool to automatically detect whether aircraft are using the correct registration code, and to prevent spoofing attacks. All these models are brought together and run in real time in a library called AdsbAnomalyDetector.

In the future, we plan to improve the variety of attacks that could be handled by the model. For example, we could train the models to falsify trajectories generated by interpolation, which would be too smooth. We will also try to add redundancy to confirm the model's predictions by using additional models such as autocoders. Finally, we will try to fine-tune the architecture of our model and go beyond our current accuracies.

References

[1] Andrei Costin and Aurélien Francillon. *Ghost in the Air(Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices*. July 2012.

[2] Ralph Karam, Michel Salomon, and Raphaël Couturier. "A comparative study of deep learning architectures for detection of anomalous ADS-B messages". In: *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*. Vol. 1. IEEE. 2020, pp. 241–246.

[3] Ralph Karam, Michel Salomon, and Raphaël Couturier. "Supervised ADS-B Anomaly Detection Using a False Data Generator". In: *2022 2nd International Conference on Computer, Control and Robotics (ICCCR)*. 2022, pp. 218–223. DOI: 10.1109/ICCCR54399.2022.9790149.

[4] Edan Habler and Asaf Shabtai. "Using LSTM encoder-decoder algorithm for detecting anomalous ADS-B messages". In: *Computers Security* 78 (2018), pp. 155–173. ISSN: 0167-4048. DOI: 10.1016/j.cose.2018.07.004.

[5] Antoine Chevrot, Alexandre Vernotte, and Bruno Legeard. "CAE: Contextual auto-encoder for multivariate time-series anomaly detection in air transportation". In: *Computers & Security* 116 (2022), p. 102652.

[6] Yunkai Zou Jing Wang and Jianli Ding. "ADS-B spoofing attack detection method based on LSTM". In: *J Wireless Com Network* 160 (2020). DOI: 10.1186/s13638-020-01756-8.

[7] *OpenSky's Historical Database*. 2013. URL: https://opensky-network.org/data/historical-flight-data.

[8] David Megginson. *OurAirports*. 2007. URL: https://ourairports.com/.

[9] Avery Wang. "An Industrial Strength Audio Search Algorithm." In: Jan. 2003.

[10] Theodoros Ntakouris. *Timeseries classification with a Transformer model*. 2021. URL: https://keras.io/examples/timeseries/timeseries_classification_transformer/.

[11] Nathan Trouvain, Nicolas Rougier, and Xavier Hinaut. "Create Efficient and Complex Reservoir Computing Architectures with ReservoirPy". In: *From Animals to Animats 16*. Cham: Springer International Publishing, 2022, pp. 91–102. DOI: 10.1007/978-3-031-16770-6_8.

[12] Haixu Wu et al. *Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting*. 2022. arXiv: 2106.13008 [cs.LG].