Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Master in Data Science

Master Thesis

Analysis of the Main Causes of Aborted Landings and Real-Time Prediction using Deep Learning Techniques

Author: Unai Zuazo Angulo
Supervisors: Antonio Jiménez Martín
Manuel Ángel Amaro Carmona

Madrid, July - 2025

This Master Thesis has been deposited in ETSI Informáticos de la Universidad Politécnica de Madrid.

Master Thesis
Master in Data Science

Title: Analysis of the Main Causes of Aborted Landings and Real-Time Prediction using Deep Learning Techniques
July - 2025

Author:	Unai Zuazo Angulo
Supervisor:	Antonio Jiménez Martín Departamento de Inteligencia Artificial ETSI Informáticos Universidad Politécnica de Madrid
Supervisor:	Manuel Ángel Amaro Carmona Centro de Referencia de Investigación, Desarrollo e Innovación ATM (CRIDA)

Acknowledgements

The main body of this Master thesis is in English, but I wish to express my gratitude in my native language below.

Habiendo acabado este Trabajo Fin de Máster (TFM), miro hacia atrás y no puedo sino sentirme agradecido por todas las personas que me han rodeado este último año y las que por desgracia no han podido.

En primer lugar quiero agradecer a todo el equipo de CRIDA por brindarme esta oportunidad, por la cercanía día sí día también, por recibirme con los brazos abiertos como a uno más y apoyarme durante toda mi estancia en la empresa. En especial, agradecer a Manuel Ángel Amaro, mi tutor dentro de la empresa, por estar dispuesto siempre a apoyarme cuando tenía dudas y a escucharme aunque no las tuviese; por su disposición a hacer huecos entre sus reuniones para charlar sobre mis avances, consultar dudas y charlar de asuntos ajenos al trabajo también. Asimismo a Diego Piva Álvarez, por la paciencia y por tomarse el tiempo de explicarme todas las nociones de aviación necesarias, un mundo nuevo para mí, y echarme una mano cuando necesitaba, aunque claramente tenía muchos otros asuntos de los que preocuparse. No puedo citar uno a uno a todos mis compañeros pero os doy encarecidamente las gracias a todos.

Obviamente quiero agradecer a Antonio Jiménez Martín, y Alfonso Mateos Caballero por guiarme a lo largo de todo este proceso que ha sido mi TFM; por tomarse el tiempo regularmente de atender mi progreso y aportar conocimiento experto cuando era necesario.

Por último, pero no menos importante, quiero agradecer a mi familia, especialmente a mi aita y mi ama, por su apoyo incondicional desde que era niño hasta ahora. Por motivarme a superarme todos los días y a ser mejor persona, tanto en lo laboral como en lo personal. Quiero dedicar este TFM a mis abuelos, que sé que les habría encantado verme terminar mis estudios, y mencionar a mi aitite, que se fue antes de que le pudiese conocer realmente y a mi amama, a quien ni siquiera tuve la oportunidad de ver.

"No hay mayor expresión de cariño que dedicarle a alguien tu tiempo." es una frase que siempre me ha dicho mi ama, y que siempre llevaré en el corazón. Me habría gustado entenderla antes.

A todos los que me habéis acompañado hasta aquí, gracias.

This Master Project was supported by grants PID2021-122209OB-C31 and RED2022-134540-T funded by MICIU/AEI/10.13039/501100011033.

Resumen

Las aproximaciones frustradas, o go-arounds, son eventos críticos en aviación que suponen desafíos operativos tanto para los pilotos como para los controladores aéreos. Su carácter impredecible puede generar retrasos, aumentar la carga de trabajo del controlador e impactar negativamente en la eficiencia general del aeropuerto. Este trabajo de fin de máster presenta un enfoque basado en datos para la predicción en tiempo real de go-arounds mediante técnicas de inteligencia artificial, con especial atención a modelos de deep learning aplicados a datos reales de trayectorias de vuelo.

El problema se modela como una tarea de *regresión extrínseca de series temporales*, donde el sistema estima la probabilidad de una aproximación frustrada en cada punto durante la fase final de descenso. Utilizando datos históricos de aeropuertos españoles como Barcelona-El Prat y Madrid-Barajas, se entrenan y evalúan distintas arquitecturas de deep learning: LSTM, LSTM con atención y transformers. Un conjunto personalizado de características derivadas, incluyendo alineación con la pista, desviación del rumbo, velocidad vertical, energía del avión y datos meteorológicos, mejora significativamente la precisión del modelo.

El mejor rendimiento se obtiene con una arquitectura basada en transformers, alcanzando una precisión superior al 80% hasta dos minutos antes del evento de go-around. Se emplea análisis SHAP para interpretar el comportamiento del modelo y confirmar que sus decisiones se alinean con factores causales conocidos. Además, se desarrolla una interfaz de predicción en tiempo real (LandIA), capaz de monitorizar vuelos activos y asistir a los controladores en un entorno operativo.

Los resultados confirman que la predicción en tiempo real de aproximaciones frustradas es tanto viable como efectiva, especialmente cuando el modelo se adapta a aeropuertos concretos. Este proyecto contribuye a mejorar la seguridad, reducir la carga de trabajo de los controladores y avanzar en la integración de la inteligencia artificial en los sistemas de gestión del tráfico aéreo.

Abstract

Go-arounds, or aborted landings, are critical events in aviation that pose operational challenges for both pilots and air traffic controllers. Their unpredictable nature can cause delays, increase controller workload, and impact overall airport efficiency. This master thesis presents a data-driven approach to real-time go-around prediction using artificial intelligence techniques, with a focus on deep learning models applied to real-world flight trajectory data.

The problem is modeled as a *time series extrinsic regression* task, where the system estimates the probability of a missed approach at each point during final descent. Using historical data from Spanish airports such as Barcelona-El Prat and Madrid-Barajas, various deep learning architectures, LSTM, LSTM with attention, and transformers, are trained and evaluated. A custom set of engineered features including runway alignment, heading deviation, vertical speed, aircraft energy, and wind data significantly improves prediction accuracy.

The best-performing model, a Transformer-based architecture, achieves an accuracy of over 80% up to two minutes before a go-around event. SHAP analysis is used to interpret model behavior and confirm its decisions align with known causal factors. Additionally, a real-time prediction interface (LandIA) is developed, capable of monitoring active flights and assisting air traffic controllers in operational environments.

The results confirm that real-time go-around prediction is both viable and effective, especially when tailored to specific airports. The project contributes to improving safety, reducing controller workload, and advancing the integration of artificial intelligence into air traffic management systems.

Contents

Chapter 1

Introduction

Air traffic management is a complex and dynamic system that requires constant monitoring and precise decision-making to ensure the safety and efficiency of flight operations. One of the most critical phases of a flight is the final approach and landing, where pilots and air traffic controllers must work in tandem to achieve a smooth and stable descent. However, not all landings proceed as planned, and in some cases, pilots must execute a go-around or an aborted landing due to unstable approach conditions, external disruptions, or safety concerns.

Go-Arounds are a standard safety procedure designed to prevent hazardous landings, yet they introduce additional challenges for Air Traffic Controllers (ATCOs), pilots, and overall airport operations [10]. They require rapid decision-making, immediate flight path adjustments, and careful coordination to avoid conflicts with other aircraft. Additionally, the unpredictability of go-arounds can lead to increased air traffic congestion, potential delays, and added fuel consumption. Given these challenges, the ability to predict the probability of an aborted landing in real time can significantly enhance situational awareness and decision-making, reducing the operational impact and improving overall flight safety.

Recent advancements in artificial intelligence and machine learning have demonstrated the potential for data-driven predictive models in aviation [9, 12, 20, 23]. Traditional methods for identifying go-around likelihood rely on historical trends, predefined safety parameters, and expert judgment. However, these approaches often fail to provide real-time, adaptive insights necessary for dynamic flight conditions. The advent of deep learning and real-time data processing presents an opportunity to develop more sophisticated predictive models that can continuously assess flight parameters and estimate the probability of a go-around event.

This Master thesis proposes a deep learning-based approach to real-time probability prediction of aborted landings, leveraging different model architectures trained on real-world airport data. A detailed workflow can be seen in Figure 1.1. Note that this project is a continuation of previous work at CRIDA A.I.E. and aims to complete research on aborted landing prediction. As can be seen in the workflow diagram, this work uses a prelabeled database that has already been thoroughly worked on. An XGBoost classifier [5] has been trained on historical flight data, and is able to distinguish aborted landings from non-aborted. This model was then used to label all historical flights up to 2024, which set the foundation for this project. Using this

database, together with more data collected from various sources, we aim to develop a model, based on deep learning, that continuously updates the go-around probability throughout the final approach. We hope for robust and generalizable results that offer ATCOs and pilots critical foresight into potential go-arounds before they are initiated. In addition, we will analyze the most contributing factors when talking about Go-Arounds (GAs) to gain further insight into their predictions.

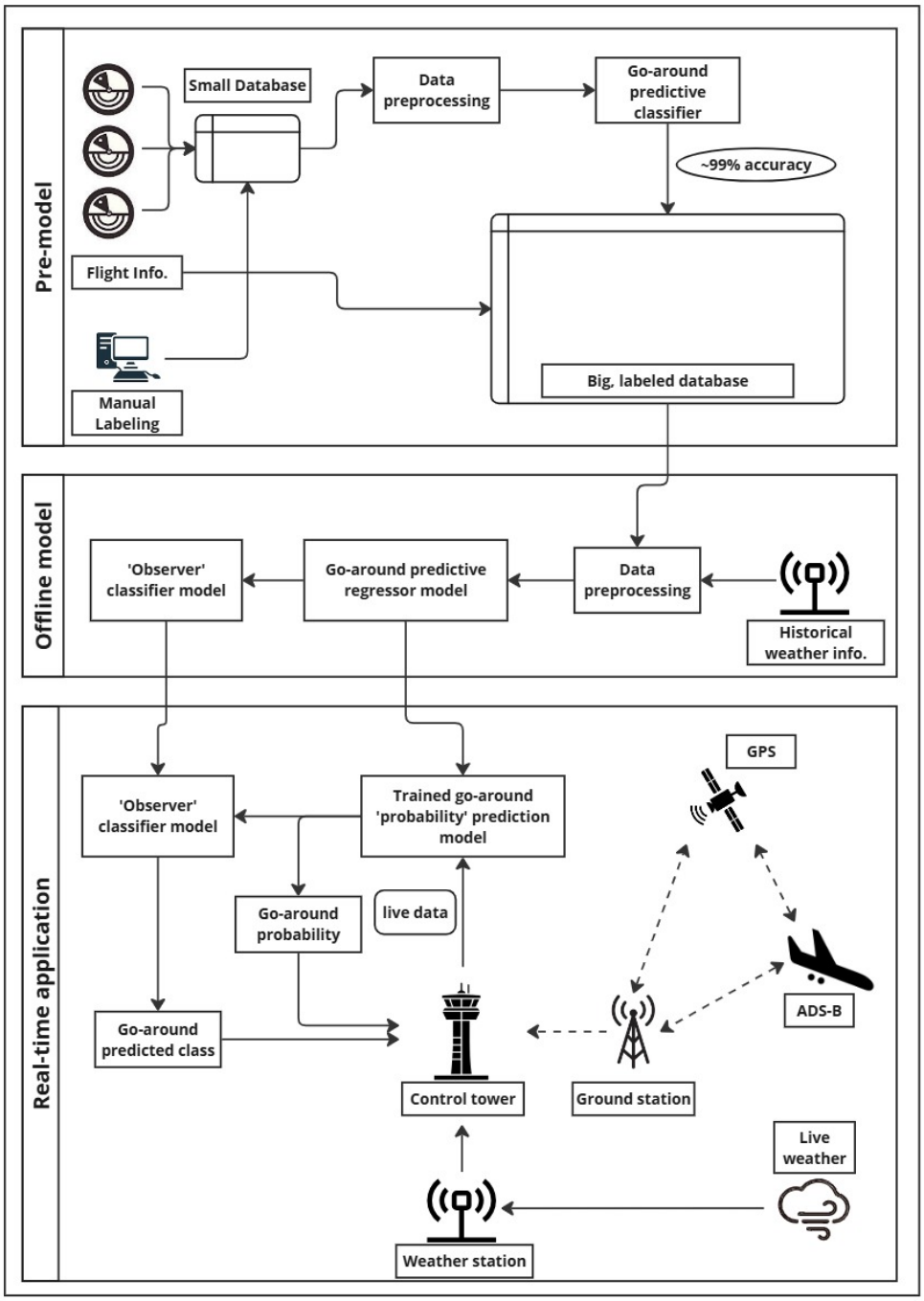


Figure 1.1: Workflow of the proposed project for real-time probability prediction of aborted landings (image inspired by [9])

The study aims to address the following key research questions:

- Can a deep learning model accurately predict the probability of a go-around before it occurs?
- How does the proposed model compare with traditional and tree-based machine learning approaches?
- What are the most influential flight parameters that contribute to go-around decisions?

The remainder of this Master thesis is structured as follows: Chapter 2 outlines the technological basis of the study, including a problem description and a review of state-of-the-art methodologies. Chapter 3 describes the methodology, including dataset selection, preprocessing, exploratory data analysis, and the proposed deep learning models. Chapter 4 details the development, implementation and results of the prediction tool. Finally, Chapter 5 concludes with a final discussion of findings, potential applications, and future research directions. To assist the reader, an acronym table can be found in Annex C and a glossary of important terms in D.

By developing a real-time probability prediction tool for aborted landings, this research contributes to enhancing air traffic control efficiency, improving flight safety, and advancing the integration of machine learning into aviation operations.

Chapter 2

Technological Bases

While aborted landings and go-arounds are among the most critical events that increase Air Traffic Control (ATC) complexity when dealing with two or three airports at the same time [18], little research has been done on tools to help ATCOs visualize data more effectively. Providing them with weather information showed a significant increase in controller efficiency, as shown in [1]. Developing visual tools that highlight aborted landings and accurately predict them would reduce ATCO workload, increasing their awareness at all times, leading to more efficient operations. Main advantages include helping ATCOs to be prepared to stop launching departures during a go-around event and it would therefore provide more time to re-sequence further departures and/or landings.

2.1 Problem Description

From an abstract point of view, the problem to be tackled is called *time series extrinsic regression* (TSER) [35]. It consists of learning the relationship between a time series and a continuous scalar variable. It differs from *time series classification* (TSC) in the fact that TSC maps a time series to a finite set of discrete labels while TSER predicts a continuous value from the time series. Additionally, if the value we are going to predict is a continuation of the input time series, it becomes a *time series forecasting* (TSF) problem.

Little research has been done on TSER. Typical regression algorithms do not work well when applied directly to time-dependent problems because they do not take into account the temporal aspect of the data. Traditionally in Machine Learning (ML), the features used for regression are static and have no relation to time. Our goal is to consider time series data as the features to see how changing data over time affects predictions.

2.2 State of the Art

Research into GA prediction has explored both environmental influences and flight dynamics, leveraging diverse machine learning techniques.

Khattak et al. [20] introduced a deep learning model studying wind shear effects on aborted landings at Hong Kong International Airport, using features such as wind

shear intensity, runway assignment, and shear proximity to the runway. While insightful, their approach may not fully capture the temporal dependencies of sequential flight data.

Dhief et al. [9] developed a tree-based model to predict GA probability using pilot-in-the-loop simulation data from Philadelphia and Van Nuys airports. They advocate for more advanced modelling approaches, supporting our hypothesis that deep learning with real-world data may improve predictive performance.

Dai et al. [6] applied *principal component logistic regression* to identify latent factors across approach and environmental variables, though its linear structure may miss nonlinear dynamics inherent in real-world landing scenarios.

Physiological studies, such as by Ahlstrom and Friedman-Berg [1] and Dehais et al. [7], emphasize pilot workload and scanning behavior, but rely on sensor data that is typically unavailable to ATC. Instead, our work focuses on trajectory and weather data to enable real-time, scalable predictions.

Jiao et al. [17], Lai et al. [21], and Moriarty & Jarvis [27] explored unstable approach detection and mental model mismatches, but none leverage deep sequential architectures to learn predictive patterns directly from trajectory data.

Figuet et al. [12] introduced macroscopic (GAM-based hourly forecasting) and microscopic (ADS-B trajectory classification) models at Zurich. Their microscopic classifier detected only 50% of GAs (7% FPR), indicating room for improvement via deeper sequence-aware models.

A recent study by Liu et al. [23] deployed an *LSTM* model for real-time GA prediction at JFK, integrating in-trail spacing and runway configuration. This work established a strong benchmark for trajectory-based, real-time GA prediction.

Newer literature includes: *"Go-around occurrence prediction with rule-induction, rule evolution"* (2025), applying ML methods on meteorological events and generating sequential GA alerts [25]. Wiley's *"Missed Approach, a Safety-Critical Go-Around Procedure in Low-Level Wind Shear"* (2023) using ensemble learning under low-level wind shear showed high accuracy [19].

In summary, the literature illustrates:

1. Environmental models (wind shear, weather conditions).
2. Linear or tree-based classifiers (e.g., GAM, logistic, random forests).
3. Temporal deep learning (*LSTM* with ADS-B, runway environment).

Our work transforms these strands by:

- Leveraging modern deep architectures (*LSTM* + *attention*, *transformers*) to better capture sequential and contextual dependencies in trajectory and meteorological time series.
- Training and evaluating on real-world data from Barcelona (LEBL) and potentially other European airports, not just simulations or single sites, to enhance model generalizability.
- Incorporating explainability (*SHAP*), enabling ATCOs to understand model drivers

in real-time and bridge the gap between performance and operational transparency.

This positions this Master thesis at the intersection of temporal model innovation, multi-airport realism, and operational interpretability, representing a novel contribution to GA prediction research.

2.3 Theoretical Framework

In this section, we introduce the main machine learning models and concepts that form the foundation of this work. We begin with a review of classical models such as *feed-forward neural networks* and *decision trees*, followed by more advanced sequential architectures like *LSTM* networks and *transformers*. Special attention is given to the role of *attention* mechanisms and ensemble methods, particularly *random forests*, which are used as interpretable observers for the outputs of our main models. These theoretical tools are essential to understanding the design choices and evaluation criteria used throughout the project.

2.3.1 Decision Trees

A *decision tree* [4] is a non-parametric supervised learning algorithm used for classification and regression tasks. It works by recursively partitioning the input space into regions based on feature thresholds, with the goal of minimizing a certain impurity measure at each step.

Each internal node of the tree corresponds to a decision based on the value of a feature, and each leaf node corresponds to a final output prediction. For classification problems, the most common impurity criteria are the Gini index and Entropy. The Gini index for a node with classes $k = 1, ..., K$ is defined as:

$$G = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2,$$

where p_k is the proportion of samples in the node that belong to class k.

The *decision tree* is built by selecting the feature and corresponding threshold that produce the largest decrease in impurity. This process continues recursively until some stopping criterion is met, such as maximum depth, minimum number of samples per node, or zero impurity.

Although individual *decision trees* are easy to interpret, they are prone to overfitting, especially with noisy data. This limitation is precisely what *random forests* aim to overcome by aggregating multiple decorrelated trees.

In this work, *decision trees* serve as the foundational building blocks of the *random forest* observer model, used to classify probability sequences into binary go-around decisions.

2.3.2 Random Forest Classifier

Random forest [3] is an ensemble learning method that builds a large collection of *decision trees* and merges their predictions to produce more accurate and stable results. Each tree in the forest is trained on a random subset of the training data (bootstrapping) and, during construction, chooses the best split among a random subset of features at each node. This randomness introduces diversity among trees, reducing the risk of overfitting and improving generalization.

Mathematically, let $\{T_1, T_2, ..., T_k\}$ be the set of *decision trees* built on bootstrapped datasets $D_i \subset D$, where D is the training set. Each tree T_i returns a prediction y_i for an input x. For classification tasks, the final prediction $\hat{y}$ is obtained via majority voting:

$$\hat{y} = \text{mode}\{T_1(x), T_2(x), ..., T_k(x)\}.$$

In our setup, a *random forest* is used as an observer to interpret the probability sequences output by the main regressor (e.g. *transformer* or *LSTM* model). Each probability sequence is converted into a feature vector, and the *random forest* classifier is trained to output a binary prediction indicating whether a go-around is expected. This setup allows us to decouple the regression task from the final decision-making, offering greater flexibility and interpretability.

Random forests are particularly useful in this context because they are robust to noise, require little parameter tuning, and provide good insight into feature importance. Additionally, they allow for fast inference, which is crucial for real-time applications in air traffic control environments.

2.3.3 Feed-forward Neural Networks

Feed-forward neural networks (FNN), also known as *multi-layer perceptrons* (MLPs), are among the most basic deep learning architectures. Their main characteristic is that information moves in only one direction, forward, from input nodes, through hidden layers (if any), and finally to the output layer. There are no cycles or loops in the network, hence the name “feed-forward”.

Mathematically, an FNN is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ composed of a series of affine transformations followed by non-linear activations. Each layer l computes the following:

$$\mathbf{h}^{(l)} = \phi\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right), \tag{2.1}$$

where $\mathbf{h}^{(l)}$ represents the output of the l-th layer, ϕ is an activation function (typically ReLU, sigmoid or tanh), and $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$ are the weight matrix and bias vector, respectively [13].

While simple in structure, *FNNs* are not well suited for tasks involving temporal dependencies, such as time series data, since they treat all inputs as independent. This limitation makes them a suboptimal choice for predicting go-arounds in aviation, where the sequence of aircraft states over time carries vital contextual information.

2.3.4 Recurrent Neural Networks (LSTM)

Recurrent neural networks (RNNs) are designed to handle sequential data, making them a natural candidate for time series analysis. They introduce a feedback loop in the architecture, allowing them to maintain a hidden state that evolves over time and captures temporal dependencies between inputs.

At each time step t, an *RNN* cell receives an input vector $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$, and produces a new hidden state $\mathbf{h}_t$:

$$\mathbf{h}_t = \phi\left(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b}\right). \tag{2.2}$$

However, traditional *RNNs* suffer from vanishing and exploding gradient problems, making them ineffective at capturing long-range dependencies. To address this, the *long short-term memory* (LSTM) architecture was proposed [15]. It introduces gated mechanisms to control the flow of information and selectively retain or forget past data.

An *LSTM* unit consists of a memory cell $\mathbf{c}_t$, an input gate $\mathbf{i}_t$, a forget gate $\mathbf{f}_t$, and an output gate $\mathbf{o}_t$, computed as follows:

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f\right), \tag{2.3}$$
$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right), \tag{2.4}$$
$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c\right), \tag{2.5}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \tag{2.6}$$
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right), \tag{2.7}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{2.8}$$

where σ is the sigmoid function and $\odot$ denotes element-wise multiplication. This gating mechanism enables *LSTMs* to capture both short-term and long-term dependencies in sequences, which is crucial for modeling the evolving nature of aircraft trajectories and detecting precursors to missed approaches.

In our work, *LSTMs* are used as the foundational recurrent architecture due to their proven effectiveness in sequential prediction tasks, including aviation-related research [23]. Furthermore, we experiment with *attention* mechanisms and Transformer-based models to enhance the ability of the system to capture long-range interactions and contextual dependencies in flight data.

2.3.5 Attention in NN

At its core, an *attention* function is nothing else but a mapping between a query and a set of key-value pairs to an output, being all of them vectors. The output is computed as a weighted sum of the values, where the different assigned weights are given by a compatibility function of the query with the corresponding key.
First, let us see how exactly *scaled dot-product attention* works. Let d_k be the dimension of the queries and keys and d_v be the dimension of the values. The weights of

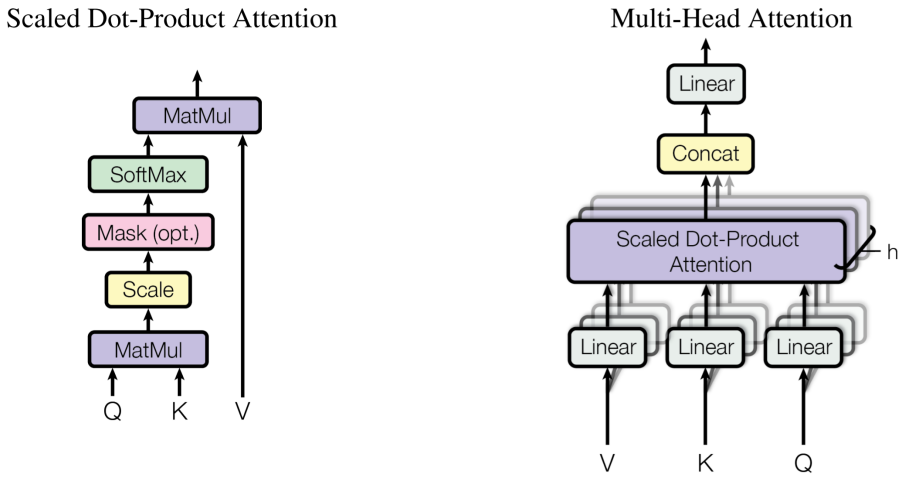


Figure 2.1: Scaled Dot-Product (left) and Multi-Head (right) Attention [36]

the values will be computed as follows:

$$w^{(i)} = \text{softmax}\left(\frac{1}{\sqrt{d_k}}\sum_{k=1}^{d_k} q_k^{(i)} \cdot k_k^{(i)}\right) \quad i = 1, \ldots, n\,, \tag{2.9}$$

where $w^{(i)}$ is the weight associated to the i-th value, $q_k^{(i)}$ is the k-th value of the i-th query, $k_k^{(i)}$ is the k-th value of the i-th key and n is the total number of queries, keys and values. In matrix notation, if Q is the queries matrix, K the keys one and V the values one, we obtain the matrix of outputs by

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.10}$$

Multi-head attention on the other hand consists of several *attention* layers running in parallel. It linearly projects the queries, keys and values h times with different linear projections to d_k, d_k and d_v dimensions respectively. The projections are computed in parallel to each other and return d_v-dimensional outputs, which are concatenated and projected one final time to yield the final values. In matrix notation, this process looks like

$$\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O\,, \tag{2.11}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \tag{2.12}$$

and the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

2.3.6 Transformers

In short, a *transformer* is a deep learning model created by Google, introduced for the first time in 2017 in the paper "Attention Is All You Need" [36]. It revolutionized deep learning by getting rid of recurrence in its entirety, replacing it with exclusively

multi-head attention mechanisms. On the inside, it uses encoder and decoder stacks. The encoder takes a sequence of discrete input tokens $(x_1, \ldots, x_n)$ and maps them to another sequence of continuous representations $\mathbf{z} = (z, \ldots, z_n)$. After this, the decoder will then return an output sequence $(y_1, \ldots, y_m)$ of symbols one element at a time.

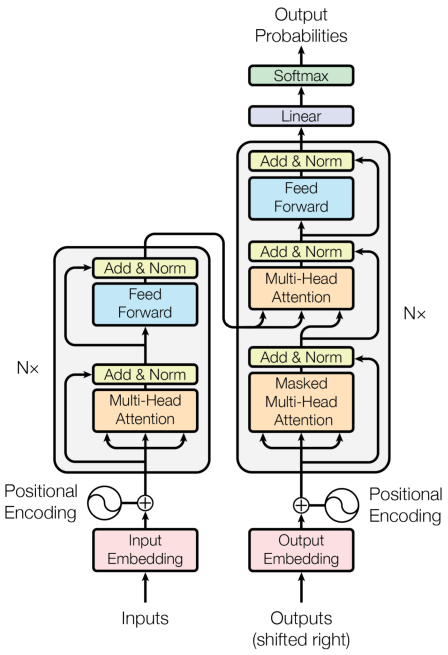


Figure 2.2: Architecture of a transformer [36]

The encoder and decoder can be seen represented in Figure 2.2 in the left and right half, respectively. This is the main structure the transformer follows, using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder.

Encoder and Decoder Stacks

As shown in Figure 2.2, each stack is repeated N times to form an encoder and/or decoder. It is specified that N will take the value 6, meaning encoder and decoder will consist of a stack of $N = 6$ identical layers.
The encoder layers will comprise two sub-layers. The first is a *multi-head self-attention* mechanism and the second is a simple, position-wise fully connected feed-forward network like the ones we have already seen. Additionally, a residual connection is employed around each of the two sub-layers, combined with layer normalization. In other words, each sub-layer output will be $\mathrm{LayerNorm}\,(x + \mathrm{Sublayer}(x))$, being $\mathrm{Sublayer}(x)$ the function implemented by the same sub-layer. In general, outputs will have dimension $d_{\mathrm{model}} = 512$ for the purpose of ease.
In comparison, decoder layers will have a third sub-layer on top of the two we have already seen in encoders. It will perform multi-head attention over the output of the encoder stack. As in the encoder, residual connections and normalization are added around each of the sub-layers. Additionally, the self-attention sub-layer in the decoder stack is modified to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

Chapter 3

Methodology

3.1 Assumptions

In this study, some assumptions were made to facilitate the modeling and analysis of go-around events and landing outcomes:

1. **Sigmoidal-like Probability Evolution of Aborted Landings:** It is assumed that the probability of an aborted landing or go-around follows a sigmoidal-like progression as the aircraft approaches the runway[1]. Initially, this probability remains low, increasing gradually until it reaches 1 for flights that ultimately execute a missed approach. Conversely, for successful landings, the probability is assumed to remain 0 at all times. Further explanation in Section 3.3.2.
2. **Negligible Effect of Earth's Curvature on Distance Calculations:** Given that the study focuses on aircraft trajectories in the vicinity of an airport, it is assumed that the curvature of the Earth does not significantly affect distance calculations in Section 3.3.1. The spatial scales involved are sufficiently small that planar approximations of distance measurements provide accurate results.

3.2 Main Dataset Description

The main dataset that will be used contains information related to entire flights, including a unique flight identifier (flightKey), a non-unique callsign (which may be repeated across different days), the destination airport (ades), and the aircraft's wake turbulence category (wake), among other attributes.

In addition to these flight-level attributes, the dataset also includes time-series data recorded at approximately five-second intervals throughout each flight. These time-dependent variables capture dynamic flight characteristics such as latitude, longitude, altitude, speed, and other relevant parameters.

Furthermore, the dataset includes records of missed approaches (go-arounds). For these events, additional data points are provided, including the geographic coordi-

[1]Notice that we talk about probabilities, but they do not have a mathematical meaning as such. For example, if we at some point get a probability of 10%, this does not mean one in every ten flights with that probability will abort. It acts like a 'score' such that if it becomes too big it will likely perform a go-around.

nates (latitude and longitude) where the go-around was initiated, the exact timestamp of the event, and the aircraft's altitude at that moment.

Table 3.1: Description of Flight Features

Feature	Description
Flight Key	Unique identifier for each flight.
Callsign	Designated callsign used by the flight.
Adep	Departure airport code.
Ades	Planned destination airport code.
Flight Type	Type of flight (e.g., REGULAR, NON REGULAR, etc.).
Wake Category	Wake turbulence category (L, M, H, J).
Aircraft Model	Model of the aircraft used.
Lat	Latitude at a given trace point.
Long	Longitude at a given trace point.
Hdg	Heading of the aircraft at a given trace point.
Altitude	Flight level (FL) at a given trace point.
Vel	Ground speed of the aircraft (in knots).
Vel_z	Vertical speed component (in ft/min).
(missed) Runway	Runway used for landing or missed approach.
Missed Lat	Latitude where the missed approach occurred (if any).
Missed Long	Longitude where the missed approach occurred (if any).
Missed Instant	Timestamp of the missed approach (if applicable).
Missed FL	Flight level at the missed approach point (if applicable).

3.3 Data Preprocessing

The aircraft altitude is often provided as a flight level (FL) by the data. Flight levels are converted to meters using:

$$h = h_{FL} \times 100 \times 0.3048 = h_{FL} \times 0.48. \tag{3.1}$$

This conversion assumes standard atmospheric pressure conditions.

3.3.1 Feature Engineering

Understanding an aircraft's alignment with a runway is essential for assessing landing precision and safety. This analysis involves computing key metrics that capture the aircraft's deviation from the ideal approach path. By engineering specific features we can quantify how well an aircraft aligns with the runway threshold. These metrics provide valuable insights for evaluating flight performance and ensuring safer landings, hopefully easing the model's task.

3.3.1.1 Heading Difference

The heading difference is calculated as the angular difference between the aircraft's heading and the runway heading. For example, runway 24R at LEBL, has a heading of 244°. The heading of the aircraft, provided by its navigation system, is compared to this reference to determine misalignment. The difference is computed as:

$$\Delta hdg = (hdg_p - hdg_A + 180) \mod 360 - 180 \in [-180, 180], \tag{3.2}$$

where hdg_p is the heading of an aircraft's trace point and hdg_A is the heading of the runway.

3.3.1.2 Distance to Runway and Angle of Descent

Two distance calculations are performed. **Ground Distance** to Runway is the horizontal distance from the aircraft's position to the extended centerline of the runway. It is calculated using a geodesic distance formula based on latitude and longitude.

Proposition 1 *Given two points P_1, P_2 on a sphere, given by their latitude ϕ_1, ϕ_2 and longitude λ_1, λ_2, respectively, the geodesic distance between both points can be calculated via the haversine formula:*

$$d_{hav} = 2R \times \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \times \cos(\phi_2) \times \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right), \tag{3.3}$$

where $\Delta\phi = \phi_2 - \phi_1$ and $\Delta\lambda = \lambda_2 - \lambda_1$.

Proof 1 *A proof for this formula can be found in [34].*

We also calculated the three-dimensional **Euclidean distance** between a trace point and the runway threshold. For this, we assumed the plane would be at a small enough distance that the curvature of the earth would have little to no impact. By doing this, we were able to apply the Pythagorean theorem:

$$d = \sqrt{h^2 + d_{hav}^2}, \tag{3.4}$$

where h is the aircraft's height, obtained in 3.1 Additionally, applying basic trigonometry, we obtain the **angle of descent** (α) as follows:

$$\alpha = \arctan\left(\frac{h}{d_{hav}}\right). \tag{3.5}$$

These calculations were also performed for distance and angle with respect to to the runway end. A visual representation can be seen in Figure 3.1, represented by d_hav - Haversine (Ground) distance, d - Euclidean distance, h - altitude, α - angle of descent.

3.3.1.3 Deviation from Runway Centerline

As it may result difficult for a neural network to interpret if a plane is off trajectory, we introduced two variants for measuring deviation from the imaginary extended centerline of the runway. On the one hand, we calculated **lateral deviation**, in meters,

corresponding to the perpendicular distance from the aircraft's position to the centerline. Again, for simplicity, it was assumed that distances would be too small for the earth's curvature to have any major impact. To project the aircraft's position onto the runway centerline and measure the shortest distance we first calculated the Easting and Northing differences between the aircraft's position and the runway threshold as $x_0 = (lng_p - lng_A) \times 111.194 \times \cos(lat_1)$, $y_0 = (lat_p - lat_A) \times 111.194$, respectively. Then the lateral deviation can be calculated as follows.

$$\Delta align = |x_0 \sin\theta - y_0 \cos\theta|, \tag{3.6}$$

where θ represents the standard mathematical angle (counterclockwise from the positive x-axis, which is East) corresponding to the runway heading. Based on the same principle as Eq. 3.2, we can calculate $\theta = (90° - hdg_A) \mod 360°$.

On the other hand, we calculated the **angular deviation**, so that the same lateral deviation is 'punished' if it is near the runway, as the angle will grow. The formula for this measure is

$$\Delta\varphi = \arccos\left(\frac{x_0 \cos(\theta) + y_0 \sin(\theta)}{\sqrt{x_0^2 + y_0^2}}\right). \tag{3.7}$$

Visually, Figure 3.1 represents these variables, shown as Δalign - Lateral deviation, $\Delta\varphi$ - Angular deviation.

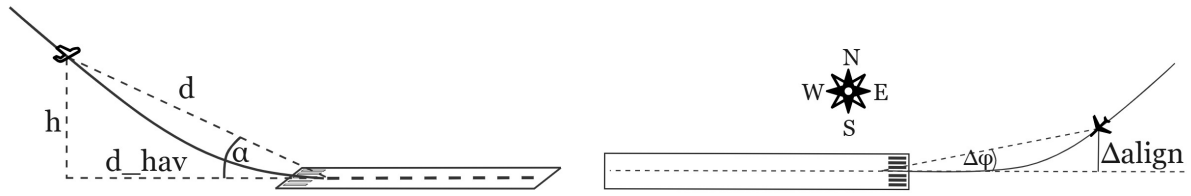


Figure 3.1: Visualization of the calculated features.

3.3.1.4 Aircraft Specific Energy

In the context of aborted landings, the aircraft's **specific energy** plays a crucial role in predicting whether the aircraft can safely stop in a runway's length. Since the aircraft has no brakes in the air, it must intentionally lose energy (speed and/or altitude) by flying in circle-like shapes or increasing drag. We assume that if the energy is too high at the time of landing, the aircraft will have to perform a go-around. The specific energy is the sum of the potential and kinetic energies per unit weight

$$E_s = h + \frac{v^2}{2g}, \tag{3.8}$$

where h is the altitude in feet (Flight Level $\times 100$), v is the ground speed in knots (converted to feet per second using $1\,\text{knots} = 1.68781\,\text{ft/s}$), g is the gravitational constant, $g = 32.174\,\text{ft/s}^2$.

3.3.1.5 Distance to Preceding Arrival

As will be seen in Section 3.6, the second most common reason for a go-around is a lack of separation with the previous plane. This is especially concerning if the

previous aircraft is big and therefore creates a lot of turbulence, known as 'wake'. That is why it is necessary to find out, at each trace point, what the distance to the previous arrival is. To do this, we will see in which order the flights cross the 4NM point (a point far away enough so hardly any flights have already aborted and close enough that the planes are already aligned with the runway). Doing so, we are able to establish an order of landing for each runway, and can therefore calculate the distance from each plane to its preceding aircraft at each trace point. If there is no preceding aircraft or it has already landed and stopped emitting GPS signals, the distance to it will be said to be 999km, which should signal the model not to take it into consideration.

3.3.2 Labeling

There is no doubt that this is a supervised learning task. However, we lack labels for each training instance. This is because we only have whole flights labeled as 'aborted' or 'successful' approach. Sometimes there is even more than one missed approach in a same flight, but every one of them ends with a successful landing. This raises the doubt of what label we should assign to each training instance, may it be a single trace point or a sequence of them.

The most straight-forward approach is to label each trace point of a missed approach as aborted - 1, and as not aborted - 0, elsewise. However, this seems kind of counter-intuitive if we think of the build-up behind a missed approach. They typically happen in the middle of the final approach of a plane, and it is normally not clear, especially in the more previous points. Therefore, a less rudimentary approach is proposed and has already been seen in some papers, like in [9], where they assumed a linear growth of the abort probability, up to the point of the go-around. This, however, seems artificial and unlikely, so we also tried for a more sigmoidal-like approach, where the initial probability will be 0 and will rise very slowly, at some point spiking to values close to 1 and remaining there throughout the go-around, as shown in Figure 3.2. We will see how each modeling performs and chose the more appropriate one at the end.

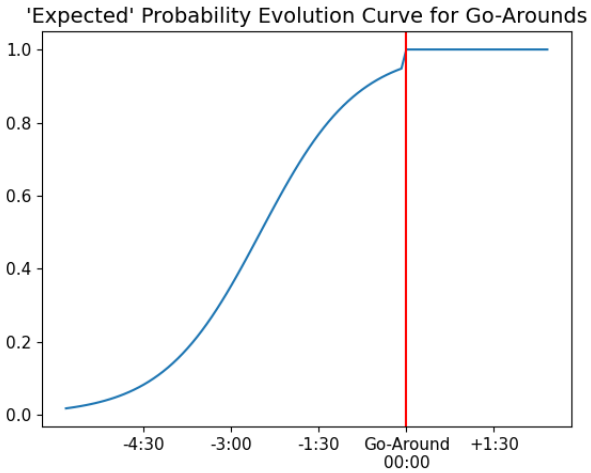


Figure 3.2: Expected Probability Evolution of Missed Approach

3.3.3 Sequence Extraction

It is crucial to determine a meaningful sequence length we are going to feed the model. This is, how many trace points prior to a missed point and/or a successful landing. If we give the model too many points in advance, it will not be able to focus on the imminent deviations and relations that lead to a missed approach. However, if we do not provide enough points in advance to the model, it will not be capable of detecting them, as they would not appear in the training set at all.

To determine what an appropriate sequence would look like, we first have to understand some concepts in aviation. To land, a plane has to perform an approach, a process that can be subdivided into different stages. Figure 3.3 schematically shows them. A typical approach begins at the *Initial Approach Fix* (IAF), which marks the point where the aircraft transitions from the en-route phase to the approach procedure. At this point, the aircraft is usually vectored toward the airport and begins to align itself with the intended landing path. Following the IAF, the aircraft reaches the *Intermediate Fix* (IF), where descent and lateral alignment are further refined. The path becomes more constrained and the aircraft begins to adopt the correct speed and configuration for landing.

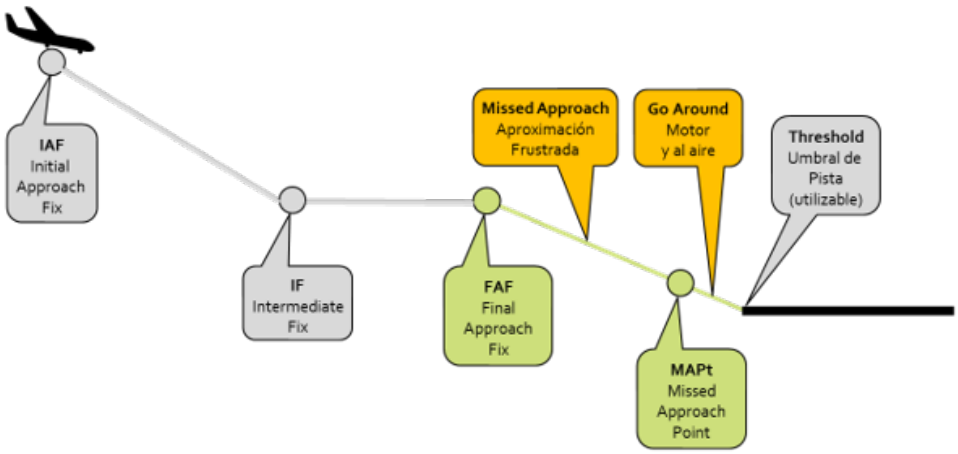


Figure 3.3: Different Stages of Approach in Commercial Aviation

As it continues, the aircraft arrives at the *Final Approach Fix* (FAF). This is a key moment in the approach: from here, the aircraft should be fully stabilized, both in vertical and lateral guidance, and begin a continuous descent toward the runway threshold. The approach segment that follows is known as the final leg, and it ends at the *Missed Approach Point* (MAPt), a predefined location in time or space by which the pilot must have the runway in sight (in the case of non-precision approaches), or decide to abort the landing if visibility or safety criteria are not met.

This go/no-go decision is guided by predefined minimums: either the Decision Height (DH), used in precision approaches, or the Minimum Descent Altitude (MDA), used in non-precision approaches. These values define the lowest altitude a pilot can descend to without visual confirmation of the runway environment. If the runway is not in sight at or before reaching this altitude, a missed approach must be initiated.

It is important to distinguish between two commonly used terms: *go-around* and *missed approach*. While both refer to an aborted landing, they are not exactly the

same. A missed approach is a predefined, procedural response triggered at the MAPt, as outlined in the approach chart. A go-around, on the other hand, is a more general maneuver that can be initiated at any moment during the approach, even after touchdown, usually due to safety concerns like unstable trajectory, late clearance, or runway incursions.

In this work, however, we will not differentiate between these two. Whether the maneuver occurs before or after the MAPt is irrelevant for our purposes: we are only interested in identifying the fact that the aircraft did not land, regardless of the technical or procedural label it receives. This choice allows for a more unified treatment of the dataset and is justified by the observation that, from a trajectory standpoint, both events exhibit similar patterns: a stable descent, followed by a sudden pitch-up and climb-out.

3.4 Exploratory Data Analysis

In this section we will be diving into the data, examining the distribution of the different features and preparing it for usage. This step is crucial, as nonuniform data can lead to a model that does not predict accurately. We have decided that it is best to let a model be airport-specific. That is, a model will be specialized on predicting missed landings in a certain airport. This is why it is important to understand how much data we have.

In Table 3.2 we can see how many classified missed flights (n) we have in each of the major airports. Based on this information, we will develop tools for prediction specific for the LEBL airport, although once it has been done, it can be adapted for any other airport of choice. From this point forward, unless specified otherwise, data comes from LEBL.

ICAO	n	Full Airport Name
LEBL	2685	Aeropuerto Josep Tarradellas Barcelona-El Prat
LEMD	2404	Aeropuerto Adolfo Suárez Madrid-Barajas
LEPA	2009	Aeropuerto de Palma de Mallorca
GCTS	1701	Aeropuerto de Tenerife Sur
GCXO	1658	Aeropuerto de Tenerife Norte-Ciudad de La Laguna
LEMG	1294	Aeropuerto de Málaga - Costa del Sol
LEJR	1093	Aeropuerto de Jerez
LEZL	1013	Aeropuerto de Sevilla
...	...	...

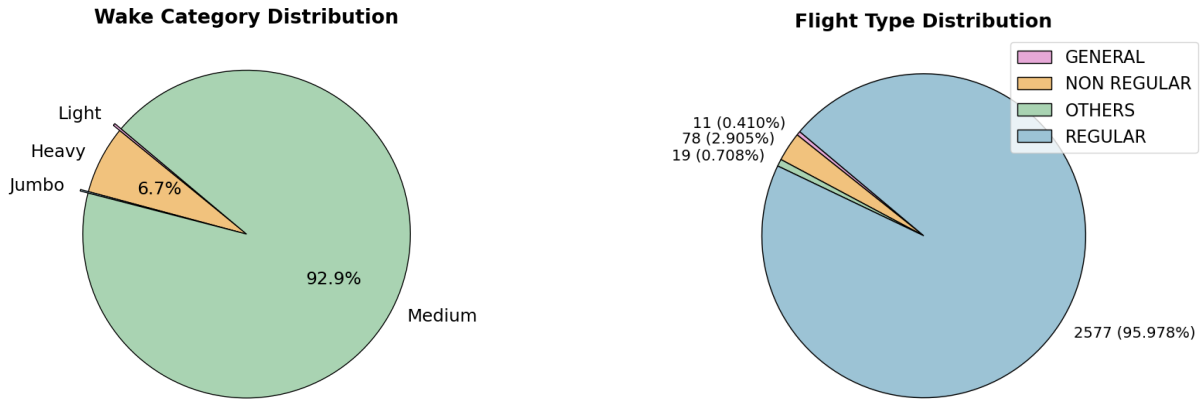
Table 3.2: Number of missed flights by airport

Wake and Flight Type

A small plane approximation may differ significantly from a bigger airliner, so we should check how these cases are reflected on the data. To tackle this question

we check the distribution of the wake category (RECAT) in missed flights. Wake turbulence is a function of an aircraft producing lift, resulting in the formation of two counter-rotating vortices trailing behind the aircraft. The intensity of the vortices is proportional to the size of the aircraft.

Examining the frequency of appearance in the missed flights dataset of the different wake categories, we obtain the pie chart shown in Figure 3.4a. Note that this data comes from LEBL, as specified earlier. However, similar relations are present in most commercial airports. We can observe that most planes landing have wake class 'Medium'. This means aircraft of weights of 7,000 kg to less than 136,000 kg. Very common aircraft of this category are *Airbus*'s A320 and *Boeing*'s 737 [16]. Due to the huge unbalance of the dataset and the majority of the data corresponding to 'Medium' wake aircraft, we have decided to make a model specific for this kind of aircraft, as it is the most common. This does not mean that the model will not be able to make predictions for other aircraft categories, rather that they will not be as accurate.



(a) Wake Category Distribution

(b) Flight Type Distribution

Figure 3.4: Distribution of the features 'wake' and 'flightType'

On the other hand, if we observe the pie chart in Figure 3.4b, we observe the distribution of the 'FlightType' label. This refers to the type of flight (General, non regular, regular or other). Regular flights refer to flights that repeat with a certain periodicity, like a Madrid-Barcelona every day at 18:00. Non regular flights are those that do not follow these patterns, but otherwise are quite similar. General flights refer to non-commercial business aviation, instructional flying, pleasure flying, aerial work (agriculture, construction, surveying, photography, advertisement, SaR, etc.).

Finally, other flights contains flights that do not fall into the previous categories, like military flights. We do not wish to include military flights or ones that could use some uncommon planes or flight paths, so we can remove the categories GENERAL and OTHERS, as they represent a very small part of the database.

Runway Distribution & Runway Dependency

It is also interesting to examine the distribution of missed flights across different runways. Figure 3.5 shows this spread for missed flights in LEBL airport.[2] The first thing that catches the eye is that more than half of the missed approaches happen in runway 24R. However, this is not necessarily because of a difficult approach or some other reason that might make landing on this runway more difficult, but rather the fact that 67% (seen in Table 3.3) of landings are directed to 24R. In fact, it has the lowest calculated probability of aborted landings. This is expected, given that it is the longest and widest runway at LEBL, making it the preferred option for landings. We can also see that, in general, some runways are not usually used for landings.

For example, runway 20 only has one recorded successful landing in the last 11 years (and 79 missed landings). This could be due to orientation and/or flight over residential areas, so they are normally avoided[3]. If a plane were to approach runway 20 for landing, it will most likely be redirected by ATCO's instructions.

Runway	**Safe Landings**	**Go-Arounds**	P(Aborted | Runway)
02	209133	537	0.002561
20	1	79	0.9875
06L	224082	307	0.001368
06R	500	96	0.1611
24L	56036	143	0.002545
24R	1015823	1305	0.001283

Table 3.3: Runway Distribution at LEBL

The question arises if we should develop a model specific to each runway, thus further reducing our data quantity. On the one hand, if we take a look at the side profiles of some landings, grouped by the target runway in Figure 3.6, it shows that there could potentially be some underlying pattern that indicates runway-dependency (landings in 06L seem to fly lower than 24R).

On the other hand, to be sure, we performed a hypothesis test for correlation between the runway and missed label. The first step was to perform a ratio comparison test like the *Chi-square test of Independence* for all runways. It would have as null-hypothesis H_0 : All runways have the same ratio of aborted landings vs. H_1 : at least two runways have different ratios of aborted landings. This test gave a p-value of 0, indicating that at least two of the runways have different abort probabilities. This is no wonder, as runway 20 has a probability of around 0.9875 (likely due to instructions from ATCO). However, this p-value stands even when repeating the test pairwise, with the main runways for landings (02,06L,24L,24R) and adjusting with the Bonferroni correction for 15 tests ($\alpha = 0.05/15 = 0.0033$).

[2]Due to the change in magnetic declination on the 24th March 2022 Barcelona (LEBL) changed the designator of two of its runways. Old 07/25L and 07/25R became 06/24L and 06/24R. The labels have been corrected for this visualization and for the rest of the project.

[3]Runway 20 figures as 'Not available for landing' in the AIP documentation for LEBL [2].

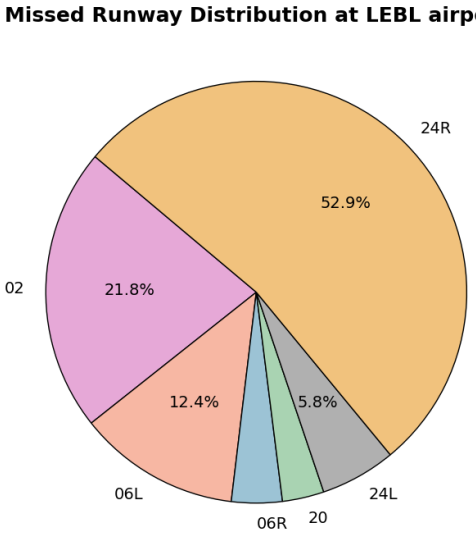


Figure 3.5: Missed Runway Distribution at LEBL

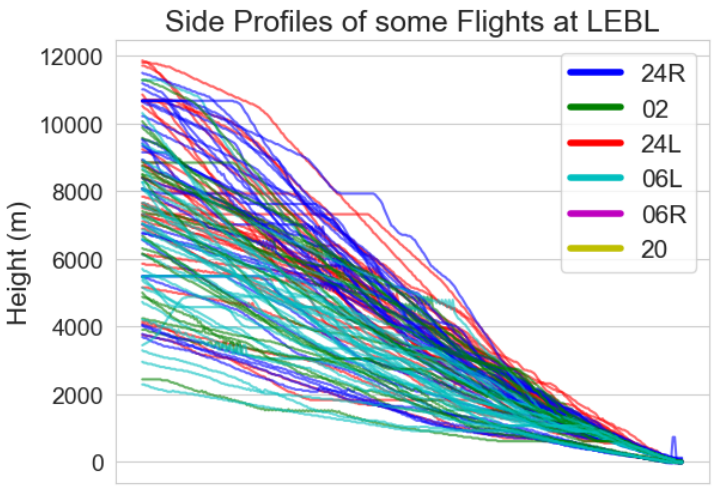


Figure 3.6: Side Profiles of Some Safe Landings at LEBL

It is then interesting to detect which runways significantly differ from each other. Looking at Table 3.4, we can separate runways 02 and 24L (no significant difference between them) from runways 24R and 06L (which also do not show any difference). This makes sense, as runway 24R/06L is the biggest and widest of them, allowing for a higher tolerance on path deviations.

Because of all the calculated features, that abstract the problem from a physiscal runway, we have decided to use one model for the whole airport. However, we have just proved that it is convenient to inform the model on which runway is being used. Because the runway is a categorical nominal variable, we have to encode, associating each runway to a number. This assignment is not a trivial matter, as the model could perceive some sense of order or ranking among the runways. What we will do is take the calculated probabilities of aborted landings on each runway from Table 3.3 to rank them, from lowest to highest. Following this method, we end up with the order: 1 - 24R; 2 - 06L; 3 - 24L; 4 - 02; 5 - 06R; 6 - 20.

Table 3.4: Pairwise p-values for Go-Around Rate Comparisons between Runways

Runway	02	06L	06R	20	24L	24R
02	–	4.84×10^{-19}	0.000	0.000	**0.948**	4.25×10^{-43}
06L		–	0.000	0.000	4.49×10^{-10}	**0.310**
06R			–	1.51×10^{-56}	0.000	0.000
20				–	0.000	0.000
24L					–	2.09×10^{-15}
24R						–

Flight Filtering

In order to obtain the best results, it is necessary to filter the rest of the flights. After a thorough examination, the most common occurrences to be addressed are:

- More than one missed approach
- Mislabeled flight destination and/or airport of missed approach
- Erratic flight paths due to GPS errors

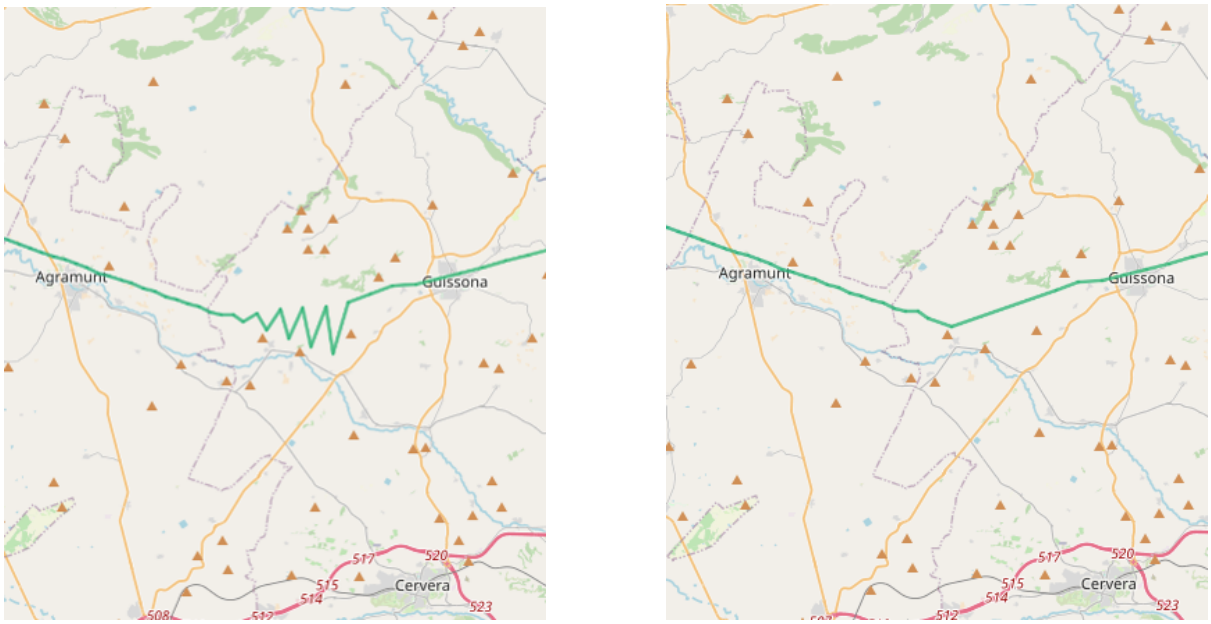
To take care of the first problem, we examine how many flights actually have to perform multiple go-arounds. In fact, less than 2.5% of flights show more than one missed approach, so if this is the case, we can afford to consider only the first one for simplicity's sake.
It also appears that some flights are labeled as 'missed at LEBL', but they not only do not perform a go-around, they do not even land at LEBL at all. To detect these kind of flights, we discard those which do not have at least one trace point in a 20km radius around the airport in question.

Flight Path Smoothening

It is quite common to see flight paths that present erratic patterns like the one in Figure 3.7a, or a trace point that is very far away from the rest (Example: a flight over Madrid suddenly has a trace point in Marrakech and comes back, obviously it is an error). If these irregularities appear during cruising there is no problem, but if they happen during an approach it can really mess with our model's perception.

However, we cannot just address the distance between two consecutive trace points, as flights from/to the Canary Islands fly over Moroccan airspace, so there are gaps in the trace in our database. However, if we calculate the mean speed of the aircraft between two consecutive trace points, we can detect these anomalies, as they would present abnormally high speeds. Doing this, we can smooth the path of an aircraft, as can be seen in Figure 3.7.

In some works like Figuet's [12], they used a Savitzky-Golay filter [32] to smooth flight paths, but we considered it could oversee crucial deviations, specially in the last stages of the approach.



(a) Erratic Trace (b) Smooth Trace

Figure 3.7: Example of smoothening a flight path

3.5 Meteo

It is no wonder that meteorological phenomena can significantly impact the difficulty of a landing. Head winds, for example, can even make it easier to land, as the plane can maintain lift much easier with less ground speed. However, crosswinds or tail wind can make landing very difficult, sometimes leading to go-arounds. That is why it is important that we include meteorological data in our study.

For this, we extracted data from the Open-Meteo API [38]. This service aggregates high-resolution weather data from several authoritative sources, including the German Meteorological Service (DWD) [8], the European Centre for Medium-Range Weather Forecasts (ECMWF) [11], and the US National Weather Service (NWS) [28].

The API provides historical, real-time, and forecast weather variables, all accessible with spatial and temporal precision suitable for aviation applications. Given the sequential nature of our input data and the need to synchronize meteo features with each trace point, we restricted our query to wind-related variables (direction and speed) at the surface level, specifically, at the coordinates of the runway threshold. More complex atmospheric data such as fog, visibility, or wind at higher altitudes was discarded for the time being due to either its inconsistent availability or insufficient temporal resolution.

3.6 eDina

Go-arounds can be triggered by a wide range of factors. Studies show that the probability of an aborted landing is strongly influenced by flight separation, approach stability, the presence of departing aircraft, and the altitude of the aircraft above the runway [6]. Among these, unstable approaches are particularly critical, as they have

been identified as major contributors to commercial aviation accidents during the landing phase [17, 21, 27]. A stable approach typically requires compliance with specific criteria regarding configuration, speed, and flight path alignment. When these criteria are not met, the approach is classified as unstable, significantly increasing the likelihood of a go-around or runway incident. Furthermore, an analysis of pilot performance and visual scanning behavior during go-arounds revealed that nearly two-thirds of pilots made errors, often involving substantial deviations from the intended flight path [7].

Of course, we want to check these statements by ourselves. eDina is a repository that ATCOs use to submit reports on aborted landings. Here, they note down the reason for aborted landings, sometimes also an explanation of why it happened or what went wrong. They will note down the callsign of the plane that aborted, the approximate time of the go-around, and will select from a drop-down list the cause that best suits the situation (options can be seen in Table C.1, found in Annex A). Additional information like the runway, model of the aircraft, a small explanation, etc. are optional. In the data provided, the callsign contained typos, so an extensive corrective task was needed to correct all info and cross-check with a database containing all historical flight info.

3.7 Data Balancing

Missed approaches are a rare occurrence, only about 0.2% of approaches have to perform a go-around maneuver. Due to this, we cannot use the flights from the last 6 years for example as a dataset. It would be very unbalanced, leading to unacceptable biases. This is why we decided to perform an undersampling of successful landings to match the number of missed approaches. We will also experiment to see if the model performs better with a slightingly unbalanced database, as it may result in a less conservative prediction and thus enhance differences.

3.8 Proposed Models

In order to estimate the likelihood of a missed approach at each trace point, the problem is formulated as a time series extrinsic regression task. The input consists of a sequence of engineered features derived from the aircraft's trajectory, and the model is expected to return a probability score $p_t \in [0, 1]$ for each time step t.

Due to the sequential nature of the data, we explored several architectures capable of capturing temporal dependencies across trace points. The models evaluated fall under the family of deep neural networks, specifically those designed for time-dependent inputs.

3.8.1 Recurrent Neural Networks

The first models considered were Long Short-Term Memory networks (LSTM), a recurrent architecture that extends vanilla RNNs by incorporating memory cells and gating mechanisms. These models are particularly well-suited for processing sequences with long-term dependencies, making them a natural candidate for trajectory analysis.

Each trace point is represented as a fixed-length vector of features, and the sequence of vectors is passed through several stacked LSTM layers. The network outputs a probability at each time step, which can be interpreted as the model's belief of whether a go-around is imminent, given the past trajectory.

3.8.2 LSTM with Attention Mechanism

To enhance the performance of the LSTM, we then introduced an attention mechanism. The main motivation behind this modification is to allow the model to focus selectively on the most relevant trace points in the sequence when making a decision.

Instead of relying solely on the final hidden state of the LSTM, the attention layer computes a weighted sum over all hidden states, effectively allowing the model to extract information from any part of the input sequence. This results in more flexible and potentially more accurate probability outputs, especially in cases where the signal is not strictly local in time.

3.8.3 Transformer Architecture

Finally, we experimented with Transformer-based architectures. By eliminating recurrence altogether and relying on self-attention to capture temporal relationships between elements in the sequence, Transformers become highly parallelizable and particularly effective at modeling long-range dependencies.

In our implementation, each sequence of trace points is passed through several Transformer encoder layers. The output is a sequence of hidden representations, one per time step, which are linearly projected to obtain the final probability values. This model architecture forms the core of our go-around predictor, given its balance between expressiveness, stability, and scalability.

While all three architectures are designed to handle sequential inputs, the way they capture temporal patterns varies substantially. The LSTM maintains a memory of the past through recurrent connections, the attention-based LSTM improves this by allowing soft selection over all time steps, and the Transformer models all pairwise interactions in parallel through multi-head attention.

3.9 Evaluation

Evaluating the model performance is not as straightforward as calculating a simple accuracy score. Since the model outputs a probability at each trace point, we are faced with the challenge of interpreting a sequence of probabilities without clear-cut thresholds. That is, although the model might output something like $[0.01, 0.03, 0.08, ..., 0.97, 1.00]$ for a missed approach, these values do not have an absolute meaning on their own.

The probability scale is entirely learned and might not align linearly with the concept of "certainty." What we are interested in, then, is the evolution of that signal and how it anticipates the missed landing.

As a first approach for an evaluation metric we introduced the trivial metric, that compares a predicted probability with the 'ideal' outcome, which is binary (1 if the

flight is going to abort, 0 else). It will be called DTI (Distance To Ideal) computed as

$$DTI = \frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{n} \left| \hat{y}_i^{(j)} - y^{(j)} \right|, \tag{3.9}$$

where n is the length of the flight sequence presented to the model, N is the test set size, $\hat{y}_i^{(j)}$ represents the predicted probability of the model at time step i of the j-th flight and $y^{(j)}$ is the binary label (1 if the flight is going to abort, 0 else) of the j-th flight.

To further gain insight into the evaluation, we introduced a second model, an observer, if you will. A Random Forest classifier was trained to interpret the temporal evolution of these probabilities and decide, at a given trace point, whether the flight will eventually miss the approach. With this setup, we are able to transform a continuous-time probabilistic signal into a sequence of binary decisions, one per time step, with the objective to provide both to the ATCO in order to aid with interpretation.

3.9.1 Accuracy Over Time

The Random Forest observer was trained on sequences of probabilities, each associated with the true outcome (missed or successful landing). For each timestamp t, we can extract the TP, TN, FP, FN counts, establishing a confusion matrix at each time step. With this information, we can then calculate the accuracy (together with other metrics) with the usual formulas. This is, how well it predicts the outcome based solely on the partial probability sequence up to time t.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{3.10}$$

$$Precision = \frac{TP}{TP + FP}, \tag{3.11}$$

$$Recall = \frac{TP}{TP + FN}. \tag{3.12}$$

Plotting these metrics over time gives us a clear view of how early and how confidently we can anticipate a go-around. Ideally, the observer should start detecting a potential go-around early, and its confidence (reflected in accuracy) should grow as the aircraft gets closer to the missed point.

3.9.2 Area Under the Accuracy Curve

We define a new custom metric: the Area Under the Accuracy Curve (AUAC). This metric is not to be confused with the well-known AUC (Area Under the ROC Curve), which measures classification performance across probability thresholds. Instead, AUAC quantifies how early and consistently accurate a model is in predicting a missed approach, reflecting the intuition that earlier correct predictions are more operationally valuable.

Let acc_t be the accuracy of the model at time step t, and let T be the total number of time steps leading up to the missed approach. The AUAC is defined as the area under the curve formed by plotting acc_t over time $t = 1, 2, \ldots, T$.

To compute this area, we apply the trapezoidal rule, which approximates the integral by dividing the area under the curve into trapezoids rather than rectangles.

$$\text{AUAC} = \sum_{t=1}^{T-1} \frac{(acc_t + acc_{t+1})}{2}. \tag{3.13}$$

This calculation effectively averages the accuracy at consecutive time steps and multiplies by the time step size. This metric rewards not just eventual accuracy, but also how early the model becomes reliably accurate.

Its value is bounded above by T and serves as a proxy for both early and reliable detection. In our setup, $T = 70$, corresponding to the last 70 trace points before the go-around (approximately the final 6 minutes). A perfect model would then achieve AUAC $= 70$. Of course, a higher *AUAC* does not necessarily imply a better model, as a model could achieve better accuracy with less notice, but make very bad predictions with more notice. However, it serves as a metric to establish an approximate ranking. The final decision on models to use should balance high accuracy and the time in advance the model is able to predict.

3.9.3 Confusion Matrices and TP,TN,FP,FN rates

Also, the False Positive/Negative rate can be assessed using confusion matrices and/or ROC curves. However, these can only be applied to predictions at a certain time step rather than capture the temporal evolution of the predictions. That is why they will be provided for different, dedicated steps over time. For a more continuous visualization of the TP,TN,FP,FN, plotting their rates seems to be more appropriate. The TP and TN rates are expected to grow and the FP and FN rates to decrease over time.

In addition, we will also calculate the PPV (a.k.a. Precision, see 3.11) and NPV rates to answer the questions: 1. “Of all the cases predicted as positive, how many were actually positive?” and 2. “Of all the cases predicted as negative, how many were actually negative?”.

$$NPV = \frac{TN}{TN + FN}. \tag{3.14}$$

The selected split for our data set is train-validation-test. as it enables us to present the model with unseen data in the test set for a final evaluation, while allowing us to train the observer with validation data. So, the DTI-score will be provided for the validation and test set, while the AUAC-score and rates over the confusion matrices will only be calculated in the test set.

In order to avoid biases, we will shuffle the flights presented to the model. Additionally, it is really important to present the data in order when testing, to avoid forward-looking data sequences that lead to misleading scores.

3.10 SHAP Analysis

With the increasing complexity of machine learning models, interpretability has become a significant challenge. Many algorithms function as ’black boxes’, performing

intricate computations and yielding outputs without transparent reasoning. Understanding the internal decision-making process is crucial for identifying key contributing factors and ensuring model reliability. This presents a trade-off between accuracy and interpretability.

SHAP (SHapley Additive exPlanations), introduced by Lundberg and Lee [24], offers a unified framework for interpreting model predictions. It leverages concepts from cooperative game theory, specifically Shapley values, to assign each feature an importance value for a particular prediction. This approach has demonstrated improved performance and alignment with human intuition compared to previous methods.

Fundamentally, SHAP values quantify the contribution of each feature to the difference between the actual prediction and the average prediction across the dataset. For a given instance, the model's output can be expressed as:

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i,$$

where ϕ_0 is the base value (mean model output over the training data), M is the number of features, and ϕ_i represents the SHAP value for feature i, indicating its contribution to the prediction for instance x.

To visualize the impact of features across multiple predictions, the beeswarm plot is employed. This plot displays each feature on the y-axis and the corresponding SHAP values on the x-axis. Each point represents an individual prediction, with color indicating the feature value (e.g., red for high, blue for low). The distribution of points illustrates both the magnitude and direction of a feature's impact, providing insights into feature importance and interaction effects.

In the context of our study, SHAP analysis enables us to interpret the predictions of our deep learning models for go-around events. By examining the SHAP values, we can identify which features most significantly influence the model's decisions, thereby enhancing the transparency and trustworthiness of our predictive system.

Chapter 4

Development and Results

In this chapter, we will describe the development process of this project. For the most part, the implementation was done using Python 3.12.7 and it has been properly noted when using external packages. The results are summarized and visualized concisely for clarity.

4.1 Data Preprocessing

This section is a walk through the data processes that required additional work and was not already explained in Chapter 3.

4.1.1 eDina

As explained in Section 3.6, there was an extensive corrective task regarding the data tables received from eDina containing the reasons behind aborted landings. This is because, on the one hand, they lack a unique identifier and cannot be quickly identified in our internal database. This is the main task we want to address. On the other hand, however, because of how the data is collected, callsigns, aircraft models and other information pieces often contain typos. For example, instead of 'ABC361Z', the registry may show 'ABC3610'. This makes it more difficult to contrast databases and more than once requires manual examination, making it a resource-intensive task.

Figure 4.1 outlines the logical procedure implemented to clean and cross-reference the database with valid flight identifiers. The first step consisted of assigning a unique identifier to flights whose callsign matched one already present in our dataset. In cases where a given callsign was associated with exactly one go-around on that same day, it was safe to assume we had correctly identified the flight. On the other hand, if the callsign did not exist in the database, it was likely incorrect.

In most of these cases, the erroneous callsign differed from the correct one by at most two character. For this reason, we computed the Levenshtein distance [22] between callsigns, defined as the minimum number of single-character edits (insertions, deletions or substitutions) required to transform one string into another. If there existed a callsign in the missed approach dataset with a Levenshtein distance less than or

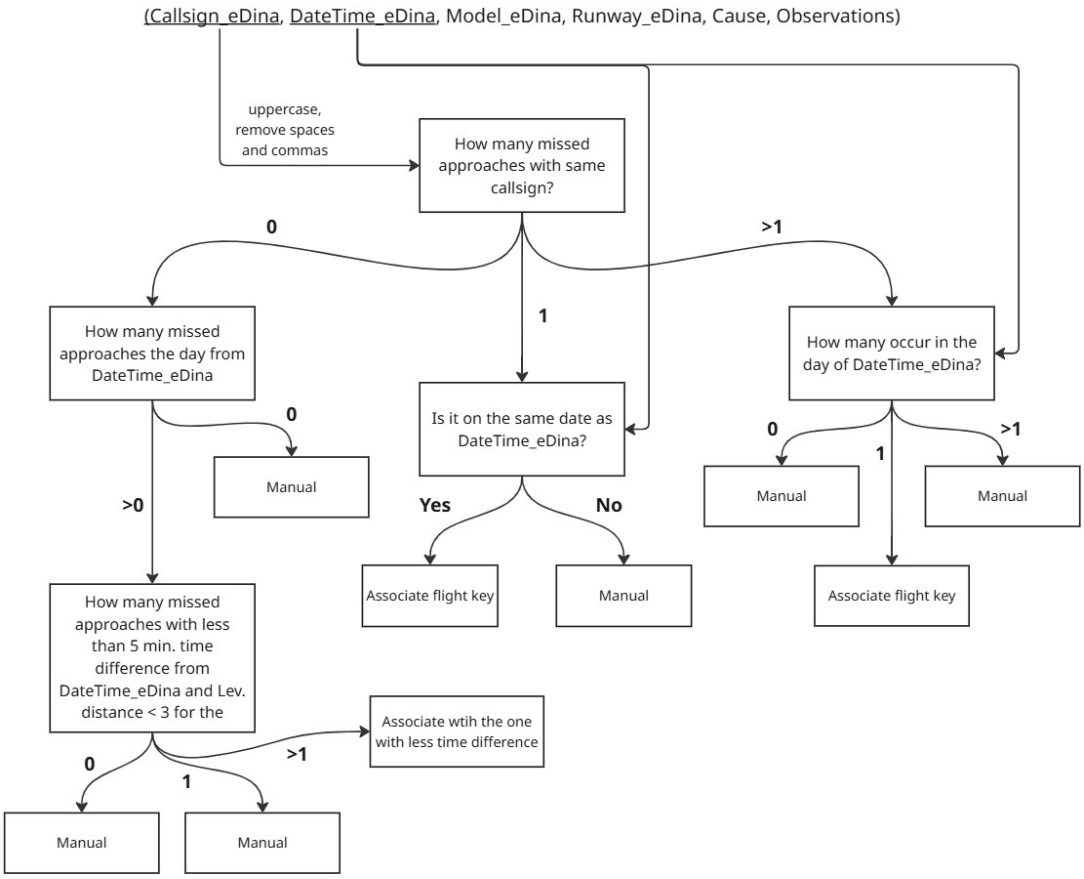


Figure 4.1: Logic behind eDina correction

equal to 2 from the observed one, and it occurred on the same date, we inferred that this was the corresponding flight.

However, if the ambiguous callsign matched more than one flight in our database, we further filtered the matches by date. If this narrowed the result to a single flight, we assigned the identifier accordingly. In any other case, the ambiguity could not be resolved mechanically and a manual more in-depth inspection was necessary to determine the correct association.

After all of the processing, we can finally see the approximate distribution of the reasons behind go-arounds in Figure 4.2. Looking at the figure, we can see that the statements made at the beginning of the section approximately align with our observations. Using these labels, we can now remove the flights from our database that we will not be able to classify (at least for now). These are especially those related to obstacles in the runway, so those starting with 'PIP'. The complete list of the approaches removed were the ones related to any of the following: PIP (all), RTA, ISDP, COM, MET_VIS, MET_CLO, MET_OTH, APE.

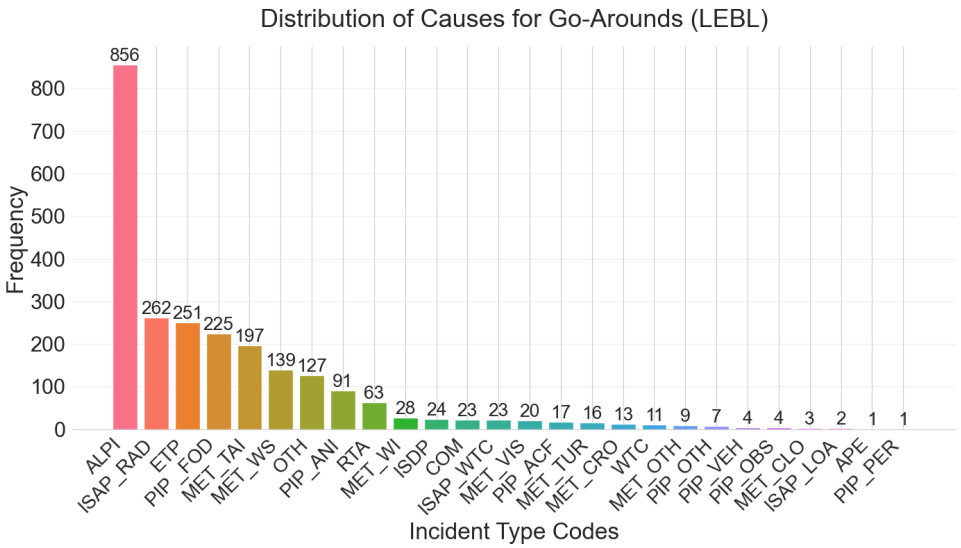


Figure 4.2: Distribution of GA (out of ~2.400)

4.1.2 Data Scaling

Some input features, like distance to the runway threshold, take very large values at some points. Because of this, it is necessary to include some scaling technique. The *Standard Scaler* from *sklearn* [30] proved to be effective at this task, so all data has been previously scaled before feeding it to the model.

4.1.3 Final Set of Features

After all features that have been collected and/or calculated in the Methodology (Section 3), we proceed to sum up all of them. The final set of attributes that will be fed to the model(s) can be seen in Table 4.1. All features are presented (and most of them get updated) at each time step. Measuring units are not relevant due to the scaler.

Table 4.1: Final Set of Features

Feature	Description
vel_z	Vertical speed
hdg	Heading of the aircraft
vel	Ground speed of the aircraft
height	Altitude of the aircraft over sea level
runway	Runway the aircraft is headed to
hdg_diff_rwy	Heading difference between the aircraft and the runway
dist_rwy	Distance to runway threshold
angle_descent	Angle of descent of the aircraft (runway threshold as reference point)

Continues in the next page.

Feature	Description
lateral_deviation	Lateral deviation of the aircraft
angular_deviation	Angular deviation of the aircraft (runway threshold as reference point)
dist_rwy_end	Distance to the runway stop end
angle_final_rwy	Angle the aircraft makes with the runway stop end
specific_energy	Specific energy of the aircraft
wake	Wake category of the aircraft
distToPrecedent	Distance to precedent aircraft
wakePrec	Wake category of the precedent aircraft
directionWind	Direction of wind at runway level at the time of landing or go-around
knotsWind	Wind speed at runway level at the time of landing or go-around

4.2 Model Architectures

Following the architecture proposals outlined in Section 3.8, we proceed to describe the practical implementation of each of the models. The first decision to make was regarding the format of the input data. On the one hand, we considered feeding each trace point individually and relying on the model internal memory to retain relevant information from previous steps. On the other hand, we explored feeding a full sequence of trace points at once, allowing the model to observe the temporal structure directly. While this second option reduces dependence on recurrence or attention, it may lead to repeated and redundant information, which can hinder learning by distracting the model from newer inputs. This trade-off will be a subject of further experimentation.

All models were implemented using the *PyTorch* framework [29], which allowed for a flexible and detailed definition of the network architecture.

The first model is based on Long Short-Term Memory (LSTM) networks and was developed using the `torch.nn.LSTM` module. It consists of a shallow architecture with two hidden LSTM layers, each followed by a dropout layer with probability 0.3 to avoid overfitting. Each LSTM layer includes 64 hidden units. The output is then passed through a fully connected layer with a sigmoid activation function to produce a probability estimate.

To improve on the LSTM's ability to focus on relevant temporal segments, a second model was built by incorporating an attention mechanism on top of the recurrent layers. Aside from the addition of attention, the architecture remains the same, including the number of layers, hidden units, and dropout probability.

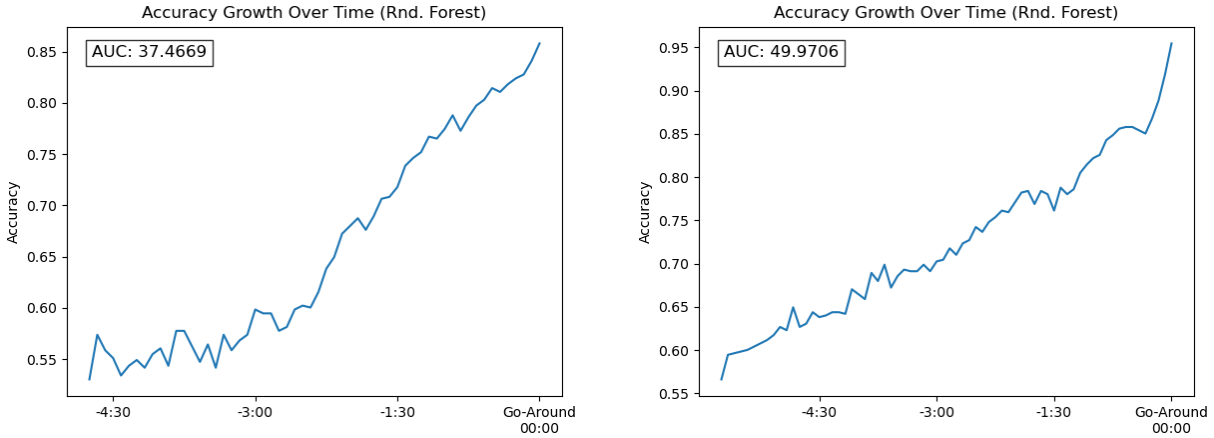
Lastly, we implemented a Transformer-based model, inspired by its success across a wide range of sequence modeling tasks. Using the `torch.nn.TransformerEncoder` module, the model was constructed with 4 encoder layers and 2 attention heads per layer. This model is slightly deeper than the LSTM variants, but otherwise similar in terms of hidden size and dropout.

All models were trained using the Adam optimization algorithm and Binary Cross-Entropy loss, which yielded consistent and stable performance. Hyperparameter selection was conducted via grid search, varying the number of layers, hidden units, dropout rate, and learning rate to determine the optimal configuration.

The final comparisons can be seen in Table 4.2 and show that the best-performing model is a *transformer*-based network using individual trace points as inputs.

Model	**Seq**	**NoSeq**
LSTM	**DTI**: 23.19	**DTI**: 29.46
	AUAC: 37.30	**AUAC**: 48.39
LSTM + att.	**DTI**: 23.18	**DTI**: 29.60
	AUAC: 38.04	**AUAC**: 47.04
Transformer	**DTI**: 22.89	**DTI**: 30.37
	AUAC: 37.47	**AUAC**: 49.97

Table 4.2: Scores on test set of different models, comparing sequenced inputs and single trace points. All have been trained and tested on a balanced dataset, with logistic labeling. Scores come from the same balanced test set.



(a) Accuracy curve with seq. inputs

(b) Accuracy curve with non-seq. inputs

Figure 4.3: Comparison of accuracy curves with sequenced and non-sequenced inputs. Notice that the left one start at about -4:30, while the right one starts earlier, at around -5:30.

Note that, although the scores and distances for sequenced inputs are, in general, lower, this is not necessarily because a significantly worse model. Because we did not use any padding for early data points, we can only start predicting once we have 12 points registered. For this reason, predictions start 11 points later and therefore *DTI* and *AUAC*, because of how they are calculated, are lower because they lack the part of the sequences. Looking at Figure 4.3, we can see that the accuracy curves start significantly later when using sequenced inputs. We can over-correct this comparison by subtracting from the *AUAC* from non-sequenced inputs $0.7 \cdot 11 = 7.7$ units. Even with this over-correction, we see that non-sequenced inputs perform better overall. A similar correction can be done for *DTI*, but we obtain the same results.

Additionally, comparing the graphs (like the ones in Figure 4.3) we see that, indeed, accuracy at each time step individually is higher when using non-sequenced inputs.

4.3 Data Balancing

For now, tests have been conducted using a balanced dataset. However, sometimes the best results come from a slightly unbalanced dataset, leading to a more/less conservative model. This also allows us sometimes to tweak the FP/FN ratio, depending on whether we want to favor one of them. In our case for example, we do not care so much about false positives (predicting an aborted landing which finally does not abort), but rather want to minimize false negatives (flights that are predicted as safe landings, but end up performing a go-around.

Ratio Successful/Missed		0.75	0.9	1	1.1	1.25
Score(s):	DTI:	30.37	30.53	30.37	29.50	30.10
	AUAC:	49.79	48.26	49.97	50.67	48.58

Table 4.3: Comparison of different balance rations on the training set. Scores come from the same balanced test set.

It seems like we get better results with a slightly unbalanced dataset. For that, we will finally include 1231 missed approaches in the training set, together with $1231 \cdot 1.1 = 1354$ successful landings. Both the validation and test sets will consist of 528 approaches, half missed and half non-missed. A balanced dataset will allow us to gain better insights into the performance, but we will need to look closer at the recall/precision in order to answer questions like how many missed flights are we detecting and how many of our predictions actually have to perform a go-around.

4.4 Labeling Strategies

Continuing with the analysis, we want to explore which labeling strategy performed the best. We compared a binary labeling strategy, which is the most rudimentary. It means that a data point would have associated '1' as label if it is part of a missed approach and '0' otherwise. We also wanted to try the linear approach shown in [9] and the sigmoidal curve presented in Section 3.3.2.

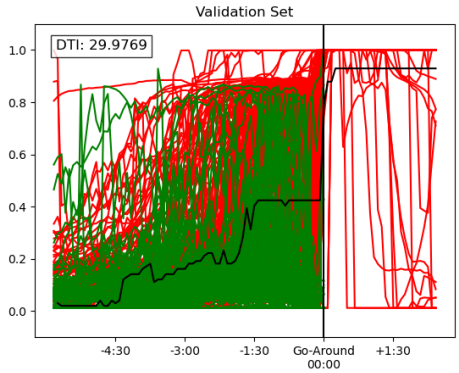
Labeling Strategy	Binary	Linear	Logistic
Score(s)	DTI: 42.02	DTI: 30.00	DTI: 29.50
	AUAC: 35.22	AUAC: 49.23	AUAC: 50.67

Table 4.4: Scores of different labeling strategies

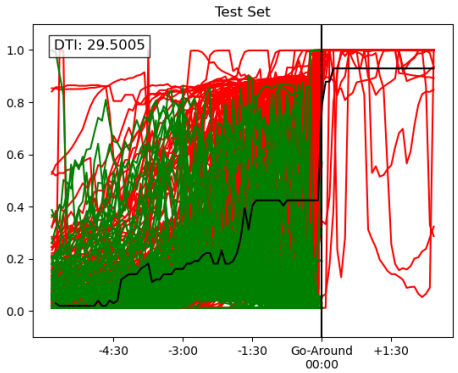
The labeling strategy that returned the best results was the logistic approach, so we will continue to use it.

4.5 Threshold-based Observer

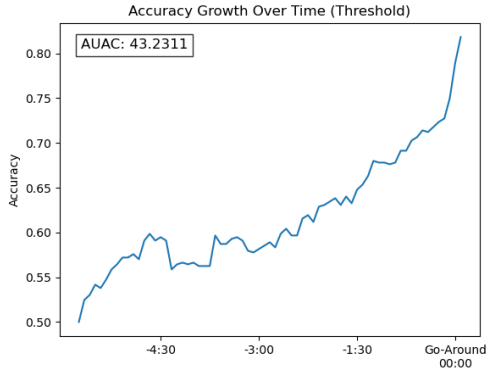
In some papers like [23], we saw a different approach to classification. Once they extracted the probability sequences for the regressor, they classified missed flights based on the fact if their probability at some point surpasses a certain threshold (e.g. 0.6). We assumed this approach is less effective than applying a Random Forest observer, but we wanted to test it anyway. Using the data from the validation set, we grid-search an optimal threshold for each time step (maximizing accuracy). This is, the threshold may be 0.2 at a cutoff point far away, but 0.6 at a point closer to the runway. Then, using these thresholds, we would classify the flights from the test set and establish an accuracy curve like the one we have created for the RF. On the one hand, Figure 4.4 shows the optimal thresholds found, both drawn into the validation and test set probability curves. On the other hand, the figure shows a comparison of the accuracy curves between the threshold-based approach and the RF classifier we originally used. Clearly, the RF is a much better classifier and we will continue using it.



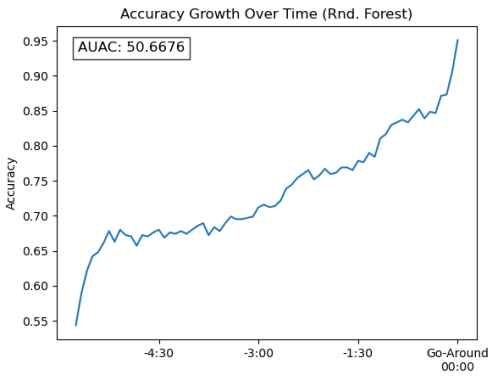
(a) Prediction curves for validation set and optimal thresholds



(b) Prediction curves for test set and optimal thresholds



(c) Temporal evolution curve of accuracy and AUAC of threshold-based classifier



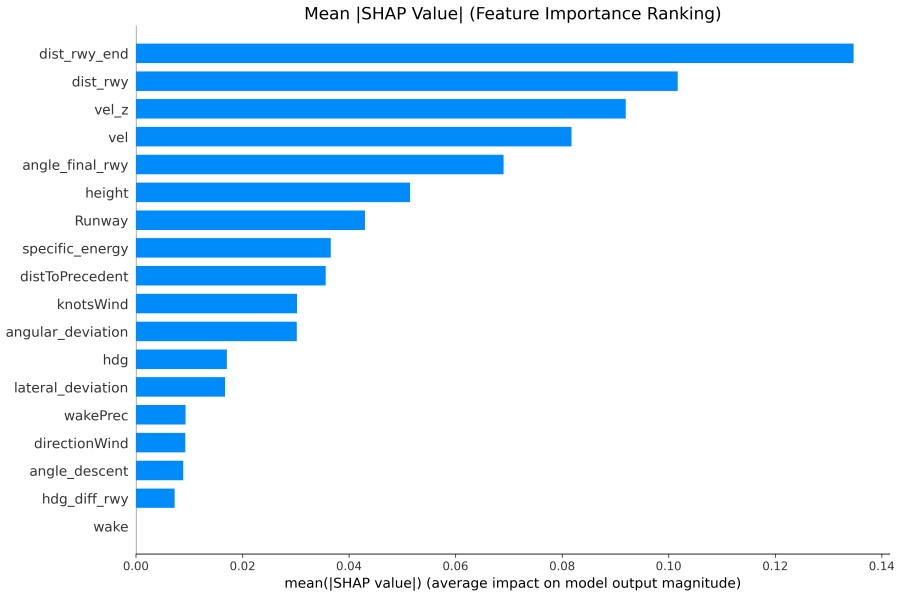
(d) Temporal evolution curve of accuracy and AUAC of RF observer

Figure 4.4: Visualization of the threshold evolution (in black) and comparison of a threshold-based classifier and a Random Forest Observer

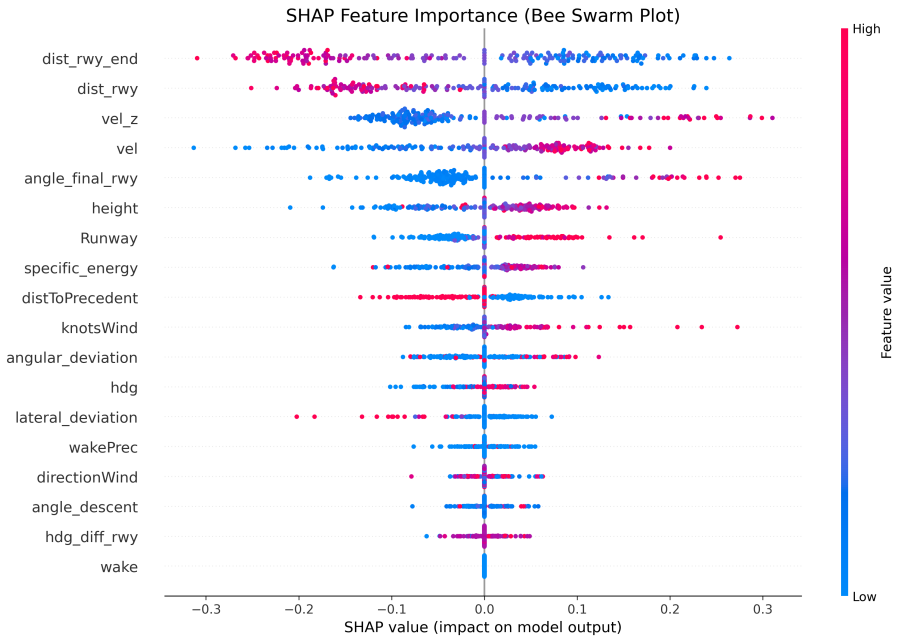
4.6 SHAP Analysis

In this section we will be doing a SHAP analysis to find out which features contribute the most to the model decision-making and how they affect.

We will analyze each feature one by one to understand the output exactly. In general, the beeswarm plot 4.5b is most interesting, as it shows how a features value directly affects the outputs. The higher (or lower) the SHAP value, the more impact the feature has.



(a) SHAP Bar Plot



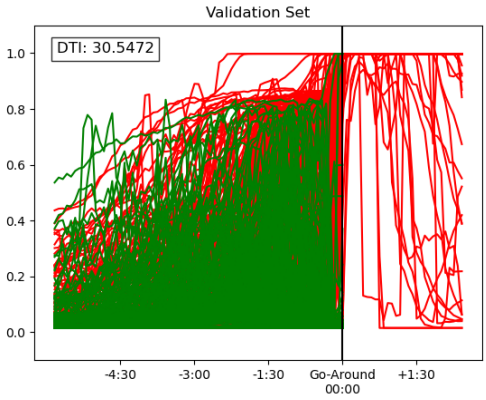
(b) SHAP Beeswarm Plot

Figure 4.5: SHAP Analysis Plots

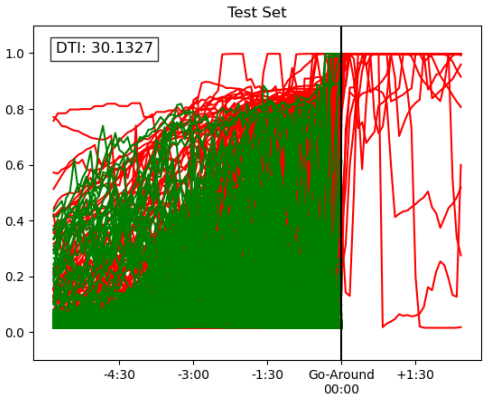
- *dist_rwy_end:* If the airplane is very far away from the airport, the model tends to give lower probabilities and only outputs higher probabilities once the plane gets closer. This is probably very influenced by the sigmoidal approach, because all trace points that are very far away from the airport will have lower associated probabilities.
- *dist_rwy:* In a similar way as the distance to the runway end, if the distance to the runway threshold is very big, the model will not tend to return high probabilities.
- *vel_z:* If a plane has a very high vertical velocity, it means it is ascending, so in many cases initiating a go-around. This means the probability will have to be very high if the vertical velocity is high. Otherwise, if it is somewhat low, it is as expected.
- *vel:* If a plane is going very slowly, it is a good sign that the approach is controlled. Otherwise, if it is going too fast, the chances of a go-around significantly rise.
- *angle_final_rwy:* If the angle the plane makes with the runway end is low, it is as expected. However, if it increases, this most likely means the plane has run out of runway, so it will have to perform a go-around.
- *height:* If the plane is too high, the probability of a GA is going to rise, as its descent rate is limited. Approaches at lower altitudes do not seem to pose a risk.
- *Runway:* Looking back at the encoding we did for the runways, we encoded 'riskier' runways with higher values. This has been reflected in the SHAP analysis, as higher values of the Runway variable seem to produce higher GA probabilities.
- *specific_energy:* As expected, a higher specific energy means the GA probability will rise, as the plane has to lose more energy.
- *distToPrecedent:* If a plane distance to the previous arrival is low, the missed landing probability will rise. If it is higher, in turn, there does not seem to be any risks.
- *knotsWind:* Intuitively, higher values of wind speed mean GA probability will rise.
- *angular_deviation:* Higher angular deviations are associated with higher GA probabilities. This is expected.
- *hdg:* The interpretation of the outputs of this variable is more difficult, as too high or too low values could mean deviations with respect to the optimal heading. This optimal heading will change depending on the runway, so to analyze this variable SHAP is not too accurate.
- *lateral_deviation:* Surprisingly, unlike angular deviation, high values of lateral deviations are associated with a lower GA probability. This is most likely due to the fact that high lateral deviations are normally found at trace points further away from the airport, where it is not as problematic, as the pilot can still correct its path. High lateral deviations are dangerous when close to the runway, but

at that point angular deviation will also be very big. This variable is a candidate for removal.

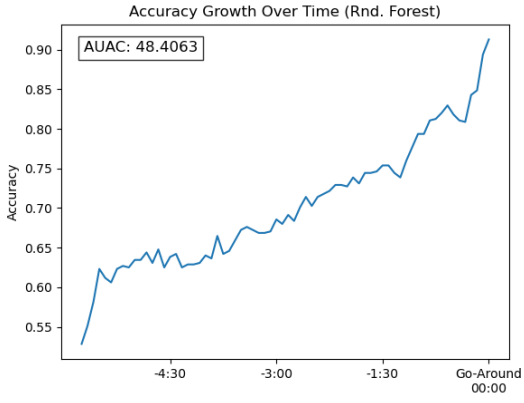
- *wakePrec:* This variable seems to have little impact in the output and there is no obvious pattern. This variable is a candidate for removal.
- *directionWind:* Similarly as the heading variable, it is difficult to analyze it with SHAP, as the magnitude of the variable is rather irrelevant. The interesting thing is to check the difference between heading and wind direction, which we assume the model is able to do.
- *angle_descent:* This variable seems to have little impact in the output and there is no obvious pattern. This variable is a candidate for removal.
- *hdg_diff_rwy:* This variable seems to have little impact in the output and there is no obvious pattern. This variable is a candidate for removal.
- *wake:* This variable seems to be pretty much irrelevant, probably because flights have mostly wake category 'Medium'. This variable is a candidate for removal.



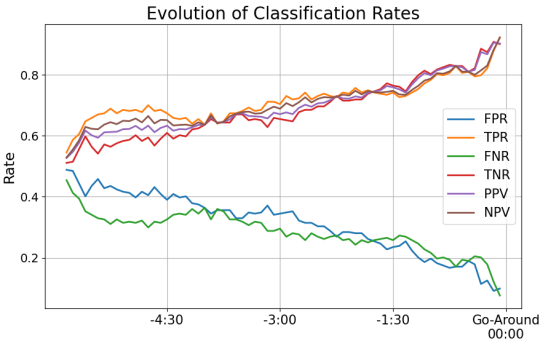
(a) Prediction curves for validation set



(b) Prediction curves for test set



(c) Temporal evolution of accuracy and AUAC



(d) Temporal evolution of TP,FP,. . . rates

Figure 4.6: Results of removing the candidates for removal and retraining the model

To sum up, the features affect the model as one may expect. We did, however, gain some insights into some of the variables, which could be removed. Candidates for

removal are: lateral_deviation, wakePrec, angle_descent, hdg_diff_rwy and wake. By removing these variables, we expect the model to become less complex. By simplifying its inner structure, we hope that it will make better predictions as it will be able to focus on the more important variables. To test this hypothesis, we train a new model after removing the candidates for removal. The results obtained can be seen in Figure 4.6. The scores have significantly dropped, so it seems best to keep all features, as they do appear to retain some significant information.

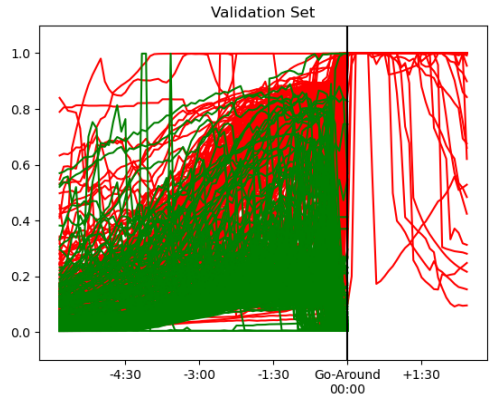
4.7 Final Model

At this point we have explored the different possibilities for model implementation. We have experimented with different model algorithms (LSTM, Transformers, ...), different data balances, different labeling strategies, different feature combinations, etc. It is time for one final grid-search to find the optimal hyperparameters for these settings. After multiple training and testing executions, we conclude that the final best model consists of a transformer with 4 encoder layers, 2 attention heads per layer and 64 hidden units per layer. Other adjustments include a learning rate of 0.001, paired with an Adam optimizer and Binary Cross-Entropy loss criterion. To avoid overfitting, a dropout of 0.3 was used to randomly remove some neurons from the network. With this setup, we finally obtain the results in Figure 4.7.

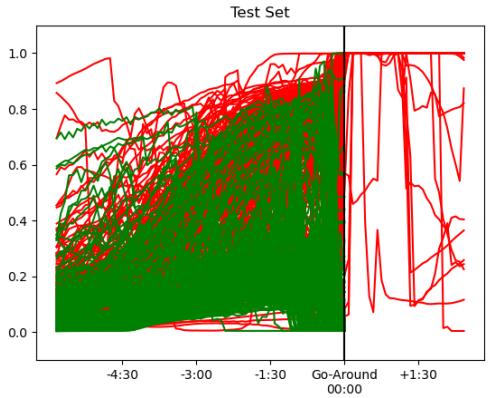
These are, as expected, the best results we have obtained overall. Not only do they return the best scores (DTI = 27.47 and AUAC = 54), but looking at the different score evolutions (accuracy, precision, etc.), these also show the best evolution. Figure 4.7c shows that we can predict GAs with an accuracy of 83%, 1:30 in advance. This accuracy decreases as we try to predict with more advance and increases as we approach the GA point, reaching an accuracy of 87%, 0:30 in advance. More details on scores at different time points can be seen in Table 4.5.

Advance	**-4:30**	**-3:00**	**-1:30**	**-0:30**
Accuracy	0.73	0.77	0.83	0.87
Precision (PPV)	0.68	0.72	0.79	0.85
NPV	0.81	0.84	0.87	0.9
TPR	0.86	0.86	0.89	0.91
TNR	0.6	0.69	0.77	0.84
FPR	0.4	0.31	0.23	0.16
FNR	0.14	0.14	0.11	0.08

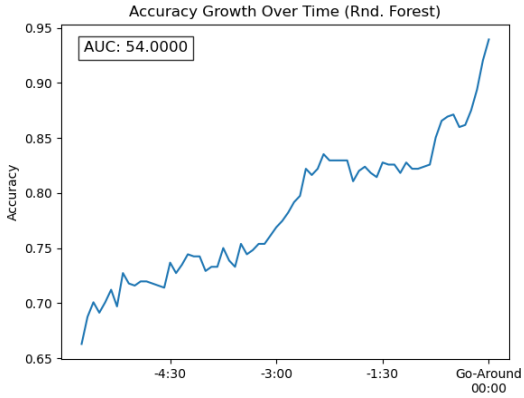
Table 4.5: Different metrics at -4:30, -3:00, -1:30, -0:30; referring to the advance with respect to the GA or successful landing.



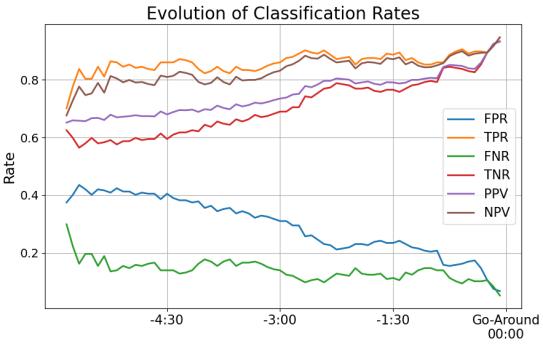
(a) Prediction for validation set



(b) Prediction curves for test set



(c) Temporal evolution of accuracy and AUAC



(d) Temporal evolution of TP,FP,. . . rates

Figure 4.7: Results of the final model in LEBL

4.8 Model Precision Exploration

We want to find out if the model is better suited to predict GAs at different stages of the approach. For this, we have decided that a missed approach will be correctly classified (the whole flight), when the observer assigns 1 more often than 0 among the last stages of the approach. After some tests, we settled on using the last 30 trace points before the GA occurs. In other words, $30 \times 5s = 150s$, so we would be taking into account the predictions from the last two and a half minutes. There are some example plots in Appendix B that show the probability evolution, color-coded by the classification of the observer (red - going to abort; blue - going to land).

By doing this, we can classify the approaches as 'predicted correctly' or not. Out of the 256 missed approaches from the test set, 209 were correctly classified by this rule and 55 were not. This represents an accuracy of 80%. With this information we can then plot the different points into a scatter plot, based on the distance to the airport at the point of abortion.

We obtain the results in Figure 4.8. It does seem like the GAs that were not detected were initiated closer to the airport. However, to be certain, we will check the distributions. Figure 4.9 shows a histogram-like distribution of these data points,

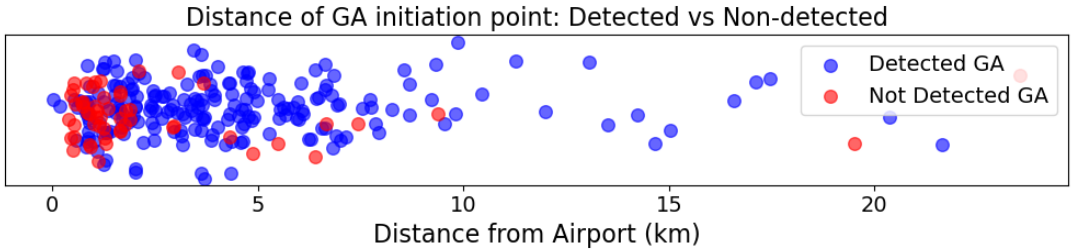


Figure 4.8: Scatter Plot of distance when GA was initiated, grouped by if the model successfully predicted the aborted landing

together with a KDE plot (Kernel Density Estimation) [37]. It shows the probability distribution of the two groups, and it is pretty clear that the distributions cannot be the same. This indicates that there is a difference, in turn, meaning that the model works better at predicting GAs that occur further away from the airport, rather than closer to it.

To further prove this point, we performed a Kolmogorov-Smirnov test to determine whether the two samples could come from the same distribution (two-sample K–S test) [31]. The test returned a KS Statistic of 0.4967 and a p-value of 2.4330e-10, concluding that the distributions are significantly different ($p << 0.05$). To sum up, we have exhaustively proven that the model will predict GAs more accurately if they abort further away from the airport ($\sim > 2.5km$).

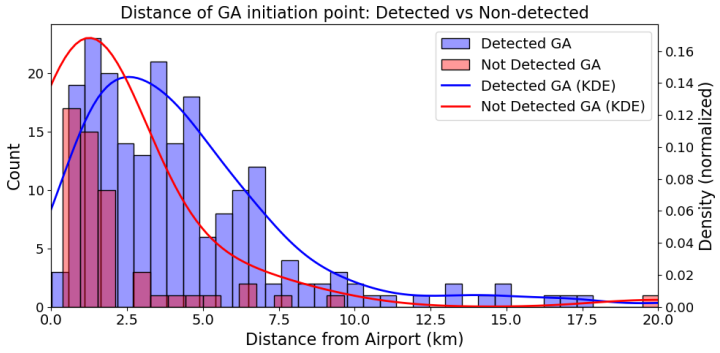
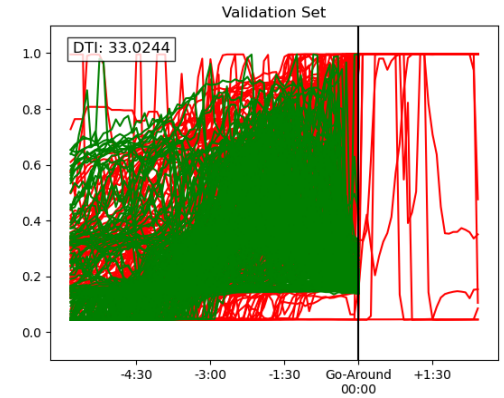


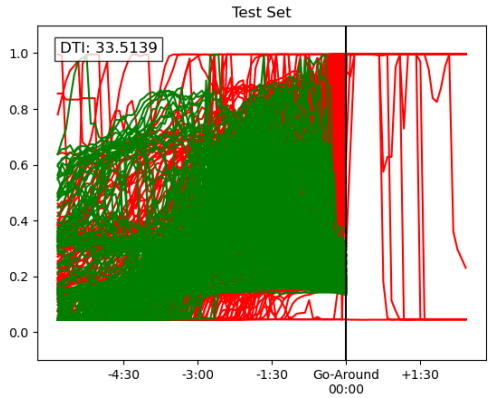
Figure 4.9: Histogram and KDE plots of the distributions of detected vs. non-detected GAs. Scales on the y-axes correspond to the histogram (left) and to the KDE plot (right).

4.9 Final model in LEMD

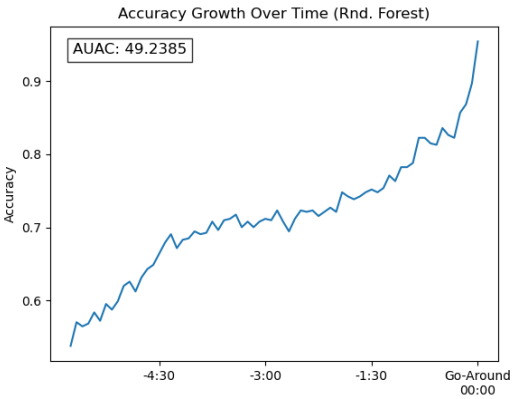
This raises the question of how a similar model would perform in LEMD. Until now, we have only tested in LEBL, so it is naturally alluring to check how a model would perform having been trained only with LEMD's flights. Using the same settings and hyperparameters as in the final model described in Section 4.7, we obtain the following results.



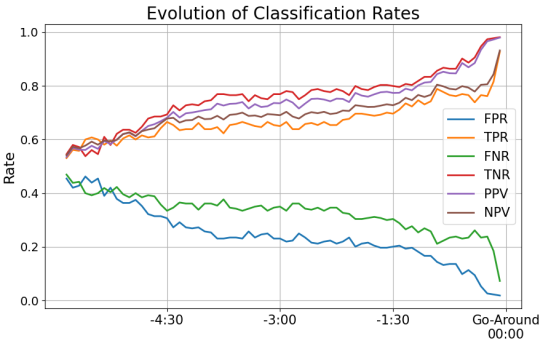
(a) Prediction curves for validation set



(b) Prediction curves for test set



(c) Temporal evolution of accuracy and AUAC



(d) Temporal evolution of TP, FP,. . . rates

Figure 4.10: Results of the final model in LEMD

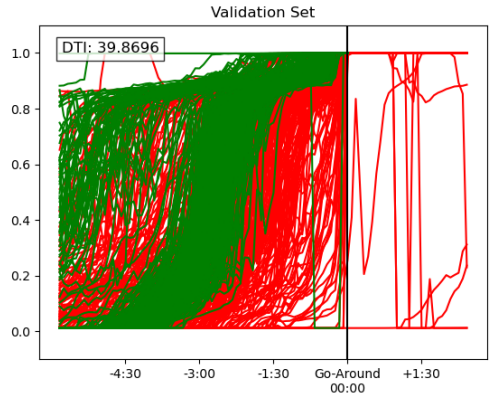
Figure 4.10 shows that predictions are still accurate. Although scores have substantially dropped (DTI = 33.051; AUAC = 49.24), the model retains significant predictive power, achieving an accuracy of 80% at a minute in advance. Additionally, these scores have been seen to increase with minimal grid-search, so it seems like the optimal hyperparameters in LEBL do not necessarily have to be the best ones in LEMD.

4.10 Experiment 1: LEBL model in LEMD

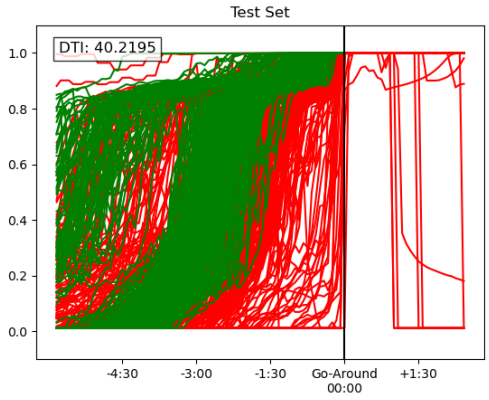
A very interesting experiment we wanted to conduct was if the model trained with flights on LEBL would be able to make accurate predictions in LEMD. Our hypothesis was that, because we have calculated many features like lateral and angular deviation from runway. distance to its threshold, etc., we have abstracted the model enough for it be able to make predictions in any runway in any airport. If this were true, we would have come up with a universal model, that could be implemented in any airport (given an airport has insufficient data to train its own model).

To carry out this experiment, we used the final pre-trained model described in Section 4.7, and applied validation and test sets from LEMD. However, some adjustments had to be made to the data. For example, the "height" variable measures altitude above

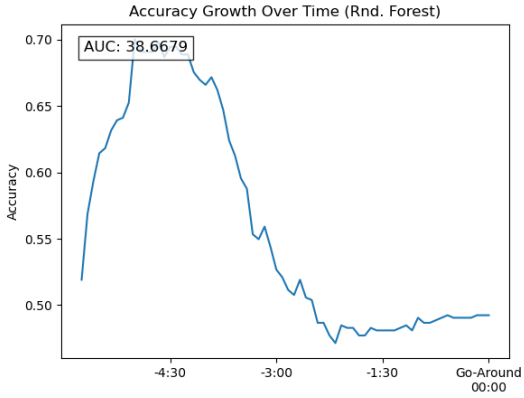
sea level. This means that to land at LEBL an aircraft would reach flight level 0, while at LEMD, the runway is at FL19. To compensate for this, we subtracted the height difference at each trace point. The results obtained are shown in Figure 4.11.



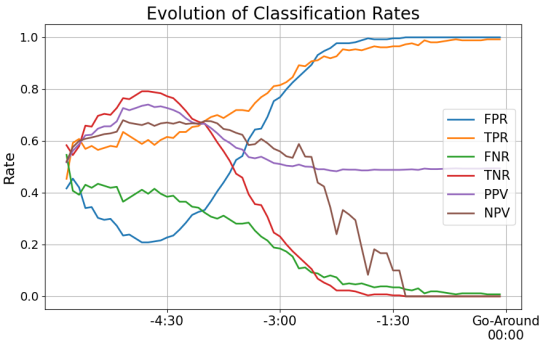
(a) Prediction curves for validation set



(b) Prediction curves for test set



(c) Temporal evolution of accuracy and AUAC



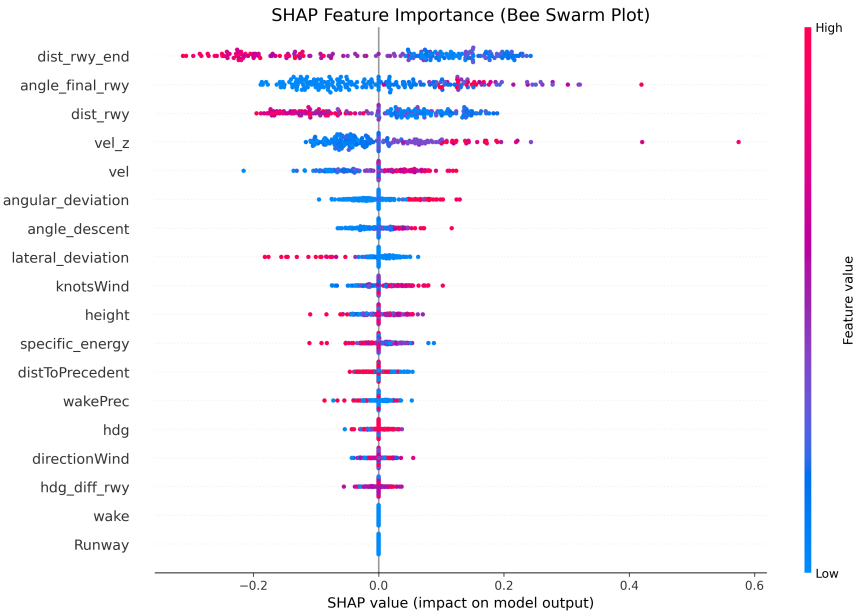
(d) Temporal evolution of TP,FP,. . . rates

Figure 4.11: Results of testing a LEBL regressor and observer to predict GAs in LEMD

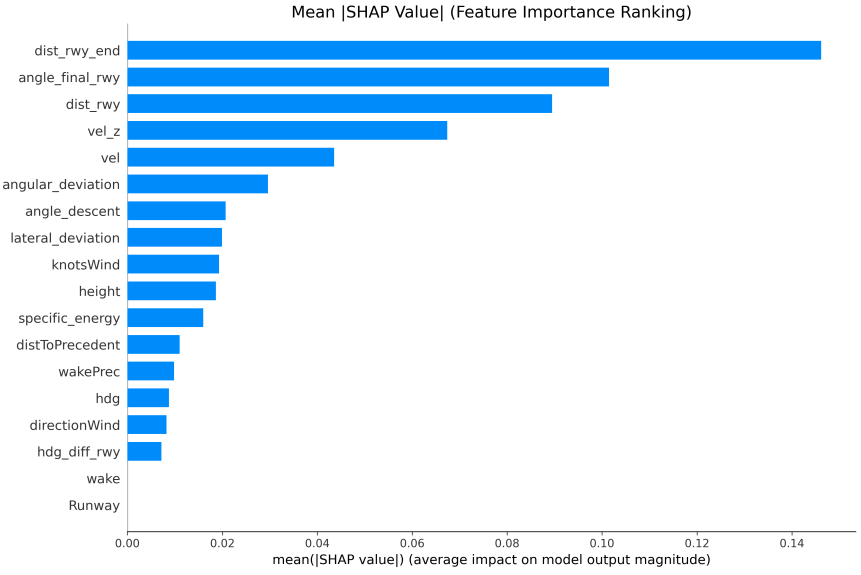
We can see that the predictions are not as expected. Even though the data has been abstracted from a specific airport (as much as possible), it is still not capable of making accurate predictions, especially in the final stages of the approach. Even though the models seem to capture some sort of pattern in the earlier stages ($\sim$ 70% accuracy at -4:30), it loses those insights as soon as the plane approaches the runway any further.

Looking at the prediction curves and at the temporal evolution of the prediction rates (4.11d), we can see that the main problem is that it detects all flights as aborted. This is, both the regressor and the observer tend to predict aborted landings, even looking at those flights that are not. The green curves in 4.11a and 4.11b show successful landings, but we can see that the model is assigning very high probabilities to these flights, making them indistinguishable from missed approaches. Therefore, if we look at the FP and TP rates in 4.11d, we can see that the observer is classifying almost all flights as missed. The most likely hypothesis is that the model is seeing something in the data that is confusing it, leading to it thinking all approaches are going to abort.

It is interesting to now perform a SHAP analysis to see if one variable is the main source of this confusion. The plots in Figure 4.12 suggest that the model is getting confused by the runway length and therefore the distance to the runway end and the angle that forms. This makes sense, as each runway at each airport has its own unique characteristics, which makes it unique. It seems like this feature calls for some adjustments or a new model altogether. However, we could try to remove this feature and see if the predictions remain unreliable.



(a) SHAP Beeswarm Plot

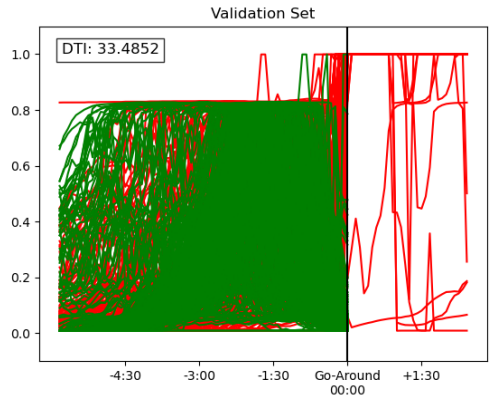


(b) SHAP Bar Plot

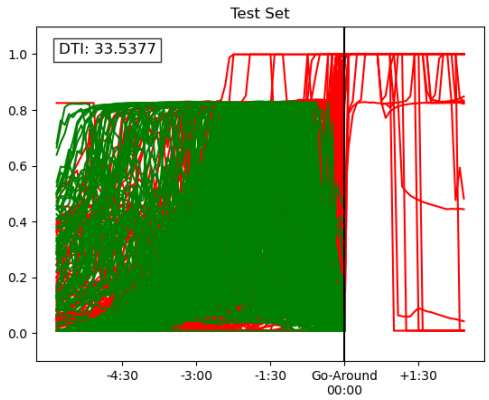
Figure 4.12: SHAP Analysis of LEBL Model in LEMD Experiment

After successfully removing those features, training a new model and testing it in LEMD, we get the results shown in Figure 4.13. Right away we can see that the

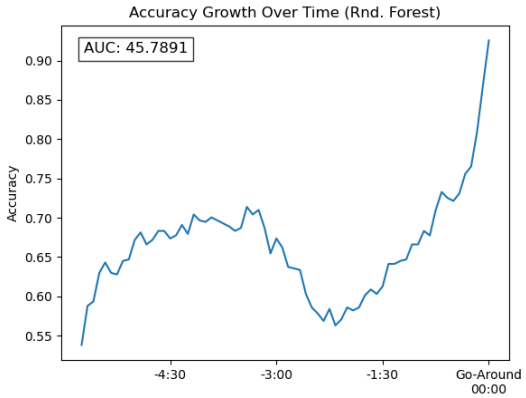
predictions have improved (as seen in 4.13c). Although Figure 4.13a and Figure 4.13b do not show any pretty curves whatsoever, it does seem like we have gained some predictive power by removing the problematic features. Figure 4.13d also shows that the FPR is no longer catastrophic and starts decreasing at some point instead of approaching 1.



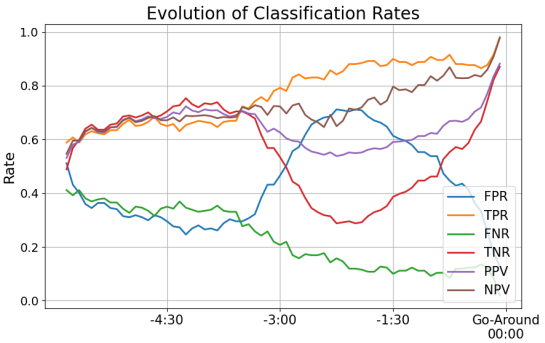
(a) Prediction curves for validation set



(b) Prediction curves for test set



(c) Temporal evloution of accuracy and AUAC



(d) Temporal evolution of TP,FP,. . . rates

Figure 4.13: Results of testing a LEBL regressor and observer to predict GAs in LEMD, having removed the features dist_rwy_end and angle_rwy_end

In general, although the results are not as good as desired (the same model obtains a DTI of 30 and an AUAC of 49 in LEBL), it still keeps some predictive capabilities even when not having been trained on any data from LEMD. We have shown that if we desire to gain generalizability in the model, we have to sacrifice information on runway limits and therefore a bit of performance locally. In other words, while we can use a model with all features in a certain airport, we could potentially train a secondary model, which does not include data on runway end limits, that will perform, at least decently, in other airports. This is especially helpful when talking about smaller or more remote airports that do not have enough data to train their own model. In this case, a model of an airport with similar characteristics could be used to predict aborted landings (which is much better than not having anything).

4.11 LandIA

Until now we have worked exclusively with historical data. This raises the question of how this model would perform in a real-time situation. To do this, we collect information from ADS-B (Automatic Dependent Surveillance–Broadcast) sources. ADS-B is a technology used in aviation for aircraft surveillance, where aircraft automatically broadcast their position, altitude, and velocity to other aircraft and air traffic control. This information is transmitted via radio waves, and unlike traditional radar, it does not require a ground-based signal to be sent to the aircraft for it to transmit its location. This data is open and can be fetched using free APIs in Python (e.g. OpenSky).

As the whole analysis was conducted on LEBL airport, we will also implement the real-time model there. To do this, we first filter out all flights on a 10-km radius around LEBL. Then we remove all flight that are either grounded or not performing an approach at LEBL. Most information we are able to retrieve directly from the API. Engineered features like lateral deviation, distance to runway threshold, etc. have to be calculated at every point, but calculations are made pretty much instantly by the computer. However, out of the flights approaching LEBL, we have to find out which runway they are directed to. This information can not be found in the API, so we have to calculate it manually.

We propose Algorithm 1. In a nutshell, the algorithm takes a flight current position

Algorithm 1 Infer runway from flight position and heading

Require: flight with attributes `latitude`, `longitude`, `heading`; runway metadata `runway_info[airport]`
Ensure: inferred runway identifier or `None`

if flight lacks any of latitude/longitude/heading **then**
  return `None`
end if
$(\varphi, \lambda) \leftarrow$(flight.latitude, flight.longitude)
$\theta \leftarrow$ flight.heading or 0
$P_{\text{proj}} \leftarrow$ destination_point$(\varphi, \lambda, \theta, 5)$
$best\_runway \leftarrow$ None, $best\_perp\_dist \leftarrow \infty$, $best\_geo\_dist \leftarrow \infty$
for all runway r in `runway_info[airport]` **do**
  retrieve $R_{\text{start}}, R_{\text{end}}, \alpha_r$ from runway metadata
  $d_{\perp} \leftarrow$ point_line_distance$(R_{\text{start}}, (\varphi, \lambda), P_{\text{proj}})$
  $d_{\text{geo}} \leftarrow$ geodesic$((\varphi, \lambda), R_{\text{start}})$ in NM
  $\Delta\theta \leftarrow |(\theta - \alpha_r + 180) \bmod 360 - 180|$
  if $\Delta\theta > 25$ **then**
    continue
  end if
  if $d_{\perp} < best\_perp\_dist \ \vee \ (|d_{\perp} - best\_perp\_dist| < 0.3 \wedge d_{\text{geo}} < best\_geo\_dist)$ **then**
    update $best\_runway$, $best\_perp\_dist$, $best\_geo\_dist$
  end if
end for
return $best\_runway$

and heading, projects a point 5 NM ahead along that heading, and then checks every

runway at the specified airport to find the best match. It does this by:

1. Calculating the perpendicular distance from the flight projected path to each runway's threshold.
2. Measuring the geographic distance between the aircraft position and the runway start point.
3. Ensuring the aircraft's heading is within ±25° of the runway's alignment.
4. Selecting the runway with the smallest perpendicular distance (and if tied, the shorter geographic distance).

After having assigned a target runway for each approaching flight, we can use this information to establish a landing order in each runway. This allows us to figure out which the preceding aircraft is for each flight and therefore calculate the distance between a flight and its preceding approach.

Additionally, using historic information, we created a dictionary that assigns the corresponding wake category to each of the aircraft models. Doing so, if the model aircraft is, for example, 'A320' we will know its wake category is 'M'.

Finally, we used the Open-Meteo API to retrieve real-time wind information at the runway level. The API offers information on wind direction and speed, which we input directly into the model.

Due to the memory mechanism of the main regressor model, we will load a separate instance of the trained model for each flight. This is, once an aircraft enters the 10-km monitoring range of our system, a fresh new model will be loaded for that specific flight, and will receive only updates from that flight. This means we will parallelly run two or three models, but this does not make an impact on performance.

On top of that, each model output is saved into a dictionary and stored individually for each flight. By doing this, we can load the observer model (Random Forest) and show it the outputs so it can interpret them and classify the flights.

For visualization purposes, we developed a web-based prototype, called LandIA (see Figure 4.14), where we showed real-time landing approaches at LEBL on a map. Flights are color-coded based on the aborted risk offered by the regressor model. On the right side you can find a list of the current approaches, together with some other relevant information, including the outputs of both the regressor model and the observer, aircraft model, etc. The display is updated every second, which gives a good sense of continuity. If less frequency is acceptable, the update time can be tweaked to relieve stress on the system, or it can be set to update even faster if needed, but it is still limited by computing capacity.

At the moment of publishing, models have been trained for both LEBL and LEMD and can be implemented into their own LandIA interface. Further models can be easily trained for more airports, given the data is processed correctly. As ATCOs only work in one airport at a time, it is not considered necessary to offer the visualization of more than one airport at a time, however, these visualizations could be provided with minimal work, as the code should only be slightly adapted.

The main objective of this interface is to serve as aid for ATCOs, complementing the visualizations they already have. If given access to them, the system could be seam-

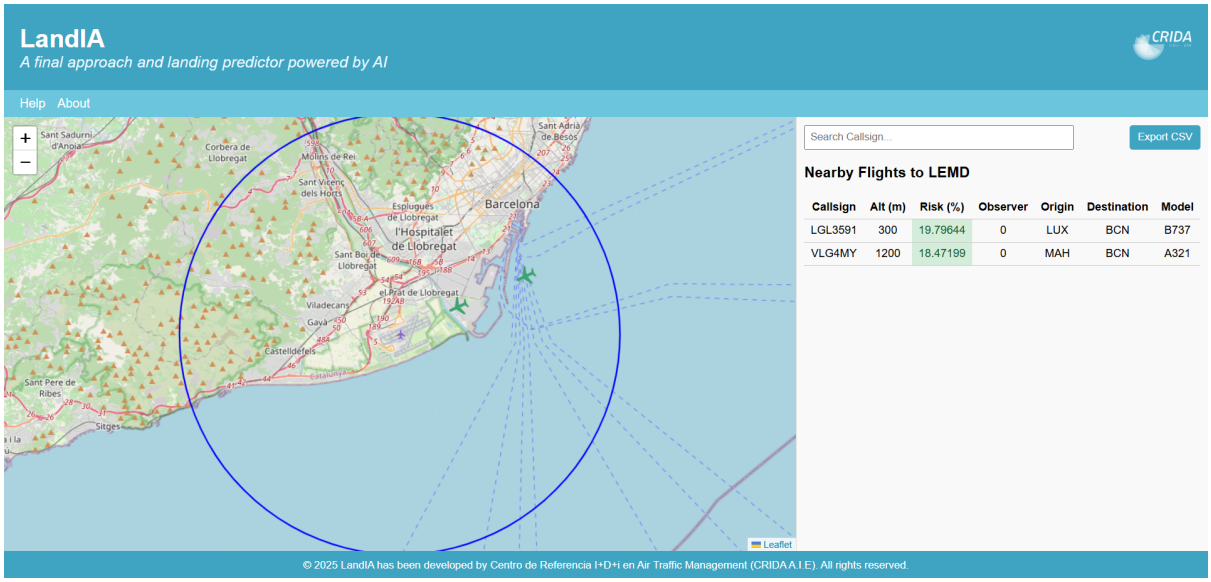


Figure 4.14: Screenshot from the LandIA website (retrieved 23.06.2025 14:28:32), LEBL

lessly implemented into the existing one, offering additional information on flight status and prediction.

Chapter 5

Conclusions

The results of this project show that it is possible to develop a robust model that accurately estimates the probability of an aborted landing during the final approach phase. By combining aircraft trajectory data with engineered features like heading deviation, specific energy, distance to the previous arrival and meteorological conditions, the system was able to anticipate go-arounds with considerable precision, up to 87% accuracy just 30 seconds before the maneuver, and over 80% two minutes in advance.

Among all the evaluated architectures, Transformer-based models proved to be the most effective, outperforming LSTMs in terms of both prediction stability and scalability. The model achieved a DTI (Distance to Ideal) score of 27.47 and an AUAC (Area Under Accuracy Curve) of 54, marking the best performance across all tests, again proving transformers flexibility among a wide range of tasks.

Moreover, SHAP analysis confirmed that the model was learning the correct patterns: proximity to the runway threshold and end, speed (both vertical and absolute), height evolution, runway used and deviation from alignment were key contributors to go-around decisions. These findings not only validate the predictive power of the model, but also make its decisions explainable, and thus more trustworthy from an operational perspective.

The system was further tested in real-time using ADS-B data, demonstrating that it could be integrated into a live ATCO interface, such as the LandIA tool developed during the project. While the model performed best when trained on airport-specific data (e.g. LEBL), it showed promising generalizability to other airports (e.g. LEMD), especially when adjusted to account for runway-dependent features.

However, several limitations remain. The current dataset uses 5-second intervals, which may smooth out rapid changes critical for decision-making. Some flight anomalies are also caused by ATC instructions or rare events not captured in the data. Despite these limitations, the model offers a substantial step forward toward real-time decision support for ATCOs.

This work proves that real-time go-around prediction is not only feasible, but it can be made operationally meaningful through good feature engineering, modern sequential models, and proper interpretability. While challenges remain, this work establishes a strong foundation for operational go-around prediction systems.

Future work on this project can incorporate improving data quality. This includes incorporating more meteorological information (wind shear, fog, visibility,...), more data from other sources (e.g. OpenSky) and information on deceleration rates or precise pilot inputs.

Additionally, if we can get experts to analyze flights individually and identify where an aircraft starts to deviate or where a go-around starts to be likely, we could refine the labeling strategy, which in turn should improve results.

One could also develop interval-specific models. That is, if a model is specialized for the interval between the 1000ft and 800ft mark, it could potentially make better predictions than one single model that has to learn patterns on all height or distance intervals. This is quite promising, as we have seen in Section 4.8 that GAs at different distances/altitudes may constitute different problems on their own.

Finally, there are works that have attempted to detect unstable approaches [33], predict aircraft arrival runway occupancy time [14] or predicting runway conditions using weather and flight data [26]. If we could parallelly run these models with our regressor and input their predictions, we could gain insight into other variables not seen before.

References

[1] U. Ahlstrom and F. J. Friedman-Berg. “Using eye movement activity as a correlate of cognitive workload”. *International Journal of Industrial Ergonomics* 36.7 (2006), pp. 623–636.

[2] *AIP España: AD 2-LEBL – Aeropuerto Josep Tarradellas Barcelona-El Prat*. Aeronautical Information Publication AIRAC AMDT 04/25. Madrid, Spain: AIS España (Aeronautical Information Service), May 2025.

[3] L. Breiman. “Random Forests”. *Machine Learning* 45.1 (2001), pp. 5–32.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Belmont, California: Wadsworth International Group, 1984.

[5] T. Chen and C. Guestrin. “XGBoost: A Scalable Tree Boosting System”. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’16)*. New York: ACM, 2016, pp. 785–794.

[6] L. Dai, Y. Liu, and M. Hansen. “Modeling go-around occurrence using principal component logistic regression”. *Transportation Research Part C: Emerging Technologies* 129 (2021), p. 103262.

[7] F. Dehais, J. Behrend, V. Peysakhovich, M. Causse, and C. D. W. and. “Pilot Flying and Pilot Monitoring’s Aircraft State Awareness During Go-Around Execution in Aviation: A Behavioral and Eye Tracking Study”. *The International Journal of Aerospace Psychology* 27.1-2 (2017), pp. 15–28.

[8] Deutscher Wetterdienst (DWD). *German Meteorological Service – Deutscher Wetterdienst*. 2025.

[9] I. Dhief, S. Alam, N. Lilith, and C. C. Mean. “A machine learned go-around prediction model using pilot-in-the-loop simulations”. *Transportation Research Part C: Emerging Technologies* 140 (2022), p. 103704.

[10] EUROCONTROL and Flight Safety Foundation. *Go-Around Decision Making*. 2013.

[11] European Centre for Medium-Range Weather Forecasts (ECMWF). *ECMWF Open Data*. 2025.

[12] B. Figuet, R. Monstein, M. Waltert, and S. Barry. “Predicting Airplane Go-Arounds Using Machine Learning and Open-Source Data”. *Proceedings of the OpenSky Symposium* 59.1 (2020), p. 6.

[13] J. M. Font, D. Manrique, and J. Ríos. *Redes de Neuronas Artificiales y Computación Evolutiva*. Madrid, España: Editorial Universitaria, 2021.

[14] H. Gao, Y. Xie, C. Yuan, X. He, and T. Niu. “Prediction of Aircraft Arrival Runway Occupancy Time Based on Machine Learning”. *International Journal of Computational Intelligence Systems* 16 (Sept. 2023).

[15] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. *Neural computation* 9.8 (1997), pp. 1735–1780.

[16] International Civil Aviation Organization (ICAO). *Procedures for Air Navigation Services – Air Traffic Management (PANS-ATM), Amendment 9*. Doc 4444, Chapter 4, Section 4.9; Chapter 5, Section 5.8; Chapter 8, Section 8.7. 2020.

[17] Y. Jiao, H. Sun, C. Wang, and J. Han. "Research on Unstable Approach Detection of Civil Aviation Aircraft". *Procedia Computer Science* 131 (2018), pp. 525–530.

[18] B. Josefsson, J. Jakobi, A. Papenfuß, T. Polishchuk, C. Schmidt, and L. Sedov. "Identification of Complexity Factors for Remote Towers". *SESAR Innovation Days 2018*. Nov. 2018.

[19] A. Khattak, P.-W. Chan, and F. Chen. "Missed Approach, a Safety-Critical Go-Around Procedure in Low-Level Wind Shear". *Journal of Aerospace Engineering* 2023 (2023), p. D 9119521.

[20] A. Khattak, J. Zhang, P.-W. Chan, F. Chen, A. Hussain, and H. Almujibah. "Wind Shear and Aircraft Aborted Landings: A Deep Learning Perspective for Prediction and Analysis". *Atmosphere* 15.5 (2024).

[21] H.-Y. Lai, C.-H. Chen, L.-P. Khoo, and P. Zheng. "Unstable approach in aviation: Mental model disconnects between pilots and air traffic controllers and interaction conflicts". *Reliability Engineering System Safety* 185 (2019), pp. 383–391.

[22] V. I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals". *Soviet Physics Doklady* 10.8 (1966), pp. 707–710.

[23] K. Liu, K. Ding, L. Dai, M. Hansen, K. Chan, and J. Schade. "Real-Time Go-Around Prediction: A case study of JFK airport". *arXiv* (2024).

[24] S. M. Lundberg and S. Lee. "A unified approach to interpreting model predictions". *CoRR* abs/1705.07874 (2017).

[25] C. M. Marina, J. Pérez-Aracil, E. Lorente-Ramos, C. Casanova-Mateo, and S. Salcedo-Sanz. "Go-around occurrence prediction with rule-induction, rule evolution and Machine Learning algorithms". *Advanced Engineering Informatics* 65 (2025), p. 103171.

[26] A. D. Midtfjord, R. D. Bin, and A. B. Huseby. "A decision support system for safer airplane landings: Predicting runway conditions using XGBoost and explainable AI". *Cold Regions Science and Technology* 199 (2022), p. 103556.

[27] D. Moriarty and S. Jarvis. "A systems perspective on the unstable approach in commercial aviation". *Reliability Engineering System Safety* 131 (2014), pp. 197–202.

[28] National Weather Service (NWS). *National Weather Service*. 2025.

[29] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". *Advances in Neural Information Processing Systems* 32 (2019), pp. 8024–8035.

[30] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[31] J. W. Pratt and J. D. Gibbons. "Kolmogorov-Smirnov Two-Sample Tests". *Concepts of Nonparametric Theory*. New York: Springer, 1981, pp. 318–344.

[32] A. Savitzky and M. J. E. Golay. "Smoothing and Differentiation of Data by Simplified Least Squares Procedures". *Analytical Chemistry* 36.8 (1964), pp. 1627–1639.

[33] N. P. Singh, S. K. Goh, and S. Alam. "Real-time Unstable Approach Detection Using Sparse Variational Gaussian Process". *2020 International Conference on*

Artificial Intelligence and Data Analytics for Air Transportation (AIDA-AT). 2020, pp. 1–10.

[34] R. W. Sinnott. "Virtues of the Haversine". *Sky and Telescope* 68.2 (1984), p. 159.

[35] C. W. Tan, C. Bergmeir, F. Petitjean, and G. I. Webb. "Time Series Extrinsic Regression". *Data Mining and Knowledge Discovery* (2021), pp. 1–29.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need". *CoRR* abs/1706.03762 (2017).

[37] S. Weglarczyk. "Kernel density estimation and its application". *ITM Web of Conferences* 23 (Nov. 2018), p. 00037.

[38] P. Zippenfenig. *Open-Meteo.com Weather API*. 2023.

Appendix A

Appendix A - eDina Missed Approach Codes

Table A.1: Missed Approach Cause Codes and Descriptions

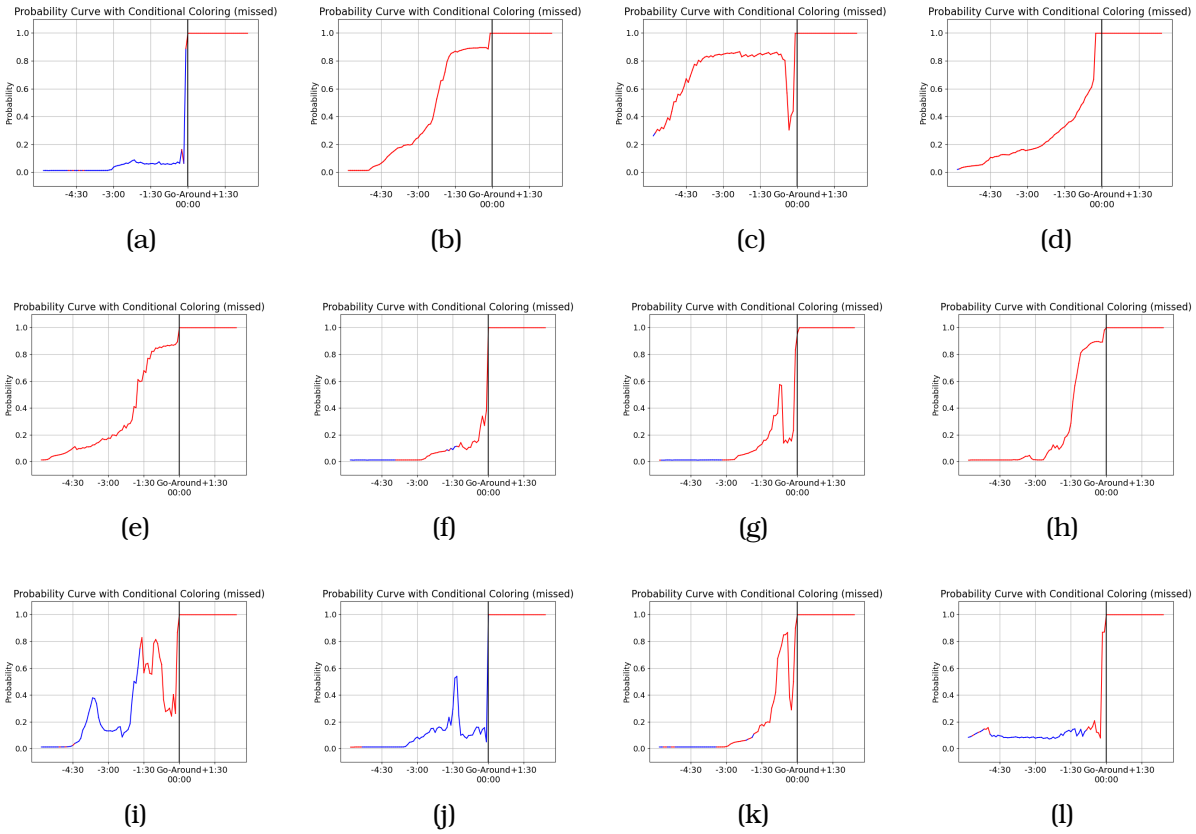
Code	Description
ALPI	Unstabilized approach according to crew notification.
MET_WS	Meteorological factors related to wind shear.
MET_TAI	Meteorological factors related to tailwind.
MET_VIS	Meteorological factors related to visibility.
MET_WI	Meteorological factors related to strong winds (not tailwind or crosswind).
MET_TUR	Meteorological factors related to turbulence (not wake turbulence).
MET_CRO	Meteorological factors related to crosswind.
MET_CLO	Meteorological factors related to cloud ceiling.
MET_OTH	Other meteorological factors not included above.
ISAP_RAD	Insufficient separation with the preceding arrival.
ISAP_WTC	Insufficient separation to maintain regulatory minima.
ISAP_LOA	Separation based on TWR-APP letter of agreement.
ISDP	Insufficient separation with the preceding departure.
ETP	Excessive runway occupancy time by preceding arrival.
WTC	Wake turbulence.
PIP_FOD	Presence of FOD (foreign object debris) on the runway.
PIP_ANI	Unauthorized presence of an animal on the runway.
PIP_OBS	Unauthorized presence of obstacles on the runway.

Continues in the next page.

Code	Description
PIP_VEH	Unauthorized presence of a vehicle on the runway.
PIP_ACF	Unauthorized presence of an aircraft on the runway.
PIP_PER	Unauthorized presence of personnel on the runway.
PIP_OTH	Other unauthorized presence on the runway.
COM	Communication problems or failure.
APE	Wrong or unauthorized runway approach.
RTA	Technical issues with the aircraft, according to crew notification.
OTH	Other cause not included in the above categories.

Appendix B

Appendix B - Multiple Probability Prediction Plots Colour-Coded by Observer Output

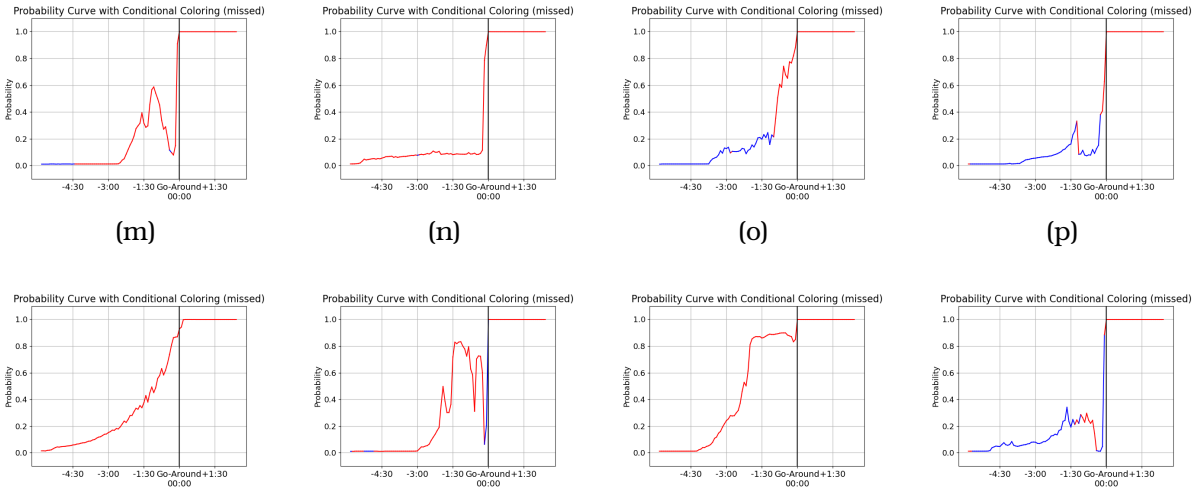


(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

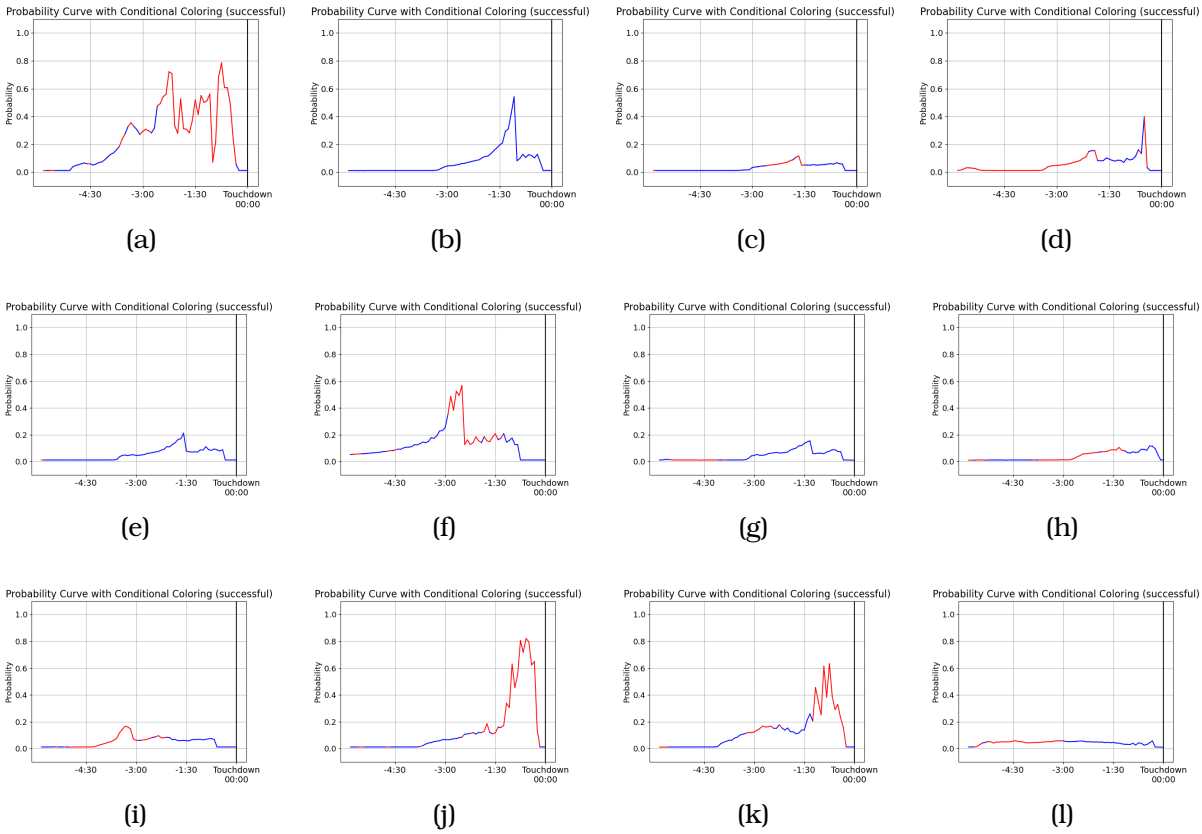
Figure B.1: Color-coded probability curves of missed approaches (1–12)



(m) (n) (o) (p)

(q) (r) (s) (t)

Figure B.1: Color-coded probability curves of missed approaches (13–20)



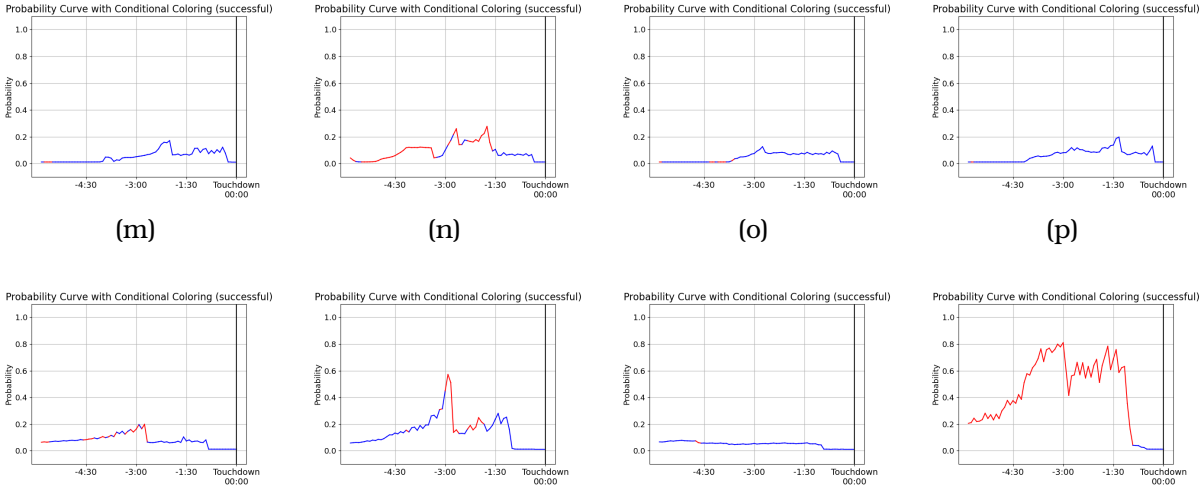
(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

Figure B.2: Color-coded probability curves of successful landings (1–12)

Appendix B - Multiple Probability Prediction Plots Colour-Coded by Observer Output



(m) (n) (o) (p)

(q) (r) (s) (t)

Figure B.2: Color-coded probability curves of successful landings (13–20)

Appendix C

Appendix C - Glossary of Acronyms

Table C.1: Glossary of Acronyms

Term	Definition
ADS-B	Automatic Dependent Surveillance–Broadcast
ANN	Artificial Neural Network
API	Application Programming Interface
ATC	Air Traffic Control
ATCO	Air Traffic Controller
AUAC	Area Under the Accuracy Curve
DH	Decision Height
DL	Deep Learning
DTI	Distance To Ideal
FAF	Final Approach Fix
FL	Flight Level
FNN	Feed-forward Neural Network
FN(R)	False Negative (Rate)
FP(R)	False Positive (Rate)
GA	Go-Around
GAM	Generalized additive model
IAF	Initial Approach Fix
IF	Intermediate Fix
KDE	Kernel Density Estimation
LEBL	Barcelona-El Prat Airport (ICAO code)

Continues in the next page.

Term	Definition
LEMD	Madrid-Barajas Airport (ICAO code)
LSTM	Long Short-Term Memory
MAPt	Missed Approach Point
MDA	Minimum Descent Altitude
ML	Machine Learning
NPV	Negative Predictive Value
PPV	Positive Predictive Value (Precision)
RECAT	Re-categorization (of wake turbulence classes)
RF	Random Forest
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SHAP	Shapley Additive exPlanations
TP(R)	True Positive (Rate)
TN(R)	True Negative (Rate)
TSC	Time Series Classification
TSER	Time Series Extrinsic Regression
TSF	Time Series Forecasting

Appendix D

Appendix D - Glossary of Important Terms

Aborted Landing: A landing that is interrupted and does not result in touchdown, typically leading to a go-around procedure.

Accuracy Curve: A visual plot showing how the model's classification accuracy evolves over time or at different decision thresholds.

Air Traffic Control (ATC): The system of ground-based personnel and technology that monitors and directs aircraft in controlled airspace to ensure safety and efficiency.

Air Traffic Controller (ATCO): A person responsible for directing aircraft on the ground and in the air to prevent collisions and optimize traffic flow.

Attention Mechanism: A method that allows models to dynamically focus on different parts of an input sequence when making predictions, improving performance in sequential tasks.

Deep Learning (DL): A subset of machine learning that uses neural networks with many layers to learn from large amounts of data, especially effective in pattern recognition tasks.

Feature Engineering: The process of creating new input variables from raw data to improve the performance of a machine learning model.

Go-Around: A maneuver performed by an aircraft that aborts its landing attempt and ascends to try again, usually due to unstable approach or external factors.

LandIA: A real-time prediction tool developed in this project that monitors ongoing approaches and estimates go-around probabilities to assist ATCOs.

Long Short-Term Memory (LSTM): A type of RNN designed to remember information over long sequences using special gates that control memory flow.

Neural Network: A machine learning model inspired by the human brain, composed of interconnected units (neurons) that process data through weighted connections.

Observer Model: A secondary model used to interpret the primary model's output, often translating continuous probabilities into binary decisions.

Probability Curve: A sequence of predicted probabilities over time, indicating the model's evolving confidence in a certain event (e.g., go-around).

Recurrent Neural Network (RNN): A type of neural network that processes sequential data by maintaining a memory of previous inputs through loops in the architecture.

Regression: A type of supervised learning where the model predicts a continuous output (as opposed to classification, which predicts discrete categories).

Sequence Modeling: A machine learning task involving inputs that are ordered (time series, text, etc.), where temporal relationships are important for predictions.

SHAP Values: Interpretability scores derived from game theory that quantify the contribution of each input feature to a model's output.

Trajectory: The path that an aircraft follows through space, typically described by its position, altitude, heading, and speed over time.

Transformer: A deep learning architecture based on attention mechanisms, highly effective in processing sequences such as text or time series without using recurrence.

Wake Turbulence: Disturbances in the air left behind by an aircraft, which can be hazardous to following aircraft, especially during approach and landing.